XMC7000 MCU family architecture

Reference manual

About this document

Scope and purpose

The XMC7000 device is a microcontroller targeted at industrial applications. These devices have up to two Arm® Cortex®-M7 CPUs for primary processing, and a Cortex®-M0+ CPU for peripheral and security processing.

Intended audience

This document is primarily intended for anyone who use the XMC7000 MCU device.

Table of contents

About this document . . . 1
Table of contents . . . 2
1 Introduction . . . 27
1.1 Device characteristics . . . 27
1.1.1 CPU subsystem . . . 27
1.1.2 Communication . . . 27
1.1.3 Memory interfaces . . . 28
1.1.3.1 Serial memory interface . . . 28
1.1.3.2 Secure digital high capacity . . . 28
1.1.4 Audio subsystem . . . 28
1.1.5 Miscellaneous . . . 28
1.2 Top level architecture . . . 29
1.2.1 CPU subsystem . . . 29
1.2.1.1 CPU . . . 29
1.2.1.2 DMA Controllers . . . 30
1.2.1.3 Flash . . . 30
1.2.1.4 SRAM with 32-KB retention granularity . . . 30
1.2.1.5 ROM . . . 30
1.2.1.6 Cryptography accelerator for security . . . 30
1.2.2 System resources . . . 30
1.2.2.1 Power system . . . 30
1.2.2.2 Regulators . . . 31
1.2.2.3 Clock system . . . 31
1.2.2.4 IMO clock source . . . 32
1.2.2.5 ILO clock source . . . 32
1.2.2.6 PLL and FLL . . . 32
1.2.2.7 Clock supervisor . . . 32
1.2.2.8 EXT_CLK . . . 32
1.2.2.9 ECO . . . 32
1.2.2.10 WCO . . . 32
1.2.2.11 Reset . . . 32
1.2.2.12 Power modes . . . 33
1.2.3 Peripherals . . . 33
1.2.3.1 Peripheral clock dividers . . . 33
1.2.3.2 Peripheral protection unit . . . 33
1.2.3.3 12-bit SAR ADC . . . 33
1.2.3.4 Timer/counter/PWM block (TCPWM) . . . 34
1.2.3.5 Serial communication blocks (SCB) . . . 34
1.2.3.6 CAN FD . . . 34

Table of contents

1.2.3.7 Ethernet MAC . . . 35
1.2.3.8 External memory interface . . . 35
1.2.3.9 SDHC interface . . . 35
1.2.3.10 One-time-programmable (OTP) eFuse . . . 35
1.2.3.11 Event generator . . . 35
1.2.3.12 Trigger multiplexer . . . 35
1.2.4 I/Os . . . 36
1.2.4.1 GPIO . . . 36
1.2.4.2 HSIO . . . 36
1.2.4.3 Drive modes . . . 36
1.2.4.4 Port nomenclature . . . 36
1.2.4.5 Smart I/O . . . 36

2 Getting started . . . 37
2.1 Support . . . 37
2.2 Product upgrades . . . 37
2.3 Development kits . . . 37
2.3.1 Evaluation board . . . 37
2.3.2 Peripheral driver library (PDL) . . . 38
2.3.3 Development tools . . . 38
2.3.4 Open OCD . . . 38
2.4 Application notes . . . 38

3 Document construction . . . 39
3.1 Major sections . . . 39
3.2 Documentation conventions . . . 39
3.2.1 Register conventions . . . 39
3.2.2 Numeric naming . . . 39
3.2.3 Unit of measure . . . 39
3.2.4 Acronyms and abbreviations . . . 41

4 CPU subsystem (CPUSS) . . . 45
4.1 Features . . . 45
4.2 How it works . . . 46
4.3 Address map . . . 46
4.4 Operating modes and privilege levels . . . 47
4.5 Instruction set . . . 47
4.6 TCM interface . . . 47
4.6.1 TCM read wait state . . . 48
4.6.2 TCM ECC . . . 48
4.6.2.1 TCM read modify write configuration . . . 48
4.6.2.2 ECC failure detection . . . 50
4.6.2.3 ECC fault reporting . . . 50
4.6.2.4 TCM initialization . . . 50

Table of contents

4.6.2.5 ECC error injection . . . 51
4.6.2.6 ECC parity generation by software . . . 52
4.6.2.7 TCM enabling . . . 53
4.7 Cache ECC fault reporting . . . 55
4.8 Registers . . . 55
4.8.1 Arm® specification registers . . . 55
4.8.2 MCU control registers . . . 58
5 Inter-processor communication . . . 59
5.1 Features . . . 59
5.1.1 IPC channel . . . 59
5.1.2 IPC interrupt . . . 60
5.1.3 IPC channels and interrupts . . . 60
5.2 Implementing locks . . . 61
5.3 Message passing . . . 61
5.4 Registers . . . 63
6 Protection unit . . . 64
6.1 Features . . . 64
6.2 Configuration . . . 65
6.2.1 Block diagram . . . 65
6.2.2 Protection unit structure . . . 65
6.2.3 Master with missing access attributes . . . 66
6.3 Protection context . . . 66
6.3.1 Protection context configuration . . . 66
6.3.2 Protection context 0 and 1 . . . 67
6.4 Protection structure . . . 68
6.4.1 Address region . . . 68
6.4.2 Access control attributes . . . 69
6.4.3 Protection violation . . . 70
6.4.4 Protection of protection structures . . . 70
6.4.5 MPU . . . 71
6.4.6 SMPU . . . 71
6.4.7 PPU . . . 72
6.4.7.1 ECC for SRAM . . . 74
6.4.7.2 ECC error injection . . . 74
6.4.7.3 ECC parity generation by software . . . 74
6.4.8 Protection structure types . . . 76
6.5 SWPU . . . 79
6.6 SWPU layout . . . 80
6.6.1 SWPU configuration . . . 81
6.7 Registers . . . 85
7 Direct memory access . . . 88

Table of contents

7.1 Peripheral DMA (P-DMA(DW)) . . . 88
7.1.1 Overview . . . 88
7.1.2 Channels . . . 89
7.1.3 Descriptors . . . 90
7.1.4 Interrupts . . . 92
7.1.5 P-DMA Controller Status registers . . . 93
7.1.6 P-DMA controller design . . . 94
7.1.6.1 P-DMA channel configuration SRAMs . . . 96
7.1.6.2 ECC for P-DMA channel configuration SRAMs . . . 96
7.1.7 Functionality . . . 98
7.1.7.1 Example 1 - Single transfer . . . 98
7.1.7.2 Example 2 - 2D transfer . . . 99
7.1.8 P-DMA descriptor structure . . . 100
7.2 Memory DMA (M-DMA (DMAC)) . . . 105
7.2.1 Overview . . . 105
7.2.2 Channels . . . 106
7.2.3 Descriptors . . . 107
7.2.4 Interrupts . . . 110
7.2.5 Control and Active registers . . . 110
7.2.6 M-DMA controller design . . . 110
7.2.7 Examples . . . 111
7.2.8 M-DMA descriptor structure . . . 113
7.3 Registers . . . 119

8 Code flash . . . 125
8.1 Features . . . 125
8.2 Configuration . . . 125
8.2.1 Block diagram . . . 125
8.2.2 Flash controller . . . 126
8.2.2.1 Bus error . . . 127
8.2.2.2 Wait cycle count . . . 128
8.2.2.3 Power modes . . . 129
8.2.2.4 CM0+ CPU cache . . . 129
8.2.2.5 Code flash ECC . . . 131
8.2.2.6 Software generating code flash ECC . . . 133
8.2.2.7 Cache ECC . . . 134
8.2.2.8 Software generating cache ECC . . . 135
8.2.3 Flash geometry . . . 136
8.2.3.1 Interface, regions, and type of use . . . 136
8.2.3.2 Geometries . . . 136
8.2.3.3 Logical bank . . . 137
8.2.4 Over-the-air (OTA) support . . . 138
8.2.4.1 Dual bank mode and remap functionality . . . 138

Table of contents

8.2.4.2 Address mapping for 4 MB memory . . . 139
8.2.4.3 Address mapping for 8 MB memory . . . 140
8.3 Operation . . . 141
8.3.1 SROM APIs . . . 141
8.4 Registers . . . 141

9 Work flash . . . 143
9.1 Features . . . 143
9.2 Configuration . . . 143
9.2.1 Block diagram . . . 143
9.2.2 Flash controller . . . 144
9.2.2.1 Bus error . . . 144
9.2.2.2 Work flash ECC . . . 145
9.2.2.3 Software generating work flash ECC . . . 147
9.2.3 Flash geometry . . . 147
9.2.3.1 Interface, regions, and type of use . . . 147
9.2.3.2 Geometries . . . 148
9.2.3.3 Logical bank . . . 150
9.2.4 Over-the-air (OTA) support . . . 150
9.2.4.1 Dual bank mode and remap functionality . . . 150
9.2.5 Address map of work flash . . . 151
9.2.5.1 Address mapping for 128 KB memory . . . 151
9.2.5.2 Address mapping for 256 KB memory . . . 152
9.3 Operation . . . 153
9.3.1 Read . . . 153
9.3.2 SROM APIs . . . 154
9.4 Registers . . . 154

10 SRAM interface . . . 156
10.1 Features . . . 156
10.2 Configuration . . . 157
10.2.1 Block diagram . . . 157
10.2.2 Wait states . . . 158
10.2.3 Operation . . . 158
10.2.4 Write buffer . . . 159
10.3 ECC details . . . 159
10.3.1 ECC parity generation for SRAM write accesses . . . 160
10.3.2 ECC syndrome generation for SRAM read accesses . . . 160
10.3.3 ECC error injection . . . 160
10.3.4 ECC parity generation by software . . . 161
10.4 RAM retention configuration . . . 162
10.5 Registers . . . 163

11 BootROM . . . 164

Table of contents

11.1 Features . . . 164
11.2 ROM controller . . . 164
11.2.1 Wait states . . . 164
11.3 ROM boot process . . . 165
11.3.1 Life-cycle stages and protection states . . . 165
11.3.2 Multicore boot . . . 165
11.3.3 “Secure Boot” . . . 165
11.3.4 Protection setting . . . 165
11.3.4.1 SMPU configuration in SFlash . . . 166
11.3.4.2 SWPU configuration in SFlash . . . 166
11.3.4.3 PPU configuration in SFlash . . . 167
11.3.4.4 Boot protection settings in SFlash . . . 169
11.3.4.5 Security enhancement PPU configuration in SFlash . . . 170
11.3.5 Debug and test access restrictions . . . 172
11.3.6 SWD/JTAG initialization . . . 172
11.3.7 Waking up from hibernate . . . 172
11.3.8 ROM boot flow chart . . . 173
11.4 MMIO Registers and eFuse used by ROM boot . . . 174
11.4.1 MMIO Registers . . . 174
11.4.2 eFuse bits . . . 175

12 Interrupts . . . 178
12.1 Features . . . 178
12.2 How it works . . . 179
12.3 Interrupts and exceptions – operation . . . 180
12.3.1 Interrupt/exception handling . . . 180
12.3.2 Level interrupts . . . 181
12.3.3 Exception vector table . . . 181
12.4 Exception sources . . . 184
12.4.1 Reset exception . . . 184
12.4.2 Non-maskable interrupt exception . . . 184
12.4.3 HardFault exception . . . 184
12.4.4 Memory management fault exception . . . 184
12.4.5 Bus fault exception . . . 185
12.4.6 Usage fault exception . . . 185
12.4.7 Supervisor call (SVCall) exception . . . 185
12.4.8 PendSV exception . . . 185
12.4.9 SysTick exception . . . 186
12.5 Interrupt sources . . . 186
12.6 Exception priority . . . 188
12.7 Enabling and disabling interrupts . . . 188
12.8 Exception states . . . 189
12.8.1 Pending exceptions . . . 190

Table of contents

12.9 Stack usage for exceptions . . . 190
12.10 Interrupts and low-power modes . . . 191
12.11 Exception – initialization and configuration . . . 191
12.12 Registers . . . 192

13 Device security . . . 196
13.1 Features . . . 196
13.2 How it works . . . 196
13.2.1 Life-cycle stages . . . 196
13.2.2 Memory and peripheral protection . . . 197
13.2.3 Flash write and eFuse read/write protection . . . 198
13.2.4 Hardware-based cryptography . . . 198

14 Chip operational modes . . . 199
14.1 Boot . . . 199
14.2 User . . . 199
14.3 Trusted . . . 199
14.4 Debug . . . 199

15 Fault subsystem . . . 200
15.1 Fault report structure . . . 200
15.2 Fault and reset . . . 202
15.3 Fault and power modes . . . 202
15.4 Registers . . . 203

16 Power supply and monitoring . . . 204
16.1 Features . . . 205
16.2 Power supply . . . 205
16.2.1 Core regulators . . . 206
16.2.2 Power pins and rails . . . 207
16.2.3 Power sequencing requirements . . . 207
16.2.4 Power supply sources . . . 207
16.2.5 Usage of high-current regulator controller . . . 208
16.2.5.1 Transitioning from active regulator to high-current regulator controller with external transistor . . . 211
16.2.5.2 Transitioning from high-current regulator controller with external transistor to active regulator . . . 211
16.2.5.3 Transitioning from active regulator to high-current regulator controller with external PMIC . . . 213
16.2.5.4 Transitioning from high-current regulator controller with external PMIC to active regulator . . . 215
16.2.5.5 Internal regulator configuration when using PMIC . . . 216
16.2.5.6 Transitioning from active/sleep to reset . . . 217
16.2.5.7 Deep sleep entry/exit . . . 219
16.3 Voltage monitoring . . . 219

Table of contents

16.3.1 Power-on-reset (POR) . . . 220
16.3.2 Brownout-detection (BOD) . . . 220
16.3.2.1 BOD on V_{DDD} . . . 220
16.3.2.2 BOD on V_{DDA} . . . 220
16.3.2.3 BOD on V_{CCD} . . . 221
16.3.3 Over-voltage detection (OVD) . . . 221
16.3.3.1 OVD on V_{DDD} . . . 221
16.3.3.2 OVD on V_{DDA} . . . 221
16.3.3.3 OVD on V_{CCD} . . . 221
16.3.4 Low-voltage-detection (LVD) . . . 221
16.3.5 Over-current detection . . . 222
16.3.6 Voltage monitoring by ADC . . . 222
16.4 Registers . . . 225

17 Device power modes . . . 226
17.1 Features . . . 226
17.2 Device power modes . . . 226
17.2.1 Active and sleep modes . . . 227
17.2.1.1 Low-power profiles - LPACTIVE and LPSLEEP . . . 228
17.2.1.2 CM7 power domains . . . 228
17.2.2 Deep sleep mode . . . 229
17.2.3 Hibernate mode . . . 230
17.2.4 Other operational states . . . 231
17.2.4.1 XRES/OFF state . . . 231
17.2.4.2 Reset . . . 231
17.3 Power mode transitions . . . 232
17.3.1 Power-up transitions . . . 233
17.3.2 Low-power mode transition . . . 234
17.3.3 Wakeup . . . 237
17.3.4 Internal reset transitions . . . 238
17.3.5 Powering down/brownout/overvoltage . . . 238
17.3.6 Debugger effect on device power modes . . . 238
17.4 Summary . . . 239
17.5 Registers . . . 240

18 Clocking system . . . 241
18.1 Block diagram . . . 241
18.2 Clock sources . . . 243
18.2.1 Internal main oscillator (IMO) . . . 243
18.2.2 External crystal oscillator (ECO) . . . 243
18.2.2.1 ECO trimming . . . 243
18.2.3 External clock (EXT_CLK) . . . 245
18.2.4 Internal low-speed oscillator (ILO) . . . 245

Table of contents

18.2.5 Watch crystal oscillator (WCO) . . . 246
18.2.6 ECO prescaler . . . 246
18.3 Clock generation . . . 246
18.3.1 PLL without SSCG and fractional operation . . . 246
18.3.2 PLL with SSCG and fractional operation (400-MHz PLL) . . . 247
18.3.2.1 Spread spectrum clock generation (SSCG) . . . 247
18.3.2.2 Fractional operation . . . 248
18.3.3 Frequency locked loop (FLL) . . . 249
18.4 Clock trees . . . 251
18.4.1 Path clocks . . . 251
18.4.2 High-frequency root clocks . . . 252
18.4.3 Low-frequency root clocks . . . 252
18.4.4 Timer clock . . . 252
18.4.5 Clock output function . . . 253
18.5 CLK_HF distribution . . . 253
18.5.1 CLK_MEM . . . 253
18.5.2 CLK_SLOW . . . 253
18.5.3 CLK_FAST_0 and CLK_FAST_1 . . . 253
18.5.4 CLK_PERI . . . 253
18.5.5 PCLK . . . 253
18.5.6 CLK_GR . . . 254
18.5.7 CLK_TRC_DBG . . . 254
18.6 Peripheral clock dividers . . . 254
18.6.1 Fractional clock dividers . . . 254
18.6.2 Peripheral clock divider configuration . . . 254
18.6.2.1 Phase aligning dividers . . . 255
18.7 Clock calibration counters . . . 256
18.8 Clock supervision (CSV) . . . 257
18.8.1 Overview . . . 257
18.8.2 CSV operation . . . 257
18.9 Registers . . . 261

19 Reset system . . . 266
19.1 Reset sources . . . 266
19.1.1 Power-on reset (POR) . . . 268
19.1.2 Brownout detection (BOD) reset . . . 268
19.1.3 Overvoltage detection (OVD) reset . . . 268
19.1.4 Overcurrent reset . . . 268
19.1.5 External reset . . . 268
19.1.6 Watchdog timer reset (WDT) . . . 269
19.1.7 Internal system reset . . . 269
19.1.8 Fault detection reset . . . 269
19.1.9 Clock-supervision reset . . . 269

Table of contents

19.1.10 Hibernate wakeup reset . . . 269
19.1.11 PMIC reset . . . 269
19.2 Identifying reset sources . . . 270
19.3 Registers . . . 271
20 Watchdog timer . . . 272
20.1 Features . . . 272
20.2 Block diagram . . . 273
20.3 Basic watchdog timer . . . 273
20.3.1 Overview . . . 273
20.3.2 Watchdog reset . . . 277
20.3.3 Watchdog interrupt . . . 278
20.4 Multi-counter watchdog timer . . . 279
20.4.1 Overview . . . 279
20.4.2 How it works . . . 280
20.4.2.1 Subcounter 0/1 operation . . . 280
20.4.2.2 32-bit counter operation . . . 286
20.4.3 Enabling and disabling MCWDT . . . 288
20.4.4 Watchdog reset . . . 289
20.4.5 Watchdog interrupt . . . 290
20.5 Reset cause detection . . . 290
20.6 Debug mode . . . 291
20.7 CPU select . . . 291
20.8 Registers . . . 292
21 Real-time clock . . . 294
21.1 Features . . . 294
21.2 Block diagram . . . 294
21.3 Power supply . . . 295
21.4 Clocking . . . 295
21.5 Reset . . . 295
21.6 Real-time clock . . . 296
21.6.1 Reading RTC User registers . . . 297
21.6.2 Writing to RTC User registers . . . 297
21.7 WCO calibration . . . 297
21.7.1 Absolute accuracy calibration . . . 298
21.8 Alarm feature . . . 298
21.9 Backup registers . . . 299
21.10 Real time Clock registers . . . 299
22 I/O system . . . 301
22.1 Features . . . 301
22.2 GPIO interface overview . . . 302
22.3 I/O cell architecture . . . 303

Table of contents

22.4 High speed I/O (HSIO) . . . 304
22.5 Digital input buffer . . . 304
22.6 Digital output driver . . . 305
22.6.1 Drive modes . . . 305
22.6.2 Slew rate control . . . 309
22.7 High-speed I/O matrix . . . 311
22.8 I/O state on power up . . . 313
22.9 Behavior in low-power modes . . . 314
22.10 Interrupt . . . 314
22.11 Peripheral connections . . . 316
22.11.1 Firmware-controlled GPIO . . . 316
22.11.2 Analog I/O . . . 317
22.11.3 Serial communication block (SCB) . . . 317
22.12 Smart I/O . . . 317
22.12.1 Overview . . . 317
22.12.2 Block components . . . 318
22.12.2.1 Clock and reset . . . 318
22.12.2.2 Synchronizer . . . 319
22.12.2.3 LUT . . . 320
22.12.2.4 Data unit . . . 324
22.12.3 Routing . . . 327
22.12.4 Operation . . . 328
22.12.5 Example application . . . 329
22.13 Registers . . . 334

23 Serial communications block (SCB) . . . 337
23.1 Features . . . 337
23.2 Block diagram . . . 338
23.2.1 AHB-Lite bus interface . . . 338
23.2.2 Trigger interface . . . 339
23.2.2.1 DMA/DW trigger signals . . . 339
23.2.2.2 SCB_I2C_SCL_TR_OUT[x] signal . . . 339
23.2.2.3 Serial protocol interfaces . . . 339
23.2.3 Clock and reset interface . . . 339
23.2.4 Block enable . . . 340
23.2.5 Interrupt interface . . . 340
23.3 Operation modes . . . 342
23.3.1 Buffer modes . . . 342
23.3.1.1 FIFO mode . . . 342
23.3.1.2 EZ mode . . . 343
23.3.1.3 CMD_RESP mode . . . 343
23.3.2 Clocking modes . . . 343
23.4 Serial peripheral interface (SPI) . . . 344

Table of contents

23.4.1 Features . . . 344
23.4.2 General description . . . 345
23.4.3 SPI modes of operation . . . 346
23.4.3.1 Motorola SPI . . . 346
23.4.3.2 Texas Instruments SPI . . . 349
23.4.3.3 National Semiconductors SPI . . . 351
23.4.4 SPI buffer modes . . . 352
23.4.4.1 FIFO mode . . . 352
23.4.4.2 EZSPI mode . . . 352
23.4.4.3 Command-response mode . . . 355
23.4.5 Clocking and oversampling . . . 357
23.4.5.1 Clock modes . . . 357
23.4.5.2 Using SPI master to clock slave . . . 359
23.4.5.3 Oversampling and bit rate . . . 359
23.4.6 SPI master SELECT output timing control . . . 360
23.4.7 SPI parity functionality . . . 362
23.4.8 Loop-back . . . 362
23.4.9 Enabling and initializing SPI . . . 363
23.4.10 I/O pad connection . . . 363
23.4.10.1 SPI master . . . 363
23.4.10.2 SPI slave . . . 364
23.4.11 SPI Registers . . . 366
23.5 UART . . . 366
23.5.1 Features . . . 366
23.5.2 General description . . . 367
23.5.3 UART modes of operation . . . 367
23.5.3.1 Standard protocol . . . 367
23.5.3.2 UART multi-processor mode . . . 371
23.5.3.3 SmartCard (ISO 7816) . . . 372
23.5.3.4 IrDA . . . 373
23.5.4 Clocking and oversampling . . . 374
23.5.5 Loop-back . . . 375
23.5.6 Enabling and initializing UART . . . 375
23.5.7 I/O pad connection . . . 375
23.5.7.1 Standard UART mode . . . 375
23.5.7.2 SmartCard mode . . . 377
23.5.7.3 IrDA mode . . . 377
23.5.8 UART Registers . . . 378
23.6 Inter integrated circuit (I^2C) . . . 378
23.6.1 Features . . . 378
23.6.2 General description . . . 379
23.6.3 Terms and definitions . . . 379

Table of contents

23.6.3.1 Clock stretching . . . 379
23.6.3.2 Bus arbitration . . . 380
23.6.4 I²C modes of operation . . . 380
23.6.4.1 Write transfer . . . 381
23.6.4.2 Read transfer . . . 382
23.6.5 I²C buffer modes . . . 382
23.6.5.1 FIFO mode . . . 382
23.6.5.2 EZI2C mode . . . 383
23.6.5.3 Command-response mode . . . 385
23.6.6 Clocking and oversampling . . . 387
23.6.6.1 Glitch filtering . . . 388
23.6.6.2 Oversampling and bit rate . . . 389
23.6.7 Loop-back . . . 391
23.6.8 Enabling and initializing the I²C . . . 391
23.6.8.1 Configuring for I²C FIFO mode . . . 391
23.6.8.2 Configuring for EZ and CMD_RESP modes . . . 392
23.6.9 I/O pad connections . . . 392
23.6.10 I²C Registers . . . 392
23.7 SCB interrupts . . . 393
23.7.1 SPI interrupts . . . 393
23.7.2 UART interrupts . . . 394
23.7.3 I²C interrupts . . . 395
23.8 Registers . . . 396
23.8.1 SPI Registers . . . 396
23.8.2 UART Registers . . . 397
23.8.3 I²C Registers . . . 398

24 CAN FD controller . . . 400
24.1 Overview . . . 400
24.1.1 Features . . . 400
24.1.2 Features not supported . . . 401
24.2 Configuration . . . 401
24.2.1 Block diagram . . . 401
24.2.2 Dual clock sources . . . 401
24.2.3 Interrupt lines . . . 401
24.3 Functional description . . . 403
24.3.1 Operation modes . . . 403
24.3.1.1 Software initialization . . . 403
24.3.1.2 Normal operation . . . 404
24.3.1.3 CAN FD operation . . . 404
24.3.1.4 Transmitter delay compensation . . . 405
24.3.1.5 Restricted operation mode . . . 407
24.3.1.6 Bus monitoring mode . . . 407

Table of contents

24.3.1.7 Disable automatic retransmission . . . 407
24.3.1.8 Power down (sleep mode) . . . 408
24.3.1.9 Test mode . . . 408
24.3.1.10 Application watchdog . . . 409
24.3.2 Timestamp generation . . . 410
24.3.3 Timeout counter . . . 410
24.3.4 RX handling . . . 411
24.3.4.1 Acceptance filtering . . . 411
24.3.4.2 RX FIFOs . . . 414
24.3.4.3 Dedicated RX buffers . . . 417
24.3.4.4 Debug on CAN support . . . 418
24.3.5 TX handling . . . 419
24.3.5.1 Transmit pause . . . 420
24.3.5.2 Dedicated TX buffers . . . 420
24.3.5.3 TX FIFO . . . 421
24.3.5.4 TX queue . . . 421
24.3.5.5 Mixed dedicated TX buffers/TX FIFO . . . 422
24.3.5.6 Mixed dedicated TX buffers/TX queue . . . 422
24.3.5.7 Transmit cancellation . . . 423
24.3.5.8 TX event handling . . . 423
24.3.6 FIFO acknowledge handling . . . 423
24.3.7 Configuring the CAN bit timing . . . 424
24.3.7.1 CAN bit timing . . . 424
24.3.7.2 CAN bit rates . . . 426
24.4 Message RAM . . . 427
24.4.1 Message RAM configuration . . . 427
24.4.2 RX buffer and FIFO element . . . 428
24.4.3 TX buffer element . . . 431
24.4.4 TX event FIFO element . . . 434
24.4.5 Standard message ID filter element . . . 435
24.4.6 Extended message ID filter element . . . 437
24.4.7 Trigger memory element . . . 439
24.4.8 ECC for message RAM . . . 441
24.4.8.1 Correctable ECC error . . . 441
24.4.8.2 Non-correctable ECC error . . . 441
24.4.8.3 Address error . . . 442
24.4.8.4 ECC error injection . . . 442
24.4.8.5 ECC parity generation by software . . . 443
24.4.9 Message RAM OFF . . . 443
24.4.10 RAM watchdog (RWD) . . . 444
24.5 TTCAN operation . . . 444
24.5.1 Reference message . . . 444

Table of contents

24.5.1.1 Level 1 . . . 444
24.5.1.2 Level 2 . . . 445
24.5.1.3 Level 0 . . . 445
24.5.2 TTCAN configuration . . . 445
24.5.2.1 TTCAN timing . . . 445
24.5.2.2 Message scheduling . . . 447
24.5.2.3 Trigger memory . . . 448
24.5.2.4 TTCAN schedule initialization . . . 451
24.5.3 TTCAN gap control . . . 452
24.5.4 Stop watch . . . 453
24.5.5 Local time, cycle time, global time, and external clock synchronization . . . 453
24.5.6 Synchronization triggers . . . 456
24.5.7 TTCAN error level . . . 456
24.5.8 TTCAN message handling . . . 457
24.5.8.1 Reference message . . . 457
24.5.8.2 Message reception . . . 458
24.5.8.3 Message transmission . . . 458
24.5.9 TTCAN interrupt and error handling . . . 460
24.5.10 Level 0 . . . 460
24.5.10.1 Synchronizing . . . 461
24.5.10.2 Handling error levels . . . 461
24.5.10.3 Master slave relation . . . 462
24.5.11 Synchronization to external time schedule . . . 463
24.6 Setup procedures . . . 464
24.6.1 General program flow . . . 464
24.6.2 Clock stop request . . . 465
24.6.3 Message RAM OFF operation . . . 465
24.6.4 Message RAM ON operation . . . 466
24.6.5 Consolidated interrupt handling . . . 466
24.6.6 Procedures specific to M_TTCAN channel . . . 467
24.6.6.1 CAN bus configuration . . . 468
24.6.6.2 Message RAM configuration . . . 468
24.6.6.3 Interrupt configuration . . . 471
24.6.6.4 Transmit frame configuration . . . 472
24.6.6.5 Interrupt handling . . . 472
24.7 Registers . . . 478

25 Timer, counter, and PWM . . . 483
25.1 Features . . . 483
25.2 Block diagram . . . 484
25.2.1 Enabling and disabling counters in TCPWM block . . . 484
25.2.2 Clocking . . . 485
25.2.2.1 Clock prescaling . . . 485

Table of contents

25.2.2.2 Count event . . . 485
25.2.3 Trigger inputs . . . 485
25.2.4 Synchronization of multiple counters . . . 491
25.2.5 Trigger outputs . . . 492
25.2.6 Internal events . . . 493
25.2.6.1 Underflow event . . . 493
25.2.6.2 Overflow event . . . 493
25.2.6.3 TC event . . . 494
25.2.6.4 cc0_match (cc1_match) event . . . 495
25.2.7 Interrupts . . . 496
25.2.8 Debug mode . . . 497
25.2.9 PWM outputs . . . 498
25.2.10 Power modes . . . 500
25.3 Operation modes . . . 501
25.3.1 Timer mode . . . 502
25.3.1.1 Configuring counter for timer mode . . . 508
25.3.2 Capture mode . . . 509
25.3.2.1 Configuring counter for capture mode . . . 512
25.3.3 Quadrature decoder mode . . . 512
25.3.3.1 Quadrature QUAD_RANGE0 mode . . . 516
25.3.3.2 Configuring counter for quadrature mode (QUAD_RANGE0 mode) . . . 518
25.3.3.3 Quadrature QUAD_RANGE0_CMP mode . . . 519
25.3.3.4 Quadrature QUAD_RANGE1_CMP mode . . . 521
25.3.3.5 Quadrature QUAD_RANGE1_CAPT mode . . . 524
25.3.4 Pulse width modulation (PWM) mode . . . 524
25.3.4.1 PWM mode functionalities . . . 528
25.3.4.2 Configuring Counter for PWM Mode for Stepper Motor Control (SMC) . . . 540
25.3.4.3 Configuring counter for PWM mode . . . 547
25.3.5 Pulse width modulation with dead time mode . . . 547
25.3.5.1 Configuring counter for PWM with dead time mode . . . 549
25.3.6 Pulse width modulation pseudo-random mode (PWM PR) . . . 549
25.3.6.1 Configuring counter for pseudo-random PWM mode . . . 554
25.3.7 Shift Register (SR) . . . 555
25.3.7.1 SR mode functionality overview . . . 555
25.3.7.2 Features of SR mode . . . 557
25.4 Recovery . . . 558
25.5 Initialize . . . 558
25.6 Pin status . . . 558
25.7 Registers . . . 559

26 Cryptography block . . . 561
26.1 Features overview . . . 561
26.2 System diagram . . . 561

Table of contents

26.3 Block diagram . . . 562
26.4 Function description . . . 563
26.4.1 Operating mode . . . 563
26.4.2 Memory map and register definitions . . . 563
26.4.3 Instruction set . . . 563

27 Event generator (EVTGEN) . . . 564
27.1 Features . . . 564
27.2 Block diagram . . . 564
27.2.1 Enabling and disabling EVTGEN block . . . 565
27.2.2 Counters . . . 565
27.2.2.1 Clock and prescaling . . . 565
27.2.2.2 Ratio . . . 566
27.2.2.3 Software control . . . 566
27.2.2.4 Hardware control . . . 566
27.2.3 Counter status . . . 566
27.2.4 Deep Sleep Counter update . . . 567
27.2.5 Comparator structures . . . 568
27.2.6 Interrupts . . . 570
27.2.7 Usage Guidelines . . . 571
27.2.8 Deep sleep interrupt accuracy analysis . . . 571
27.2.8.1 Higher the RATIO, lower the error . . . 572
27.2.9 Use case . . . 572
27.2.10 Register list . . . 573

28 Trigger multiplexer . . . 575
28.1 Features . . . 575
28.2 Description . . . 575
28.3 Trigger multiplexing . . . 576
28.4 Trigger functionality . . . 578
28.5 Registers . . . 580

29 Ethernet MAC . . . 581
29.1 Overview . . . 581
29.1.1 Supported features . . . 581
29.2 Block diagram . . . 582
29.3 Ethernet MAC operation . . . 582
29.3.1 DMA interface . . . 582
29.3.1.1 AXI interface . . . 583
29.3.1.2 Partial store and forward using packet buffer DMA . . . 583
29.3.1.3 Full store and forward mode using packet buffer DMA . . . 583
29.3.1.4 DMA transaction . . . 584
29.3.1.5 Receive buffers . . . 584
29.3.1.6 Transmit buffers . . . 589

Table of contents

29.3.1.7 DMA burst . . . 592
29.3.1.8 DMA packet buffer . . . 592
29.3.1.9 Priority queuing in EMAC DMA . . . 594
29.3.2 Transmit scheduling algorithm . . . 595
29.3.2.1 802.1Qav support - credit based shaping . . . 596
29.3.2.2 Fixed priority . . . 596
29.3.2.3 Deficit weighted round robin (DWRR) . . . 596
29.3.2.4 Enhanced transmission selection (ETS) . . . 597
29.3.3 MAC transmitter . . . 597
29.3.4 MAC receiver . . . 598
29.3.5 Checksum offload for IP, TCP, and UDP . . . 598
29.3.5.1 Receive checksum offload . . . 598
29.3.5.2 Transmit checksum offload . . . 599
29.3.6 Jumbo frame support . . . 599
29.3.7 MAC filtering block . . . 600
29.3.7.1 Broadcast address . . . 601
29.3.7.2 Hash addressing . . . 601
29.3.7.3 Copy all frames . . . 602
29.3.7.4 Disable copy of pause frames . . . 602
29.3.7.5 VLAN support . . . 602
29.3.8 IEEE 1588 and IEEE 802.1AS support . . . 603
29.3.8.1 Support for time stamping and timestamp accuracy . . . 607
29.3.8.2 Single step time stamping . . . 607
29.3.8.3 Timestamp capture registers . . . 608
29.3.8.4 Timestamp capture in DMA descriptors . . . 608
29.3.8.5 Controlling timestamp unit . . . 608
29.3.8.6 Increment mode . . . 608
29.3.8.7 Alternate increment mode . . . 609
29.3.9 MAC 802.3 pause frame support . . . 609
29.3.9.1 IEEE Std 802.3 pause frame reception . . . 609
29.3.9.2 IEEE Std 802.3 pause frame transmission . . . 609
29.3.10 MAC PFC priority-based pause frame support . . . 610
29.3.10.1 PFC pause frame reception . . . 610
29.3.10.2 PFC pause frame transmission . . . 610
29.3.11 Energy efficient Ethernet support . . . 611
29.3.11.1 LPI operation in EMAC . . . 612
29.3.12 MDIO interface . . . 612
29.3.13 Interrupts . . . 612
29.3.14 Media independent interfaces . . . 613
29.3.14.1 MII . . . 613
29.3.14.2 RGMII . . . 613
29.3.14.3 RMII . . . 614

Table of contents

29.3.14.4 Clock sources for PHY interface . . . 615
29.3.14.5 PHY interface . . . 616
29.3.15 Clocks to EMAC . . . 617
29.3.15.1 Power modes . . . 617
29.4 Registers . . . 618
30 Serial memory interface . . . 628
30.1 Features . . . 628
30.2 System diagram . . . 629
30.3 Block diagram . . . 629
30.3.1 Clocks . . . 630
30.4 Functional description . . . 631
30.4.1 Operating modes . . . 631
30.4.1.1 MMIO_MODE . . . 631
30.4.1.2 XIP_MODE . . . 632
30.4.1.3 Continuous transfer merging . . . 632
30.4.2 Off-chip interfaces . . . 635
30.4.2.1 Clock polarity and phase . . . 635
30.4.2.2 Specifying memory devices . . . 635
30.4.2.3 Connecting SPI memory devices . . . 637
30.4.2.4 Connecting HB memory devices . . . 643
30.4.2.5 SPI data transfer . . . 644
30.4.2.6 SPI - putting it all together . . . 646
30.4.2.7 SPI- slave select signal during power up . . . 648
30.4.2.8 HB data transfers . . . 649
30.4.2.9 HB - putting it all together . . . 651
30.4.3 AXI interface . . . 654
30.4.3.1 Support for multiple outstanding transactions . . . 656
30.4.3.2 Wrapping bursts . . . 657
30.4.3.3 Fixed bursts . . . 657
30.4.3.4 Splitting bursts to smaller blocks . . . 657
30.4.3.5 Write strobes . . . 657
30.4.4 Triggers . . . 659
30.4.5 Interrupts . . . 659
30.4.6 Monitor signals . . . 660
30.5 Supply rails and power domains . . . 660
30.5.1 Power modes . . . 660
30.6 Sub block descriptions . . . 661
30.6.1 Address space . . . 661
30.6.1.1 XIP address space . . . 661
30.6.1.2 MMIO address space . . . 662
30.6.2 TX and RX FIFOs . . . 662
30.6.2.1 TX command FIFO . . . 662

Table of contents

30.6.2.2 TX data FIFO . . . 663
30.6.2.3 RX data FIFO . . . 664
30.6.3 Interrupts and triggers . . . 665
30.6.4 Cache . . . 665
30.6.5 Arbitration . . . 667
30.6.6 Cryptography . . . 667
30.6.6.1 Mapping of bytes . . . 669
30.6.6.2 Software and MMIO register requirements . . . 669
30.6.6.3 Cryptography support for AXI interface . . . 670
30.6.7 Serial memory interface logic . . . 670
30.6.7.1 TX and RX logic . . . 670
30.6.7.2 Flow control . . . 672
30.6.7.3 Data capture . . . 672
30.7 Registers . . . 676

31 SDHC host controller . . . 679
31.1 Features . . . 679
31.1.1 Features not supported . . . 679
31.2 Block diagram . . . 680
31.3 Clocking . . . 680
31.3.1 Clock gating . . . 681
31.3.2 Base clock (CLK_HFx) configuration . . . 681
31.3.3 Card clock (SDCLK) configuration . . . 681
31.3.4 Timeout (TOUT) configuration . . . 681
31.4 Bus speed modes . . . 682
31.5 Power modes . . . 682
31.5.1 Standby mode . . . 682
31.6 Interrupts to CPU . . . 682
31.6.1 SDIO interrupt . . . 683
31.7 I/O interface . . . 684
31.8 Packet buffer SRAM . . . 684
31.8.1 Packet buffer full/empty . . . 685
31.9 DMA engine . . . 685
31.10 Initialization sequence . . . 686
31.10.1 Enabling SDHC . . . 687
31.10.2 Card detection . . . 688
31.10.3 SDHC initialization . . . 690
31.10.4 Clock setup . . . 691
31.11 Error detection . . . 692
31.12 Registers . . . 692

32 Audio subsystem . . . 697
32.1 Features . . . 697

Table of contents

32.2 Architecture . . . 697
32.3 Digital audio interface formats . . . 698
32.3.1 Standard I²S format . . . 698
32.3.2 Left justified (LJ) format . . . 700
32.3.3 Time division multiplexed (TDM) format . . . 700
32.4 Clocking polarity and delay options . . . 701
32.5 Interfacing with audio codecs . . . 703
32.6 Clocking features . . . 704
32.7 FIFO buffer and DMA support . . . 705
32.8 Interrupt support . . . 708
32.9 Watchdog timer . . . 709
32.10 MCLK output function . . . 709
32.11 Registers . . . 710

33 SAR ADC . . . 712
33.1 Features . . . 712
33.2 Block diagram . . . 714
33.3 Operation . . . 714
33.3.1 SAR ADC conversion flow . . . 715
33.3.2 Result data format . . . 716
33.3.2.1 Signed/unsigned result . . . 716
33.3.2.2 Alignment . . . 717
33.3.3 Acquisition/sample time . . . 717
33.4 SARMUX . . . 718
33.4.1 Preconditioning . . . 719
33.4.2 Overlap diagnostic . . . 720
33.4.3 SARMUX diagnostics . . . 720
33.5 SAR sequencer . . . 721
33.5.1 Analog input selection . . . 721
33.5.2 External analog multiplexer . . . 722
33.5.3 Port selection . . . 722
33.5.4 Averaging . . . 722
33.5.5 Right shifting . . . 723
33.5.6 Range detect . . . 723
33.5.7 Pulse detect . . . 724
33.5.8 Double buffer . . . 726
33.5.9 Group coherency . . . 727
33.5.10 Status . . . 727
33.6 Triggering and scheduling . . . 728
33.6.1 Channel grouping . . . 728
33.6.2 Triggers . . . 728
33.6.3 Arbitration, preemption, and acquisition scheduling . . . 729
33.6.4 Debug freeze . . . 732

Table of contents

33.6.5 Auto idle power down . . . 732
33.6.6 Channel disable/software abort . . . 732
33.7 Output triggers and interrupts . . . 733
33.7.1 Trigger outputs . . . 733
33.7.1.1 Channel done trigger . . . 733
33.7.1.2 Range violation trigger . . . 733
33.7.1.3 Generic trigger output . . . 734
33.7.2 Channel interrupts . . . 734
33.7.2.1 Range detect interrupt . . . 734
33.7.2.2 Pulse detect interrupt . . . 734
33.7.2.3 Channel overflow interrupt . . . 734
33.7.3 Group interrupts . . . 735
33.7.3.1 Group done interrupt . . . 735
33.7.3.2 Group canceled interrupt . . . 735
33.7.3.3 Group overflow interrupt . . . 735
33.8 Calibration . . . 735
33.8.1 Analog calibration . . . 735
33.8.2 Alternate calibration . . . 737
33.8.3 Coherent calibration update . . . 737
33.9 Temperature measurement . . . 739
33.9.1 Example measurement flow . . . 739
33.9.2 Temperature sensor calibration and SFlash address . . . 739
33.9.3 Temperature calculation . . . 741
33.9.3.1 Procedure to calculate the temperature . . . 741
33.10 Diagnostic reference generator . . . 742
33.10.1 Diagnostic reference configuration . . . 742
33.11 Reference buffer . . . 742
33.12 Registers . . . 744

34 Program and debug interface . . . 747
34.1 Features . . . 748
34.2 Functional description . . . 748
34.2.1 Debug access port (DAP) . . . 749
34.2.1.1 DAP security . . . 749
34.2.1.2 DAP power domain . . . 749
34.2.2 ROM tables . . . 750
34.2.3 Trace . . . 750
34.2.4 Embedded cross-triggering . . . 750
34.3 Serial wire debug (SWD) interface . . . 752
34.3.1 SWD timing details . . . 754
34.3.2 ACK details . . . 754
34.3.3 Turnaround (Trn) period details . . . 755
34.4 JTAG interface . . . 755

Table of contents

34.5 Pin configuration of debug interface on BootROM . . . 758
34.6 Programming the XMC7000 device . . . 758
34.6.1 SWD port acquisition . . . 758
34.6.1.1 SWD port acquire sequence . . . 758
34.6.2 SWD programming mode entry . . . 759
34.6.3 SWD programming routines executions . . . 759
34.7 Registers . . . 759

35 Non-volatile memory programming . . . 761
35.1 Functional description . . . 761
35.2 System call implementation . . . 764
35.2.1 System call via CM0+ or CM7_0 or CM7_1 . . . 764
35.2.1.1 System call via DAP . . . 764
35.2.2 Exiting from a system call . . . 764
35.3 SROM API library . . . 764
35.4 System calls . . . 767
35.4.1 BlankCheck . . . 768
35.4.2 BlowFuseBit . . . 769
35.4.3 CheckFactoryHash . . . 770
35.4.4 CheckFMStatus . . . 771
35.4.5 Checksum . . . 773
35.4.6 Compute basic hash . . . 775
35.4.7 ConfigureFMInterrupt . . . 776
35.4.8 EraseAll . . . 777
35.4.9 EraseResume . . . 778
35.4.10 EraseSector . . . 779
35.4.11 EraseSuspend . . . 780
35.4.12 GenerateHash . . . 781
35.4.13 SwitchOverRegulators . . . 782
35.4.14 ConfigureRegulator . . . 784
35.4.15 ProgramRow . . . 787
35.4.16 ProgramWorkFlash . . . 789
35.4.17 ReadFuseByte . . . 791
35.4.18 ReadFuseByteMargin . . . 792
35.4.19 ReadSWPU . . . 793
35.4.20 ReadUniqueID . . . 794
35.4.21 SetEnforcedApproval . . . 795
35.4.22 SiliconID . . . 796
35.4.23 SoftReset . . . 799
35.4.24 Transition to RMA . . . 800
35.4.25 Transition to Secure . . . 801
35.4.26 DirectExecute . . . 802
35.4.27 WriteRow . . . 803

Table of contents

35.4.28 WriteSWPU . . . 807
35.4.29 Debug power up down . . . 808
35.4.30 LoadRegulatorTrims . . . 809
35.4.31 OpenRMA . . . 811
35.5 System call status . . . 812
35.6 eFuse memory . . . 816
35.6.1 Features . . . 816
35.6.2 Customer eFuses . . . 816
36 Flash boot . . . 817
36.1 Features . . . 817
36.2 Using flash boot . . . 818
36.2.1 Flash boot shared functions . . . 818
36.2.1.1 Cy_FB_VerifyApplication . . . 818
36.2.1.2 Cy_FB_IsValidKey . . . 818
36.2.2 Using a bootloader . . . 819
36.2.2.1 Bootloader host requirements . . . 819
36.2.2.2 Using CAN . . . 822
36.2.2.3 CAN driver configuration . . . 823
36.3 Flash boot internals . . . 824
36.3.1 Definitions . . . 824
36.3.2 SFlash address mapping . . . 825
36.3.3 Flash boot flow . . . 826
36.3.3.1 Entry from ROM boot (1) . . . 828
36.3.3.2 Set-up SP (2) . . . 828
36.3.3.3 Initialization (3) . . . 828
36.3.3.4 Validate TOC2 (5) . . . 829
36.3.3.5 Is TOC2 valid (6) . . . 829
36.3.3.6 Bootloading (8) . . . 829
36.3.3.7 Get app #{0, 1} reset handler (9) . . . 829
36.3.3.8 Valid reset handler (10) . . . 830
36.3.3.9 App authentication (11) . . . 830
36.3.3.10 Is public key valid (12) . . . 830
36.3.3.11 Valid digital signature (13) . . . 830
36.3.3.12 Enable system calls (14) . . . 830
36.3.3.13 Set up DAP from AR (15) . . . 830
36.3.3.14 Set PC (16) . . . 830
36.3.3.15 Is DAP enabled (17) . . . 831
36.3.3.16 Configure SWJ (18) . . . 831
36.3.3.17 Wake-up from hibernate (19) . . . 831
36.3.3.18 Listen window (20) . . . 831
36.3.3.19 Test mode (21) . . . 831
36.3.3.20 Is single-core (22) . . . 831

Table of contents

36.3.3.21 Launch CM0+ application (24) . . . 831
36.3.3.22 Idle loop (25) . . . 832
36.3.3.23 Branch DEAD (8) . . . 832
36.3.3.24 Branch bootloader (28) . . . 832
36.3.3.25 Interrupts and system calls (29) . . . 832
36.3.3.26 Set error code (30) . . . 832
36.3.3.27 PROTECTION = VIRGIN (31) . . . 833
36.3.3.28 LifeCycle = SECURE (32) . . . 833
36.3.3.29 PROTECTION = DEAD (33) . . . 833
36.3.3.30 Deploy AR (34) . . . 833
36.3.3.31 Apply system protection (35) . . . 834
36.3.3.32 Disable WDT (40) . . . 834
36.3.3.33 Set VECTOR_TABLE_BASE (41) . . . 834
36.3.3.34 Launch bootloader (42) . . . 834
36.3.3.35 CM0+ core reset (50) . . . 834
36.3.3.36 Launch a user app by ROM boot (51) . . . 834
36.3.4 Data structures . . . 835
36.3.4.1 Application formats . . . 835
36.3.4.2 RSA public key format . . . 837
36.3.4.3 TOC2 structure . . . 839
36.3.5 Internal bootloader . . . 841
36.3.5.1 Terms and definitions . . . 842
36.3.5.2 Using bootloader . . . 842
36.3.5.3 Bootloader activation conditions . . . 842
36.3.5.4 Basic bootloader function flow . . . 843
36.3.5.5 End-of-line programming . . . 846
References . . . 848
Revision history . . . 849
Disclaimer . . . 852

1 Introduction

The XMC7000 device is a microcontroller targeted at industrial applications. These devices have up to two Arm® Cortex®-M7 CPUs for primary processing, and a Cortex®-M0+ CPU for peripheral and security processing. These devices contain embedded peripherals supporting controller area network with flexible data rate (CAN FD) and gigabit Ethernet. XMC7000 is manufactured on an advanced 40-nm process. XMC7000 incorporates Infineon low-power flash memory, multiple high-performance analog and digital peripherals, and enables the creation of a secure computing platform.

1.1 Device characteristics

1.1.1 CPU subsystem

- Two 350-MHz 32-bit Arm® Cortex®-M7 CPUs, each with
  - Single-cycle multiply
  - Single/double-precision floating point unit (FPU)
  - 16-KB data cache, 16-KB instruction cache
  - Memory protection unit (MPU)
  - 16-KB instruction and 16-KB data tightly-coupled memory (TCM)
- 100-MHz 32-bit Arm® Cortex® M0+ CPU with single-cycle multiply and MPU
- Interprocessor communication in hardware
- Two types of DMA controllers – one to support peripheral-to-memory (and vice versa) and one for memory-to-memory data transfers over the AHB bus
- Up to 8384 KB of code-flash with an additional, up to 256 KB of work-flash and an internal SRAM of up to 1024 KB
  - Flash programming on JTAG/SWD interface
  - Read-while-write (RWW) allows updating the code-flash and work-flash while executing from it
  - Single- and dual-bank modes (specifically for firmware over-the-air (FOTA) update)
- Crypto engine to support enhanced secure hardware extension (eSHE) and hardware secure module (HSM). The Crypto engine and software support the following functions:
  - RSA-2048, RSA-3072, RSA-4096, ECC-256, ECC-384, SHA-2, SHA-3, AES-128/256, and 3DES
  - True random number generator (TRNG) and pseudo random number generator (PRNG)
  - Hash function
  - Galois/counter mode (GCM)
- Hardware error correction (SECDED ECC) on all safety-critical memories (SRAM and flash)

1.1.2 Communication

- High-speed CAN FD communication supporting up to 8 Mbps data rate
- Serial interface to support various serial communication (UART/SPI/I^2C)
- 10/100/1000 Mbps Ethernet MAC interfaces with audio video bridging (AVB) and precision time protocol (PTP) support conforming to IEEE-802.3az. These interfaces support the following PHY interfaces:
  - Media-independent interface (MII)
  - Reduced media-independent interface (RMII)
  - Reduced gigabit media-independent interface (RGMII)

1.1.3 Memory interfaces

1.1.3.1 Serial memory interface

- Supports SPI (single, dual, quad, or octal) or HYPERBUS™ interface, with on-the-fly encryption and decryption along with execute-in-place (XIP)

1.1.3.2 Secure digital high capacity

- Secure digital high capacity (SDHC) interface supporting embedded MultiMediaCard (eMMC), or secure digital (SD), or secure digital input output (SDIO) compliant to eMMC 5.1, SD 6.0, and SDIO 4.10 specifications
- Supports data rates up to SD SDR50 (single data rate) or eMMC 52 MHz DDR

1.1.4 Audio subsystem

- Inter-IC sound (I^2S) interfaces to connect digital audio devices
- Supports I^2S, left justified, or time division multiplexed (TDM) audio formats
- Supports independent transmitter or receiver operation, each in master or slave mode

1.1.5 Miscellaneous

- Low-power 2.7 V to 5.5 V operation, with two robust brownout detect (BOD) and over-voltage detect (OVD) options
- Programmable GPIOs and smart I/O to perform Boolean operations on signals going to and from I/O pins
- High-performance 12-bit analog-to-digital converter (ADC)
  - Supports 12-bit resolution and sampling rates up to 1 Msps
- High-current linear regulator
  - Supports generation of 1.1 V nominal core supply from a 2.7 V to 5.5 V input supply
  - Supports control of external low-dropout (LDO) or power management IC (PMIC) regulators
- Hardware watchdog function
- Real-time clock with auto-calibration
- Timing and pulse-width modulation with support for timer, capture, quadrature, pulse-width modulation (PWM outputs), PWM with dead time (PWM_DT), pseudo-random PWM (PWM_PR), and Shift-Register (SR) modes; some PWM channels also support stepper motor control
- Deep sleep and hibernate power modes for low-power solution
- Event generator to support cyclic wakeup from deep sleep mode and peripheral trigger in active power mode
- Debugging via JTAG controller (interface compliant IEEE-1149.1-2001) and Arm® SWD port
- Supports Arm® embedded trace macrocell (ETM) trace
  - Data trace using SWD
  - Instruction and data trace using JTAG

1.2 Top level architecture

Figure 1 shows the XMC7000 architecture block diagram, giving a simplified view of the interconnection between subsystems and blocks. The device has four major subsystems: CPU, system resources, peripherals, and I/O.

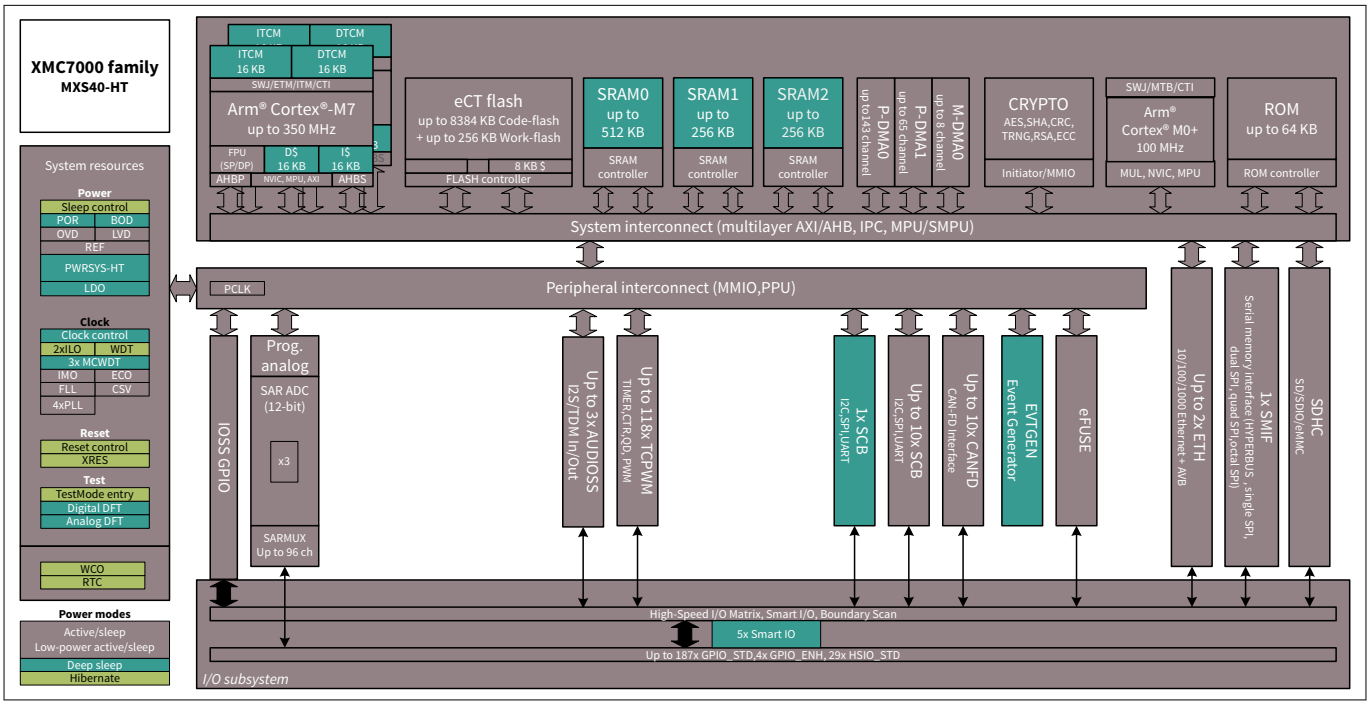


Figure 1 XMC7000 architecture diagram

The device provides extensive support for programming, testing, debugging, and tracing of both hardware and firmware. Debug-on-chip functionality enables in-system debugging using the production device. It does not require special interfaces, debugging pods, simulators, or emulators. The JTAG interface is fully compatible with industry-standard third-party probes such as I-jet and J-Link. The debug circuits are enabled by default. The microcontroller provides a high level of security with robust flash protection and the ability to disable features such as debug. Additionally, each device interface can be permanently disabled for applications concerned with phishing attacks due to a maliciously reprogrammed device or attempts to defeat security by starting and interrupting flash programming sequences. All programming, debug, and test interfaces are disabled when maximum device security is enabled.

1.2.1 CPU subsystem

1.2.1.1 CPU

The XMC7000 CPU subsystem contains a 32-bit Arm® Cortex®-M0+ CPU with MPU, up to two 32-bit Arm® Cortex®-M7 CPUs each with MPU, single/double-precision FPU, and 16-KB data and instruction caches. This subsystem also includes P-/M-DMA controllers, a cryptographic accelerator, up to 8384 KB of code-flash, up to 256 KB of work-flash, up to 1024 KB of SRAM, and 64 KB of ROM.

The Cortex®-M0+ CPU provides a secure, un-interruptible boot function. This guarantees that after the boot function is complete, system integrity is valid and privileges are enforced. Shared resources such as flash, SRAM, and peripherals can be accessed through bus arbitration, and exclusive accesses are supported by an interprocessor communication (IPC) mechanism using hardware semaphores.

Each Cortex®-M7 CPU has 16 KB of instruction and 16 KB of data TCM with an option of programmable read wait states. Each TCM is clocked by the associated Cortex®-M7 CPU clock.

1.2.1.2 DMA Controllers

XMC7000 has two types of DMA controllers: P-DMA and M-DMA. P-DMA is used for peripheral-to-memory and memory-to-peripheral data transfers and provides low latency for a large number of channels. Each P-DMA controller uses a single data-transfer engine that is shared by the associated channels. General-purpose channels have a rich interconnect matrix including P-DMA cross triggering, which enables demanding data-transfer scenarios. Dedicated channels have a single triggering input (such as an ADC channel) to handle common transfer needs. M-DMA is used for memory-to-memory data transfers and provides high memory bandwidth for a small number of channels. M-DMA uses a dedicated data-transfer engine for each channel. They support independent accesses to peripherals using the AHB multi-layer bus.

1.2.1.3 Flash

XMC7000 has up to 8384 KB of code-flash with an additional work-flash of up to 256 KB. Work-flash is optimized for reprogramming many more times than code-flash. Code-flash supports read-while-write (RWW) operation allowing flash to be updated while the CPU is active. Both the code-flash and work-flash areas support dual-bank operation for over-the-air (OTA) programming.

1.2.1.4 SRAM with 32-KB retention granularity

XMC7000 has up to 1024 KB of SRAM with three independent controllers. The first controller SRAM0 provides deep-sleep retention in 32 KB increments while SRAM1/2 is selectable between fully retained and not retained.

1.2.1.5 ROM

XMC7000 has 64 KB of ROM that contains boot and configuration routines. This ROM enables secure boot and authentication of user flash to guarantee a secure system.

1.2.1.6 Cryptography accelerator for security

The cryptography accelerator implements (3)DES block cipher, AES block cipher, SHA hash, cyclic redundancy check, pseudo random number generation, true random number generation, galois/counter mode, and a vector unit to support asymmetric key cryptography such as RSA and ECC.

Depending on the part number, this block is either completely or partially available or not available at all.

1.2.2 System resources

1.2.2.1 Power system

The power system ensures that the supply voltage levels meet the requirements of each power mode, and provides a full-system reset when these levels are not valid. Internal power-on reset (POR) guarantees full-chip reset during the initial power ramp.

Three brown-out detection (BOD) circuits monitor the external supply voltages (V_{DDD}, V_{DDA}, V_{CCD}). The BOD on V_{DDD} and V_{CCD} are initially enabled and cannot be disabled. The BOD on V_{DDA} is initially disabled and can be enabled by the user. For the external supplies V_{DDD} and V_{DDA}, BOD circuits are software configurable with two settings; a 2.7 V minimum voltage, which is robust for all internal signaling and a 3.0-V minimum voltage, which is also robust for all I/O specifications (that are guaranteed at 2.7 V). The BOD on V_{CCD} is provided as a safety measure and is not a robust detector.

Three over-voltage detection (OVD) circuits are provided for monitoring external supplies (V_{DDD}, V_{DDA}, V_{CCD}), and over-current detection circuits (OCD) for monitoring internal and external regulators. OVD thresholds on V_{DDD} and V_{DDA} are configurable with two settings; a 5.0 V and 5.5 V maximum voltage.

Two voltage detection circuits are provided to monitor the external supply voltage (V_{DDD}) for falling (low-voltage detector, LVD) and rising levels (high-voltage detector, HVD), each configurable for one of the 26 selectable levels.

All BOD, OVD, and OCD circuits on V_{DDD} and V_{CCD} generate a reset, because these protect the CPUs and fault logic. The BOD and OVD circuits in V_{DDA} can be configured to generate either a reset, or a fault.

1.2.2.2 Regulators

XMC7000 contains three regulators that provide power to the low-voltage core transistors: deep sleep, core internal, and core external. These regulators accept a 2.7 – 5.5-V V_{DDD} supply and provide a low-noise 1.1 V supply to various parts of the device. These regulators are automatically enabled and disabled by hardware and firmware when switching between power modes. The core internal and core external regulators operate in active mode, and provide power to the CPU subsystem and associated peripherals.

- Deep sleep: The deep sleep regulator is used to maintain power to a small number of blocks when in deep sleep mode. These blocks include the ILO and WDT timers, BOD detector, SCB0, SRAM memories, smart I/O, and other configuration memories. The deep sleep regulator is enabled when in deep sleep mode, and when the core internal regulator is disabled. It is disabled when XRES_L is asserted (LOW) and when the core internal regulator is disabled.
- Core internal: The core internal regulator supports load currents up to 300 mA, and is operational during device start up (boot process), and in active/sleep modes.
- Core external: To support worst case loading, with both CM7 CPUs and the CM0®+ CPU at their maximum clock frequency and all integrated peripherals operating, the core external regulator must handle load current up to 600 mA. While the control and monitor circuits for the core external regulator are internal to XMC7000, the power regulating element (NPN pass transistor, PMIC, or LDO) is external. This reduces the overall power dissipation within the XMC7000 package, while maintaining a well-regulated core supply. The core external regulator may be implemented with either an external NPN pass transistor, linear regulator (LDO), or PMIC. Each implementation requires different external components on the PCB, and different connections to XMC7000 for both regulation and control.

***Note**: Check the device datasheet for a list of external regulators that are supported.*

1.2.2.3 Clock system

The XMC7000 clock system provides clocks to all subsystems that require them, and glitch-free switching between different clock sources. In addition, the clock system ensures that no metastable conditions occur.

The clock system for XMC7000 consists of the 8 MHz IMO, ILOs, watchdog timers, PLLs, an FLL, clock supervisors (CSV), an ECO, and a WCO.

The clock system supports three main clock domains: CLK_HF, CLK_SLOW, and CLK_LF.

- CLK_HFx are the active domain clocks. Each can use any of the high frequency clock sources including IMO, EXT_CLK, ECO, FLL, or PLL.
- CLK_SLOW provides reference clock for the CM0+ CPU, Crypto, P-/M-DMA, and other slow infrastructure blocks of CPU subsystem.
- CLK_LF is a deep sleep domain clock and provides source for MCWDT or RTC modules. The reference clock for the CLK_LF domain is selectable from ILO0, ILO1, WCO, or disabled.

1.2.2.4 IMO clock source

The IMO is the frequency reference in XMC7000 when no external reference is available or enabled. The IMO operates at a frequency around 8 MHz.

1.2.2.5 ILO clock source

An ILO is a low-power oscillator, which generates clocks for a watchdog timer when in deep sleep mode. There are two ILOs to ensure CSV capability in the deep sleep mode. ILO-driven counters can be calibrated to the IMO, WCO, or ECO to improve their accuracy. ILO1 is also used for clock supervision.

1.2.2.6 PLL and FLL

A PLL or FLL may be used to generate high-speed clocks from the IMO, ECO, or EXT_CLK. The FLL provides a much faster lock than the PLL in exchange for a small amount of frequency error and a lower maximum output frequency. 400-MHz PLLs supports spread spectrum clock generation (SSCG) and down spreading (down spreading is where the maximum frequency of the spread spectrum clock is the same as that of the non-modulated clock).

1.2.2.7 Clock supervisor

Each clock supervisor (CSV) allows one clock (reference) to supervise the behavior of another clock (monitored). Each CSV has counters for both monitored and reference clocks. Parameters for each counter determine the frequency of the reference clock as well as the upper and lower frequency limits of the monitored clock. If the frequency range comparator detects a stopped clock or a clock outside the specified frequency range, an abnormal state is signaled and either a reset or an interrupt is generated.

1.2.2.8 EXT_CLK

Dedicated GPIOs can be used to provide an external clock. This clock can be used as the source clock for either the PLL or FLL, or can be used directly by the CLK_HF domain.

1.2.2.9 ECO

The ECO provides high-frequency clocking using an external crystal connected to the ECO_IN and ECO_OUT pins. It supports fundamental mode (non-overtone) quartz crystals. When used in conjunction with the PLL, it generates CPU and peripheral clocks up to device's maximum frequency. ECO accuracy depends on the selected crystal. If the ECO is disabled, the associated pins can be used for any of the available I/O functions.

1.2.2.10 WCO

The WCO is a low-power, watch-crystal oscillator intended for real-time-clock applications. It requires an external crystal connected to the WCO_IN and WCO_OUT pins. The WCO can also be configured as a clock reference for CLK_LF, which is the clock source for the MCWDT and RTC.

1.2.2.11 Reset

XMC7000 can be reset from a variety of sources, including software. Reset events are asynchronous and guarantee reversion to a known state. The reset cause (POR, BOD, OVD, overcurrent, XRES_L, WDT, MCWDT, software reset, fault, CSV, hibernate wakeup, or debug) is recorded in a register, which is sticky through reset and allows software to determine the cause of the reset. An XRES_L pin is available for external reset.

1.2.2.12 Power modes

XMC7000 supports six power modes.

- Active - All peripherals are available
- Low-power active (LPACTIVE) - Low-power profile of active mode where all peripherals and the CPUs are available, but with limited capability
- Sleep - All peripherals except the CPUs are available
- Low-power sleep (LPSLEEP) - Low-power profile of sleep mode where all peripherals except the CPUs are available, but with limited capability
- Deep sleep - Only peripherals that work with CLK_LF are available
- Hibernate - The device and I/O states are frozen; the device resets on wakeup

1.2.3 Peripherals

1.2.3.1 Peripheral clock dividers

Integer and fractional clock dividers are provided for peripheral and timing purposes.

1.2.3.2 Peripheral protection unit

The peripheral protection unit (PPU) controls and monitors unauthorized access from all masters (CPU, P-/M-DMA, Crypto, and the debug interface) to the peripherals. It allows or restricts data transfers on the bus infrastructure. The access rules are enforced based on specific properties of a transfer, such as an address range for the transfer and access attributes (such as read/write, user/privilege, and secure/non-secure).

1.2.3.3 12-bit SAR ADC

XMC7000 contains three 1-Msps SAR ADCs, which can be clocked at up to 26.67 MHz and provide a 12-bit result in 26 clock cycles.

The references for all three SAR ADCs come from a dedicated pair of inputs: VREFH and VREFL.

Each ADC has a sequencer, which autonomously cycles through the configured channels (sequencer scan) with zero-switching overhead (that is, the aggregate sampling bandwidth, when clocked at 26.67 MHz, is equal to 1 Msps whether it is for a single channel or distributed over several channels). The sequencer switching is controlled through a state machine or firmware. The sequencer prioritizes trigger requests, enables the appropriate analog channel, controls ADC sampling, initiates ADC data conversion, manages results, and initiates subsequent conversions for repetitive or group conversions without CPU intervention.

Each SAR ADC has an analog multiplexer used to connect the signals to be measured to the ADC. It has 32 GPIO_STD inputs, one special GPIO_STD input for motor-sense, and six additional inputs to measure internal signals such as a band-gap reference, a temperature sensor, and power supplies. The device supports synchronous sampling of one motor-sense channel on each of the three ADCs.

XMC7000 has one temperature sensor that is shared by all three ADCs. The temperature sensor must only be sampled by one ADC at a time. Software post processing is required to convert the temperature sensor reading into kelvin or celsius values.

To accommodate signals with varying source impedances and frequencies, it is possible to have different sample times programmed for each channel. Each ADC also supports range comparison, which allows fast detection of out-of-range values without having to wait for a sequencer scan to complete and for the CPU firmware to evaluate the measurement for out-of-range values.

The ADCs are not usable in the deep sleep and hibernate modes as they require a high-speed clock. The ADC input reference voltage (VREFH) range is 2.7 V to V_{DDA} and VREFL is V_{SSA}.

1.2.3.4 Timer/counter/PWM block (TCPWM)

The TCPWM block consists of 16-bit and 32-bit counters with user-programmable period. Some of the 16-bit counters are optimized for motor-control operations. Each TCPWM counter contains a capture register to record the count value at the time of an event, a Period Register (used to either stop or auto-reload the counter when its count is equal to the Period Register), and compare registers to generate signals that are used as PWM duty-cycle outputs.

Each counter within the TCPWM block supports several functional modes such as timer, capture, quadrature, PWM with dead-time insertion (PWM_DT, 8-bit), pseudo-random PWM (PWM_PR), and Shift-Register.

In motor control applications, the counter within the TCPWM block supports enhanced quadrature mode with features such as asymmetric PWM generation, dead-time insertion (16-bit), and association of different dead times for PWM output signals.

The TCPWM block also provides true and complement outputs, with programmable offset between them, to allow their use as deadband complementary PWM outputs. The TCPWM block also has a kill input (only for the PWM mode) to force outputs to a predetermined state; for example, this may be used in motor-drive systems when an overcurrent state is detected and the PWMs driving the FETs need to be shut off immediately (no time for software intervention).

1.2.3.5 Serial communication blocks (SCB)

XMC7000 contains up to 11 serial communication blocks, each configurable to support I^2C, UART, or SPI.

- I^2C interface: An SCB can be configured to implement a full I^2C master (capable of multi-master arbitration) or slave interface. Each SCB configured for I^2C can operate at speeds of up to 1 Mbps (fast-mode plus) and has flexible buffering options to reduce the interrupt overhead and latency of the CPU. In addition, each SCB supports FIFO buffering for receive and transmit data, which, by increasing the time for the CPU to read the data, reduces the need for clock stretching. The I^2C interface is compatible with standard, fast-mode, and fast-mode plus devices as specified in the NXP I^2C-bus specification and user manual (UM10204). The I^2C-bus I/O is implemented with GPIO in open-drain modes.
- UART interface: When configured as a UART, each SCB provides a full-featured UART with maximum signaling rate determined by the configured peripheral-clock frequency and over-sampling rate. It supports infrared interface (IrDA) and SmartCard (ISO 7816) protocols, which are minor variants of the UART protocol. It also supports the 9-bit multiprocessor mode that allows addressing of peripherals connected over common RX and TX lines. Common UART functions such as parity, number of stop bits, break detect, and frame error are supported. FIFO buffering of transmit and receive data allows greater CPU service latencies to be tolerated.
- SPI interface: The SPI configuration supports full Motorola SPI, TI synchronous serial protocol (SSP, essentially adds a start pulse that is used to synchronize SPI-based codecs), and National Microwire (a half-duplex form of SPI). The SPI interface can use the FIFO. SCB also supports EZSPI[7] mode.

SCB0 supports the following additional features:

- Operable as a slave in deep sleep mode
- I^2C slave EZ (EZI2C) mode with up to 256-byte data buffer for multi-byte communication without CPU intervention
- I^2C slave externally-clocked operations
- Command/response mode with a 512-byte data buffer for multi-byte communication without CPU intervention

1.2.3.6 CAN FD

XMC7000 contains CAN FD controller blocks, each supporting one or more CAN FD channel. All CAN FD controllers are compliant with the ISO 11898-1:2015 standard; an ISO 16845:2015 certificate is available. It also

implements the time-triggered CAN (TTCAN) protocol specified in ISO 11898-4 (TTCAN protocol levels 1 and 2) completely in hardware. All functions concerning the handling of messages are implemented by the RX and TX handlers. The RX handler manages message acceptance filtering, transfer of received messages from the CAN core to a message RAM, and provides receive-message status. The TX handler is responsible for the transfer of transmit messages from the message RAM to the CAN core, and provides transmit-message status.

1.2.3.7 Ethernet MAC

XMC7000 supports Ethernet channels with transfer rates of 10, 100, or 1000 Mbps. The input/output frames and flow control are complaint to the Ethernet/IEEE 802.3az standards, and also IEEE-1588 precision-time protocol (PTP). The device supports full-duplex data transport using external PHY devices. The MAC supports glue-free connection to PHYs through IEEE standard MII, RMII and RGMII interfaces. The device also supports audio-video bridging (AVB).

1.2.3.8 External memory interface

In addition to the 8 MB of internal flash memory, XMC7000 supports direct connection to an external flash or RAM memory. This connection is made through either a HYPERBUS™ or SPI. HYPERBUS™ allows connection to HYPERFLASH™ and HYPERRAM™ devices, while SPI (single, dual, quad, or octal SPI) can connect with serial flash memory. Code stored in memory connected through this interface allows execute-in-place (XIP) operation, which does not require the instructions to be first copied to internal memory, and on-the-fly encryption and decryption for environments requiring secure external data and code.

1.2.3.9 SDHC interface

XMC7000 supports secure digital host capacity (SDHC) interface, which conforms to SD 6.0, secure digital input output (SDIO) 4.10, and embedded multimedia card (eMMC) 5.1 specifications, along with host control interface (HCI) 4.2 specification. The interface supports system DMA (SDMA), advance DMA (ADMA2, ADMA3), and command queuing (CQ) features. Supports data rates of SD DS (default speed, 4-bits at 25 MHz), SD HS (high speed, 4-bits at 50 MHz, and eMMC 52-MHz DDR (8-bits at 52-MHz card clock).

1.2.3.10 One-time-programmable (OTP) eFuse

XMC7000 devices contain a 1024-bit OTP eFuse memory that can be used to store and access a unique and unalterable identifier or serial number for each device. eFuses are also used to control the device lifecycle (manufacturing, programming, normal operation, end-of-life, and so on) and the security state.

1.2.3.11 Event generator

The event generator supports generation of interrupts and triggers in the active mode and interrupts in the deep sleep mode. The event generators are used to trigger a specific device function (execution of an interrupt handler, a SAR ADC conversion, and so on) and provide a cyclic wakeup mechanism from the deep sleep mode. They provide CPU-free triggers for device functions, and reduce CPU involvement in triggering device functions, thus reducing overall power consumption and processing overhead.

1.2.3.12 Trigger multiplexer

XMC7000 supports connecting various peripherals using trigger signals. Triggers are used to inform a peripheral of the occurrence of an event or change of state. These triggers are used to effect or initiate some action in other peripherals. The trigger multiplexer is used to route triggers from a source peripheral to a destination. Triggers provide active logic functionality and are supported in the active mode.

1.2.4 I/Os

XMC7000 has three types of programmable I/Os: GPIO standard, GPIO enhanced, and HSIO standard.

The I/Os are organized as logical entities called ports, which are a maximum of eight bits wide. During power-on, hibernate, and reset, the I/Os are forced to the High-Z state.

Every I/O can generate an interrupt (if enabled) and each port has an interrupt request (IRQ) and interrupt service routine (ISR) associated with it.

1.2.4.1 GPIO

- GPIO standard (GPIO_STD): Supports standard signaling across the 2.7-V to 5.5-V V_{DDIO} range. GPIO Standard I/Os have multiple configurable drive levels, drive modes, and selectable input levels.
- GPIO enhanced (GPIO_ENH): Supports extended functionality signaling across the 2.7-V to 5.5-V V_{DDIO} range with higher currents at lower voltages (full I^2C timing support and slew-rate control).

Both GPIO_STD and GPIO_ENH implement the following:

- Configurable input threshold (CMOS, TTL)
- Hold mode for latching previous state (used to retain the I/O state in deep sleep mode)
- Analog input mode (input and output buffers disabled)

1.2.4.2 HSIO

HSIO standard (HSIO_STD): These I/Os are optimized exclusively for high-speed signaling (supports signaling across the 2.7-V to 3.6-V V_{DDIO} range) and do not support slew-rate control, deep sleep operation, POR mode control, analog connections, or non-CMOS signaling levels. HSIO_STD supports high-speed peripherals such as QSPI, HYPERBUS™, Ethernet, and SDHC controller. It also supports programmable drive strength. These I/Os are available only in active mode and retain state in deep sleep modes.

1.2.4.3 Drive modes

All I/Os support the following programmable drive modes:

- High impedance
- Resistive pull-up
- Resistive pull-down
- Open drain with strong pull-down
- Open drain with strong pull-up
- Strong pull-up or pull-down
- Weak pull-up or pull-down

1.2.4.4 Port nomenclature

Px.y describes a particular bit “y” available within an I/O port “x.”

For example, P4.2 reads “port 4, bit 2”.

1.2.4.5 Smart I/O

Smart I/O allows Boolean operations on signals going to the GPIO from the device subsystems or on signals coming into the chip. Operation can be synchronous or asynchronous and the blocks operate in all device power modes except for the hibernate mode.

2 Getting started

2.1 Support

Free support for XMC7000 MCU products is available online at www.infineon.com. Resources include training seminars, application notes, CRM technical support email, knowledge base, and application support engineers.

2.2 Product upgrades

Infineon provides scheduled upgrades and version enhancements for the peripheral driver library (PDL) and CYPRESS™ programmer (CYP) free of charge. Upgrades are available at www.infineon.com.

2.3 Development kits

XMC7200 MCU evaluation kit is an XMC7000 MCU series-based Arduino compatible development platform. The kit showcases different features of the XMC7000 MCU series device.

2.3.1 Evaluation board

XMC7200 MCU evaluation kit shall have all the components required to ensure the proper functioning of the XMC7000 MCU series. The kit primarily demonstrates the key feature of this silicon and provides spins on an Arduino UNO compatible header to evaluate analog, digital communication, and other digital features. The kit features peripheral interfaces such as CAN FD interface, Gigabit Ethernet interface, QSPI flash, and M.2 connector for WLAN and Bluetooth® verification.

The XMC7200 MCU evaluation kit board will contain the XMC7000 MCU series device along with all the necessary passives such as decoupling capacitors. The board will be compatible with to Arduino “Uno” form factor to support many Arduino shields to showcase the functionalities of the XMC7000 MCU series device.

The board shall contain a PSoC™ 5LP MCU which will be used as an on-board programmer/debugger and for USB-serial interface for the XMC7200 MCU device. The firmware on PSoC™ 5LP MCU will enable boot loading over USB to upgrade the firmware.

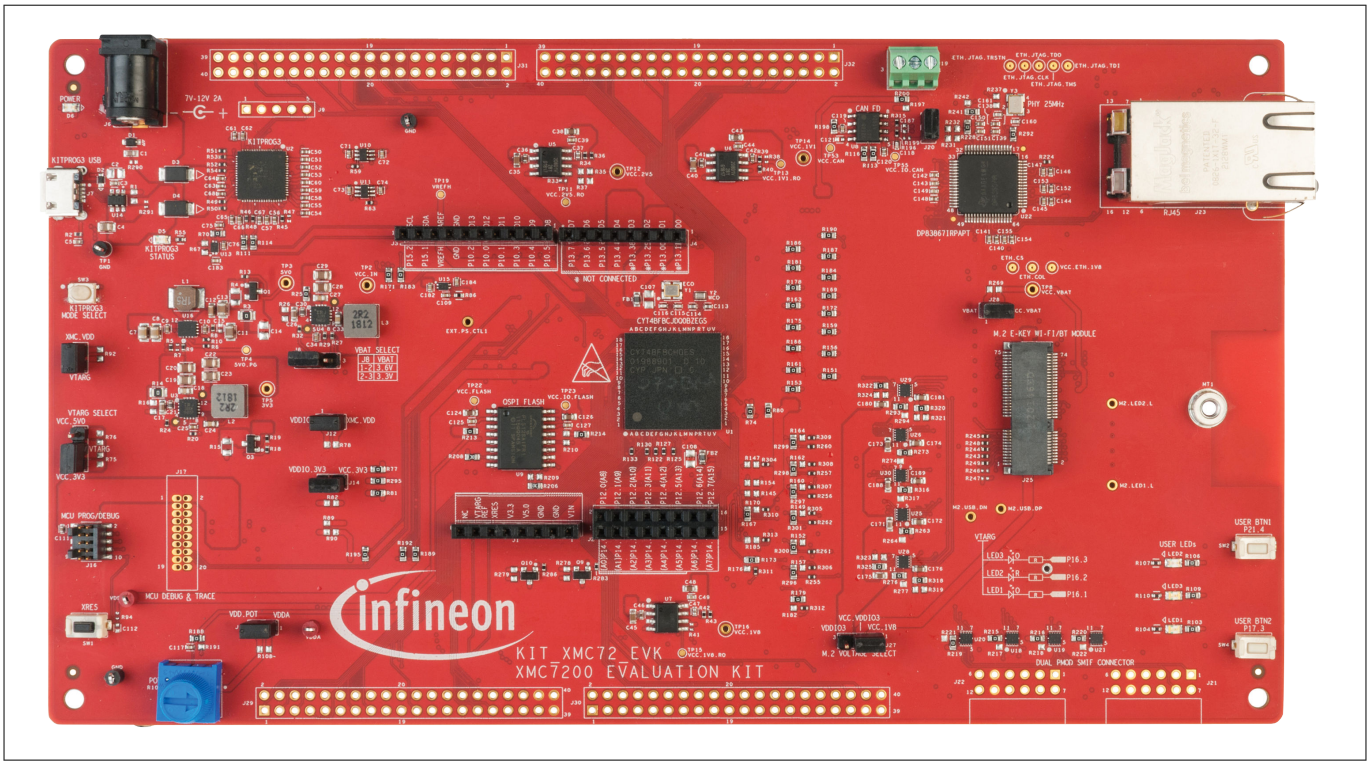


Figure 2 Example evaluation board

2.3.2 Peripheral driver library (PDL)

The Infineon PDL simplifies software development for XMC7000 (or CAT1A) family of devices. The PDL integrates device header files, startup code, and peripheral drivers into a single package. The drivers abstract the hardware functions into a set of easy-to-use APIs.

2.3.3 Development tools

Table 1 Supported development tools

Vendor	Emulators/probes	Compiler
IAR	I-JET	EWARM(8.42.x)
MTB	Minigprog4	J-link GCC

2.3.4 Open OCD

Open OCD provides on-chip programming and debugging support with a layered architecture of JTAG interface and TAP.

2.4 Application notes

See application note for additional information on XMC7000 device capabilities such as:

- Hardware connection information
- PDL folder structure, driver support, and its usage with third-party IDEs
- Startup sequence related to the device and individual cores

3 Document construction

3.1 Major sections

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

- Section – Presents the top-level architecture, how to get started, and conventions and overview information of the product.
- Chapter – Presents the chapters specific to an individual aspect of the section topic. These are the detailed implementation and information for some aspect of the integrated circuit.
- Registers technical reference manual – Supplies all device register details summarized in the technical reference manual. This is an additional document.

3.2 Documentation conventions

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

- The first is the use of *italics* when referencing a document title or file name.
- The 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 XMC7100 registers technical reference manual and XMC7200 registers technical reference manual.

3.2.2 Numeric naming

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

3.2.3 Unit of measure

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

Table 2 Unit of measure

Abbreviation	Unit of measure
bps	bits per second
°C	degrees celsius
dB	decibels
dBm	decibels-milliwatts
fF	femtofarads
G	giga
Hz	hertz
k	kilo, 1000

(table continues...)

Table 2 (continued) Unit of measure

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

3.2.4 Acronyms and abbreviations

This table lists the acronyms used in this document.

Table 3 Acronyms and abbreviations

Acronym	Description
A/D	analog to digital
ABS	absolute
ADC	analog-to-digital converter
AES	Advanced Encryption Standard
AHB	AMBA (advanced microcontroller bus architecture) high-performance bus, Arm® data transfer bus
Arm®	Advanced RISC Machine, a CPU architecture
BD	buffer descriptor
BOD	brownout detection
CAN FD	controller area network with flexible data rate
CBS	credit-based shaping
CFI	canonical format indicator
CMOS	complementary metal-oxide-semiconductor
CPHA	(SPI) clock phase
CPOL	(SPI) clock polarity
CPU	central processing unit
CPUSS	CPU subsystem
CRC	cyclic redundancy check, an error-checking protocol
CSV	clock supervisor
DDR	double data rate (also see SDR)
DES	data encryption standard
DMA	direct memory access
DW	data wire, same as P-DMA
ECC	error correcting code
ECO	external crystal oscillator
EEE	Energy Efficient Ethernet (IEEE Std 802.3az)
EOF	end of frame
ETM	embedded trace macrocell
FCS	frame check sequence
FIFO	first in first out
FLL	frequency locked loop
FPU	floating point unit

(table continues...)

Table 3 (continued) Acronyms and abbreviations

Acronym	Description
GHS	Green Hills tool chain with IDE
GPIO	general-purpose input/output
HSIOM	high-speed I/O matrix
HSM	hardware security module
IF	interface
I/O	input/output
IP	Internet protocol
IPG	inter-packet gap
I^2C	Inter-Integrated Circuit, a communications protocol
ILO	internal low-speed oscillator
IMO	internal main oscillator
IPC	inter-processor communication
IrDA	infrared interface
IRQ	interrupt request
JTAG	Joint Test Action Group
LAN	local area network (IEEE Std 802)
LLDP	link layer discovery protocol (IEEE Std 802.1AB)
LPI	low-power idle (IEEE Std 802.3az)
LVD	low-voltage detection
MAC	media access control (IEEE Std 802)
MCU	microcontroller unit
MCWDT	multi-counter watchdog timer
M-DMA	memory-direct memory access
MDC	management data clock
MII	media independent interface
MISO	master-in slave-out
MMIO	memory mapped I/O
MOSI	master-out slave-in
MPU	memory protection unit
NSP	non-standard preamble
NVIC	nested vectored interrupt controller
OTA	over-the-air programming
OTP	one-time programmable

(table continues...)

Table 3 (continued) Acronyms and abbreviations

Acronym	Description
OVD	over voltage detection
P-DMA	peripheral-direct memory access same as DW
PCS	physical coding sublayer
PFC	priority-based flow control (IEEE Std 802.1Qbb)
PLL	phase-locked loop
PHY	physical sublayer
POR	power-on reset
PPB	private peripheral bus
PPPoE	point-to-point protocol over ethernet
PPU	peripheral protection unit
PRNG	pseudo-random number generator
PTP	precision time protocol (IEEE Std 1588)
PWM	pulse-width modulation
RAM	random access memory
RISC	reduced-instruction-set computing
ROM	read-only memory
RTC	real-time clock
RX	reception
SAR	successive approximation register
SCB	serial communication block
SCL	I^2C serial clock
SDA	I^2C serial data
SDR	single data rate (also see DDR)
SECDED	single error correction double error detection
SerDes	serializer/deserializer
SHA	secure hash algorithm
SHE	secure hardware extension
SFD	start of frame delimiter
SGMII	serial Gigabit media independent interface
SMIF	serial memory interface
SMPU	shared memory protection unit
SNAP	subnetwork access protocol
SOF	start of frame

(table continues...)

Table 3 (continued) Acronyms and abbreviations

Acronym	Description
SPI	serial peripheral interface, a communications protocol
SRAM	static random-access memory
SWD	single wire debug
Tbit	bit period
TCP	transfer control protocol
TCPWM	timer/counter pulse-width modulator
TTL	transistor-transistor logic
TRNG	true random number generator
TS	timestamp
TSU	timestamp unit
TX	transmission
UART	universal asynchronous transmitter receiver, a communications protocol
UDP	user datagram protocol
VLAN	virtual LAN (IEEE Std 802.1Q)
WCO	watch crystal oscillator
WDT	watchdog timer reset
XIP	execute-in-place
XRES_L	external reset I/O pin (active low)
XTAL	crystal

4 CPU subsystem (CPUSS)

Top-level architecture

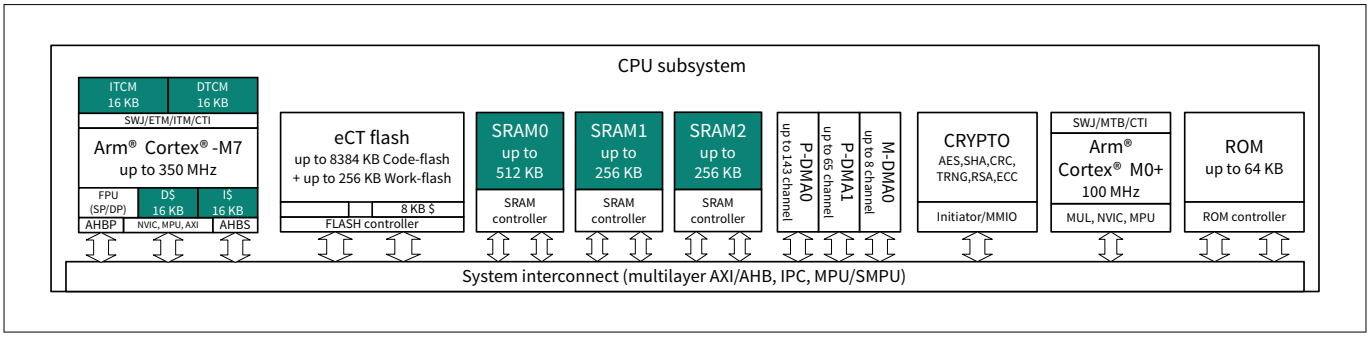


Figure 3 CPU system block diagram

The CPU subsystem is based on dual 32-bit Arm® Cortex® CPUs. The two Cortex®-M7 are the main CPUs. The Cortex®-M7 is designed for short interrupt response time, high code density, and high 32-bit throughput while maintaining a strict cost and power consumption budget. An additional Cortex®-M0+ CPU can implement security, safety, and protection features.

This section provides only an overview of the Arm® Cortex® CPUs in XMC7000 MCUs. For details, see the Arm® documentation sets for Cortex®-M7 and Cortex®-M0+.

LDREX/STREX and other exclusive access instructions are not supported. Therefore, it is recommended to implement inter process communication in a system that has multiple CPUs and shared memory.

4.1 Features

The XMC7000 MCU Arm® Cortex® CPUs have the following features:

- Little-endian byte ordering for data, memory, and CPU registers
- Each Cortex®-M7 has 16-KB instruction cache and 16-KB data cache with ECC support.
- Flash Controller has 8-KB cache for Cortex®-M0+, described In Code flash.
- Besides the shared system SRAM each Cortex®-M7 has 16 -KB of ITCM and 16-KB of DTCM (instruction/data tightly coupled memory) with ECC support. XMC7000 can access ITCM and DTCM with read and write wait “0”. See the device datasheet for address allocation of ITCM and DTCM.
- Cortex®-M7 has a floating-point unit (FPU) with single and double precision that supports single-cycle digital signal processing (DSP) instructions, and a memory protection unit (MPU). Cortex®-M0+ has an MPU.
- The Cortex®-M7 supports a subset of the Thumb instruction set (defined in the Arm® v7-M architecture reference manual). The Cortex®-M0+ supports the Arm® v6-M thumb instruction set (defined in the Arm® v6-M architecture reference manual). See Instruction set.
- Both CPUs have nested vectored interrupt controllers (NVIC) for rapid and deterministic interrupt response.
- Both CPUs have extensive debug support. For details, see the Program and debug interface.
  - SWJ: combined serial wire debug (SWD) and joint test action group (JTAG) ports
  - Breakpoints
  - Watchpoints
- Trace: Cortex®-M7: instrumentation trace macrocell (ITM) and embedded trace macrocell (ETM) with embedded trace buffer (ETB) and trace port interface unit (TPIU). Cortex®-M0+: micro trace buffer (MTB)
- Inter-processor communication (IPC) hardware - see the Inter-processor communication.

4.2 How it works

Both Cortex® CPUs are 32-bit processors with a 32-bit data path, 32-bit registers, and a 32-bit memory interface. They support a wide variety of instructions in the thumb instruction set. The CPUs support two operating modes (see Operating modes and privilege levels).

The Cortex®-M7 instruction set includes:

- Signed and unsigned, 32×32 -> 32-bit and 32×32 -> 64-bit, multiply and multiply-accumulate, all single-cycle
- Signed and unsigned 32-bit divides that take two to 12 cycles
- DSP instructions
- Complex memory-load and store access
- Complex bit manipulation

The Cortex®-M7 FPU has its own set of registers and instructions. It is compliant with the ANSI/IEEE Std 754-2008, IEEE standard for binary floating-point arithmetic.

The Cortex®-M0+ has a single cycle 32×32 -> 32-bit signed multiplication instruction.

4.3 Address map

Both CPUs have a fixed address map, with shared access to memory and peripherals except the PPB area, which is a private address space for each core. The 32-bit (4 GB) address space is divided into the regions shown in Table 4. Note that code can be executed from the code and SRAM regions.

Table 4 Address map for Cortex®-M7 and Cortex®-M0+

Address range	Name	Use
0x00000000 - 0x1FFFFFFF	Code	Program code region. You can also put data here. It includes the exception vector table, which starts at address 0.
0x20000000 - 0x3FFFFFFF	SRAM	Data region. You can also execute code from this region.
0x40000000 - 0x5FFFFFFF	Peripheral	All Peripheral registers. You cannot executed code from this region.
0x60000000 - 0x9FFFFFFF	External RAM	SMIF. You can execute code from this region.
0xA0000000 - 0xDFFFFFFF	External device	External device memory
0xE0000000 - 0xE00FFFFF	PPB	Peripheral registers within the CPU core.
0xE0100000 - 0xFFFFFFFF	Device	Device-specific system registers.

***Note**: Gaps in the address space are reserved. Do not access these gaps; if accessed, it can result in hard faults or BUS ERROR depending on which bus segment or peripheral an address is allocated to.*

4.4 Operating modes and privilege levels

Both CPUs support two operating modes and two privilege levels:

- Operating modes:
  - Thread mode – used to execute application software. The processor enters thread mode when it comes out of reset.
  - Handler mode – used to handle exceptions. The processor returns to thread mode when it has finished all exception processing.
- Privilege levels:
  - Unprivileged – the software has limited access to the MSR and MRS instructions, and cannot use the CPS instruction. It cannot access the system timer, NVIC, or system control block. It may have restricted access to memory or peripherals.
  - Privileged – the software can use all the instructions and has access to all resources.

In thread mode, the CONTROL register controls whether software execution is privileged or unprivileged. In handler mode, software execution is always privileged.

Only privileged software can write to the CONTROL register to change the privilege level. Unprivileged software can use the SVC instruction to transfer control to privileged software.

In handler mode, the MSP is always used. The exception entry and return mechanisms automatically update the CONTROL register, which may change whether MSP/PSP is 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.

4.5 Instruction set

Cortex®-M0+ is based on the Arm®v6-M architecture and supports all the 16-bit thumb instructions defined by the Arm®v7-M architecture excluding CBZ, CBNZ, and IT. In addition, it supports the following 32-bit thumb instructions: BL, DMB, DSB, ISB, MRS, and MSR.

Cortex®-M7 is based on the Arm®v7E-M architecture (Arm®v7-M with DSP extension) and supports all thumb instructions defined by that architecture, including the optional floating point instructions.

For details, see one of the Arm® Cortex® generic user guides, technical reference manuals or architecture reference manuals.

4.6 TCM interface

Cortex®-M7 has three TCM interfaces:

- ITCM, 64-bit data
- D0TCM, 32-bit data
- D1TCM, 32-bit data

The TCM interface implements the following functionality:

- Programmable read wait state (0 or 1).
- ECC functionally
  - Single-bit error automatic correction
  - Single- and multi-bit error detection and fault reporting
  - ECC error injection

4.6.1 TCM read wait state

The ITCM interface has a programmable read wait states. The ITCM and DTCM wait states are independently programmable through the CPUSS_CM7_x_CTL.ITCM_READ_WS and CPUSS_CM7_x_CTL.DTCM_READ_WS fields. When the CPUSS_CM7_x_CTL.ITCM_READ_WS or DTCM_READ_WS is '1', read access of TCM that set to '1' will get two cycles.

Note that TCM writes are always zero wait states.

Table 5 TCM read wait state configuration

Register	Bit field and bit name	Description
CPUSS_CM7_x_CTL[31:0]	ITCM_READ_WS	ITCM read wait states (writes have no wait states). 0: 0 wait state 1: 1 wait state
	DTCM_READ_WS	DTCM read wait states (writes have no wait states). 0: 0 wait state 1: 1 wait state

***Note**: The 'x' in the register name denotes the Cortex®-M7 core number.*

XMC7000 can access ITCM and DTCM with read wait '0'. Therefore, XMC7000 can use ITCM and DTCM with CPUSS_CM7_x_CTL.ITCM_READ_WS and CPUSS_CM7_x_CTL.DTCM_READ_WS fields with '0'.

4.6.2 TCM ECC

Both ITCM and DTCMs have ECC with SECDED scheme (single error correction, dual error detection) for functional safety. ITCM and DTCMs interfaces have different data widths the ECC is slightly different:

- ITCM 64-bit data/8-bit ECC
- DTCMs 32-bit data/7-bit ECC

4.6.2.1 TCM read modify write configuration

To calculate the correct ECC parity, all data bits in the stored word must be included. Therefore, when partial write is done (for example, a byte write to the DTCM), it works as follows.

1. The stored data word is read from memory.
2. Read data word is merged with the partial data as new data word.
3. ECC parity of the new data word is calculated.
4. The new data word and calculated ECC parity are stored to memory.

The Cortex®-M7 core already does all this provided the CPU-specific registers CM7_ITCMCR.RMW and CM7_DTCMCR.RMW bits are set. When ECC is used, they must be set to '1' for correct operation.

CM7_ITCMCR.RMW and CM7_DTCMCR.RMW bit initialization out of reset are controlled by the CPUSS_CM7_x_CTL.INIT_RMW_EN[1:0] register.

Note that this Cortex®-M7 feature also works for partial write from the AHB slave interface (AHBS) to the TCMs.

Table 6 TCM read-modify-write configuration

Register	Bit field and bit name	Description
CPUSS_CM7_x_CTL[31:0]	INIT_RMW_EN[1:0]	TCM read-modify-write enable initialization after reset: Bit 0: ITCM. '0': disabled; '1': enabled. Bit 1: DTCM. '0': disabled; '1': enabled. ***Note**: When TCM ECC is enabled, the read-modify-write functionality should be enabled. This prevents partial (sub-word) writes to the TCM.*

***Note**: The 'x' in the register name denotes the Cortex®-M7 core number.*

If they are disabled at reset then the following code example can be used to enable them in software:

```
CM7_ITCMCR EQU 0xE000EF90
CM7_DTCMCR EQU 0xE000EF94
LDR r11, =CM7_ITCMCR
LDR r0, [r11]
ORR r0, r0, #0x1:SHL:1 ; Set CM7_ITCMCR.RMW field
ORR r0, r0, #0x1:SHL:2 ; Set CM7_ITCMCR.RETEN field
STR r0, [r11]
LDR r11, =CM7_DTCMCR
LDR r0, [r11]
ORR r0, r0, #0x1:SHL:1 ; Set CM7_DTCMCR.RMW field
ORR r0, r0, #0x1:SHL:2 ; Set CM7_DTCMCR.RETEN field
STR r0, [r11]
DSB
ISB
```

See the Arm® documentation sets for Cortex®-M7 for more details.

4.6.2.2 ECC failure detection

If the ECC logic detects a single-bit error, calculate the corrected data and then write the corrected data back to the SRAM. Therefore, subsequent read (retry) for the same data will get the corrected data. However, if the ECC logic detects that the data has more than one-bit error or hard single-bit errors, then data cannot be corrected.

Thus, Cortex®-M7 may retry the same read a few cycles later and cause an ECC failure detection again. Note that the retry read is not guaranteed because the Cortex®-M7 core sometimes issues speculative reads.

In this case, if the retry read does happen, then this creates a live-lock situation; that is, the Cortex®-M7 repeats the same read over and over and the TCM logic keeps responding with retry. Because of the retry, it is guaranteed that the firmware will not use the erroneous data.

This live-lock will need to be resolved by a high-level interrupt, for example, the Cortex®-M7 watchdog timer (WDT) interrupt or an interrupt from the fault structure. Note that this may include the Cortex®-M0+ processor resetting and rebooting the Cortex®-M7.

4.6.2.3 ECC fault reporting

Both the correctable and the non-correctable ECC errors are reported to the central fault structure. For both correctable and non-correctable errors, a separate set of registers and a separate fault report channel is used.

This allows a more important/urgent non-correctable error to be reported immediately even if there was a preceding correctable error.

Note however that the three TCM interfaces share the registers and fault report channels for correctable and the non-correctable errors. This means that when two or more TCM interfaces detect an error at (almost) the same time, only one error will be reported and the other is lost

4.6.2.4 TCM initialization

The TCM memories need to be initialized before reading when ECC is enabled to avoid unwanted ECC faults.

ITCM writes for initialization may result in 64-bit read-modify-writes from CM7. The reads in this process may also cause ECC faults. To avoid this, use CPUSS_CM7_x_CTL.ITCM_ECC_CHECK_DIS bit. This bit need to be set (only) during initialization of ITCM to avoid ECC checking during initialization.

Table 7 TCM initialization

Register	Bit field and bit name	Description
CPUSS_CM7_x_CTL[31:0]	ITCM_ECC_CHECK_DIS	Disable ECC checking and thus fault reports. This also disables ECC correction (required to enable initialization). Intended use is for initialization. This bit is ignored when TCM_ECC_EN = 0.

***Note**: The 'x' in the register name denotes the Cortex®-M7 core number.*

4.6.2.5 ECC error injection

The TCM interface ECC logic supports ECC error injection through the following registers:

- CPUSS_CM7_x_CTL.ITCM_ECC_INJ_EN
- CPUSS_CM7_x_CTL.DTCM_ECC_INJ_EN
- CPUSS_ECC_CTL.WORD_ADDR
- CPUSS_ECC_CTL.PARITY

If CPUSS_CM7_x_CTL.ITCM_ECC_INJ_EN or CPUSS_CM7_x_CTL.DTCM_ECC_INJ_EN set to '1', a write transfer to word address specified in the CPUSS_ECC_CTL.WORD_ADDR field is performed, ECC parity generation uses the parity specified in the CPUSS_ECC_CTL.PARITY field rather than the calculated parity. Reading the same location will cause ECC fault.

***Note**: Set the RETRY bit to '0' in CM7 DTCMCR/ITCMCR before the read to avoid live-lock during ECC injection test. The RETRY can be enabled back to '1' after the ECC injection test is completed.*

Table 8 TCM ECC error injection register

Register	Bit field and bit name	Description
CPUSS_CM7_x_CTL[31:0]	ITCM_ECC_INJ_EN	ITCM ECC error injection enable: 0: Disabled 1: Enabled
	DTCM_ECC_INJ_EN	DTCM ECC error injection enable: 0: Disabled 1: Enabled
CPUSS_ECC_CTL[31:0]	WORD_ADDR[23:0]	Specifies the word address where an error will be injected. • On a write transfer to this SRAM address and when the corresponding RAM0/RAM1/RAM2/ ITCM/DTCM ECC_INJ_EN bit is '1', the parity (PARITY) is injected.
	PARITY[31:24]	ECC parity to use for ECC error injection at address WORD_ADDR. For the DTCM that has only seven parity bits, PARITY[30:24] is used as ECC parity.

***Note**: The 'x' in the register name denotes the Cortex®-M7 core number.*

4.6.2.6 ECC parity generation by software

To inject the ECC error for fault generation, ECC parity must be generated by software. Follow this procedure to generate an 8-bit ECC parity for ITCM. Parity generation calculates an 8-bit parity[7:0] over a 64-bit data word W[63:0]. First, a 128-bit ECC code word CS_SW[127:0] is created:

```
AW[119:0] = 120{1'b0};
AW[WORD_WIDTH-1:0] = W[WORD_WIDTH-1:0];
```

CW_SW[127:0] = {{8{1'b0}}, AW[119:0]};

Then, the 8-bit parity is calculated as the reduction XOR of the 128-bit code word CW_SW[127:0] ANDed with the following parity bit specific constants:

```
ECC_P0_SW =
128b00000001_10111111_10111011_01110101_10111110_00111010_01110010_11011100_01000100_10000100_01
001010_10001000_10010101_00101010_10101101_01011011;
ECC_P1_SW =
128b00000010_11011111_01110110_11111001_11011101_10011001_10111001_01110001_00010001_00001000_10
010011_00010001_00100110_10110011_00110110_01101101;
ECC_P2_SW =
128b00000100_11101111_11001111_10011111_10011010_11010101_11001110_10010111_00000110_00010001_00
011100_00100010_00111000_11000011_11000111_10001110;
ECC_P3_SW =
128b00001000_11110111_11101100_11110110_11101101_01100111_01001110_01101100_10011000_00100001_11
100000_01000011_11000000_11111100_00000111_11110000;
ECC_P4_SW =
128b00010000_11111011_01111011_10101111_01101011_10100110_10110101_10100110_11100000_00111110_00
000000_01111100_00000000_11111111_11111000_00000000;
ECC_P5_SW =
128b00100000_11111101_10110111_11001110_11110011_01101100_10101011_01011011_11111111_11000000_00
000000_01111111_11111111_00000000_00000000_00000000;
ECC_P6_SW =
128b01000000_11111110_11011101_01111011_01110100_11011011_01010101_10101011_11111111_11111111_11
111111_10000000_00000000_00000000_00000000_00000000;
ECC_P7_SW =
128b10000000_01111111_00000000_00000000_00000111_11111111_11111111_11111111_11010100_01000010_00
100101_10000100_01001011_10100110_01011100_10110111;
```

The parity bits are calculated as follows:

```
parity[0] = ^ (CW_SW[127:0] & ECC_P0_SW)
parity[1] = ^ (CW_SW[127:0] & ECC_P1_SW)
…
parity[7] = ^ (CW_SW[127:0] & ECC_P7_SW)
```

Follow this procedure to generate a 7-bit ECC parity for DTCM. Parity generation calculates a 7-bit parity[6:0] over a 32-bit data word W[31:0]. First, a 64-bit ECC code word CW_SW[63:0] is created:

```
CW_SW[63:0] = 64{1'b0};
CW_SW [31:0] = W[31:0];
```

Then, the 7-bit parity is calculated as the reduction XOR of the 64-bit code word CW_SW[63:0] ANDed with the following parity bit specific constants:

```
ECC_P0_SW = 64b00000011_01111111_00110110_11011011_00100010_01010100_00101010_10101011;
ECC_P1_SW = 64b00000101_10111101_11101011_01011010_01000100_10011001_01001101_00110101;
ECC_P2_SW = 64b00001001_11011101_11011100_11101110_00001000_11100010_01110001_11000110;
ECC_P3_SW = 64b00010001_11101110_10111011_10101001_10001111_00000011_10000001_11111000;
ECC_P4_SW = 64b00100001_11110110_11010111_01110101_11110000_00000011_11111110_00000000;
ECC_P5_SW = 64b01000001_11111011_01101101_10110100_11111111_11111100_00000000_00000000;
ECC_P6_SW = 64b10000001_00000011_11111111_11111000_00010001_00101100_10010110_01011111;
```

The parity bits are calculated as follows:

```
parity[0] = ^ (CW_SW[63:0] & ECC_P0_SW)
parity[1] = ^ (CW_SW[63:0] & ECC_P1_SW)
…
parity[6] = ^ (CW_SW[63:0] & ECC_P6_SW)
```

4.6.2.7 TCM enabling

The TCM interfaces can be enabled at reset in the system by CPUSS_CM7_x_CTL.INIT_TCM_EN.

Table 9 TCM enabling

Register	Bit field and bit name	Description
CPUSS_CM7_x_CTL[31:0]	INIT_TCM_EN[1:0]	TCM enable initialization after reset: Bit 0: ITCM. ‘0’: disabled; ‘1’: enabled. Bit 1: DTCM. ‘0’: disabled; ‘1’: enabled.

***Note**: The ‘x’ in the register name denotes the Cortex®-M7 core number.*

If they are disabled at reset, then the following code example can be used to enable both the instruction and data TCM interfaces in software:

```
CM7_ITCMCR EQU 0xE000EF90
CM7_DTCMCR EQU 0xE000EF94
LDR r11, =CM7_ITCMCR
LDR r0, [r11]
ORR r0, r0, #0x1 ; Set CM7_ITCMCR.EN field
STR r0, [r11]
LDR r11, =CM7_DTCMCR
LDR r0, [r11]
ORR r0, r0, #0x1 ; Set CM7_DTCMCR.EN field
STR r0, [r11]
DSB
ISB
```

See the Arm® documentation sets for Cortex®-M7 for more details.

Other masters can access ITCM/DTCM through specific address. Cortex®-M7_0 can access ITCM/DTCM of Cortex®-M7_1 through this address and Cortex®-M7_1 can access ITCM/DTCM of Cortex®-M7_0. Note that Cortex®-M7 can not access its own TCM memories through this address. Such an access will result in an address decode failure and will be returned as a bus error. See the datasheet for ITCM/DTCM address mapping.

Accessing the ITCM/DTCM by other masters is possible only when the power mode of the corresponding CM7 is ENABLED and TCMC_EN is '1'.

Before changing the CM7 power mode from ENABLED to another power mode, the following steps need to be performed (not doing so and trying to access the TCM may cause hang-up):

1. Disable access to the CM7 TCM by setting the field CM7_0/1_CTL.TCMC_EN to '0'.
2. Confirm that there are no outstanding accesses to the CM7 TCM by checking that the fields CM7_0/1_STATUS.TCMC_* are '0'. Repeat step 2 if necessary.
3. Now it is safe to change the CM7 power mode.

Table 10 TCM access control and CM7 power mode control

Register	Bit field and bit name	Description
CM7_x_PWR_CTL[31:0]	PWR_MODE[1:0]	CM7_x power mode: 0: Switch CM7_x off 1: Reset CM7_x 2: Put CM7_x in retained mode 3: Switch CM7_x on
CPUSS_CM7_x_CTL[31:0]	TCMC_EN	CM7 TCMC access control: 0: Disable access to the CM7 I/D-TCM slave port (AHBS). Access attempts will get a bus error response. 1: Enable access to the CM7 I/D-TCM slave port (AHBS).

***Note**: The 'x' in the register name denotes the Cortex®-M7 core number.*

4.7 Cache ECC fault reporting

Correctable and non-correctable faults are reported for cache memories.

A single set of correctable and non-correctable faults is created for I-cache and D-cache ECC faults. If ECC error occurs on both caches at the same time, I-cache ECC error has the priority for fault reporting.

4.8 Registers

4.8.1 Arm® specification registers

Both CPUs have sixteen 32-bit registers, as Table 11 shows:

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

Table 11 Cortex®-M7 and Cortex®-M0+ registers

Name	Type[1)]	Reset value	Description
R0 - R12	RW	Undefined	R0–R12 are 32-bit General-purpose registers for data operations.
MSP (R13) PSP (R13)	RW	[0x00000000][2)]	The SP is register R13. In thread mode, bit[1] of the CONTROL register indicates the stack pointer to use: 0 = MSP. This is the reset value 1 = PSP On reset, the processor loads the MSP with the value from address 0x00000000.
LR (R14)	RW	See note[3)]	The Link register (LR) is register R14. It stores the return information for subroutines, function calls, and exceptions.
PC (R15)	RW	[0x00000004][2)]	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 (see Table 12) at reset; it must always be '1'.
PSR	RW	Undefined	The Program Status register (PSR) combines: Application Program Status register (APSR) Execution Program Status register (EPSR) Interrupt program status register (IPSR)
APSR	RW	Undefined	The APSR contains the current state of the condition flags from previous instruction executions.
EPSR	RO	0x01000000	On reset, the EPSR thumb state bit is loaded with the value bit[0] of the register [0x00000004]. It must always be 1. In Cortex®-M7, other bits in this register control the state of interrupt-continuable instructions and the if-then (IT) instruction.

(table continues...)

Table 11 (continued) Cortex®-M7 and Cortex®-M0+ registers

Name	Type[1)]	Reset value	Description
IPSR	RO	0	The IPSR contains the current exception number.
PRIMASK	RW	0	The PRIMASK register prevents activation of all exceptions with configurable priority.
CONTROL	RW	0	The CONTROL Register controls: • The privilege level in Thread mode; see Operating modes and privilege levels. • The currently active stack pointer, MSP or PSP. • Cortex®-M7 only: Whether to preserve the floating-point state when processing an exception.
FAULTMASK	RW	0	Cortex®-M7 only. Bit 0 = 1 prevents the activation of all exceptions except NMI.
BASEPRI	RW	0	Cortex®-M7 only. When set to a nonzero value, prevents processing any exception with a priority greater than or equal to the value.

1) Describes access type during program execution in thread mode and handler mode. Debug access can differ.
2) [address] denotes the value stored at address
3) LR reset value is 0xFFFFFFFF in Cortex®-M7, undefined in Cortex®-M0+.

Use the MSR and MRS instructions to access the PSR, PRIMASK, CONTROL, FAULTMASK, and BASEPRI Registers. Table 12 and Table 13 show how the PSR bits are assigned.

Table 12 Cortex®-M7 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	APSR	Q	DSP overflow and saturation flag
26 – 25	EPSR	ICI/IT	Control interrupt-continuable and IT instructions
24	EPSR	T	Thumb state bit. Must always be ‘1’. Attempting to execute instructions when the T bit is 0 results in a HardFault exception
23 – 20	–	–	Reserved
19 – 16	APSR	GE	Greater than or equal flags for the SEL instruction
15 – 10	EPSR	ICI/IT	Control interrupt-continuable and IT instructions
9	–	–	Reserved

(table continues...)

Table 12 (continued) Cortex®-M7 PSR bit assignments

Bit	PSR register	Name	Usage
8 – 0	IPSR	ISR_NUMBER	Exception number of current ISR: 0 = Thread mode 1 = Reserved 2 = NMI 3 = HardFault 4 = MemManage 5 = BusFault 6 = UsageFault 7 - 10 = Reserved 11 = SVCall 12 = Reserved for debug 13 = Reserved 14 = PendSV 15 = SysTick 16 = IRQ0 … 255 = IRQ240

Table 13 Cortex®-M0+ PSR bit assignments

Bit	PSR register	Name	Usage
31	APSR	N	Negative flag
30	APSR	Z	Zero flag
29	APSR	C	Carry or borrow flag
28	APSR	V	Overflow flag
27 – 25	–	–	Reserved
24	EPSR	T	Thumb state bit. Must always be ‘1’. Attempting to execute instructions when the T bit is ‘0’ results in a HardFault exception.
23 – 6	–	–	Reserved

(table continues...)

Table 13 (continued) Cortex®-M0+ PSR bit assignments

Bit	PSR register	Name	Usage
5 - 0	IPSR	Exception Number	Exception number of current ISR: 0 = Thread mode 1 = Reserved 2 = NMI 3 = HardFault 4 - 10 = Reserved 11 = SVCall 12, 13 = Reserved 14 = PendSV 15 = SysTick 16 = IRQ0 … 47 = IRQ31

4.8.2 MCU control registers

Table 14 List of registers

Register name	Name	Description
CPUSS_CM0_CTL	CM0+ control	Specifies the Cortex®-M0+ operation
CPUSS_CM0_STATUS	CM0+ status	Indicates status of the Cortex®-M0+, such as power mode
CPUSS_CM7_x_CTL	CM7_x control	Specifies the operation of the Cortex®-M7
CPUSS_CM7_x_STATUS	CM7_x status	Indicates status of the Cortex®-M7, such as TCM access and power mode
CPUSS_ECC_CTL	ECC control	Specifies the word address and ECC parity where an error will be injected
CPUSS_CM7_x_PWR_CTL	CM7_x power control	Controls the CM7_x power state
CPUSS_CM7_x_PWR_DELAY_CTL	CM7_x power delay control	Number of clock cycle delays needed after power domain power-up

***Note**: The 'x' in the register name denotes the Cortex®-M7 core number.*

5 Inter-processor communication

Inter-processor communication (IPC) provides the functionality for multiple processors to communicate and synchronize their activities. IPC hardware is implemented using two register structures.

- IPC channel: Communication and synchronization between processors is achieved using this structure.
- IPC interrupt: Each interrupt structure configures an interrupt line, which can be triggered by a 'notify' or 'release' event of any IPC channel.

5.1 Features

The features of IPC are as follows:

- Implements locks for mutual exclusion between processors
- Allows sending messages between processors
- Supports multiple channels for communication
- Supports multiple interrupts, which can be triggered using notify or release events from the channels

5.1.1 IPC channel

An IPC channel is implemented as six hardware registers, as shown in Figure 4. The IPC Channel Registers are accessible to all processors in the system.

- IPC_STRUCTx_ACQUIRE: This register determines the lock feature of the IPC. The IPC channel is acquired by reading this register. If the SUCCESS field returns a '1', the read acquired the lock.

  If the SUCCESS field returns a '0', the read did not acquire the lock.

  Note that a single read access performs two functions:

  - The attempt to acquire a lock.
  - Return the result of the acquisition attempt (SUCCESS field).

  The atomicity of these two functions is essential in a multi-core system with multiple CPUs.

  The register also has bit fields that provide information about the processor that acquired it. When acquired, this register is released by writing any value into the IPC_STRUCTx_RELEASE Register. If the register was already in an acquired state another attempt to read the register will not be able to acquire it.
- IPC_STRUCTx_NOTIFY: This register is used to generate an IPC notify event. Each bit in this register corresponds to an IPC interrupt structure. The notify event generated from an IPC channel can trigger multiple interrupt structures.
- IPC_STRUCTx_RELEASE: Any write to this register will release the IPC channel. This register also has a bit that corresponds to each IPC interrupt structure. The release event generated from an IPC channel can trigger multiple interrupt structures. To only release the IPC channel and not generate an interrupt, the user can write a zero into the IPC Release Register.
- IPC_STRUCTx_DATA0 and IPC_STRUCTx_DATA1: These two 32-bit registers are meant to hold data. They can be considered as the shared data memory for the channel. Typically, these registers will hold messages that need to be communicated between processors. If the messages are larger than the 32-bit size, the user can place a pointer in the IPC_STRUCTx_DATA0 or IPC_STRUCTx_DATA1 Register.
- IPC_STRUCTx_LOCK_STATUS: This register provides the instantaneous lock status for the IPC channel. If the channel is acquired, this register provides details such as processor ID and protection context. The reading of lock status only provides an instantaneous status, which can be changed in the next cycle based on the activity of other processors on the channel.

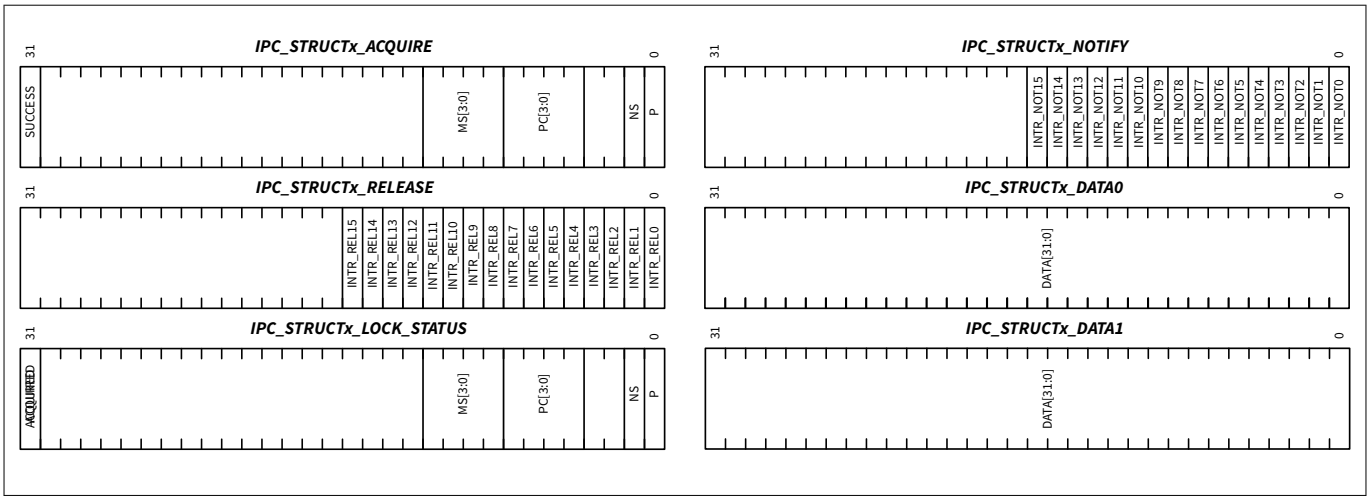


Figure 4 IPC channel structure

5.1.2 IPC interrupt

Each IPC interrupt line in the system has a corresponding IPC interrupt structure. An IPC interrupt can be triggered by a notify or a release event from any of the IPC channels in the system. The user can choose to mask any of the sources of these events using the IPC Interrupt Registers. Figure 5 shows the registers in an IPC Interrupt structure.

IPC_INTR_STRUCTx_INTR: This register provides the instantaneous status of the interrupt sources. Note that there are 16 notify and 16 release event bits in this registers. These are the notify and release events corresponding to the maximum 16 IPC channels. When a notify event is triggered in the IPC channel 0, the corresponding Notify0 bit is activated in the Interrupt Registers. A write of '1' to a bit will clear the interrupt.

IPC_INTR_STRUCTx_INTR_MASK: The bit in this register masks the interrupt sources. Only the interrupt sources with their masks enabled can trigger the interrupt.

IPC_INTR_STRUCTx_INTR_SET: A write of '1' into this register will set the interrupt.

IPC_INTR_STRUCTx_INTR_MASKED: This register provides the instantaneous value of the pending interrupts after they are masked. The value in this register is the result of the logical AND of registers IPC_INTR_STRUCTx_INTR and IPC_INTR_STRUCTx_INTR_MASK.

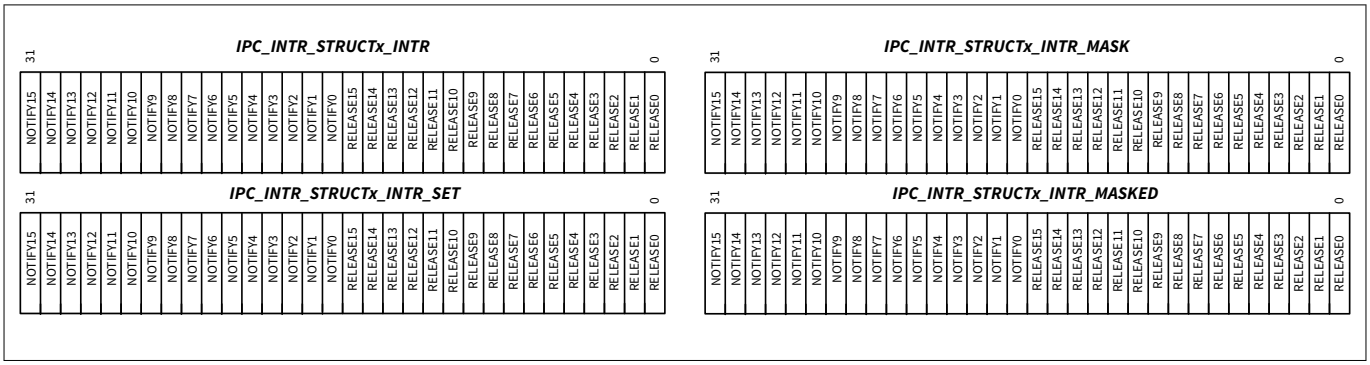


Figure 5 IPC interrupt structure

5.1.3 IPC channels and interrupts

The IPC block has a set of IPC interrupts associated with it. Each IPC Interrupt Register structure corresponds to an IPC interrupt line. This interrupt can trigger an interrupt on any of the processors in the system. The interrupt routing for processors are dependent on the device architecture.

Each IPC channel has a release and notify register, which can drive events on any of the IPC interrupts. Figure 6 shows an illustration of this relation between the IPC channels and the IPC interrupt structure.

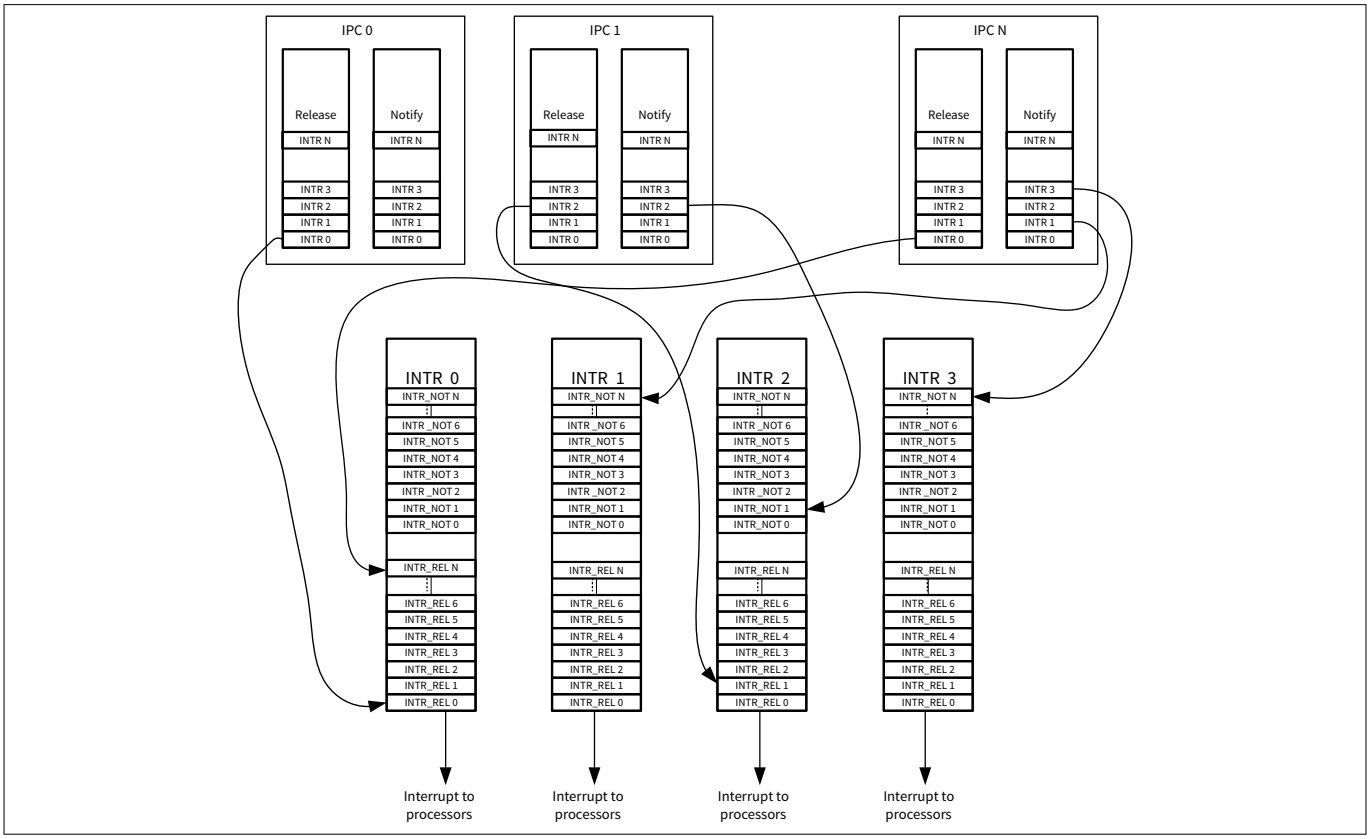


Figure 6 IPC channels and interrupts

5.2 Implementing locks

The IPC channels can be used to implement locks. Locks are typically used in multi core systems to implement some form of mutually exclusive access to a shared resource. When multiple processors share a resource, the processors are capable of acquiring and releasing the IPC channel. The processor can assume an IPC channel as a lock. The semantics of this code is that the access to the shared resource is gated by the processor's ownership of the channel. The processors need to acquire the IPC channel before they access the shared resource.

A failure to acquire the IPC channel signifies a lock on the shared resource because another processor has control of it. Note that the IPC channel will not enforce who acquires or releases the channel. All processors can acquire or release the IPC channel and the semantics of the code must make sure that the processor that acquires the channel is the one that releases it.

5.3 Message passing

IPC channels can be used to communicate messages between processors. In this use case, the channel is used in conjunction with the interrupt structures. The IPC channel is used to lock the access to the Data Register. The IPC channel is acquired by the sender and used to populate the message. The receiver reads the message and then releases the channel. Thus, between the sender placing data into the channel and receiver reading it, the channel is locked for all other tasks. The sender uses a notify event on the receiver's IPC interrupt to denote a send operation. The receiver acts on this interrupt and reads the data from the Data Register. After the reception is complete, the receiver releases the channel and can also generate a release event to the sender's IPC interrupt. Note that the action of locking the channel does not, in hardware, restrict access to the Data Register. This is a semantic that should be enforced by software.

Figure 7 portrays an example of a sender (processor A) sending data to a receiver (processor B). IPC interrupt A is configured to interrupt processor A. IPC interrupt B is configured to interrupt processor B.

1. The sender will attempt to acquire the IPC channel by reading the IPC_STRUCTx_ACQUIRE Register until the SUCCESS bit is set. Then, the sender has ownership of the channel for data transmission.
2. After the IPC channel is acquired, the sender has control of the channel for communication and places message data up to 64 bits in the IPC_STRUCTx_DATA0 and IPC_STRUCTx_DATA1 Registers.
3. Now that the message is placed in the IPC channel, the sender generates a notify event on the receiver's interrupt line. It does this by setting the corresponding bit in the IPC channel's IPC_STRUCTx_NOTIFY Register. This event creates a notify event at IPC interrupt B. If the IPC channel's notify event was enabled by setting the mask bit in the IPC interrupt B, this will generate an interrupt in the receiver.
4. When it receives IPC interrupt B, the receiver can read the IPC_INTR_STRUCTx_INTR_MASKED Register to understand which IPC channel had triggered the notify event. Based on this, the receiver identifies the channel to read and reads from the IPC channel's IPC_STRUCTx_DATA0 and IPC_STRUCTx_DATA1 Registers. The receiver has now received the data sent by the sender. It needs to release the channel so that other processors/processes can use it.
5. The receiver releases the channel. It also optionally generates a release event on the sender's IPC interrupt A. This will generate a release event interrupt on the sender if the corresponding channel release event was not masked.

On receiving the release interrupt, the sender can act on the event based on the application requirement. It can either try and reacquire the channel for further transmission or go on to other tasks because the transmission is complete.

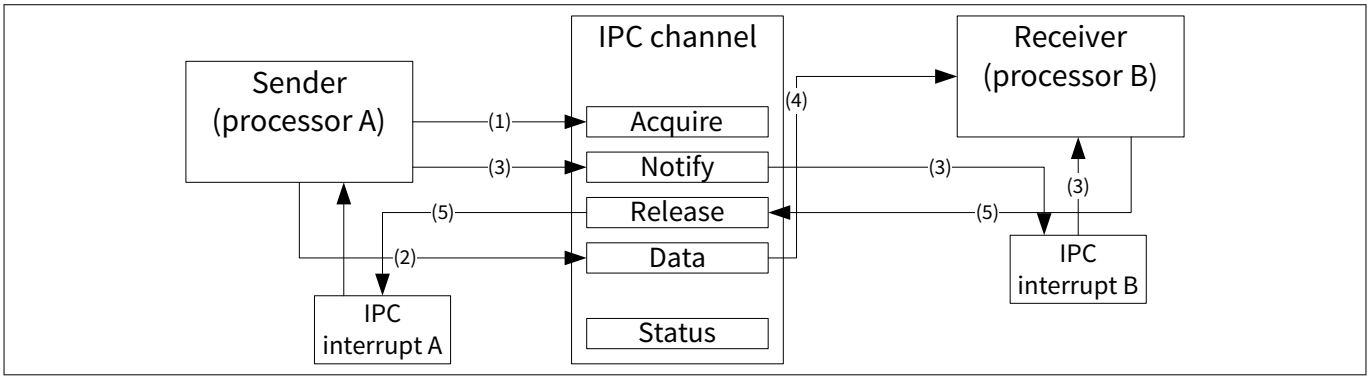


Figure 7 Sending messages using IPC

In the previous example, the size of the data being transmitted was just 32 bits. Larger messages can be sent as pointers. The sender can allocate a larger message structure in memory and pass the pointer in one of the 32-bit data registers. Figure 8 illustrates this usage. Note that the user code should implement the synchronization of the message read process.

- The implementation can stall the channel until the receiver has used up all the data in the message packet and the message packet can be rewritten. This is wasteful because it will stall other inter-processor communications as the number of IPC channels is limited.
- The receiver can release the channel as soon as it receives the pointer to the message packet. It implements the synchronization logic in the message packet as a flag, which the sender sets on write complete and receiver clears on a read complete.

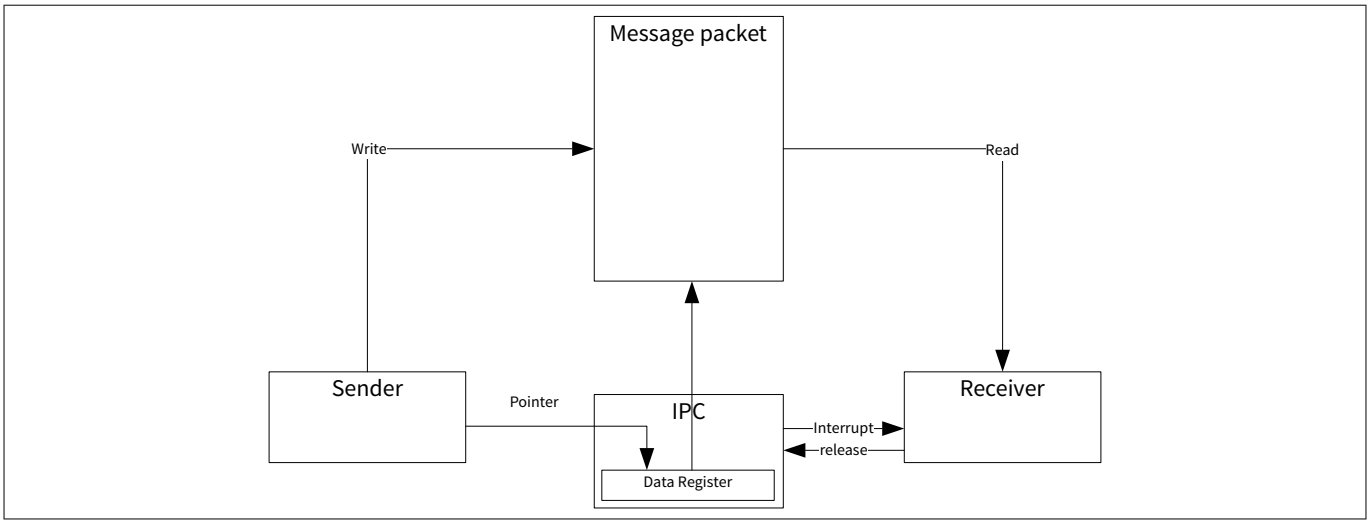


Figure 8 **Communicating larger messages**

5.4 Registers

Table 15 **IPC Registers**

Register	Name	Description
IPC_STRUCTx_ACQUIRE	IPC Structure Lock Acquire Register	This register is used to configure and acquire the lock
IPC_STRUCTx_RELEASE	IPC Structure Lock Release Register	This register is used to release the lock
IPC_STRUCTx_NOTIFY	IPC Structure Notification Register	This register is used to generate notifications for the interrupt structure
IPC_STRUCTx_DATA0	IPC Structure Data Register 0	This field holds a 32-bit data element that is associated with the IPC structure
IPC_STRUCTx_DATA1	IPC Structure Data Register 1	This field holds a 32-bit data element that is associated with the IPC structure
IPC_STRUCTx_LOCK_STATUS	IPC Structure Lock Status Register	This register shows the status of the IPC (lock status, access mode, and so on)
IPC_INTR_STRUCTx_INTR	IPC Interrupt Status Register	This register shows the status of the interrupts
IPC_INTR_STRUCTx_INTR_SET	IPC Interrupt Set Register	Writing to this register sets the corresponding IPC_INTR_STRUCTx_INTR
IPC_INTR_STRUCTx_INTR_MASK	IPC Interrupt Mask Register	This is the mask bit for corresponding bit in IPC_INTR_STRUCTx_INTR
IPC_INTR_STRUCTx_INTR_MASKED	IPC Masked Interrupt Register	This register is the bitwise AND of INTR and INTR_MASK

***Note**: In IPC_STRUCTx or IPC_INTR_STRUCT, 'x' signifies the IPC instance.*

6 Protection unit

Protection units in the XMC7000 series device enforce security based on different operations. A protection unit allows or restricts bus transfers on the bus infrastructure. The rules are enforced based on specific properties of a transfer.

6.1 Features

- An address range that is accessed by the transfer
  - Subregion: An address range is partitioned into eight equally-sized subregions and subregion can individual disables
- Access attributes such as:
  - Read/write attribute
  - Execute attribute to distinguish a code access from a data access
  - User/privilege attribute to distinguish access; for example, OS/kernel access from a task/thread access
  - Secure/non-secure attribute to distinguish a secure access from a non-secure access; the Arm® Cortex®-M CPUs do not natively support this attribute
  - A protection context attribute to distinguish accesses from different protection contexts; for peripheral-DMA (P-DMA) and memory-DMA (M-DMA), this attribute is extended with a channel identifier, to distinguish accesses from different channels
- Memory protection
- Provided by memory protection units (MPUs) and shared memory protection units (SMPUs)
  - MPUs distinguish user and privileged accesses from a single bus master
  - SMPUs distinguish between different protection contexts and between secure and non-secure accesses
- Peripheral protection
  - Provided by peripheral protection units (PPUs)
  - The PPUs distinguish between different protection contexts; they also distinguish secure from non-secure accesses and user mode accesses from privileged mode accesses
- Protection pair structure
- Software protection unit (SWPU): SWPUs define flash write (or erase) permissions, and eFuse read and write permissions. An SWPU comprises of the following:
  - Flash write protection unit (FWPU)
  - eFuse read protection unit (ERPU)
  - eFuse write protection unit (EWPU)

6.2 Configuration

6.2.1 Block diagram

Figure 9 gives an overview of the location of MPUs, SMPUs, and PPUs in the system.

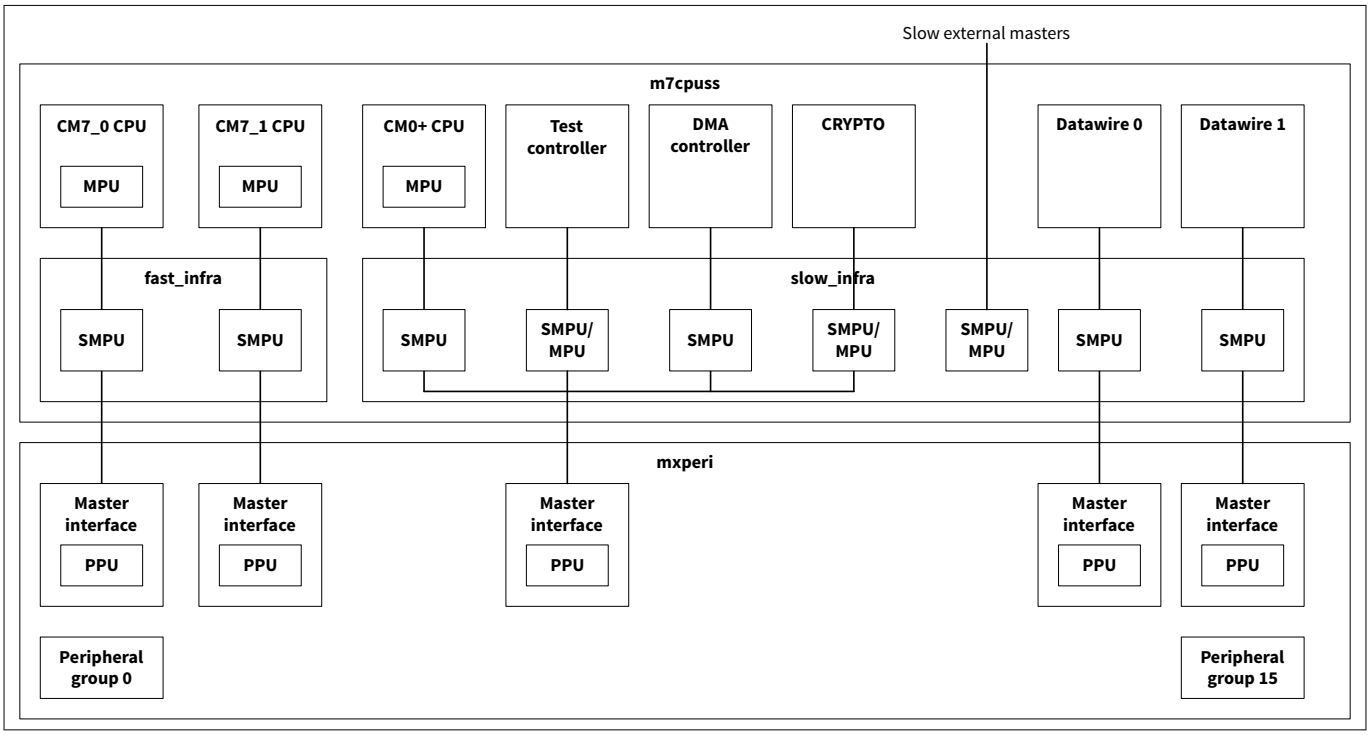


Figure 9 Protection unit locations

6.2.2 Protection unit structure

As mentioned, the MPU, SMPU, and PPU protection functionality follows the Arm® MPU definition:

- Multiple protection structures are supported.
- Each structure specifies an address range in the unified memory architecture and access attributes. Address range can be as small as 32 bytes.

A bus master may have a dedicated MPU. In a CPU bus master, the MPU is typically implemented as part of the CPU and under control of the OS/kernel. In a non-CPU bus master, the MPU is typically implemented as part of the bus infrastructure and under control of the OS/kernel of the CPU that “owns/uses” the bus master. If a CPU switches tasks or if a non-CPU switches ownership, the MPU settings are typically updated by OS/kernel software. The different MPU types are:

- An MPU that is implemented as part of the CPU. This type is found in the Arm® CM0+ and CM7 CPUs.
- CM7 MPU has 16 regions for each core, and CM0+ MPU has eight regions.
- An MPU that is implemented as part of the bus infrastructure. This type is found in bus masters such as test controller. The definition of this MPU type follows the Arm® MPU definition (in terms of memory region and access attribute definition) to ensure a consistent software interface.

The P-DMA, M-DMA, and cryptography component do not have an MPU. Instead, these components inherit the access control attributes of the bus transfer that programmed the channel or component.

The definition of SMPU and PPU follows the MPU definition and adds the capability to distinguish accesses from different protection contexts (the MPU does not include support for a protection context). If security is required, the SMPU and possibly PPUs registers must be controlled by a secure CPU that enforces system-wide protection.

Note that a peripheral group PPU only needs to provide access control to the peripherals within a peripheral group (peripherals with a shared bus infrastructure).

A protection violation is caused by a mismatch between a bus transfer's address region and access attributes and the protection structures' address range and access attributes.

A bus transfer that violates a protection structure results in a bus error. For AHB-Lite transfers, the address of each transfer beat is matched with the protection structure address range. The first violating beat in a transfer results in a bus error. A protection violation results in a bus error and the bus transfer will not reach its target. An MPU or SMPU violation that targets a peripheral will not reach the associated PPU. In other words, MPU and SMPU have a higher priority over PPU.

6.2.3 Master with missing access attributes

Not all masters provide all access attributes that are associated with a bus transfer. Some examples are:

- None of the bus masters have a natively fixed protection context attribute. This needs to be set dynamically based on the task being executed by the bus master.
- The Arm® Cortex®-M7 and Arm® Cortex®-M0+ CPUs provide a user/privilege attribute, but do not provide a secure/non-secure attribute natively.

To ensure system-wide restricted access, missing attributes are provided by register fields. These fields may be set during the boot process or by the secure CPU.

- The PROT_SMPU_MSx_CTL.PC_MASK_15_TO_1[], PROT_SMPU_MSx_CTL.PC_MASK_0, and PROT_MPUx_MS_CTL.PC[] register fields provide protection context functionality.
- The PROT_SMPU_MSx_CTL.P register field provides the user/privileged attribute for those masters that do not provide their own attribute.
- The PROT_SMPU_MSx_CTL.NS register provides the secure/non-secure attribute for those masters that do not provide their own attribute.
- Masters that do not provide an execute attribute have the execute attribute set to '0'.

The P-DMA, M-DMA and cryptography components inherit the access control attributes of the bus transfers that programmed the channels and component.

- The PROT_SMPU_MSx_CTL and PROT_MPUx_MS_CTL registers are only present for present masters.
- The PROT_MPUx_MS_CTL.PC_SAVED field (and associated protection context 0 functionality, which is discussed later in the chapter) is only present for the CM0+ master.
- The PROT_SMPU_MSx_CTL.P, PROT_SMPU_MSx_CTL.NS, PROT_SMPU_MSx_CTL.PC_MASK_15_TO_1[], and PROT_SMPU_MSx_CTL.PC_MASK_0 fields are not present for P-DMA, M-DMA and cryptography masters. The bus transfer attributes are inherited: from the master that owns the P-DMA and M-DMA channels that initiated the bus transfer.
- The PROT_MPUx_MS_CTL register is not present for the P-DMA and M-DMA and cryptography masters. The protection context (PC) bus transfer attribute is provided through inheritance.

6.3 Protection context

6.3.1 Protection context configuration

Each bus master has an PROT_MPUx_MS_CTL.PC[3:0] protection context field. This is used as the protection context attribute for all bus transfers that are initiated by the master. The SMPUs and PPUs allow or restrict bus transfers based on the protection context attribute.

Multiple masters can share a protection context. For example, a CPU and a Crypto controlled by the CPU may share a protection context (the CPU and Crypto PC[] fields are the same). Therefore, the CPU and Crypto share the SMPU and PPU access restrictions.

A bus master protection context is changed by reprogramming the master's PROT_MPUx_MS_CTL PC[] field.

Each bus master has an PROT_SMPU_MSx_CTL.PC_MASK_0 and PROT_SMPU_MSx_CTL.PC_MASK_15_TO_1, or PROT_SMPU_MSx_CTL.PC_MASK_0 only protection context mask field that identifies what protection contexts can be programmed for the bus master:

- Protection context field PROT_MPUx_MS_CTL.PC[3:0]. This register is controlled by the associated bus master and has the same access restrictions as the Bus Master's MPU registers.
- Protection context mask field PROT_SMPU_MSx_CTL.PC_MASK_15_TO_1[] and PROT_SMPU_MSx_CTL.PC_MASK_0. This register is controlled by the secure CPU and has the same access restrictions as the SMPU registers.

The PROT_SMPU_MSx_CTL.PC_MASK_15_TO_1[], PROT_SMPU_MSx_CTL.PC_MASK_0 field is a field that specifies if the PROT_MPUx_MS_CTL.PC[] field can be programmed with a specific protection context. Consider an attempt to program PROT_MPUx_MS_CTL.PC[] to '3':

- If PROT_SMPU_MSx_CTL.PC_MASK_15_TO_1[19] is '1', PROT_MPUx_MS_CTL.PC[] is set to '3'.
- If PROT_SMPU_MSx_CTL.PC_MASK_15_TO_1[19] is '0', PROT_MPUx_MS_CTL.PC[] is not changed.

As mentioned, the SMPUs and PPUs allow/restrict bus transfers based on the protection context attribute. The protection context provides an indirection between a bus master and the SMPU and PPU protection. This allows a single bus master to take on different protection roles, simply by reprogramming the protection context field PROT_MPUx_MS_CTL.PC[]. A change of protection contexts has limited CPU overhead, as the SMPU and PPUs do not have to be reprogrammed.

See the PERI_PC_NR in the datasheet for the number of available PCs.

6.3.2 Protection context 0 and 1

XMC7000 supports protection contexts to isolate software execution for security and safety purposes. Protection contexts are used to restrict access to memory and peripheral resources. XMC7000 supports eight protections contexts (PCs).

Out of eight PCs, two PCs are treated special: the entry to special PCs 0 and 1 is hardware controlled. For each PC i, a programmable exception handler address is provided: CPUSS_CM0_PCx_HANDLER.ADDR[31:0]. A CPU exception handler fetch that returns a handler address that matches the programmed CPUSS_CM0_PCx_HANDLER.ADDR[31:0] address value, causes the CM0+ PC to be changed to PC x by hardware. However, if the current PC is already 0 or 1, the current PC is not changed (an attempt to change the PC actually results in an AHB-Lite bus error).

This ensures that CPU execution in PC 0 or 1 cannot be interrupted/preempted by CPU execution in another PC 0 or 1. In other words, CPU execution in PC 0 or 1 requires cooperative multi-tasking between the different PCs. This means that handover between different PCs are software scheduled or controlled. A security implementation requires PC software to clear information that it wants to keep confidential from other PC software.

Note that each of the protection special PCs 0 and 1 have dedicated CPUSS_CM0_PC_CTL.VALID[x] field to specify that the PC's exception handler address is provided through CPUSS_CM0_PCx_HANDLER.ADDR[31:0]. If a PC's exception handler address is not provided, the PC is treated as an ordinary PC (PCs 2, 3, ..., 7).

Note that the current PC "pc" and a saved PC "pc_saved" implement a two entry stack. The hardware pushes the current PC to the stack upon entry of a special exception handler and hardware pops the saved PC from the stack upon entry of an ordinary exception handler. An attempt to enter a special exception handler from a special exception handler with a different PC results in an AHB-Lite bus error (which causes the CPU to enter the bus fault exception handler). This scenario should not occur in a carefully designed cooperative multi-tasking software implementation.

Note that pc_saved is also used upon entry of an ordinary exception handler, i.e., the PC of CM0+ will be changed to pc_saved when an ordinary exception handler is entered. User should also update pc_saved when changing the PC on CM0+.

Of the two special PCs, PC 0 is treated differently: It is the default PC value after a deep sleep reset. It has unrestricted access. Therefore, the Infineon boot code software always starts execution in PC 0. The boot code software initializes the protection structures and initializes the CPUSS_CM0_PCx_HANDLER registers.

After initialization of the protection information, the access to the protection information itself is typically restricted for all other PCs (the boot code software deploys the restrictions) and the protection information provides specific restricted access to the other special PCs and ordinary PCs (PCs 2, 3, ..., 7).

6.4 Protection structure

The MPU, SMPU, and PPU protection structure definition follows the Arm® definition. Each protection structure is defined by:

- An address region
- Access control attributes

A protection structure is always aligned on a 32-byte boundary in the memory space. Two registers define a protection structure: ADDR (Address register) and ATT (Attribute register). This alignment and organization allow straightforward protection of the protection structures by the protection scheme. This is elaborated upon later in this chapter.

As an ADDR register, MPU has PROT_MPUx_MPU_STRUCTy_ADDR, SMPU has PROT_SMPU_SMPU_STRUCTx_ADDR0/1, and PPU has PERI_MS_PPU_PRx_SL/MS_ADDR and PERI_MS_PPU_FXx_SL/MS_ADDR.

As an ATT register, MPU has PROT_MPUx_MPU_STRUCTy_ATT, SMPU has PROT_SMPU_SMPU_STRUCTx_ATT0/1, and PPU has PERI_MS_PPU_PRx_SL_ATT0-3, PERI_MS_PPU_PRx_MS_ATT, PERI_MS_PPU_FXx_SL_ATT0-3, PERI_MS_PPU_FXx_MS_ATT, PERI_MS_PPU_PRx_SL/MS_SIZE, and PERI_MS_PPU_FXx_SL/MS_SIZE.

6.4.1 Address region

The address region is defined by:

- The base address of a region as specified by ADDR.ADDR.
- The size of a region as specified by ATT.REGION_SIZE.
- Individual disables for eight subregions within the region, as specified by ADDR.SUBREGION_DISABLE.

The REGION_SIZE field specifies the size of a region. The region size is a power of 2 in the range of [256 B, 4 GB]. The base address ADDR specifies the start of the region, which needs to be aligned to the region size. A region is partitioned into eight equally-sized subregions. The SUBREGION_DISABLE field specifies individual enables for the subregions within a region.

For example, a REGION_SIZE of ‘8’ specifies a region size of 512 bytes. If the start address is 0x1000:5400 (512-byte aligned), the region ranges from 0x1000:5400 to 0x1000:55ff. This region is partitioned into the following eight 64-byte subregions:

subregion 0 from 0x1000:5400 to 0x1000:543f

subregion 1 from 0x1000:5440 to 0x1000:547f

…

subregion 7 from 0x1000:55c0 to 0x1000:55ff

If the SUBREGION_DISABLE is 0x82 (bit fields 1 and 7 are '1'), subregions 1 and 7 are disabled; subregions 0, 2, 3, 4, 5, and 6 are enabled.

In addition, an ATT.ENABLED field specifies if the region is enabled. Only enabled regions participate in the protection matching process. Matching identifies if a bus transfer address is contained within an enabled subregion (SUBREGION_DISABLE) of an enabled region (ENABLED).

6.4.2 Access control attributes

The access attributes specify access control to the region (shared by all subregions within the region). Access control is performed using a transfer's access attributes. The following access control fields are supported:

- Control for read accesses in user mode (ATT.UR field).
- Control for write accesses in user mode (ATT.UW field).
- Control for execute accesses in user mode (ATT.UX field).
- Control for read accesses in privileged mode (ATT.PR field).
- Control for write accesses in privileged mode (ATT.PW field).
- Control for execute accesses in privileged mode (ATT.PX field).
- Control for secure access (ATT.NS field).
- Control for individual protection contexts (ATT.PC_MASK_15_TO_1[] and PC_MASK_0, with PC_MASK_0 always constant at '1'). This protection context control field is present for SMPU.

The execute and read access control attributes are orthogonal. Execute transfers are typically read transfers. To allow execute or read transfers in user mode, both ATT.UR and ATT.UX need to be set to '1'. To allow data and read transfers in user mode, only ATT.UR needs to be set to '1'.

In addition, the ATT.PC_MATCH control field is supported, which controls the "matching" and "access evaluation" processes. This control field is only present for the SMPU protection structures.

For example, only protection context 2 can access a specific address range and these accesses are restricted to read and write secure accesses in privileged mode. The access control fields are programmed as follows:

- ATT.UR is '0': read accesses in user mode not allowed.
- ATT.UW is '0': write accesses in user mode not allowed.
- ATT.UX is '0': execute accesses in user mode not allowed.
- ATT.PR is '1': read accesses in privileged mode allowed.
- ATT.PW is '1': write accesses in privileged mode allowed.
- ATT.PX is '0': execute accesses in privileged mode not allowed.
- ATT.NS is '0': secure access required.
- ATT.PC_MASK_15_TO_1[10] is '1', and ATT.PC_MASK_0 is '1': protection context 0 and 2 accesses enabled (all other protection contexts are disabled).
- ATT.PC_MATCH is '0': the ATT.PC_MASK_15_TO_1[] and PC_MASK_0 field is used for access evaluation.
- Three separate access evaluation subprocesses are distinguished:
  - A subprocess that evaluates the access based on read/write, execute, and user/privileged access attributes.
  - A subprocess that evaluates the access based on the secure/non-secure attribute.
  - A subprocess that evaluates the access based on the protection context index (only used by the SMPU and PPU when ATT.PC_MATCH is '0').

If all access evaluations are successful, access is allowed. If any process evaluation is unsuccessful, access is not allowed.

Matching the bus transfer address and access evaluation of the bus transfer (based on access attributes) are two independent processes:

- Matching process. For each protection structure, the process identifies if a transfer address is contained within the address range. This identifies the matching regions.
- Access evaluation process. For each protection structure, the process evaluates the bus transfer access attributes against the access control attributes.

A protection unit typically has multiple protection structures. It evaluates the protection structures in decreasing order. The first matching structure provides the access control attributes for the evaluation of the

transfer's access attributes. In other words, higher-indexed structures take precedence over lower-indexed structures.

***Note**:*
- *If no protection structure provides a match, access is allowed.*
- *If multiple protection structures provide a match, the access control attributes for access evaluation are provided by the protection structure with the highest index.*

As mentioned, the protection unit evaluates the protection structures in decreasing order. From a security requirements perspective, this is of importance: it should not be possible for a non-secure protection context to add protection structures that have a higher index than the protection structures that provide secure access. The protection structure with a higher index can be programmed to allow non-secure accesses. Therefore, in a secure system, the higher programmable protection structures are protected to only allow restricted accesses. For more details, see Protection structure types.

6.4.3 Protection violation

If an MPU, SMPU, or PPU detects a not-allowed transfer, the bus transfer results in a bus error. Protection violations are captured in the fault report structure, and the fault report structures can generate an interrupt to indicate the occurrence of a fault. In addition, information on the violating bus transfer is communicated to the fault report structure. This is useful if the violating bus master cannot resolve the bus error by itself, but requires another CPU bus master to resolve the bus error on its behalf. Note that violating CPUs react by execution of exceptions.

For details on exceptions, see the Arm® documentation sets for Cortex®-M7 and Cortex®-M0+.

The bus transfer does not reach its target memory location or peripheral register. For write transfers that violate PPU protection, the bus master will not see the bus error if buffering is enabled (CPUSS_BUFF_CTL.WRITE_BUFF = 1). This is because the AHB-Lite bridges in the bus infrastructure will buffer the write transfer and send the OK response to masters. In this case, the system must depend on the fault reported by PPU.

6.4.4 Protection of protection structures

The MPU, SMPU, and PPU-based protection architecture is consistent and provides the flexibility to implement different system-wide protection schemes. Protection structures can be set once at boot time or can be changed dynamically during device execution. For example, a CPU RTOS can change the CPU's MPU settings; a secure CPU can change the SMPU and PPUs settings. From security requirements, it is necessary to prevent reprogramming of the protection structure from a malicious attacker.

Registers of MPU, SMPU, and PPU are the same registers as other peripherals. Furthermore, the protection structure itself can be included in the address range of the protection structure. That is, protects the protection structure by protection structure.

The first (slave) protection structure protects the resource and the second (master) protection structure protects the protection (address range of the second protection structure includes both the master and slave protection structures). We refer to the slave and master protection structures as a protection pair. Note that the address range of the master protection structure is known, that is, it is constant.

The protection architecture is flexible enough to allow for variations:

- Exclusive peripheral ownership can be shared by more than two protection contexts.
- The ability to change ownership is under control of a single protection context, and exclusive or non-exclusive peripheral ownership is shared by multiple protection contexts.

Note that in secure systems, typically a single secure CPU is used. In these systems, the ability to change ownership is assigned to the secure CPU at boot time and not dynamically changed. Therefore, it is strongly advised to assign the secure CPU its own, dedicated protection context.

Both PPU and SMPU are intended to distinguish between different protection contexts and to distinguish secure from non-secure accesses. Both PPU and SMPU protection use protection structure pairs. In the SMPU, the slave protection structure provides SMPU protection information and the master protection structure provides PPU protection information (the master and slave protection structures are registers).

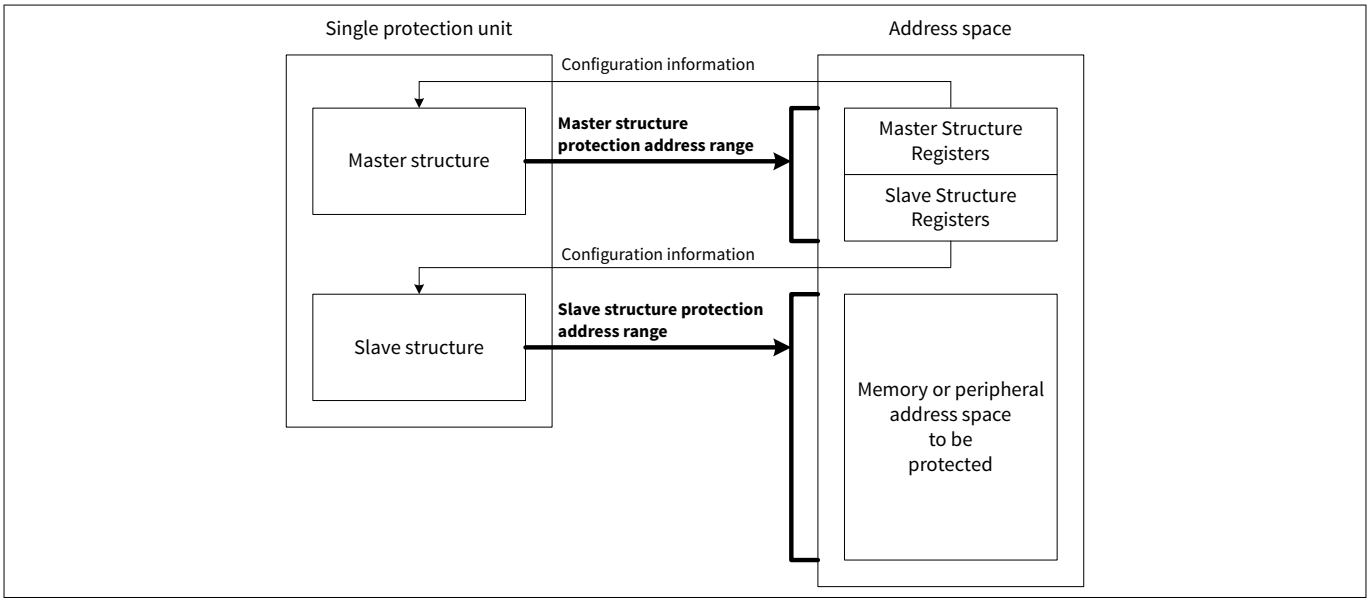


Figure 10 Concept of master and slave structure

6.4.5 MPU

The MPUs are associated to a single master. An MPU distinguishes user and privileged accesses from a single bus master. However, the capability exists to perform access control on the secure/non-secure attribute.

The MPU protection structures do not provide protection context control attributes.

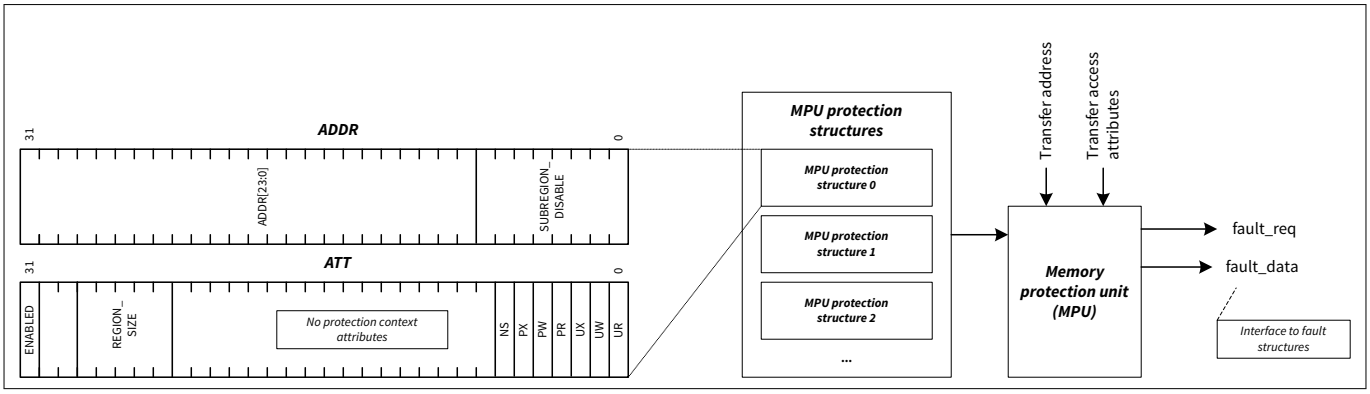


Figure 11 MPU functionality

6.4.6 SMPU

The SMPU is shared by all bus masters. The SMPU distinguishes between different protection contexts; it also distinguishes secure from non-secure accesses and user mode from privileged mode accesses.

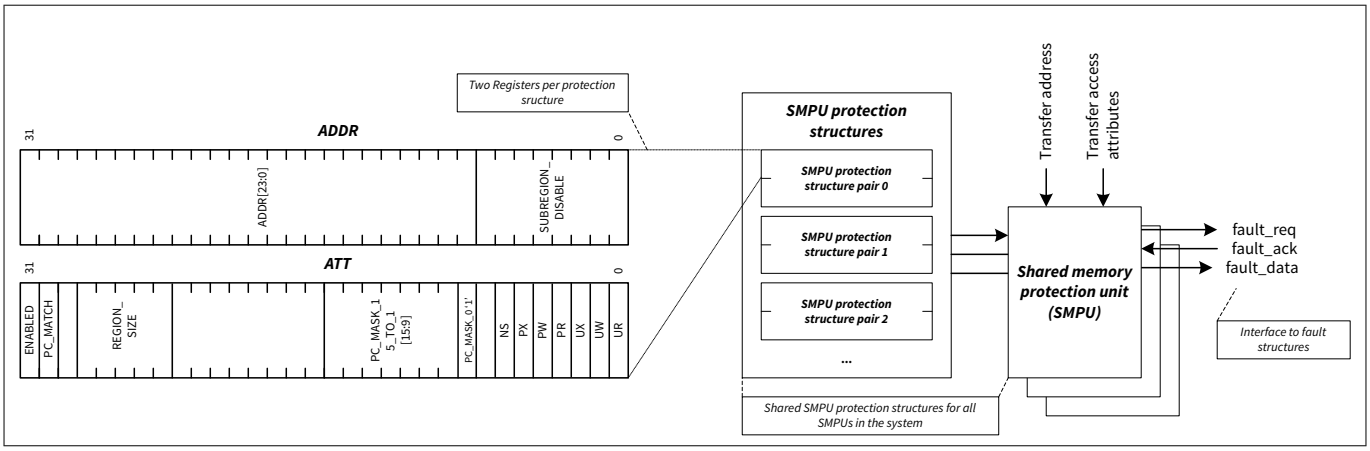


Figure 12 SMPU functionality

Single set of SMPU region structures provides the same protection information to all SMPUs in the systems.

The perform access control function on the user/privileged mode attribute in SMPU must not be used for accesses performed by the CM7 CPUs. AXI accesses performed in user mode of CM7 CPU may be marked as privileged accesses. See section 5.4.3 of the Arm® Cortex®-M7 technical reference manual for more details.

Note that there are no SMPUs on the AHBP ports. Therefore, SMPU must not be used to protect the peripherals address range. Peripherals are protected by PPU.

6.4.7 PPU

The PPUs are situated in the peripheral block and are associated with a peripheral group (peripherals with a shared AHB-Lite bus infrastructure). A PPU is shared by all bus masters. The PPU distinguishes between different protection contexts; it also distinguishes secure from non-secure accesses and user mode from privileged mode accesses.

The minimum region size of the MPU and SMPU is 10 bytes, but PPU can set the region size of at least 4 bytes. (register size).

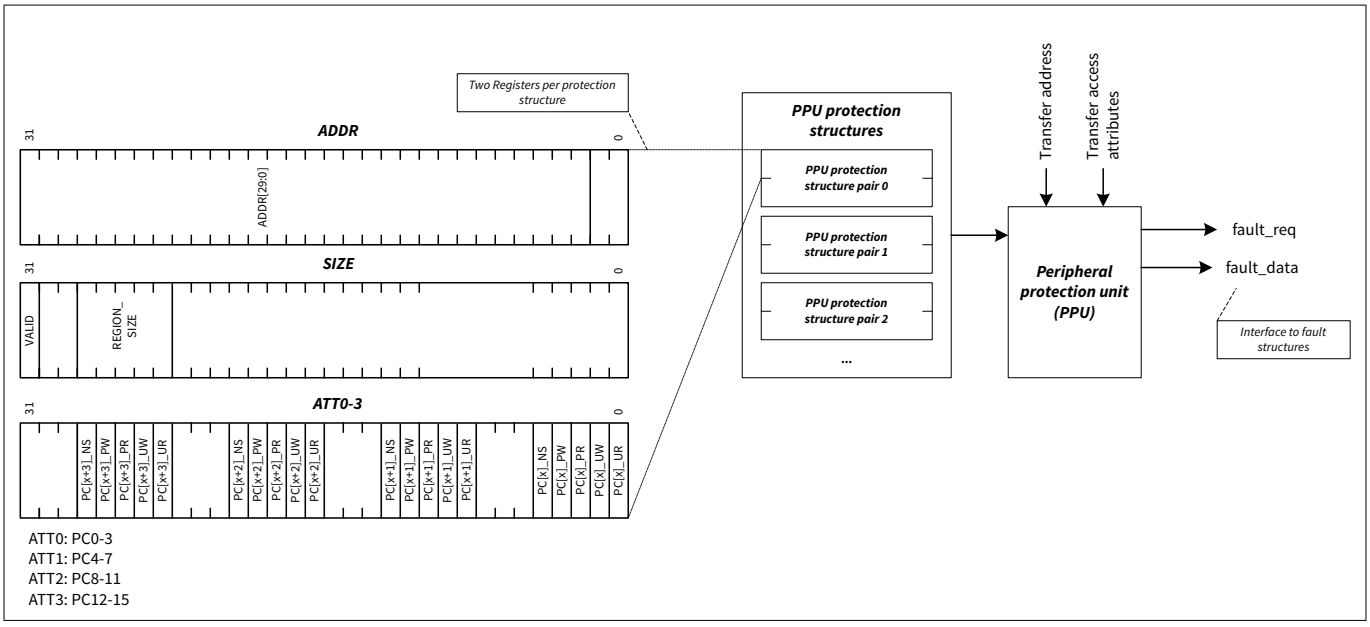


Figure 13 PPU functionality

Compared to the MPU and SMPU, a PPU have large number of protection regions. However, regions protected by PPU are mostly known address. Therefore, there are two types of PPU.

- Fixed PPU structure
  - To protect resources with a known address range. In other words, the ADDR.ADDR and SIZE.REGION_SIZE fields are fixed.
- Programmable PPU structure
  - To protect resources with an unknown address range, full programmability of a protection region's address range definition is used.

The programmable structure pairs slave address regions may overlap with other slave address regions. A transfer address is matched against all master and slave address regions. The master protection structures and programmable slave protection structures are given higher priority than the fixed slave protection structures. Following are the protection structures arranged from HIGH to LOW priority:

- Master protection structures
- Programmable slave protection structures
- Fixed slave protection structures

Note that programmable slave address regions have higher priority than fixed slave address regions.

The slave structures are programmed during the boot process when the PC is 0. For programmable protection structure pairs, this includes programming the slave address of the peripheral resource. After the boot process, slave address regions are not reprogrammed; only the master and slave attributes are reprogrammed. In other words, after the boot process, the protected peripheral resources are fixed and only the ownership of these resources (slave attributes) or the right to change the resource ownership (master attributes) are programmable/flexible.

Each protection structure provides ATT.NS, PW, PR, UW, UR access attributes for all PCs, except for PC 0. A PPU structure does not support PX and UX access attributes: peripheral transfer should have the Execute transfer attribute set to '0'. Note that execution from a peripheral address region is not allowed.

***Note**: When writing PPU attributes for any PPU region, when the write buffer in a bus infrastructure is enabled (CPUSS_BUFF_CTL.WRITE_BUFF is set to '1'), new PPU access attributes are not applied to the region immediately adjacent to the respective PPU region because the bus bridge responds. To make sure the attributes are updated properly, either configure the PPU attributes through non-buffered writes (by setting CPUSS_BUFF_CTL.WRITE_BUFF to '0') or in the case of buffered writes, make sure the attributes are updated by reading back PPU attributes before accessing the respective PPU region.*

The programmable and fixed PPU structures are shared by all PERI master interfaces. Most of the protection information uses a single SRAM memory.

6.4.7.1 ECC for SRAM

The SRAM stored protection information is supported by ECC. This ECC supports single error correction and double error detection (SECDED). The ECC is applied to the RAM word bits and the word address that is used to access the SRAM. If correctable error (single bit) is detected during SRAM read operation, the ECC corrects the data. However, the corrected data is not updated into SRAM. If non-correctable error is detected, then the current AHB transfer is aborted. These errors are communicated through the fault reporting structure.

6.4.7.2 ECC error injection

The ECC faults can be debugged through an ECC parity injection mechanism.

- PERI_ECC_CTL.PARITY: ECC parity to use for ECC error injection at PERI_ECC_CTL.WORD_ADDR. Note that this field will be used by hardware only when ECC error injection is enabled by setting PERI_ECC_INJ_EN to '1'.
- PERI_ECC_CTL.ECC_INJ_EN: Enables error injection for PERI protection structure SRAM. If this is '1', the parity (ECC_CTL.PARITY) is used when a write is done to the PERI_ECC_CTL.WORD_ADDR of the SRAM.
- PERI_ECC_CTL.WORD_ADDR: Specifies the word address where the parity is injected. When write access to this SRAM address is detected and PERI_ECC_CTL.ECC_INJ_EN bit is '1', the parity (PERI_ECC_CTL.PARITY) is injected.

6.4.7.3 ECC parity generation by software

To inject the ECC error for fault generation, ECC parity must be generated by software. Follow this procedure to generate 8-bit ECC parity.

1. Read present PPU attribute values of the target PPU structure ATT0-3 register. 'x' indicates PPU_PROG or FIXED_STRUCT number.
2. Generate ACTUALWORD [75:0].

   ACTUALWORD [75:0] = {0, {PC15_{NS, PW, PR, UW, UR}}.{PC1_{ NS, PW, PR,UW, UR}}}. Non-existing PC attributes values are set to '0'.
3. Calculate ADDR [10:0]. The ADDR can be calculated as follows:

   PERI_MS_PPU_PRx_SL_ATT0-3: x * 2

   PERI_MS_PPU_PRx_MS_ATT0-3: x * 2 + 1

   PERI_MS_PPU_FXx_SL_ATT0-3: x * 2 + 64

   PERI_MS_PPU_FXx_MS_ATT0-3: x * 2 + 65

The ‘x’ in the register name denotes the suffix number. ADDR [10:0] is set to PERI_ECC_CTL.WORD_ADDR.

4. Generate ECC parity using the following scheme.

```
CODEWORD_SW [127:0] = 128{1’b0};
CODEWORD_SW [75:0] = ACTUALWORD [75:0];
CODEWORD_SW [86: 76] = ADDR [10:0];

ECC_P0_SW =
128’b00000001_10111111_10111011_01110101_10111110_00111010_01110010_11011100_01000100_10000100_0
1001010_10001000_10010101_00101010_10101101_01011011;
ECC_P1_SW =
128’b00000010_11011111_01110110_11111001_11011101_10011001_10111001_01110001_00010001_00001000_1
0010011_00010001_00100110_10110011_00110110_01101101;
ECC_P2_SW =
128’b00000100_11101111_11001111_10011111_10011010_11010101_11001110_10010111_00000110_00010001_0
0011100_00100010_00111000_11000011_11000111_10001110;
ECC_P3_SW =
128’b00001000_11110111_11101100_11110110_11101101_01100111_01001110_01101100_10011000_00100001_1
1100000_01000011_11000000_11111100_00000111_11110000;
ECC_P4_SW =
128’b00010000_11111011_01111011_10101111_01101011_10100110_10110101_10100110_11100000_00111110_0
0000000_01111100_00000000_11111111_11111000_00000000;
ECC_P5_SW =
128’b00100000_11111101_10110111_11001110_11110011_01101100_10101011_01011011_11111111_11000000_0
0000000_01111111_11111111_00000000_00000000_00000000;
ECC_P6_SW =
128’b01000000_11111110_11011101_01111011_01110100_11011011_01010101_10101011_11111111_11111111_1
1111111_10000000_00000000_00000000_00000000_00000000;
ECC_P7_SW =
128’b10000000_01111111_00000000_00000000_00000111_11111111_11111111_11111111_11010100_01000010_0
0100101_10000100_01001011_10100110_01011100_10110111;

PARITY[0] = ^ (CW_SW[127:0] & ECC_P0_SW)
PARITY[1] = ^ (CW_SW[127:0] & ECC_P1_SW)
…
PARITY[7] = ^ (CW_SW[127:0] & ECC_P7_SW)
```

***Note**: ‘^’ means reduction XOR. For example, ^(4'b0011) = 0^0^1^1. ECC parity is set to PERI_ECC_CTL.PARITY.*

5. Set the PERI_ECC_CTL.ECC_INJ_EN to ‘1'

6. Read and write back with the same value to the target PPU structure ATT0-3. A write back will inject parity value from the ECC_CTL.PARITY[7:0] register to the SRAM PPU structure.

7. Read the target PPU structure ATT0-3. A read will generate an ECC error.

6.4.8 Protection structure types

Different protection structure types are used because some resources, such as Peripheral registers, have a fixed address range. For security, protection structures require pairs of neighboring protection structures.

Three types of protection structures, which have a consistent register interface are described here:

- Programmable protection structures – These structures are used by the MPUs.
- Fixed protection structure pairs. These structures are used by the PPUs. Both structures have a fixed, constant address region and do not have the ATT.UX and ATT.PX attributes. In addition, PC0 is permitted with all access attributes (PC0 has unrestricted access). The master structure has the ATT.UR and ATT.PR attributes as constant '1' (reading is always allowed). See Figure 14.
- Programmable protection structure pairs. These structures are used by the PPU and SMPU. The master structure has a fixed, constant address region. The slave structure has a programmable address region. The SMPU master structure has the ATT.UX and ATT.PX attributes as constant '0' (execution is never allowed) and the ATT.UR and ATT.PR attributes as constant '1' (reading is always allowed). The Both PPU structures do not have ATT.UX and ATT.PX attributes. In addition, PC0 is permitted with all access attributes (PC0 has unrestricted access). The PPU master structure has the ATT.UR and ATT.PR attributes as constant '1' (reading is always allowed). See Figure 15 and Figure 16.

Note that the master protection structure in a protection structure pair is only required to address security requirements.

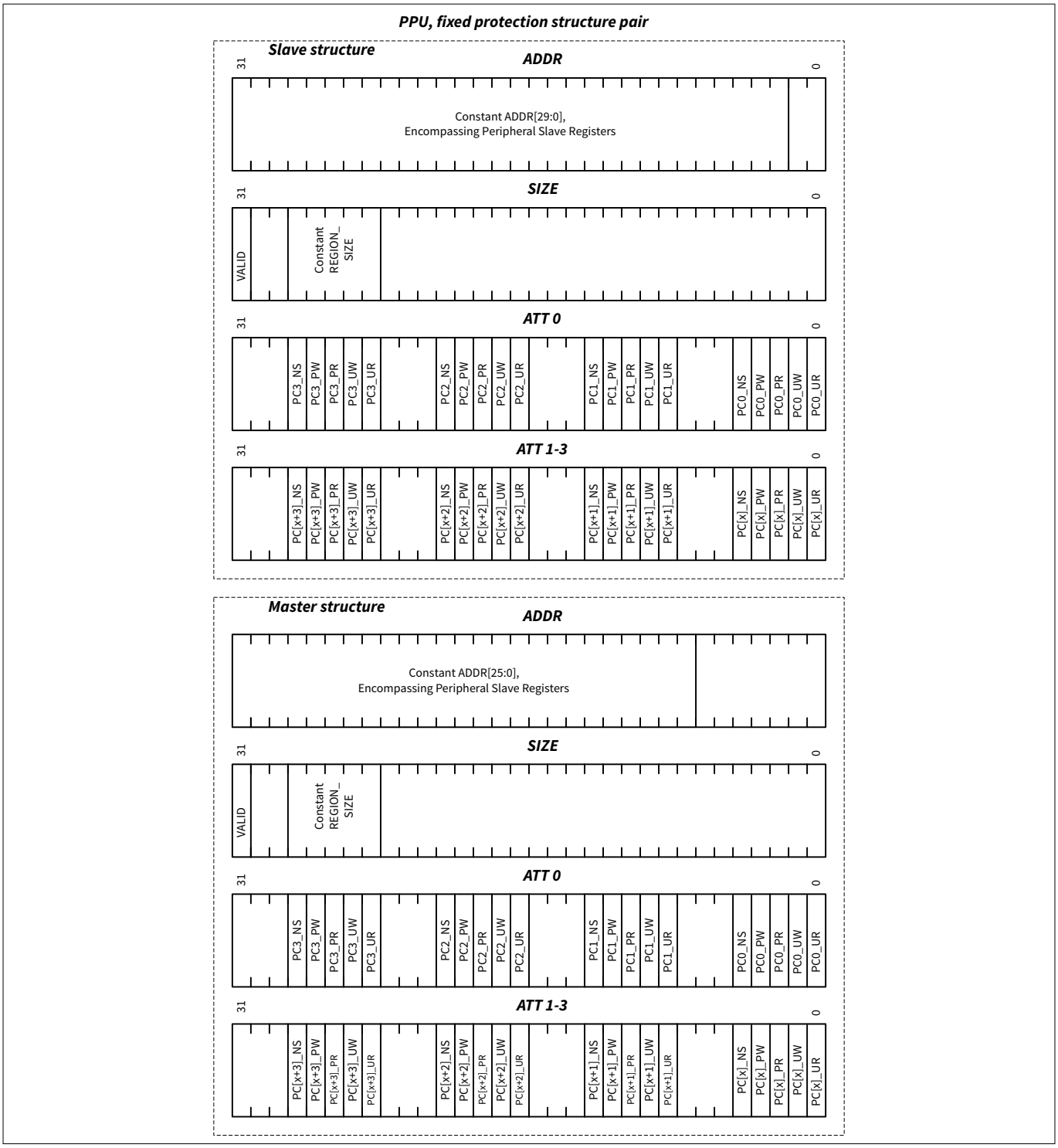


Figure 14 Fixed protection structure pair

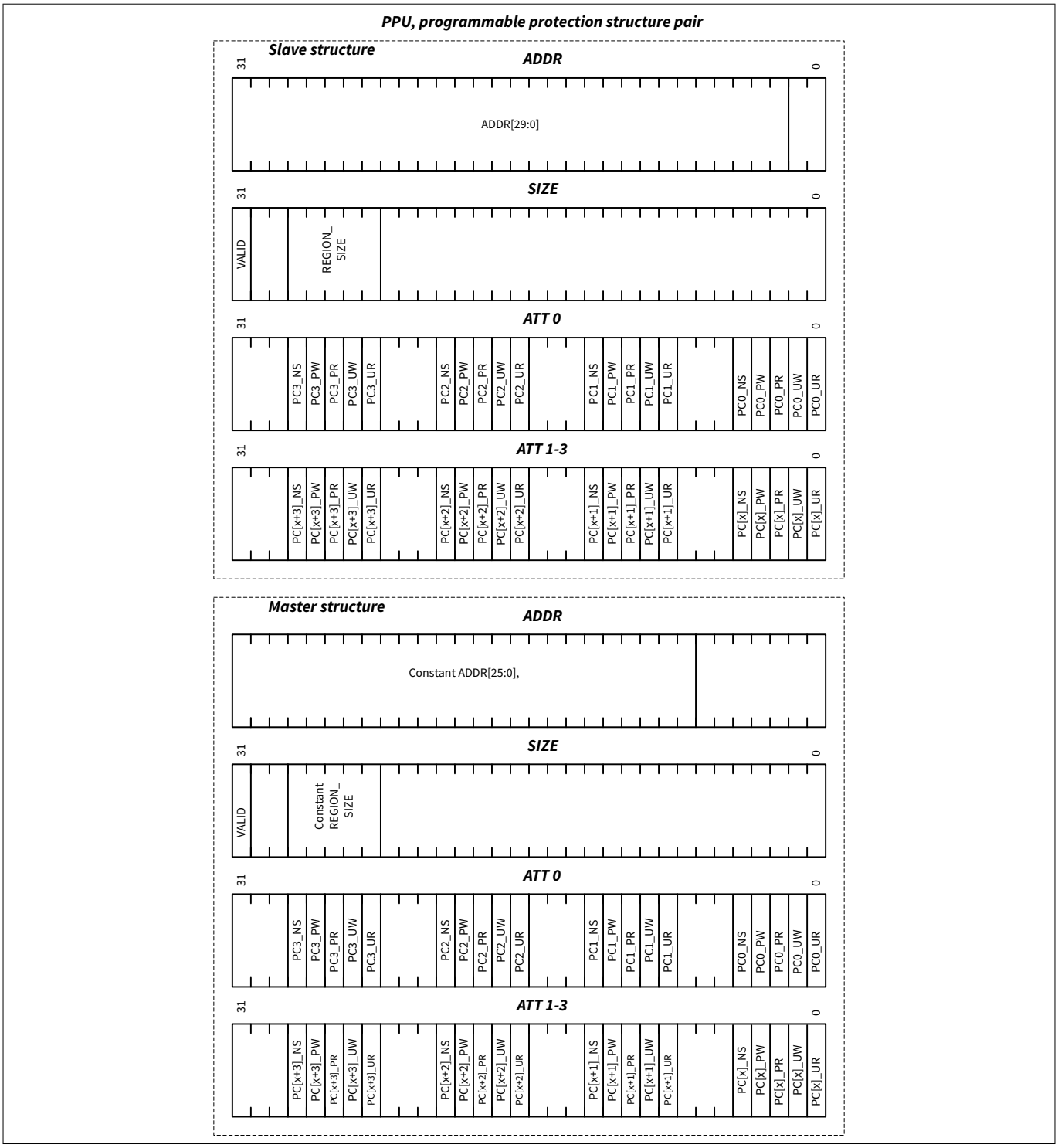


Figure 15 PPU programmable protection structure pair

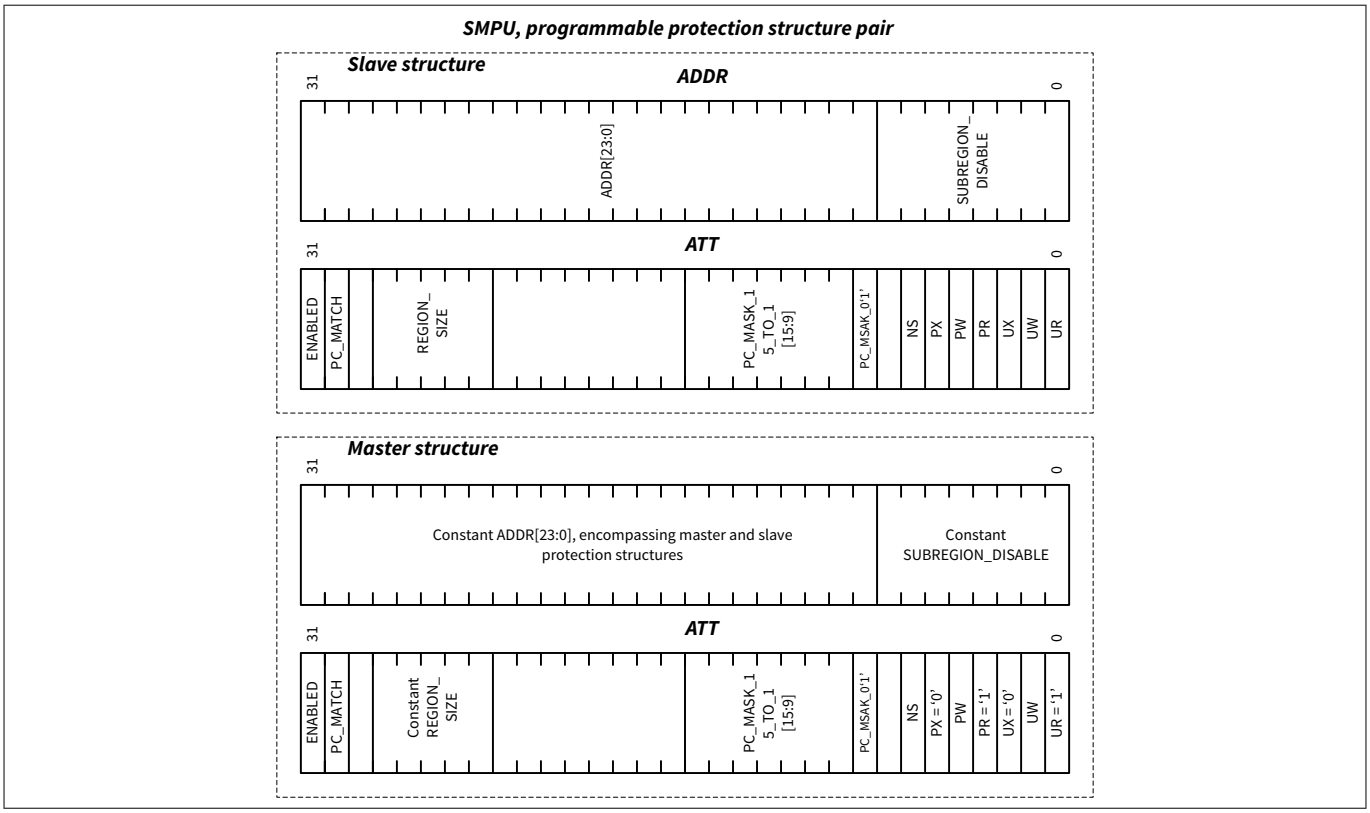


Figure 16 SMPU programmable protection structure pair

6.5 SWPU

SWPU is used to implement access restrictions to flash (program/erase) and eFuse (read/write) and stored in SFlash.

This feature prevents malicious or unintended modification of flash or eFuse, and reading of sensitive eFuse data. In addition, unauthorized changes to SWPU are detected by the secure boot operation.

SWPU has two parts – boot protection and application protection. Boot protection implements the access restrictions related to PC1 and PCx, and cannot be updated. Application protection is used by the application for additional access restrictions specific to the application. It is stored in SFlash during the NORMAL_PROVISIONED life-cycle stage and cannot be updated in SECURE. However, it can be updated more than once in the NORMAL_PROVISIONED stage by writing to the specific row in SFlash. The address ranges within each part are disjoint and in increasing order.

ROM/flash boot reads each protection of SWPU from SFlash and stores them in RAM. It also checks that the address regions are increasing and disjoint. As it reads, any overlapping entry is skipped during the merge. If there is an overlap between the boot protection entry and the application protection entry, then the application protection entry is skipped. If an SWPU protection entry is not in increasing order, then the entry is skipped.

SWPU is comprised of flash write protection unit (FWPU), eFuse read protection unit (ERPU) and eFuse write protection unit (EWPU). SWPU has slave and master protection structures as a protection pair, same as SMPU and PPU.

See Protection setting in BootROM for boot protection details.

6.6 SWPU layout

The SWPU is located at the address specified by TOC2_APP_PROTECTION_ADDR of TOC2 in SFlash. (The default address is 0x1700_7600.) FWPU has up to 16 regions, and ERPU and EWPU have up to four regions. Table 16 lists the SWPU layout.

Table 16 SWPU Layout in SFlash

SWPU	Name	Size	Description
–	PU_OBJECT_SIZE	4 bytes	The total byte number of configured elements.
FWPU	N_FWPU[3:0]	4 bytes	The number of FWPU objects. FWPU has up to 16 regions.
	FWPU0_SL_[3:0]	4 bytes	Configures the base address.
	FWPU0_SIZE_[3:0]	4 bytes	Configures the size of protection area from FWPU_SL.
	FWPU0_SL_ATT_[3:0]	4 bytes	Configures the slave attribute. This element sets the attribute for write access to Flash memory.
	FWPU0_MS_ATT_[3:0]	4 bytes	Configures the master attribute. This element sets the attribute to configure the FWPU0_SL_ATT.
	:	–	Up to 16 regions.
ERPU	N_ERPU[3:0]	4 bytes	The number of FWPU objects. ERPU has up to four regions.
	ERPU0_SL_OFFSET_[3:0]	4 bytes	Configures the offset from eFuse base address.
	ERPU0_FUSE_SIZE_[3:0]	4 bytes	Configures the size of protection area from ERPU0_SL_OFFSET.
	ERPU0_SL_ATT_[3:0]	4 bytes	Configures the slave attribute. This element sets the attribute for read access from eFuse.
	ERPU0_MS_ATT_[3:0]	4 bytes	Configures the master attribute. This element sets the attribute to configure the ERPU0_SL_ATT.
	:	–	Up to four regions.
EWPU	N_EWPU[3:0]	4 bytes	The number of FWPU objects. EWPU has up to four regions.
	EWPU0_SL_OFFSET_[3:0]	4 bytes	Configures the offset from eFuse base address.
	EWPU0_FUSE_SIZE_[3:0]	4 bytes	Configures the size of protection area from ERPU0_SL_OFFSET.
	EWPU0_SL_ATT_[3:0]	4 bytes	Configures the slave attribute. This element sets the attribute for write access to eFuse.
	EWPU0_MS_ATT_[3:0]	4 bytes	Configures the master attribute. This element sets the attribute to configure the EWPU0_SL_ATT.
	:	–	Up to four regions.

Each element is described here. The suffix ‘x’ indicates FWPU region number.

- PU_OBJECT_SIZE: This element defines the total byte number of configured elements, which includes 4 bytes of PU_OBJECT_SIZE. Note that SWPU consists of up to 512 bytes. Blanks cannot be inserted between elements of each protection unit.
- N_FWPU[3:0], N_ERPU[3:0], N_EWPU[3:0]: These elements define the number of each protection unit. There are no EWPU objects if set to ‘0’. Note that the maximum number of areas for each unit cannot be exceeded.
- FWPUx_SL_[3:0]: This element sets the base address of the Flash memory to be protected by FWPU. The absolute 32-bit address needs to be specified. Also, the last two bits should be ‘0’ for alignment purposes.

- ERPUx_SL_OFFSET_[3:0], EWPUx_SL_OFFSET_[3:0]: These elements set the offset from the eFuse base address to be protected by ERPU or EWPU.
- FWPUx_SIZE_[3:0], ERPUx_FUSE_SIZE_[3:0], EWPUx_FUSE_SIZE_[3:0]: These elements set the area size to be protected by each protection unit. The MSb indicates that the region is enabled when set to '1'. Figure 17 shows the composition of each element.

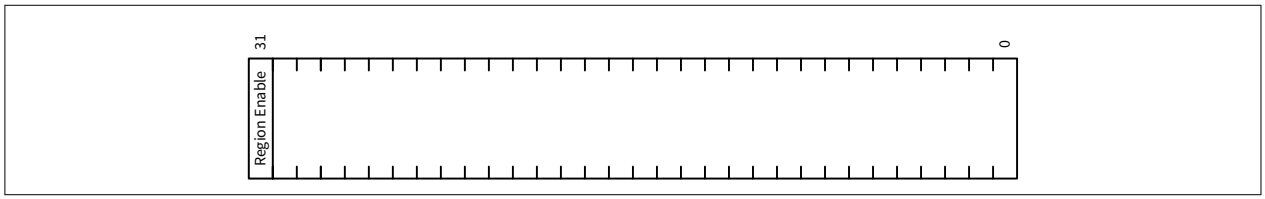


Figure 17 Composition of size elements

- FWPUx_SL_ATT_[3:0], ERPUx_SL_ATT_[3:0], EWPUx_SL_ATT_[3:0]: These elements set the attribute to access flash memory or eFuse. Figure 18 shows the composition of each attribute setting element.

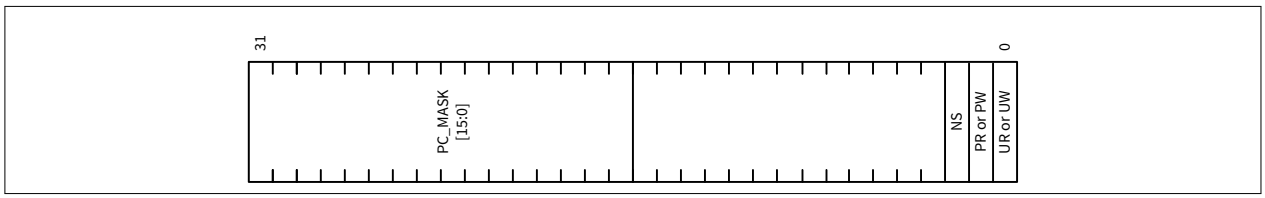


Figure 18 Composition of attribute elements

  - UR: Read accesses in user mode are allowed for ERPU, when this bit sets to '1'.
  - PR: Read accesses in privileged mode are allowed for ERPU, when this bit set to '1'.
  - UW: Write accesses in user mode are allowed for FWPU and EWPU, when this bit sets to '1'.
  - PW: Write accesses in privileged mode are allowed for FWPU and EWPU, when this bit set to '1'.
  - NS: Non-secure accesses are allowed, when this bit set to '1'.
  - PC_MASK: Accesses with protection context (PC) are allowed, when the corresponding bit is set to '1'. XMC7000 has eight PCs. Therefore, MC_MASK[15:8] is invalid.
- FWPUx_MS_ATT_[3:0], ERPUx_MS_ATT_[3:0], EWPUx_MS_ATT_[3:0]: These elements set the attribute to access slave elements. Composition of each attribute setting element is same as Figure 18.

6.6.1 SWPU configuration

The following example shows an SWPU configuration of two FWPUs, one ERPU, and one EWPU.

In the first FWPU (Region 0), base address is 0x10000000, size is 0x1000, all PCs allow full write access. In the second FWPU (Region 1), base address is 0x10008000, size is 0x8000, all PCs allow only write access with privileged. The master attribute for both FWPUs is that all PCs allow full access. The configuration of each FWPU element is as follows:

- N_FWPU[3:0]: When two FWPUs are added, N_FWPU0 should be 0x02. N_FWPU1, N_FWPU2, and N_FWPU3 should be 0x00.
- FWPU0_SL_[3:0]: The first FWPU base address is 0x10000000. Therefore, FWPU0_SL_0, FWPU0_SL_1, FWPU0_SL_2, and FWPU0_SL_3 correspond to 0x00 0x00 0x00 0x10.
- FWPU0_SIZE_[3:0]: The first FWPU size is 0x1000. Therefore, FWPU0_SIZE_0, FWPU0_SIZE_1, FWPU0_SIZE_2, and FWPU0_SIZE_3 correspond to 0x00 0x10 0x00 0x80. Note that the FWPU0_SIZE_3 is 0x80 because the MSb indicates that the region is enabled.

- FWPU0_SL_ATT_[3:0]: The first FWPU attribute is all PCs allow full access. Therefore, FWPU0_SL_ATT_0, FWPU0_SL_ATT_1, FWPU0_SL_ATT_2, and FWPU0_SL_ATT_3 correspond to 0x07 0x00 0xFF 0x00.
- FWPU0_MS_ATT_[3:0]: The first FWPU master attribute is all PCs allow full access. Therefore, FWPU0_MS_ATT_0, FWPU0_MS_ATT_1, FWPU0_MS_ATT_2, and FWPU0_MS_ATT_3 correspond to 0x07 0x00 0xFF 0x00.

The second FWPU is configured as follows:

- FWPU1_SL_[3:0]: The first FWPU base address is 0x10008000. Therefore, FWPU1_SL_0, FWPU1_SL_1, FWPU1_SL_2, and FWPU1_SL_3 correspond to 0x00 0x80 0x00 0x10.
- FWPU1_SIZE_[3:0]: The first FWPU size is 0x8000. Therefore, FWPU1_SIZE_0, FWPU1_SIZE_1, FWPU1_SIZE_2, and FWPU1_SIZE_3 correspond to 0x00 0x80 0x00 0x80. Note that the FWPU0_SIZE_3 is 0x80 because the MSb indicates the region is enabled.
- FWPU1_SL_ATT_[3:0]: The first FWPU attribute is all PCs allow only access with privileged. Therefore, FWPU1_SL_ATT_0, FWPU1_SL_ATT_1, FWPU1_SL_ATT_2, and FWPU1_SL_ATT_3 correspond to 0x06 0x00 0xFF 0x00.
- FWPU1_MS_ATT_[3:0]: The first FWPU master attribute is all PCs allow full access. Therefore, FWPU1_MS_ATT_0, FWPU1_MS_ATT_1, FWPU1_MS_ATT_2, and FWPU1_MS_ATT_3 correspond to 0x07 0x00 0xFF 0x00.

In this example, ERPU protects customer data in eFuse. The customer data is located at offset 0x68 from eFuse base address, and the size is 0x18, all PCs allow full read access. The master attribute for ERPU is all PCs allow full access. The configuration of ERPU element is as follows:

- N_ERPU[3:0]: When one ERPU is added, N_ERPU0 should be 0x01. N_ERPU1, N_ERPU2, and N_ERPU3 should be 0x00.
- ERPU0_SL_OFFSET_[3:0]: The offset is 0x68. Therefore, ERPU0_SL_OFFSET_0, ERPU0_SL_OFFSET_1, ERPU0_SL_OFFSET_2, and ERPU0_SL_OFFSET_3 correspond to 0x68 0x00 0x00 0x00.
- ERPU0_FUSE_SIZE_[3:0]: The size is 0x18. Therefore, ERPU0_SIZE_0, ERPU0_SIZE_1, ERPU0_SIZE_2, and ERPU0_SIZE_3 correspond to 0x18 0x00 0x00 0x80. Note that the ERPU0_SIZE_3 is 0x80 because the MSb indicates that the region is enabled.
- ERPU0_SL_ATT_[3:0]: The attribute is all PCs allow full access. Therefore, ERPU0_SL_ATT_0, ERPU0_SL_ATT_1, ERPU0_SL_ATT_2, and ERPU0_SL_ATT_3 correspond to 0x07 0x00 0xFF 0x00.
- ERPU0_MS_ATT_[3:0]: The master attribute is all PCs allow full access. Therefore, ERPU0_MS_ATT_0, ERPU0_MS_ATT_1, ERPU0_MS_ATT_2, and ERPU0_MS_ATT_3 correspond to 0x07 0x00 0xFF 0x00.

In the case of EWPU configuration, EWPU protects customer data in eFuse. Offset is 0x68, size is 0x18, and all PCs allow full read access. The master attribute for EWPU is all PCs allow full access. The configuration of the EWPU element is as follows:

- N_EWPU[3:0]: When one EWPU is added, N_EWPU0 should be 0x01. N_EWPU1, N_EWPU2, and N_EWPU3 should be 0x00.
- EWPU0_SL_OFFSET_ [3:0]: The offset is 0x68. Therefore, EWPU0_SL_OFFSET_0, EWPU0_SL_OFFSET_1, EWPU0_SL_OFFSET_2, and EWPU0_SL_OFFSET_3 correspond to 0x68 0x00 0x00 0x00.
- EWPU0_FUSE_SIZE_[3:0]: The size is 0x18. Therefore, EWPU0_SIZE_0, EWPU0_SIZE_1, EWPU0_SIZE_2, and EWPU0_SIZE_3 correspond to 0x18 0x00 0x00 0x80. Note that the EWPU0_SIZE_3 is 0x80 because the MSb indicates that the region is enabled.
- EWPU0_SL_ATT_[3:0]: The attribute is all PCs allow full access. Therefore, EWPU0_SL_ATT_0, EWPU0_SL_ATT_1, EWPU0_SL_ATT_2, and EWPU0_SL_ATT_3 correspond to 0x07 0x00 0xFF 0x00.
- EWPU0_MS_ATT_[3:0]: The master attribute is all PCs allow full access. Therefore, EWPU0_MS_ATT_0, EWPU0_MS_ATT_1, EWPU0_MS_ATT_2, and EWPU0_MS_ATT_3 correspond to 0x07 0x00 0xFF 0x00.

Table 17 lists the SWPU layout in SFlash in the above configuration.

Table 17 SWPU layout after configuration

Address	Element name	Setting value
0x17007600	PU_OBJECT_SIZE	0x50
0x17007601		0x00
0x17007602		0x00
0x17007603		0x00
0x17007604	N_FWPU0	0x02
0x17007605	N_FWPU1	0x00
0x17007606	N_FWPU2	0x00
0x17007607	N_FWPU3	0x00
0x17007608	FWPU0_SL_0	0x00
0x17007609	FWPU0_SL_1	0x00
0x1700760A	FWPU0_SL_2	0x00
0x1700760B	FWPU0_SL_3	0x10
0x1700760C	FWPU0_SIZE_0	0x00
0x1700760D	FWPU0_SIZE_1	0x10
0x1700760E	FWPU0_SIZE_2	0x00
0x1700760F	FWPU0_SIZE_3	0x80
0x17007610	FWPU0_SL_ATT_0	0x07
0x17007611	FWPU0_SL_ATT_1	0x00
0x17007612	FWPU0_SL_ATT_2	0xFF
0x17007613	FWPU0_SL_ATT_3	0x00
0x17007614	FWPU0_MS_ATT_0	0x07
0x17007615	FWPU0_MS_ATT_1	0x00
0x17007616	FWPU0_MS_ATT_2	0xFF
0x17007617	FWPU0_MS_ATT_3	0x00
0x17007618	FWPU1_SL_0	0x00
0x17007619	FWPU1_SL_1	0x80
0x1700761A	FWPU1_SL_2	0x00
0x1700761B	FWPU1_SL_3	0x10
0x1700761C	FWPU1_SIZE_0	0x00
0x1700761D	FWPU1_SIZE_1	0x80
0x1700761E	FWPU1_SIZE_2	0x00
0x1700761F	FWPU1_SIZE_3	0x80
0x17007620	FWPU1_SL_ATT_0	0x06

(table continues...)

Table 17 (continued) SWPU layout after configuration

Address	Element name	Setting value
0x17007621	FWPU1_SL_ATT_1	0x00
0x17007622	FWPU1_SL_ATT_2	0xFF
0x17007623	FWPU1_SL_ATT_3	0x00
0x17007624	FWPU1_MS_ATT_0	0x07
0x17007625	FWPU1_MS_ATT_1	0x00
0x17007626	FWPU1_MS_ATT_2	0xFF
0x17007627	FWPU1_MS_ATT_3	0x00
0x17007628	N_ERPU0	0x01
0x17007629	N_ERPU1	0x00
0x1700762A	N_ERPU2	0x00
0x1700762B	N_ERPU3	0x00
0x1700762C	ERPU0_SL_OFFSET_0	0x68
0x1700762D	ERPU0_SL_OFFSET_1	0x00
0x1700762E	ERPU0_SL_OFFSET_2	0x00
0x1700762F	ERPU0_SL_OFFSET_3	0x00
0x17007630	ERPU0_FUSE_SIZE_0	0x18
0x17007631	ERPU0_FUSE_SIZE_1	0x00
0x17007632	ERPU0_FUSE_SIZE_2	0x00
0x17007633	ERPU0_FUSE_SIZE_3	0x80
0x17007634	ERPU0_SL_ATT_0	0x07
0x17007635	ERPU0_SL_ATT_1	0x00
0x17007636	ERPU0_SL_ATT_2	0xFF
0x17007637	ERPU0_SL_ATT_3	0x00
0x17007638	ERPU0_MS_ATT_0	0x07
0x17007639	ERPU0_MS_ATT_1	0x00
0x1700763A	ERPU0_MS_ATT_2	0xFF
0x1700763B	ERPU0_MS_ATT_3	0x00
0x1700763C	N_EWPU0	0x01
0x1700763D	N_EWPU1	0x00
0x1700763E	N_EWPU2	0x00
0x1700763F	N_EWPU3	0x00
0x17007640	EWPU0_SL_OFFSET_0	0x68
0x17007641	EWPU0_SL_OFFSET_1	0x00

(table continues...)

Table 17 (continued) SWPU layout after configuration

Address	Element name	Setting value
0x17007642	EWPU0_SL_OFFSET_2	0x00
0x17007643	EWPU0_SL_OFFSET_3	0x00
0x17007644	EWPU0_FUSE_SIZE_0	0x18
0x17007645	EWPU0_FUSE_SIZE_1	0x00
0x17007646	EWPU0_FUSE_SIZE_2	0x00
0x17007647	EWPU0_FUSE_SIZE_3	0x80
0x17007648	EWPU0_SL_ATT_0	0x07
0x17007649	EWPU0_SL_ATT_1	0x00
0x1700764A	EWPU0_SL_ATT_2	0xFF
0x1700764B	EWPU0_SL_ATT_3	0x00
0x1700764C	EWPU0_MS_ATT_0	0x07
0x1700764D	EWPU0_MS_ATT_1	0x00
0x1700764E	EWPU0_MS_ATT_2	0xFF
0x1700764F	EWPU0_MS_ATT_3	0x00
:	–	Blank
0x170077FF	–	–

6.7 Registers

Table 18 List of MPU registers

Register	Name	Description
PROT_MPUx_MS_CTL	Master Control register	Specify the protection context of the bus transfer
PROT_MPUx_MPU_STRUCTy_ADDR	MPU Region Address register	Defines a MPU address region.
PROT_MPUx_MPU_STRUCTy_ATT	MPU Region Attributes register	Defines a MPU access control register.

***Note**: The 'x' in the registers name denotes the master number and the 'y' denotes the region number.*

Table 19 List of SMPU Registers

Register	Name	Description
PROT_SMPU_MSx_CTL	Protection Context Control register	Specify the protection context of the bus transfer.
PROT_SMPU_SMPU_STRUCTy_ADDR0	SMPU Region Address 0 (Slave Structure) register	Defines a SMPU address region (slave structure).
PROT_SMPU_SMPU_STRUCTy_ATT0	SMPU Region Attributes 0 (Slave Structure) register	Defines SMPU access control (slave structure).

(table continues...)

Table 19 (continued) List of SMPU Registers

Register	Name	Description
PROT_SMPU_SMPU_STRUCTy_ADDR1	SMPU Region Address 1 (Master structure) register	Defines a SMPU address region (master structure).
PROT_SMPU_SMPU_STRUCTy_ATT1	SMPU Region Attributes 1 (Master structure) register	Defines SMPU access control (master structure).

***Note**: The 'x' in the registers name denotes the master number and the 'y' denotes the region number.*

Table 20 List of PPU registers

Register	Name	Description
PERI_MS_PPU_PRx_SL_ADDR	Programmable PPU Slave Region, Base Address register	Specifies the base address of the slave region.
PERI_MS_PPU_PRx_SL_SIZE	Programmable PPU Slave Region, Size register	Specifies the size of the slave region and sets region enable. Typically, it is programmed by the boot process with protection context.
PERI_MS_PPU_PRx_SL_ATT0,1,2,3	Programmable PPU Slave Attributes 0, 1, 2, 3 register	Defines access control (slave structure).
PERI_MS_PPU_PRx_MS_ADDR	Programmable PPU Master Region, Base Address register	Specifies the base address of the master region. This register is fixed (non-programmable).
PERI_MS_PPU_PRx_MS_SIZE	Programmable PPU Master Region, Size register	Specifies the size of the master region. This register is fixed (non-programmable).
PERI_MS_PPU_PRx_MS_ATT0,1,2,3	Programmable PPU Master Attributes 0, 1, 2, 3 register	Defines access control (master structure).
PERI_MS_PPU_FXx_SL_ADDR	Fixed PPU Slave Region, Base Address register	Specifies the base address of the slave region.
PERI_MS_PPU_FXx_SL_SIZE	Fixed PPU Slave Region, Size register	Specifies the size of the slave region and sets region enable. Typically, it is programmed by the boot process with protection context.
PERI_MS_PPU_FXx_SL_ATT0,1,2,3	Fixed PPU Slave Attributes 0, 1, 2, 3 register	Defines access control (slave structure).
PERI_MS_PPU_FXx_MS_ADDR	Fixed PPU Master Region, Base Address register	Specifies the base address of the master region. This register is fixed (non-programmable).

(table continues...)

Table 20 (continued) List of PPU registers

Register	Name	Description
PERI_MS_PPU_FXx_MS_SIZE	Fixed PPU Master Region, Size register	Specifies the size of the master region. This register is fixed (non-programmable).
PERI_MS_PPU_FXx_MS_ATT0,1,2,3	Fixed PPU Master Attributes 0,1,2,3 register	Defines access control (master structure).
PERI_ECC_CTL	ECC Control register	Provides ECC support for the SRAM protection structures in the master interface peripherals (peripheral group 0, peripheral 1).

***Note**: The 'x' in the register name denotes the master number.*

Table 21 List of buffer control registers

Register	Name	Description
CPUSS_BUFF_CTL	Buffer Control register	Specifies if write transfer can be buffered in the bus infrastructure bridges.

7 Direct memory access

The XMC7000 device supports two kinds of DMA controllers: peripheral DMA (P-DMA) and memory DMA (M-DMA). P-DMA is used for peripheral-to-memory and memory-to-peripheral data transfers and provides low latency for a large number of channels. P-DMA controller uses a single data transfer engine that is shared by the associated channels. It supports independent accesses to peripherals using the AHB multi-layer bus. M-DMA is used for memory-to-memory data transfers and provides high memory bandwidth for a small number of channels. M-DMA uses a dedicated data transfer engine for each channel.

P-DMA and M-DMA have a similar register interface and are compared as follows:

- P-DMA focuses on peripheral-to-memory and memory-to-peripheral data transfers (but it can also perform memory-to-memory data transfers). M-DMA focuses on memory-to-memory data transfers (but it can also perform peripheral-to-memory and memory-to-peripheral data transfers).
- P-DMA focuses on achieving low latency for a large number of channels. M-DMA focuses on achieving high memory bandwidth for a small number of channels.
- P-DMA uses a single data transfer engine that is shared by all channels. M-DMA uses a dedicated data transfer engine for each channel.

***Note**: DW and P-DMA have the same meaning in this DMA chapter. Also, DMAC and M-DMA are the same. register names are labeled DW and DMAC.*

7.1 Peripheral DMA (P-DMA(DW))

P-DMA is used to transfer data between memory and peripherals without CPU involvement: the CPU configures/programs the P-DMA but the actual transfer is done by the P-DMA controller. The primary design target is P-DMA functionality at limited area overhead to the platform. Functionally, the P-DMA controller is similar to a general-purpose DMA controller.

7.1.1 Overview

The P-DMA controller is part of the CPUSS and controls data transfer between peripherals and memory. This controller can be configured/programmed to perform multiple independent data transfers. Each data transfer is managed by a channel. The number of channels varies for different part numbers; more details are available in the device datasheet.

A channel has an associated priority and is scheduled according to its priority.

A data transfer is initiated by an input trigger. This trigger may originate from the source of the transfer, destination of the transfer, CPU software, or from another SoC component. Triggers provide active/sleep functionality and are not available in deep sleep and hibernate power modes.

The data transfer specifics are specified by a descriptor. This descriptor specifies (among other things):

- The source and destination address locations and the size of the transfer.
- The actions of a channel; for example, generation of output triggers and interrupts.
- Data transfer types can be single, 1D, 2D, or CRC as defined in the descriptor structure. These types essentially define the address sequences generated for source and destination. 1D and 2D transfers are used for “scatter gather” and other useful transfer operations.

A channel's descriptor state is encoded as part of the channel's register state (and not as part of the descriptor). The following registers provide a channel's descriptor state:

- DWx_CH_STRUCTy_CH_CTL – This register provides generic channel control information.
- DWx_CH_STRUCTy_CH_CURR_PTR – This register provides the address of the memory location where the current descriptor is located. Software needs to initialize this register. Hardware sets this register to

the current descriptor's next descriptor pointer, when advancing from the current descriptor to the next descriptor in a descriptor list.

- DWx_CH_STRUCTy_CH_IDX – This register provides the current X and Y indices of the channel into the current descriptor. Software needs to initialize this register. Hardware sets the X and Y indices to '0', when advancing from the current descriptor to the next descriptor in a descriptor list.

Note that channel state is retained in deep sleep power mode.

The P-DMA controller is an active/sleep power mode functionality. Software should not initiate deep sleep system power mode entry if there are any active P-DMA controller channels transferring data. Note that there is no way of capturing the active channel data while transitioning to deep sleep system power mode.

7.1.2 Channels

P-DMA controller supports multiple independent data transfers that are managed by a channel. Each channel connects to a specific system trigger through a trigger multiplexer that is outside the P-DMA controller. See the Trigger multiplexer for more details.

- **Channel priority:** A channel is assigned a priority (DWx_CH_STRUCTy_CH_CTL.PRIO) between 0 and 3, with 0 being the highest priority and 3 being the lowest priority. Channels with the same priority constitute a priority group. Priority decoding determines the highest priority pending channel. This channel is determined as follows.
  - The highest priority group with pending channels is identified first.
  - Within this priority group, the following "round-robin" arbitration is applied. A "round" consists of a contiguous sequence of channel activations, within this priority group, without any repetition. Within a round, higher priority is given to the lower channel indices. The notion of a round guarantees that within a group, higher channel indices do not yield to lower indices indefinitely.
- **Channel state:** At any given time, there is at most one channel that is actively performing a data transfer. This channel is called the active channel. A channel can be in one of four channel states.

Table 22 P-DMA channel states

Channel state	Description
Disabled	The channel is disabled by setting DWx_CH_STRUCTy_CH_CTL.ENABLED to '0'. The channel trigger is ignored in this state. ***Note**: If an active channel is disabled by software, there should be no assumptions made about the state of the channel (current position of the transfer as reflected by the registers or descriptors). A software channel re-enable should prepare the new descriptors and reconfigure the channel.*
Blocked	The channel is enabled and is waiting for a trigger to initiate a data transfer.
Pending	The channel is enabled and has received an active trigger. In this state, the channel is ready to initiate a data transfer but waiting for it to be scheduled.
Active	The channel is enabled, has received an active trigger, and has been scheduled. It is actively performing data transfers. If there are multiple channels pending, the highest priority pending channel is scheduled.

The data transfer associated with a trigger is made up of one or more "atomic transfers" or "single transfers". For example a 1D transfer consists of X_COUNT+1 single transfers.

A channel may be marked preemptable (DWx_CH_STRUCTy_CH_CTL.PREEMPTABLE). If preemptable, and there is a higher priority pending channel, then that channel can preempt the current channel between single transfers.

A channel has two access control attributes that are SMPU and PPU for access control:

- Privileged Mode (DWx_CH_STRUCTy_CH_CTL.P) attribute can be set to privileged or user.
- Non-secure (DWx_CH_STRUCTy_CH_CTL.NS) attribute can be set to secure or non-secure.

A descriptor associated with each channel describes the data transfer. The descriptor is stored in memory and DWx_CH_STRUCTy_CH_CURR_PTR contains the descriptor address associated with channel “y”.

7.1.3 Descriptors

A descriptor is stored in memory and describes a data transfer. The descriptor is read-only for the P-DMA controller.

Descriptor type (DESCR_TYPE) – There are four types of descriptors.

Table 23 P-DMA descriptor types

Descriptor type	Description
Single transfer	This transfers a single data element (8-bit, 16-bit, or 32-bit) as shown in Figure 19. The descriptor size is four 32-bit words: DESCR_CTL, DESCR_SRC, DESCR_DST, and DESCR_NEXT_PTR.
1D transfer	This performs a one-dimensional “for loop” (described in C) as shown Figure 20. A 1D transfer is made up of X_COUNT+1 singe transfers. The descriptor size is five 32-bit words: DESCR_CTL, DESCR_SRC, DESCR_DST, DESCR_X_CTL, and DESCR_NEXT_PTR.
2D transfer	This performs a two-dimensional “for loop” (described in C) as shown in Figure 21. A 2D transfer is made up of (Y_COUNT+1) 1D transfers. The descriptor size is six 32-bit words: DESCR_CTL, DESCR_SRC, DESCR_DST, DESCR_X_CTL, DESCR_Y_CTL, and DESCR_NEXT_PTR.
CRC transfer	This performs a one-dimensional “for loop” similar to the 1D transfer. However, the source data is not transferred to a destination. Instead, a CRC is calculated over the source data as shown in Figure 22. The CRC configuration is provided through a set of registers that is shared by all P-DMA channels and the assumption is that the P-DMA channels use the CRC functionality mutually exclusive in time. These registers are: DWx_CRC_CTL0, DWx_CRC_DATA_CTL0, DWx_CRC_POL_CTL0, DWx_CRC_LFSR_CTL0, DWx_CRC_REM_CTL0, and DWx_CRC_REM_RESULT0. Note that the CRC configuration is the same as the Crypto CRC configuration.

```
// DST_ADDR is a pointer to an object of type defined by DST_TRANSFER_SIZE
// SRC_ADDR is a pointer to an object of type defined by SRC_TRANSFER_SIZE
// t_DATA_SIZE is the type associated with the DATA_SIZE
DST_ADDR[0] = (t_DATA_SIZE) SRC_ADDR[0];
```

Figure 19 Single transfer

```
// DST_ADDR is a pointer to an object of type defined by DST_TRANSFER_SIZE
// SRC_ADDR is a pointer to an object of type defined by SRC_TRANSFER_SIZE
// t_DATA_SIZE is the type associated with the DATA_SIZE
for (X_IDX = 0; X_IDX ≤ X_COUNT; X_IDX++){
   DST_ADDR[X_IDX * DST_X_INCR] =
      (t_DATA_SIZE) SRC_ADDR[X_IDX * SRC_X_INCR];
}
```

Figure 20 1D transfer

```
// DST_ADDR is a pointer to an object of type defined by DST_TRANSFER_SIZE
// SRC_ADDR is a pointer to an object of type defined by SRC_TRANSFER_SIZE
// t_DATA_SIZE is the type associated with the DATA_SIZE
for (Y_IDX = 0; Y_IDX ≤ Y_COUNT; Y_IDX++){
   for (X_IDX = 0; X_IDX ≤ X_COUNT; X_IDX++) {
      DST_ADDR[X_IDX * DST_X_INCR + Y_IDX * DST_Y_INCR ] =
         (t_DATA_SIZE) SRC_ADDR[X_IDX * SRC_X_INCR + Y_IDX * SRC_Y_INCR];
   }
}
```

Figure 21 2D transfer

```
// DST_ADDR is a pointer to an address location where the calculated CRC is stored.
// SRC_ADDR is a pointer to an object of type defined by SRC_TRANSFER_SIZE
// t_DATA_SIZE is the type associated with the DATA_SIZE CRC_STATE = CRC_LFSR_CTL;
for (X_IDX = 0; X_IDX ≤ X_COUNT; X_IDX++) {
   Update_CRC (CRC_STATE, (t_DATA_SIZE) SRC_ADDR[X_IDX * SRC_X_INCR];
}
   DST_ADDR = CRC_STATE;
```

Figure 22 CRC transfer

The variables X_IDX and Y_IDX are stored in the channel register state (DWx_CH_STRUCTy_CH_IDX register).

The parameters X_COUNT, Y_COUNT, SRC_X_INCR, SRC_Y_INCR, DST_X_INCR, DST_Y_INCR, SRC_ADDR, DST_ADDR, SRC_TRANSFER_SIZE, DST_TRANSFER_SIZE, and DATA_SIZE are stored in the descriptor.

Descriptor size – The size of a descriptor depends on its descriptor type. Only relevant parameters are stored.

For example, a 1D descriptor does not contain the Y_COUNT, SRC_Y_INCR, and DST_Y_INCR parameters.

Transfer size (SRC_TRANSFER_SIZE and DST_TRANSFER_SIZE) – In a data transfer, the source data is cast into the type specified by DATA_SIZE and assigned to the destination. The source type is determined by SRC_TRANSFER_SIZE and the destination type is determined by DST_TRANSFER_SIZE. All types are unsigned. All address computations use C semantics based on the transfer size.

Descriptor chaining – Descriptors chained together. DESCR_NEXT_PTR field contains a pointer to the next descriptor in the chain. A channel executes the next descriptor in the chain when it completes executing the current descriptor. The last descriptor in the chain has DESCR_NEXT_PTR set to '0' (null pointer). A descriptor chain is also referred to as a descriptor list. It is possible to have a circular list in which case the execution continues indefinitely until there is an error or the channel or the controller is disabled by software.

Trigger-in type (TR_IN_TYPE) – An input trigger initiates a data transfer and the TR_IN_TYPE defines the action on a trigger.

Table 24 P-DMA trigger-in types

Trigger type	Description
Type 0	Trigger results in the execution of a single transfer. In a 1D or 2D transfer, this will execute a single transfer in the loop.
Type 1	Trigger results in the execution of a single 1D transfer. If the descriptor type is "single transfer" this behaves similar to type 0. If the descriptor type is 2D, it results in executing the inner loop once.
Type 2	Trigger results in the execution of the current descriptor.
Type 3	Trigger results in the execution of a descriptor list.

Trigger-out type (TR_OUT_TYPE) – This defines when an output trigger is generated.

Table 25 P-DMA trigger-out types

Trigger type	Description
Type 0	Output trigger is generated after a single transfer. In a 1D or 2D transfer, an output trigger is generated after each transfer in the loop.
Type 1	Output trigger is generated after a single 1D transfer. If the descriptor type is “single transfer”, this behaves similar to type 0. If the descriptor type is 2D, an output trigger is generated after each execution of the inner loop.
Type 2	Output trigger is generated after the execution of the current descriptor.
Type 3	Output trigger is generated after the execution of a descriptor list.

Interrupt type (INTR_TYPE) – This defines when a completion interrupt is generated.

Table 26 P-DMA interrupt types

Trigger type	Description
Type 0	Interrupt is generated after a single transfer. In a 1D or 2D transfer, an interrupt is generated after each transfer in the loop
Type 1	Interrupt is generated after a single 1D transfer. If the descriptor type is single transfer, this behaves similar to type 0. If the descriptor type is 2D, an interrupt is generated after each execution of the inner loop.
Type 2	Interrupt is generated after the execution of the current descriptor
Type 3	Interrupt is generated after the execution of a descriptor list.

Wait for deactivation (WAIT_FOR_DEACT) – Specifies whether the P-DMA controller should wait for the input trigger to be deactivated after it has completed the data transfer corresponding to the current trigger. This field is used for level-sensitive triggers to give sufficient time for the triggering agent to deactivate the trigger. The wait specified can be ‘0’, up to four cycles, up to 16 cycles, or indefinite. Pulse-sensitive triggers should have this field set to ‘0’.

7.1.4 Interrupts

P-DMA can generate interrupts on completion and on various error conditions.

- The INTR_TYPE descriptor control defines when a completion condition (COMPLETION) is activated.
- The error conditions include SRC_BUS_ERROR, DST_BUS_ERROR, SRC_MISAL, DST_MISAL, CURR_PTR_NULL, ACTIVE_CH_DISABLED, and DESCR_BUS_ERROR.

The source of the interrupt is stored in DWx_CH_STRUCTy_CH_STATUS.INTR_CAUSE. INTR_TYPE defined in the descriptor controls when a completion interrupt is generated. Each channel has four interrupt related registers.

DWx_CH_STRUCTy_INTR – Each channel has an interrupt request register. Bit 0 is set ‘1’ when interrupt event (completion or error) is detected. Software can clear this by writing to this bit.

DWx_CH_STRUCTy_INTR_SET – Each channel has an Interrupt Set register. Software can write ‘1’ to this register to set the corresponding DWx_CH_STRUCTy_INTR register.

DWx_CH_STRUCTy_INTR_MASK – Each channel has an Interrupt Mask register. The corresponding interrupt is enabled by writing ‘1’ to this register.

DWx_CH_STRUCTy_INTR_MASKED – Each channel has an Interrupt Masked register. When read, this register reflects a bitwise AND between the interrupt request and mask registers.

The P-DMA is an active power mode peripheral; this means, it uses active functionality interrupts. Therefore, DWx_CH_STRUCTy_INTR and DWx_CH_STRUCTy_INTR_SET are not retained in deep sleep power mode (DWx_CH_STRUCTy_INTR_MASK is retained).

7.1.5 P-DMA Controller Status registers

The controller DWx_STATUS0 register contains the following information.

- DWx_STATUS0.ACTIVE - Active channel present, no/yes
- DWx_STATUS0.P - Active channel access control user/privileged
- DWx_STATUS0.NS - Active channel access control secure/non-secure
- DWx_STATUS0.B - Active channel access control non-bufferable/bufferable
- DWx_STATUS0.PC - Active channel protection context
- DWx_STATUS0.CH_IDX - Active channel index if there is an active channel
- DWx_STATUS0.PRIO - Active channel priority
- DWx_STATUS0.PREEMPTABLE - Active channel preemptable
- DWx_STATUS0.STATE - One of inactive, loading descriptor, loading data element, storing data element, or waiting for trigger deactivation

The DWx_CH_STRUCTy_CH_STATUS.PENDING register bit specifies whether the channel is currently pending or not.

7.1.6 P-DMA controller design

The following figure gives an overview of the P-DMA controller design.

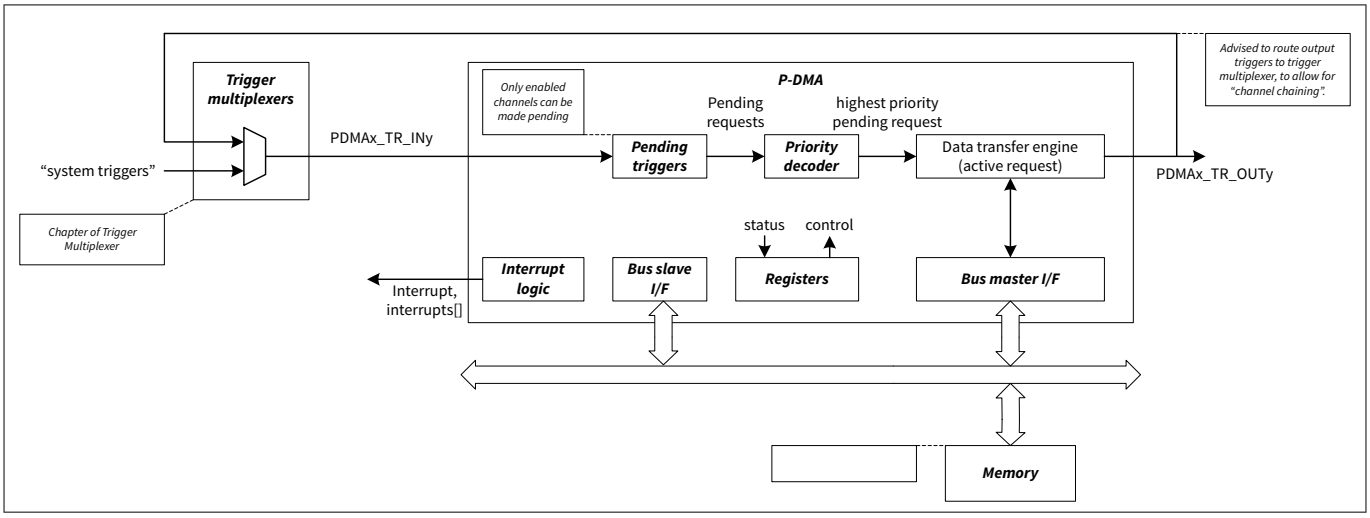


Figure 23 P-DMA controller design

In this figure, the P-DMA controller output triggers are feedback as input triggers to the component. This feedback is accomplished outside of the component.

The following design components are distinguished:

- **Trigger selection** component is outside the P-DMA controller and connects each channel to one specific system trigger. This multiplexer layer allows a controller with a limited number of channels to support a larger number of system triggers. This is an important function as the controller's area scales with the number of channels (and to a lesser degree with the number of system triggers). This is because each channel requires a channel structure. Furthermore, although the number of system triggers is large, typical use cases only use a limited subset of system triggers and as a result only a limited number of channels is required. A logical 1 on a selected trigger line indicates an activated trigger and results in a channel data transfer.

  Note that a P-DMA channel can be triggered from software directly through channel DWx_CH_STRUCTy_TR_CMD register. This is in addition to the software trigger control available in the trigger multiplexer.
- **Pending triggers** keeps track of activated triggers by locally storing them in pending bits. This is essential because multiple channel trigger may be activated simultaneously, whereas only one channel can be served by the data transfer engine at a time. This component enables the use of both level sensitive and pulse sensitive triggers.
  - Level-sensitive triggers are associated to a certain state, such as a FIFO being full. These triggers remain active as long as the state is maintained. For these triggers, keeping track of pending triggers in the P-DMA controller is not absolutely required, as the triggers are maintained outside of the controller.
  - Pulse-sensitive triggers are associated to a certain event, such as an ADC sample becoming available. For these triggers, it is essential to keep track of them in the P-DMA controller as the trigger pulse may disappear before it is served by the data transfer engine.
- **Priority decoder** determines the highest priority channel with an active trigger. Within a priority group, triggers are decoded on a round-robin basis.
- **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. It is also responsible for reading the channel descriptor from memory.

- **Master I/F** is an AHB bus master, which allows the controller to initiate AHB data transfers to the source and destination locations as well as to read the descriptor from memory.
- **Registers** description is found in the memory map. Each channel has a DWx_CH_STRUCTy_CH_CURR_PTR that points to a descriptor structure in memory that specifies the data transfer.
- **Slave I/F** is an AHB bus slave, which allows the main CPU to access controller Control/Status registers.
- **Interrupt logic** includes interrupt status for each of the channels.

A note on output triggers: Each channel has an output trigger, tr_out. The trigger is generated as defined by TR_OUT_TYPE in the descriptor. At the system level, these output triggers can be connected to the trigger multiplexer component. This connection allows a P-DMA controller output trigger to be connected to a P-DMA controller input trigger. In other words, the completion of a specific transfer of one channel can activate another channel.

As described, each design component performs a specific function, which is best illustrated by a specific example. The following figure shows the same controller design, with a trigger/data/interrupt flow superimposed on it.

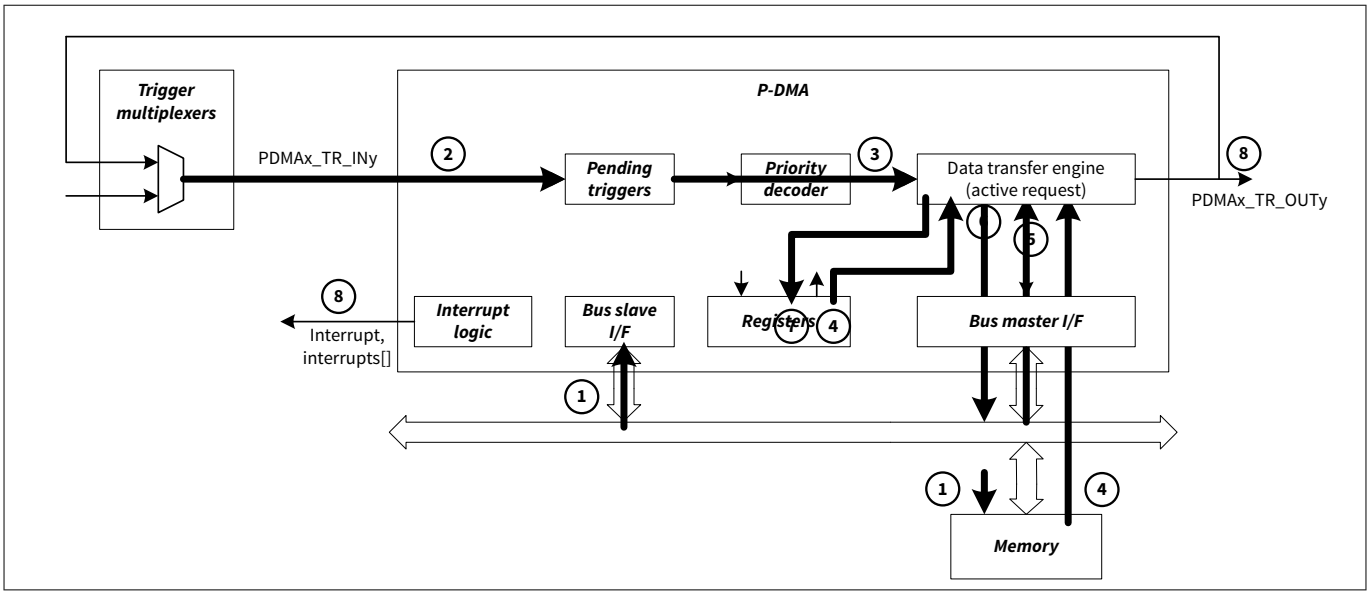


Figure 24 P-DMA controller flow

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

1. The main CPU programs the descriptor chain (in memory) and associates it with a specific channel. Further it programs the channel registers to set the desired attributes for the channel. It also programs the 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 pending channel.
4. The data transfer engine accepts the activated channel, and uses the channel identifier to load the channel's descriptor structure from memory using the master I/F. 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. The amount of data transferred depends on the TR_IN_TYPE.
7. The data transfer engine updates the active descriptor registers and the channel structure registers to reflect the data transfers. There is no update to the descriptors in memory.
8. Output trigger generation and interrupt generation are determined by TR_OUT_TYPE and INTR_TYPE respectively. P-DMA can generate an error interrupt if it encounters an error. A channel gets disabled on an error.

A note on throughput: The P-DMA controller data transfer steps can be classified as either: initialization, concurrent, or sequential step:

- 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 P-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 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. If no wait states are incurred when accessing the target memories/peripherals, the sequential steps take 12 cycles (excluding trigger synchronization and activation covered in steps 2 and 3). In other words, the P-DMA controller can sustain 100 MHz/12 cycles = 8.33 M data transfers per second.

7.1.6.1 P-DMA channel configuration SRAMs

The P-DMA controller uses SRAM memory to store some fields of the channel configuration. The following fields of the channel configuration are part of the SRAM memory.

- DWx_CH_STRUCTy_CH_CTL.P,
- DWx_CH_STRUCTy_CH_CTL.NS,
- DWx_CH_STRUCTy_CH_CTL.B,
- DWx_CH_STRUCTy_CH_CTL.PC,
- DWx_CH_STRUCTy_CH_CTL.PREEMPTABLE
- DWx_CH_STRUCTy_CH_IDX.X_IDX,
- DWx_CH_STRUCTy_CH_IDX.Y_IDX
- DWx_CH_STRUCTy_CH_CURR_PTR.ADDR

7.1.6.2 ECC for P-DMA channel configuration SRAMs

The P-DMA SRAM memory uses 7-bit SECDED parity for each 32 bits of data. Address coverage is not included. ECC functionality can be enabled or disabled through the DWx_CTL0.ECC_EN register field.

Both the correctable and non-correctable ECC errors are reported to the central fault structure.

Note that in order to avoid unwanted ECC faults, the user should initialize the P-DMA channel configuration SRAM before accessing the P-DMA channel configuration registers. This must be done by writing an all ‘0’ value to the DWx_CH_STRUCTy_SRAM_DATA0 and DWx_CH_STRUCTy_SRAM_DATA1 registers.

ECC error injection

The P-DMA SRAM ECC supports error injection through the following registers:

- DWx_CTL0.ECC_INJ_EN
- DWx_ECC_CTL0
- DWx_CH_STRUCTy_SRAM_DATA0
- DWx_CH_STRUCTy_SRAM_DATA1

DWx_CH_STRUCTy_SRAM_DATA0 and DWx_CH_STRUCTy_SRAM_DATA1 are provided for ECC fault injection functionality. These registers should not be used to control regular functionality (except that they can be used for initialization of P-DMA SRAMs).

For ECC fault injection, update a complete 32-bit SRAM data word with a user-provided ECC parity (specified by DWx_ECC_CTL0.PARITY) at a specific SRAM location (specified by DWx_ECC_CTL0.WORD_ADDR).

ECC parity generation by software

To inject the ECC error for fault generation, ECC parity must be generated by software.

Follow this procedure to generate a 7-bit ECC parity for P-DMA SRAM. Parity generation calculates a 7-bit parity[6:0] over a 32-bit data word W[31:0]. First, a 64-bit ECC code word CW_SW[63:0] is created:

```
CW_SW[63:0] = 64{1'b0};
CW_SW[31:0] = W[31:0];
CW_SW[41:32] = Addr; channel index *2 + 0 (for SRAM_DATA0) or 1 (for SRAM_DATA1;
```

Then, the 7-bit parity is calculated as the reduction XOR of the 64-bit code word CW_SW [63:0] ANDed with the following parity bit specific constants:

```
ECC_P0_SW = 64b00000011_01111111_00110110_11011011_00100010_01010100_00101010_10101011;
ECC_P1_SW = 64b00000101_10111101_11101011_01011010_01000100_10011001_01001101_00110101;
ECC_P2_SW = 64b00001001_11011101_11011100_11101110_00001000_11100010_01110001_11000110;
ECC_P3_SW = 64b00010001_11101110_10111011_10101001_10001111_00000011_10000001_11111000;
ECC_P4_SW = 64b00100001_11110110_11010111_01110101_11110000_00000011_11111110_00000000;
ECC_P5_SW = 64b01000001_11111011_01101101_10110100_11111111_11111100_00000000_00000000;
ECC_P6_SW = 64b10000001_00000011_11111111_11111000_00010001_00101100_10010110_01011111;
```

The parity bits are calculated as follows:

```
parity[0] = ^ (CW_SW[63:0] & ECC_P0_SW)
parity[1] = ^ (CW_SW[63:0] & ECC_P1_SW)
…
parity[6] = ^ (CW_SW[63:0] & ECC_P6_SW)
```

7.1.7 Functionality

This section illustrates the descriptor features and P-DMA functionality through two examples.

7.1.7.1 Example 1 - Single transfer

This example illustrates how a trigger initiates a transfer of a 16-bit sample from an ADC source to a memory destination. The ADC source has a bus interface that only supports 32-bit transfers. The memory has a bus interface that supports 8-bit, 16-bit, and 32-bit transfers. The transferred sample should be written to 16 memory bits.

The ADC sample location is at address 0x4000:0000. The memory location is at address 0x0000:0000.

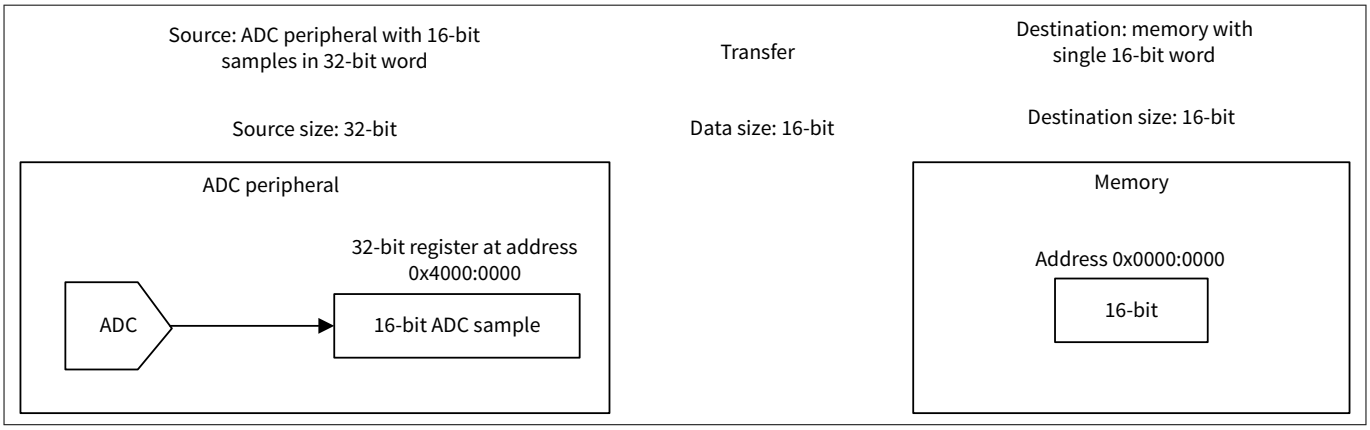


Figure 25 Single transfer

Setup:

Let us assume that P-DMA0/DW0 channel 3 is used for this data transfer an0d the trigger from ADC is connected to this channel.

Initialize the Channel registers.

1. DW0_CH_STRUCT3_CH_IDX.X_IDX = 0 and DW0_CH_STRUCT3_CH_IDX.Y_IDX = 0.
2. DW0_CH_STRUCT3_CH_CURR_PTR = address of the descriptor in memory.
3. Set the descriptor as follows:
   - DESCR_SRC = 0x4000:0000
   - DESCR_DST = 0x0000:0000
   - DESCR_CTL.DESCR_TYPE = 0 (single transfer)
   - DESCR_CTL.WAIT_FOR_DEACT = 0
   - DESCR_CTL.INTR_TYPE = 2 (interrupt is generated after the execution of the current descriptor). The CPU is interrupted after the execution of the current descriptor. Because this a single transfer, INTR_TYPE can be 0 or 1 and will have the same effect.
   - DESCR_CTL.TR_IN_TYPE = 0 (trigger results in the execution of a single transfer). Setting it to 1 or 2 will have the same effect as the descriptor type is single transfer.
   - DESCR_CTL.DATA_SIZE = 1 (16 bits).
   - DESCR_CTL.SRC_TRANSFER_SIZE = 1 (32 bits)
   - DESCR_CTL.DST_TRANSFER_SIZE = 0 (DATA_SIZE = 16 bits)
   - DESCR_X_CTL.SRC_X_INCR = 0 (FIFO)
   - DESCR_NEXT_PTR = NULL.
4. DW0_CH_STRUCT3_CH_CTL.ENABLED = 1.

Transfer:

When the trigger is received, the transfer engine will load 32 bits from the ADC location and will store the lower 16 bits to the 0x0000:0000 memory location. Successive triggers will have no impact on the transfer because the link pointer is set to NULL. If the link pointer points to itself, then successive triggers will result in the same behavior as the original single transfer.

7.1.7.2 Example 2 - 2D transfer

In this example, the data transfer is from a peripheral that gathers input from two channels. The transfer is to a buffer in memory with the following constraints. Figure 26 is a pictorial representation of the transfer.

1. The two-channel data elements are interleaved in the peripheral FIFO.
2. Each data element is 2 bytes: X0 and X1. The first data element of channel 1 is (X1, X0). The first data element of channel 2 is (X3, X2).
3. The destination is a double buffer: the CPU processes one buffer while P-DMA fills in the other buffer.
4. The destination only considers channel 1. So the buffer contains (X1, X0), (X5, X4), (x9, X8), …
5. For each trigger from the peripheral, we must transfer eight data elements or 16 B (X1, X0), (X5, X4), (x9, X8), … , (X29, X28) to the destination.
6. Each buffer is 128 data elements or 256 B. When a buffer is full, the transfer switches to the other buffer.
7. When a buffer is full, an interrupt is generated.
8. The data transfer must continue indefinitely until the CPU disables the channel.

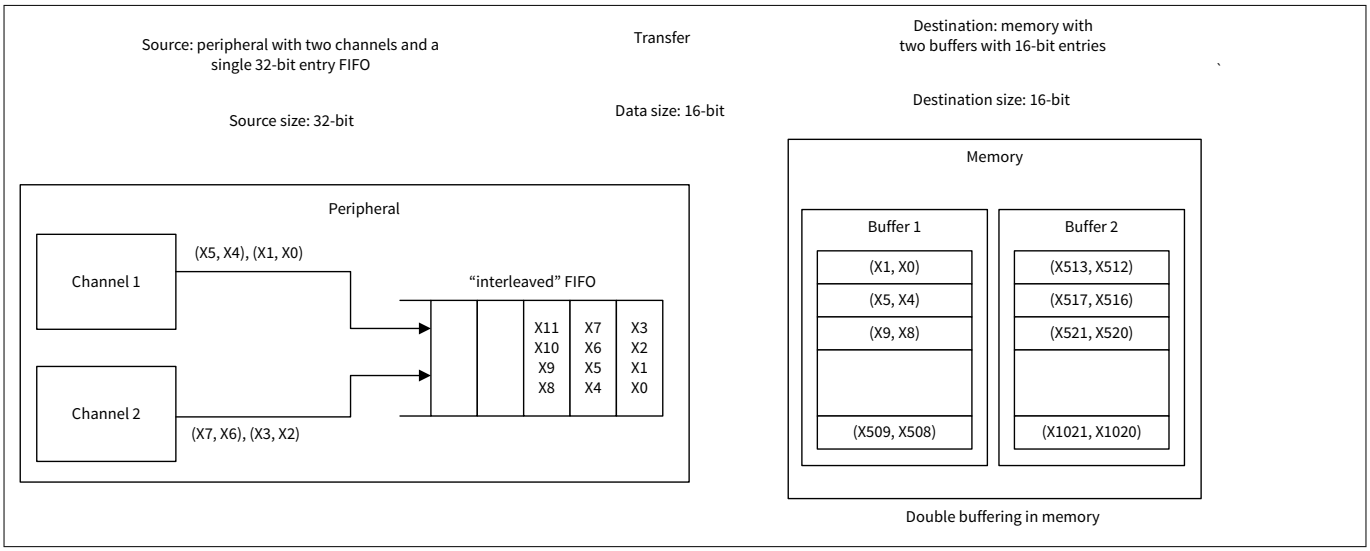


Figure 26 2D transfer

```
// DST_ADDR is a pointer to the buffer in memory of type uint16_t
// SRC_ADDR is a pointer to the FIFO of type uint32_t
for (Y_IDX = 0; Y_IDX ≤ 15; Y_IDX++) {
    for (X_IDX = 0; X_IDX ≤ 7; X_IDX++) {
        DST_ADDR[X_IDX * 1 + Y_IDX * 8 ] = (uint16_t) SRC_ADDR[0];
    }
}
```

Double buffering requires chaining of two descriptors. The basic data transfer is achieved by a 2D loop as shown in the pseudo code above.

Set the first descriptor as follows. The two-channel data elements are interleaved in the peripheral FIFO.

1. DESCR_SRC = address of the peripheral FIFO
2. DESCR_DST = address of the first buffer in memory
3. DESCR_CTL.DESCR_TYPE = 2 (2D transfer)
4. DESCR_CTL.INTR_TYPE = 2 (interrupt is generated after the execution of the current descriptor). The CPU is interrupted when one of the buffers is completely filled.
5. DESCR_CTL.TR_IN_TYPE = 1 (trigger results in the execution of a single 1D transfer). Here, eight elements are transferred.
6. DESCR_CTL.DATA_SIZE = 1 (16 bits)
7. DESCR_CTL.SRC_TRANSFER_SIZE = 1 (32 bits)
8. DESCR_CTL.DST_TRANSFER_SIZE = 0 (DATA_SIZE = 16 bits)

***Note**: The SRC_TRANSFER_SIZE of 32 bits and the DATA_SIZE of 16 bits effectively suppress the transfer of the second channel data elements.*

9. DESCR_X_CTL.SRC_X_INCR = 0 (FIFO)
10. DESCR_X_CTL.DST_X_INCR = 1
11. DESCR_X_CTL.X_COUNT = 7: 8 data elements
12. DESCR_Y_CTL.SRC_X_INCR = 0 (FIFO)
13. DESCR_Y_CTL.DST_Y_INCR = 8
14. DESCR_X_CTL.Y_COUNT = 15. The buffer size is (15+1) × (7+1) = 128 data elements.
15. DESCR_NEXT_PTR = address of the second descriptor in memory

Set the second descriptor same as the first except:

1. DESCR_DST = address of the second buffer in memory
2. DESCR_NEXT_PTR = address of the first descriptor in memory

This setting results in the required data transfer. Only channel 1 data elements are transferred according to the requirement. An interrupt is generated when a buffer is full. The destination buffers are alternated because of the chaining.

7.1.8 P-DMA descriptor structure

The P-DMA descriptor is stored in memory and it consists of six fields as follows:

Table 27 P-DMA descriptor structure

Offset	Name	Description
0x00	DESCR_CTL	Descriptor control
0x04	DESCR_SRC	Descriptor source
0x08	DESCR_DST	Descriptor destination
0x0c	DESCR_X_CTL	Descriptor X loop control
0x10	DESCR_Y_CTL	Descriptor Y loop control
0x14	DESCR_NEXT_PTR	Descriptor next pointer

The offset is based on the descriptor pointer position for each channel, which is stored in the register (DWx_CH_STRUCTy_CH_CURR_PTR).

The structure and explanation of each field are as follows:

DESCR_CTL - Descriptor control

Table 28 P-DMA descriptor control

Bit	Name	Description
1:0	WAIT_FOR_DEACT	Specifies whether the controller should wait for the input trigger to be deactivated; that is, the selected system trigger is not active. This field is used to synchronize the controller with the agent that generated the trigger. This field is used only on completion of the transfer as specified by TR_IN. For example, a TX FIFO indicates that it is empty and needs a new data sample. The agent removes the trigger only when the data sample has been written by the controller and received by the agent. Furthermore, the agent's trigger may be delayed by a few cycles before it reaches the controller. This field is used for a level-sensitive trigger, which reflects the state (pulse sensitive triggers should have this field set to '0'). The wait cycles incurred by this field reduce P-DMA controller performance. 0: Do not wait for trigger deactivation (for pulse sensitive triggers) 1: Wait for up to 4 cycles 2: Wait for up to 16 cycles 3: Wait indefinitely. This option may result in controller lockup if the trigger is not deactivated.
3:2	INTR_TYPE	Specifies when a completion interrupt is generated (CH_STATUS.INTR_CAUSE is set to COMPLETION): 0: An interrupt is generated after a single transfer. 1: An interrupt is generated after a single 1D transfer. • If the descriptor type is 'single', the interrupt is generated after a single transfer • If the descriptor type is 1D, CRC, or 2D, the interrupt is generated after the execution of a 1D transfer 2: An interrupt is generated after the execution of the current descriptor. Independent of the value of DESCR_NEXT_PTR.ADDR of the current descriptor. 3: An interrupt is generated after the execution of the current descriptor. The value of DESCR_NEXT_PTR.ADDR of the current descriptor must be '0'.
5:4	TR_OUT_TYPE	Specifies when an output trigger is generated: 0: An output trigger is generated after a single transfer 1: An output trigger is generated after a single 1D transfer • If the descriptor type is 'single', the output trigger is generated after a single transfer • If the descriptor type is 1D, CRC, or 2D, the output trigger is generated after the execution of a 1D transfer 2: An output trigger is generated after the execution of the current descriptor. 3: An output trigger is generated after the execution of a descriptor list: after the execution of the current descriptor and the current descriptor DESCR_NEXT_PTR.ADDR is '0'.

(table continues...)

Table 28 (continued) P-DMA descriptor control

Bit	Name	Description
7:6	TR_IN_TYPE	Specifies the input trigger type (not to be confused with the descriptor type): 0: A trigger results in the execution of a single transfer. The descriptor type can be single, 1D, or 2D 1: A trigger results in the execution of a single 1D transfer • If the descriptor type is 'single', the trigger results in the execution of a single transfer • If the descriptor type is 1D or 2D, the trigger results in the execution of a 1D transfer 2: A trigger results in the execution of the current descriptor 3: A trigger results in the execution of the current descriptor and continues (without requiring another input trigger) with the execution of the next descriptor using the next descriptor's information.
24	CH_DISABLE	Specifies whether the channel is disabled after completion of the current descriptor (independent of the value of the DESCR_NEXT_PTR value): 0: Channel is not disabled. 1: Channel is disabled. ***Note**: A disabled channel will ignore its input trigger.*
26	SRC_TRANSFER_SIZE	Specifies the bus transfer size to the source location: 0: As specified by DATA_SIZE. 1: Word (32 bits). Distinguishing bus transfer size from data element size allows for source components with data elements that are smaller than their 32-bit bus interface width. For example, an ADC source has a 32-bit bus transfer size, but only provides a 16-bit data element.
27	DST_TRANSFER_SIZE	Specifies the bus transfer size to the destination location: 0: As specified by DATA_SIZE 1: Word (32 bits) Distinguishing bus transfer size from data element size allows for destination components with data elements that are smaller than their 32-bit bus interface width.

(table continues...)

Table 28 (continued) P-DMA descriptor control

Bit	Name	Description
29:28	DATA_SIZE	Specifies the data element size: 0: Byte (8 bits) 1: Halfword (16 bits) 2: Word (32 bits) DATA_SIZE, SRC_TRANSFER_SIZE, and DST_TRANSFER_SIZE together determine how data elements are transferred. The following are the nine legal settings: • DATA is 8 bit, SRC is 8-bit, DST is 8-bit • DATA is 8 bit, SRC is 32-bit (higher 24 bits are dropped), DST is 8-bit • DATA is 8 bit, SRC is 8-bit, DST is 32 bit (higher 24 bits are made '0') • DATA is 8 bit, SRC is 32-bit (higher 24 bits are dropped), DST is 32-bit (higher 24 bits are made '0') • DATA is 16 bit, SRC is 16-bit, DST is 16-bit • DATA is 16 bit, SRC is 32-bit (higher 16 bits are dropped), DST is 16-bit • DATA is 16 bit, SRC is 16-bit, DST is 32 bit (higher 16 bits are made '0') • DATA is 16 bit, SRC is 32-bit (higher 16 bits are dropped), DST is 32-bit (higher 16 bits are made '0') • DATA is 32-bit, SRC is 32-bit, DST is 32-bit
31:30	DESCR_TYPE	Specifies the descriptor type (not to be confused with the trigger type): 0: Single transfer - The DESCR_X_CTL and DESCR_Y_CTL registers are not present and DESCR_NEXT_PTR is at offset 0x0c. 1: 1D transfer - The DESCR_X_CTL register is present, the DESCR_Y_CTL is not present, and DESCR_NEXT_PTR is at offset 0x10. A 1D transfer consists of DESCR_X_CTL.X_COUNT single transfers. 2: 2D transfer - The DESCR_X_CTL and DESCR_Y_CTL registers are present and DESCR_NEXT_PTR is at offset 0x14. A 2D transfer consists of DESCR_X_CTL.X_COUNT*DESCR_Y_CTL.Y_COUNT single transfers. 3: CRC transfer - The DESCR_X_CTL register is present, the DESCR_Y_CTL is not present and DESCR_NEXT_PTR is at offset 0x10. A CRC transfer consists of DESCR_X_CTL.X_COUNT single transfers. After the execution of the current descriptor, the DESCR_NEXT_PTR address is copied to the channel's DWx_CH_STRUCTy_CH_CURR_PTR address and DWx_CH_STRUCTy_CH_IDX.X_IDX and DWx_CH_STRUCTy_CH_IDX.Y_IDX are set to '0'.

DESCR_SRC - Descriptor source

Table 29 P-DMA descriptor source

Bit	Name	Description
31:0	SRC_ADDR	Base address of source location.

DESCR_DST - Descriptor destination

Table 30 P-DMA descriptor destination

Bit	Name	Description
31:0	DST_ADDR	Base address of destination location. ***Note**: For a CRC transfer descriptor, the calculated CRC value is stored at this address location. It is not subjected to post processing specified by the DWx_CRC_CTL0/DWx_CRC_REM_CTL0 registers. The CRC result after post processing is only available in the DWx_CRC_REM_RESULT0 register.*

DESCR_X_CTL - Descriptor X loop control

This register is not present for a single transfer descriptor type.

Table 31 P-DMA descriptor X loop control

Bit	Name	Description
11:0	SRC_X_INCR	Specifies increment of source address for each X loop iteration (in multiples of SRC_TRANSFER_SIZE). This field is a signed number in the range [–2048, 2047]. If this field is '0', the source address is not incremented. This is useful for reading from RX FIFO structures.
23:12	DST_X_INCR	Specifies increment of destination address for each X loop iteration (in multiples of DST_TRANSFER_SIZE). This field is a signed number in the range [–2048, 2047]. If this field is '0', the destination address is not incremented. This is useful for writing to TX FIFO structures. ***Note**: This field is not used for CRC transfer descriptors and must be set to '0'.*
31:24	X_COUNT	Number of iterations (minus 1) of the X loop (X_COUNT+1 is the number of single transfers in a 1D transfer). This field is an unsigned number in the range [0, 255], representing 1 through 256 iterations.

DESCR_Y_CTL - Descriptor Y loop control

This register is not present for the single, 1D, and CRC transfer descriptor types.

Table 32 P-DMA descriptor Y loop control

Bit	Name	Description
11:0	SRC_Y_INCR	Specifies increment of source address for each Y loop iteration (in multiples of SRC_TRANSFER_SIZE). This field is a signed number in the range [–2048, 2047].
23:12	DST_Y_INCR	Specifies increment of destination address for each Y loop iteration (in multiples of DST_TRANSFER_SIZE). This field is a signed number in the range [–2048, 2047].
31:24	Y_COUNT	Number of iterations (minus 1) of the Y loop (X_COUNT+1) × (Y_COUNT+1) is the number of single transfers in a 2D transfer). This field is an unsigned number in the range [0, 255], representing 1 through 256 iterations.

DESCR_NEXT_PTR - Descriptor next pointer

***Note**: For a single transfer descriptor type, this register is at offset 0x0c. For 1D and CRC transfer descriptor types, this register is at offset 0x10. For a 2D transfer descriptor type, this register is at offset 0x14.*

Table 33 P-DMA descriptor next pointer

Bit	Name	Description
31:2	ADDR	Address of next descriptor in the descriptor list. When this field is '0', this is the last descriptor in the descriptor list.

7.2 Memory DMA (M-DMA (DMAC))

The M-DMA controller is used to transfer data between memory and peripherals without CPU involvement:

- The CPU configures/programs the M-DMA controller
- The M-DMA controller performs the data transfers

The primary design target is to achieve high memory bandwidth with limited area overhead to the platform.

The main difference between the M-DMA and P-DMA controllers is that the M-DMA controller has dedicated channel logic (with channel state) for each channel, whereas the P-DMA reuses the channel logic for all channels. Furthermore, the M-DMA channel logic includes a 16-byte FIFO for temporary storage of data. This results in increased memory bandwidth, but comes at the cost of significant silicon area overhead for each channel. M-DMA supports an additional descriptor type called "Memory Copy". This is a special 1D transfer; DESCR_X_INCR.SRC_X_INCR and DESCR_X_INCR.DST_X_INCR are implicitly set to '1' and not part of the descriptor. This descriptor makes it possible to achieve higher bandwidth for certain class of transfers.

7.2.1 Overview

The M-DMA controller can be configured/programmed to perform multiple independent data transfers. Each data transfer is managed by a channel. The number of channels varies for different part numbers; more details are available in the device datasheet.

A channel has an associated priority. When there are multiple bus transfer requests, the priority decoder determines the highest priority channel for the request.

A data transfer is initiated by an input trigger. This trigger may originate from the source of the transfer, destination of the transfer, CPU software, or from another SoC component. Triggers provide active/sleep functionality and are not available in deep sleep and hibernate power modes.

The data transfer specifics are specified by a descriptor. This descriptor specifies (among other things):

- The source and destination address locations and the size of the transfer
- The actions of a channel; for example, generation of output triggers and interrupts
- Data transfer types can be single, 1D, or 2D as defined in the descriptor structure. These types essentially define the address sequences generated for source and destination. 1D and 2D transfers are used for "scatter gather" and other useful transfer operations.

A channel's descriptor state is encoded as part of the channel's register state (and not as part of the descriptor).

7.2.2 Channels

M-DMA supports multiple independent data transfers that are managed by different channels. Each channel connects to a specific system trigger through a trigger multiplexer that is outside the M-DMA controller. See the Trigger multiplexer for details.

- **Channel priority:**

  A channel is assigned a priority (DMAC_CHx_CTL.PRIO) between 0 and 3, with 0 being the highest priority and 3 being the lowest priority. Priority decoding determines the highest priority pending channel. Channels with the same priority constitute a priority group and within this priority group, the following round-robin arbitration is applied.

  A “round” consists of a contiguous sequence of channel activations, within this priority group, without any repetition. Within a round, higher priority is given to the lower channel indices. The notion of a round guarantees that within a group, higher channel indices do not yield to lower indices indefinitely.

  The data transfer associated with a trigger is made up of one or more atomic transfers or single transfers. For example, a 1D transfer consists of X_COUNT+1 single transfers.

- **Channel Access Control register:**

  A channel has three access control attributes that are used by the SMPUs and PPUs for access control:

  - Privileged mode (DMAC_CHx_CTL.P) attribute can be set to privileged or user.
  - Non-secure (DMAC_CHx_CTL.NS) attribute can be set to secure or non- secure.
  - PC (DMAC_CHx_CTL.PC) can be set to one of the protection contexts.

  These three fields are inherited from the write transaction and not specified by the transaction write data.

- **Channel Descriptor State registers:**

  The following registers provide a channel's descriptor state:

  - DMAC_CHx_CTL: This register provides generic channel control information.
  - DMAC_CHx_CURR: This register provides the address of the memory location where the current descriptor is located. Software needs to initialize this register. Hardware sets this register to the current descriptor's next descriptor pointer, when advancing from the current descriptor to the next descriptor in a descriptor list. When this field is ‘0’, there is no valid descriptor.
  - DMAC_CHx_IDX: This register provides the current X and Y indices of the channel into the current descriptor. Software needs to initialize this register. Hardware sets the X and Y indices to ‘0’, when advancing from the current descriptor to the next descriptor in a descriptor list.
  - DMAC_CHx_SRC: This register provides the current address of source location.
  - DMAC_CHx_DST: This register provides the current address of destination location.
  - DMAC_CHx_DESCR_STATUS: This register provides the validity of other DMAC_CHx_DESCR registers.
  - DMAC_CHx_DESCR_CTL: This register contains a copy of DESCR_CTL of the currently active descriptor.
  - DMAC_CHx_DESCR_SRC: This register contains a copy of DESCR_SRC of the currently active descriptor.
  - DMAC_CHx_DESCR_DST: This register contains a copy of DESCR_DST of the currently active descriptor.
  - DMAC_CHx_DESCR_X_INCR: This register contains a copy of DESCR_X_INCR of the currently active descriptor.
  - DMAC_CHx_DESCR_Y_SIZE: This register contains a copy of DESCR_X_SIZE of the currently active descriptor.
  - DMAC_CHx_DESCR_Y_INCR: This register contains a copy of DESCR_Y_INCR of the currently active descriptor.
  - DMAC_CHx_DESCR_Y_SIZE: This register contains a copy of DESCR_Y_SIZE of the currently active descriptor.
  - DMAC_CHx_DESCR_NEXT: This register contains a copy of DESCR_NEXT_PTR of the currently active descriptor.

- DMAC_CHx_INTR: This register contains the interrupts that are currently activated for this channel.
- DMAC_CHx_INTR_SET: Writing '1' to the appropriate bit in this register sets the corresponding DMAC_CHx_INTR field to '1'.
- DMAC_CHx_INTR_MASK: Mask for corresponding field in DMAC_CHx_INTR register.
- DMAC_CHx_INTR_MASKED: Logical and of corresponding DMAC_CHx_INTR and DMAC_CHx_INTR_MASK fields.
- DMAC_CHx_TR_CMD: This register allows the channel to be triggered through software. This is in addition to the software trigger control available in the trigger multiplexer.

Note that channel state is retained in deep sleep power mode.

The M-DMA controller is an active/sleep power mode functionality. Software should not initiate deep sleep system power mode entry if there are any active M-DMA controller channels transferring data. Note that there is no way of capturing the active channel data while transitioning to deep sleep system power mode.

7.2.3 Descriptors

A descriptor is stored in memory and describes a data transfer. The descriptor is read-only for the M-DMA controller.

Descriptor type (DESCR_TYPE) - There are five types of descriptors.

Table 34 M-DMA descriptor types

Descriptor type	Description
Single transfer	This transfers a single data element (8-bit, 16-bit, or 32-bit) as shown in Figure 27. The descriptor size is four 32-bit words: DESCR_CTL, DESCR_SRC, DESCR_DST, and DESCR_NEXT_PTR.
1D transfer	This performs a one-dimensional "for loop" (described in C) as shown in Figure 28. A 1D transfer is made up of X_COUNT+1 single transfers. The descriptor size is six 32-bit words: DESCR_CTL, DESCR_SRC, DESCR_DST, DESCR_X_INCR, DESCR_X_SIZE, and DESCR_NEXT_PTR.
2D transfer	This performs a two-dimensional "for loop" (described in C) as shown in Figure 29. A 2D transfer is made up of (Y_COUNT+1) 1D transfers. The descriptor size is eight 32-bit words: DESCR_CTL, DESCR_SRC, DESCR_DST, DESCR_X_INCR, DESCR_X_SIZE, DESCR_Y_INCR, DESCR_Y_SIZE, and DESCR_NEXT_PTR.
Memory Copy	This is a special case of 1D transfer as shown in Figure 30; DESCR_X_INCR.SRC_X_INCR and DESCR_X_INCR.DST_X_INCR are implicitly set to 1 and not part of the descriptor. The size of the descriptor is five 32-bit words. The M-DMA is optimized for performance.
Scatter	This descriptor type is intended to write a set of 32-bit data elements as shown in Figure 31, whose addresses are "scattered" around the address space. The size of the descriptor is four 32-bit words. DESCR_CTL, DESCR_SRC, DESCR_X_SIZE, and DESCR_NEXT_PTR.

The functionality of these five descriptor types is described by the following pseudo code.

```
// DST_ADDR is a pointer to an object of type defined by DST_TRANSFER_SIZE
// SRC_ADDR is a pointer to an object of type defined by SRC_TRANSFER_SIZE
// t_DATA_SIZE is the type associated with the DATA_SIZE
DST_ADDR[0] = (t_DATA_SIZE) SRC_ADDR[0];
```

Figure 27 Single transfer

```
// DST_ADDR is a pointer to an object of type defined by DST_TRANSFER_SIZE
// SRC_ADDR is a pointer to an object of type defined by SRC_TRANSFER_SIZE
// t_DATA_SIZE is the type associated with the DATA_SIZE
for (X_IDX = 0; X_IDX ≤ X_COUNT; X_IDX++) {
       DST_ADDR[X_IDX * DST_X_INCR] =
             (t_DATA_SIZE) SRC_ADDR[X_IDX * SRC_X_INCR];
}
```

Figure 28 1D transfer

```
// DST_ADDR is a pointer to an object of type defined by DST_TRANSFER_SIZE
// SRC_ADDR is a pointer to an object of type defined by SRC_TRANSFER_SIZE
// t_DATA_SIZE is the type associated with the DATA_SIZE
for (Y_IDX = 0; Y_IDX ≤ Y_COUNT; Y_IDX++) {
    for (X_IDX = 0; X_IDX ≤ X_COUNT; X_IDX++) {
       DST_ADDR[X_IDX * DST_X_INCR + Y_IDX * DST_Y_INCR ] =
             (t_DATA_SIZE) SRC_ADDR[X_IDX * SRC_X_INCR + Y_IDX * SRC_Y_INCR];
    }
}
```

Figure 29 2D transfer

```
// DST_ADDR is a pointer to an object of type uint8_t
// SRC_ADDR is a pointer to an object of type uint8_t
// This transfer type uses 8-bit, 16-bit an 32-bit transfers. The hardware ensures that
// alignment requirements are met.
for (X_IDX = 0; X_IDX ≤ X_COUNT; X_IDX++) {
    DST_ADDR[X_IDX] = SRC_ADDR[X_IDX];
}
```

Figure 30 Memory copy

```
// SRC_ADDR is a pointer to an object of type uint32_t
for (X_IDX = 0; X_IDX ≤ X_COUNT; X_IDX += 2) {
      address = SRC_ADDR[X_IDX];
      data    = SRC_ADDR[X_IDX + 1];
      *address = data;
}
```

Figure 31 Scatter

Descriptor size – The size of a descriptor depends on its type. Only relevant parameters are stored. For example, a 1D descriptor does not contain the Y_SIZE and Y_INCR parameters.

Transfer size (SRC_TRANSFER_SIZE and DST_TRANSFER_SIZE) – In a data transfer, the source data is cast into the type specified by DATA_SIZE and assigned to the destination. The source type is determined by

SRC_TRANSFER_SIZE and the destination type is determined by DST_TRANSFER_SIZE. All types are unsigned. All address computations use C semantics based on the transfer size.

Descriptor chaining – Descriptors chained together. DESCR_NEXT_PTR field contains a pointer to the next descriptor in the chain. A channel executes the next descriptor in the chain when it completes executing the current descriptor. The last descriptor in the chain has DESCR_NEXT_PTR set to '0' (NULL pointer). A descriptor chain is also referred to as a descriptor list. It is possible to have a circular list in which case the execution continues indefinitely until there is an error or the channel or the controller is disabled by software.

Trigger-in type (TR_IN_TYPE) – An input trigger initiates a data transfer and the TR_IN_TYPE defines the action on a trigger.

Table 35 M-DMA trigger-in types

Trigger type	Description
Type 0	Trigger results in the execution of a single transfer. In a 1D or 2D transfer, this will execute a single transfer in the loop.
Type 1	Trigger results in the execution of a single 1D transfer. If the descriptor type is single transfer this behaves similar to type 0. If the descriptor type is 2D, it results in executing the inner loop once.
Type 2	Trigger results in the execution of the current descriptor.
Type 3	Trigger results in the execution of a descriptor list.

Trigger-out type (TR_OUT_TYPE) – This defines when an output trigger is generated.

Table 36 M-DMA trigger-out types

Trigger type	Description
Type 0	Output trigger is generated after a single transfer. In a 1D or 2D transfer, an output trigger is generated after each transfer in the loop.
Type 1	Output trigger is generated after a single 1D transfer. In a single transfer descriptor type, this behaves similar to type 0. If the descriptor type is 2D, an output trigger is generated after each execution of the inner loop.
Type 2	Output trigger is generated after the execution of the current descriptor.
Type 3	Output trigger is generated after the execution of a descriptor list.

Interrupt type (INTR_TYPE) – This defines when a completion interrupt is generated.

Table 37 M-DMA interrupt types

Trigger type	Description
Type 0	Interrupt is generated after a single transfer. In a 1D or 2D transfer, an interrupt is generated after each transfer in the loop
Type 1	Interrupt is generated after a single 1D transfer. In a single transfer descriptor type, this behaves similar to type 0. If the descriptor type is 2D, an interrupt is generated after each execution of the inner loop.
Type 2	Interrupt is generated after the execution of the current descriptor.
Type 3	Interrupt is generated after the execution of a descriptor list.

Wait for deactivation (WAIT_FOR_DEACT) – Specifies whether the M-DMA controller should wait for the input trigger to be deactivated after it has completed the data transfer corresponding to the current trigger. This field

is used for level-sensitive triggers to give sufficient time for the triggering agent to deactivate the trigger. The wait specified can be 0, up to four cycles, up to 16 cycles, or indefinite. Pulse-sensitive triggers should have this field set to 0.

Data prefetch – If this bit is set, source data transfers are initiated as soon as the channel is enabled, the current descriptor pointer is not 0, and there is space available in the channel's data FIFO. When the input trigger is activated, the trigger can initiate destination data transfers with data that is already in the channel's data FIFO. This effectively shortens the initial delay of the data transfer. Data prefetch should be used with care, to ensure that data synchronization is not violated.

7.2.4 Interrupts

M-DMA can generate interrupts on completion and on error conditions:

- The INTR_TYPE descriptor control defines when a completion condition (COMPLETION) is activated.
- The error conditions include SRC_BUS_ERROR, DST_BUS_ERROR, SRC_MISAL, DST_MISAL, CURR_PTR_NULL, ACTIVE_CH_DISABLED, and DESCR_BUS_ERROR.

DMAC_CHx_INTR – Each channel has an Interrupt Request register. There are eight possible causes that can generate an interrupt. These causes are encoded in bits 0 to 7. Software can clear these by writing to these bits.

DMAC_CHx_INTR_SET – Each channel has an Interrupt Set register. There are eight bits (same as DMAC_CHx_INTR) and software can write 1 to any of these bits to set the corresponding DMAC_CHx_INTR bit.

DMAC_CHx_INTR_MASK – Each channel has an Interrupt Mask register. There are eight bits (same as DMAC_CHx_INTR) and they can be selectively enabled by writing 1 to the corresponding bits.

DMAC_CHx_INTR_MASKED – Each channel has an Interrupt Masked register. When read, this register reflects a bitwise “and” between the Interrupt Request and Mask registers.

The M-DMA is an active power mode peripheral; this means, it uses active functionality interrupts. Therefore, DMAC_CHx_INTR and DMAC_CHx_INTR_SET are not retained in deep sleep power mode (DMAC_CHx_INTR_MASK is retained).

7.2.5 Control and Active registers

DMAC_CHx_CTL.ENABLED indicates whether the M-DMA is enabled. Software writes to this register to enable the controller.

The ACTIVE register indicates which channels are currently active – enabled channels whose trigger is activated.

7.2.6 M-DMA controller design

The following figure gives an overview of the M-DMA controller design.

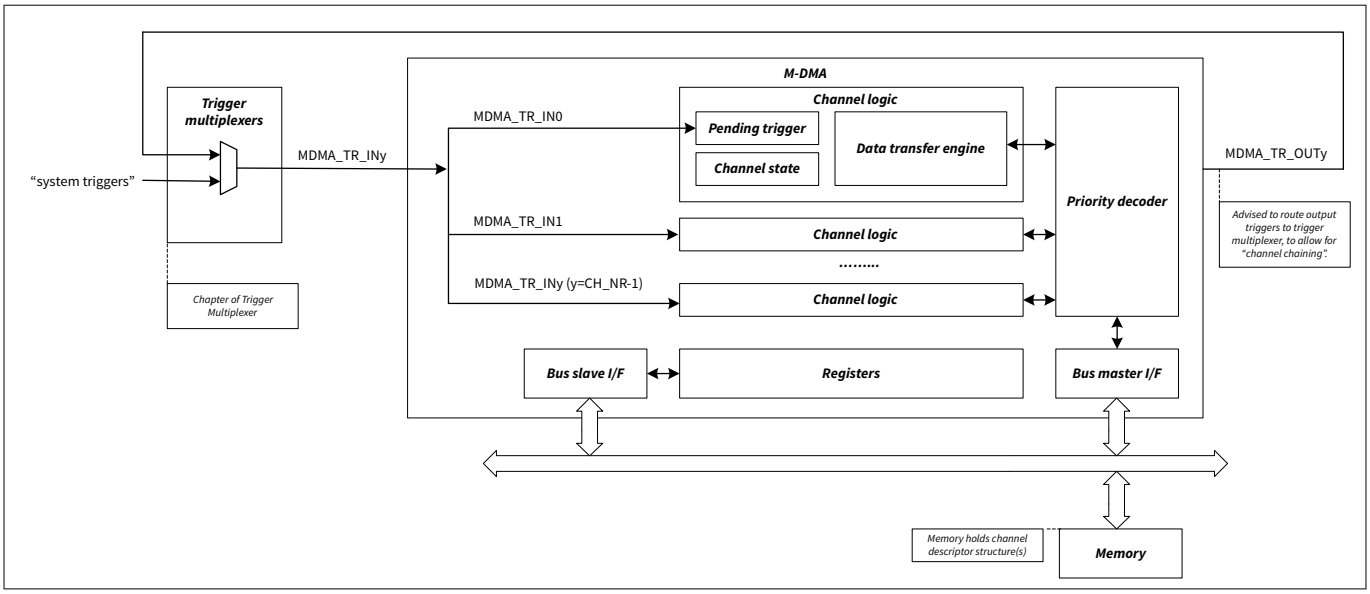


Figure 32 M-DMA controller design

The following components are distinguished:

- **Channel logic:**

Each M-DMA controller channel has its own dedicated channel logic. This logic tracks the channel's input trigger and maintains the channel state (Channel registers and a copy of the current descriptor from memory) and a data transfer engine. The data transfer engine transfers data elements from a source location to a destination location as specified by the channel state. The channels transfer requests are arbitrated by the priority decoder using channel specific priorities.

Each channel consists of two state machines that are connected through a 16-byte FIFO. The first state machine reads the descriptors from memory and data from the source location. When the current descriptor is read from memory, it is part of the channel's state. Source location data is temporarily buffered in the FIFO. The second state machine writes the buffered data in the FIFO to the source location.

- **Priority decoder** determines the highest priority channel with a bus transfer request.
- **Registers** – A description of the registers is available in the memory map. This memory map also describes the descriptors.
- **Master I/F** is an AHB-Lite bus master, which allows the controller to initiate AHB-Lite data transfers to the source and destination locations as well as to read the descriptor from memory.
- **Slave I/F** is an AHB-Lite bus slave, which allows the main CPU to access M-DMA controller Control/Status registers.

7.2.7 Examples

Example 1: The source is a 32-bit word addressable peripheral; the destination is regular memory. The M-DMA controller transfers five bytes from the source to destination. The source transfer size is a 32-bit word. The data size is an 8-bit byte. The destination transfer size is an 8-bit byte. A 1D transfer descriptor type is used.

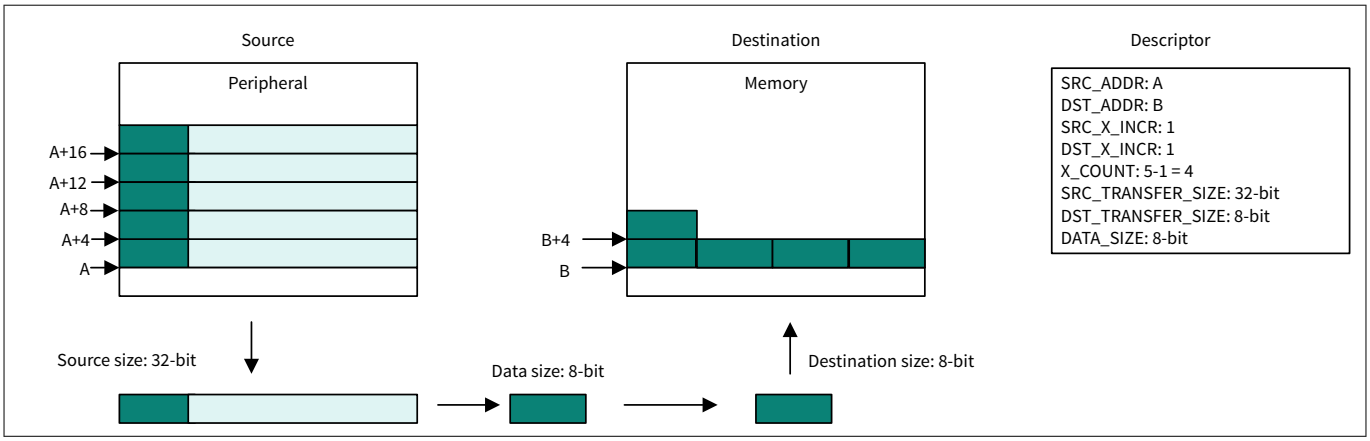


Figure 33 M-DMA 1D transfer

Example 2: The source and memory are regular memory. The M-DMA controller transfers five byte pairs and de-interleaves the pairs as part of the transfer. The source transfer size, data size, and destination transfer size are all 8-bit bytes. A 2D transfer descriptor type is used.

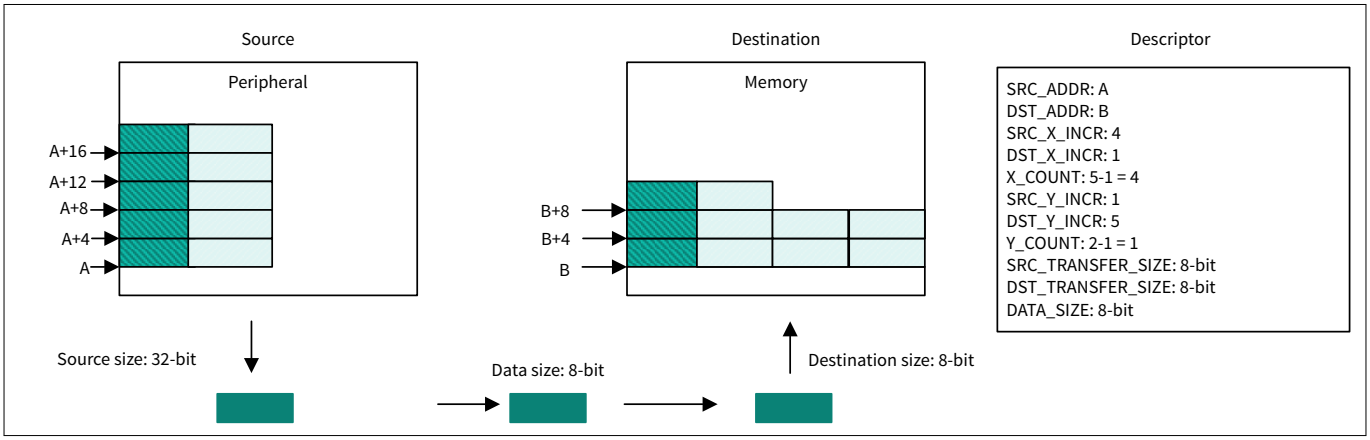


Figure 34 M-DMA 2D transfer

Example 3: The source and memory are regular memory. The M-DMA controller transfers bytes in the inverse direction of the previous example (note how the source and destination increments are reversed).

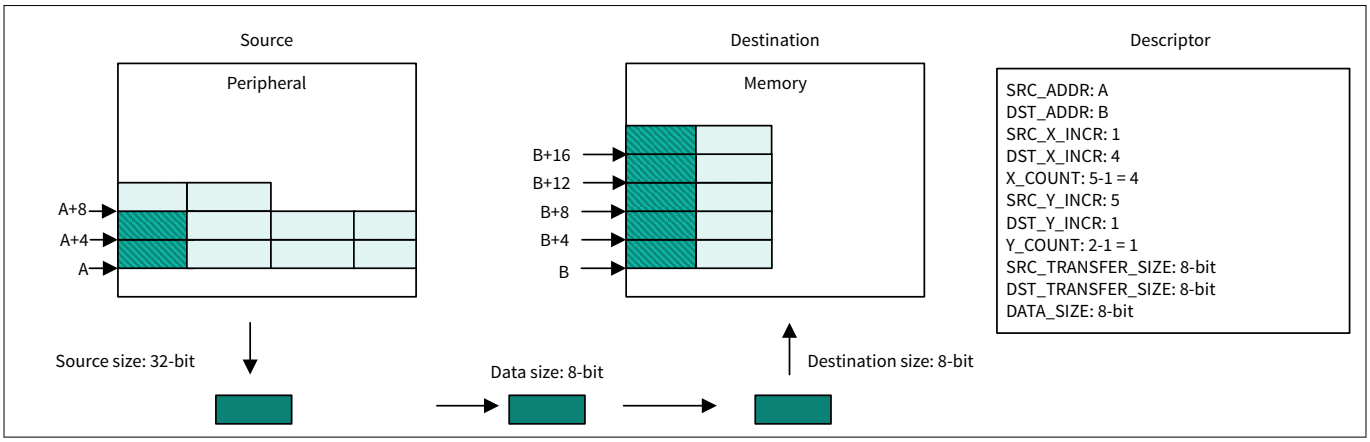


Figure 35 M-DMA inverse direction transfer

7.2.8 M-DMA descriptor structure

The M-DMA descriptor is stored in memory and it consists of eight fields as follows:

Table 38 M-DMA descriptor structure

Offset	Name	Description
0x00	DESCR_CTL	Descriptor control
0x04	DESCR_SRC	Descriptor source
0x08	DESCR_DST	Descriptor destination
0x0c	DESCR_X_SIZE	Descriptor X loop size
0x10	DESCR_X_INCR	Descriptor X loop increment
0x14	DESCR_Y_SIZE	Descriptor Y loop size
0x18	DESCR_Y_INCR	Descriptor Y loop increment
0x1c	DESCR_NEXT_PTR	Descriptor next pointer

The offset is based on the descriptor pointer position for each channel which is stored in the register (DMAC_CHx_CURR). The structure and explanation of each field are as follows:

DESCR_CTL - Descriptor control

Table 39 M-DMA descriptor

Bit	Name	Description
1:0	WAIT_FOR_DEACT	Specifies whether the controller should wait for the input trigger to be deactivated; that is, the selected system trigger is not active. This field is used to synchronize the controller with the agent that generated the trigger. This field is used only on completion of the transfer as specified by TR_IN. For example, a TX FIFO indicates that it is empty and needs a new data sample. The agent removes the trigger only when the data sample has been written by the controller and received by the agent. Furthermore, the agent's trigger may be delayed by a few cycles before it reaches the controller. This field is used for a level-sensitive trigger, which reflects the state (pulse sensitive triggers should have this field set to '0'). The wait cycles incurred by this field reduce M-DMA controller performance. 0: Do not wait for trigger deactivation (for pulse sensitive triggers) 1: Wait for up to 4 cycles 2: Wait for up to 16 cycles 3: Wait indefinitely. This option may result in controller lockup if the trigger is not deactivated.

(table continues...)

Table 39 (continued) M-DMA descriptor

Bit	Name	Description
3:2	INTR_TYPE	Specifies when a completion interrupt is generated: 0: An interrupt is generated after a single transfer. 1: An interrupt is generated after a single 1D transfer or a memory copy transfer. • If the descriptor type is “single”, the interrupt is generated after a single transfer. • If the descriptor type is 1D or 2D, the interrupt is generated after the execution of a 1D transfer. • If the descriptor type is “memory copy”, the interrupt is generated after the execution of a memory copy transfer. • If the descriptor type is “scatter”, the interrupt is generated after the execution of a scatter transfer. 2: An interrupt is generated after the execution of the current descriptor. Independent of the value of DESCR_NEXT_PTR.ADDR of the current descriptor. 3: An interrupt is generated after the execution of the current descriptor and the current descriptor's DESCR_NEXT_PTR.ADDR is ‘0’.
5:4	TR_OUT_TYPE	Specifies when an output trigger is generated: 0: An output trigger is generated after a single transfer. 1: An output trigger is generated after a single 1D transfer or a memory copy transfer. • If the descriptor type is “single”, the output trigger is generated after a single transfer. • If the descriptor type is 1D or 2D, the output trigger is generated after the execution of a 1D transfer. • If the descriptor type is “memory copy”, the output trigger is generated after the execution of a memory copy transfer. • If the descriptor type is “scatter”, the output trigger is generated after the execution of a scatter transfer. 2: An output trigger is generated after the execution of the current descriptor. 3: An output trigger is generated after the execution of a descriptor list: after the execution of the current descriptor and the current descriptor's DESCR_NEXT_PTR.ADDR is ‘0'.

(table continues...)

Table 39 (continued) M-DMA descriptor

Bit	Name	Description
7:6	TR_IN_TYPE	Specifies the input trigger type (not to be confused with the descriptor type): 0: A trigger results in the execution of a single transfer. The descriptor type can be single, 1D or 2D. 1: A trigger results in the execution of a single 1D transfer. • If the descriptor type is “single”, the trigger results in the execution of a single transfer. • If the descriptor type is 1D or 2D, the trigger results in the execution of a 1D transfer. • If the descriptor type is “memory copy”, the trigger results in the execution of a memory copy transfer. • If the descriptor type is “scatter”, the trigger results in the execution of an scatter transfer. 2: A trigger results in the execution of the current descriptor. 3: A trigger results in the execution of the current descriptor and continues (without requiring another input trigger) with the execution of the next descriptor using the next descriptor's information.
8	DATA_PREFETCH	Source data prefetch: 0: No source data prefetch. Source data transfers are only initiated after the input trigger is activated. 1: Source data prefetch. Source data transfers are initiated as soon as the channel is enabled, the current descriptor pointer is not 0 and there is space available in the channel's data FIFO. When the input trigger is activated, the trigger can initiate destination data transfers with data that is already in the channel's data FIFO. This effectively shortens the initial delay of the data transfer. ***Note**: Data prefetch should be used with care, to ensure that data coherency is guaranteed and that prefetches do not cause undesired side effects.*

(table continues...)

Table 39 (continued) M-DMA descriptor

Bit	Name	Description
17:16	DATA_SIZE	Specifies the data element size: 0: Byte (8 bits) 1: Halfword (16 bits) 2: Word (32 bits) DATA_SIZE, SRC_TRANSFER_SIZE, and DST_TRANSFER_SIZE together determine how data elements are transferred. The following are the nine legal settings: • DATA is 8-bit, SRC is 8-bit, DST is 8-bit. • DATA is 8-bit, SRC is 32-bit (higher 24 bits are dropped), DST is 8-bit. • DATA is 8-bit, SRC is 8 bit, DST is 32-bit (higher 24 bits are made '0'). • DATA is 8-bit, SRC is 32-bit (higher 24 bits are dropped), DST is 32 bit (higher 24 bits are made '0'). • DATA is 16-bit, SRC is 16-bit, DST is 16-bit. • DATA is 16-bit, SRC is 32-bit (higher 16 bits are dropped), DST is 16-bit. • DATA is 16-bit, SRC is 16-bit, DST is 32 bit (higher 16 bits are made '0'). • DATA is 16-bit, SRC is 32-bit (higher 16 bits are dropped), DST is 32-bit (higher 16 bits are made '0'). • DATA is 32 bit, SRC is 32-bit, DST is 32-bit. ***Note**: This field is not used for a "memory copy" descriptor type. It must be set to '2' for a "initialization" descriptor type.*
24	CH_DISABLE	Specifies whether the channel is disabled or not after completion of the current descriptor (independent of the value of the DESCR_NEXT_PTR value): 0: Channel is not disabled 1: Channel is disabled
26	SRC_TRANSFER_SIZE	Specifies the bus transfer size to the source location: 0: As specified by DATA_SIZE 1: Word (32 bits) Distinguishing bus transfer size from data element size allows for source components with data elements that are smaller than their 32-bit bus interface width. For example, an ADC source has a 32-bit bus transfer size, but only provides a 16-bit data element. ***Note**: This field is not used for a "memory copy" descriptor type. It must be set to '1' for a "scatter" descriptor type.*

(table continues...)

Table 39 (continued) M-DMA descriptor

Bit	Name	Description
27	DST_TRANSFER_SIZE	Specifies the bus transfer size to the destination location: 0: As specified by DATA_SIZE 1: Word (32 bits) Distinguishing bus transfer size from data element size allows for destination components with data elements that are smaller than their 32-bit bus interface width. ***Note**: This field is not used for a "memory copy" descriptor type. It must be set to '1' for a "scatter" descriptor type.*
30:28	DESCR_TYPE	Specifies the descriptor type (not to be confused with the trigger type): 0: Single transfer: The DESCR_X_SIZE, DESCR_X_INCR, DESCR_Y_SIZE, and DESCR_Y_INCR registers are not present. The DESCR_NEXT_PTR is at offset 0x0c. 1: 1D transfer: The DESCR_X_SIZE and DESCR_X_INCR registers are present, the DESCR_Y_SIZE and DESCR_Y_INCR are not present. A 1D transfer consists out of DESCR_X_SIZE.X_COUNT+1 single transfers. The DESCR_NEXT_PTR is at offset 0x14. 2: 2D transfer: The DESCR_X_SIZE, DESCR_X_INCR, DESCR_Y_SIZE, and DESCR_Y_INCR registers are present. A 2D transfer consists of (DESCR_X_SIZE.X_COUNT+1)*(DESCR_Y_SIZE.Y_COUNT+1) single transfers. The DESCR_NEXT_PTR is at offset 0x1c. 3: Memory copy: The DESCR_X_SIZE register is present, the DESCR_X_INCR, DESCR_Y_SIZE, and DESCR_Y_INCR are not present. A memory copy transfer copies DESCR_X_SIZE.X_COUNT+1 Bytes and may use Byte, halfword, and word transfers. The DESCR_NEXT_PTR is at offset 0x10. 4: Scatter transfer. The DESCR_X_SIZE register is present, the DESCR_DST, DESCR_X_INCR, DESCR_Y_SIZE, and DESCR_Y_INCR are not present. 5-7: Undefined. After the execution of the current descriptor, the DESCR_NEXT_PTR address is copied to the channel's DMAC_CHx_CURR address and DMAC_CHx_IDX.X and DMAC_CHx_IDX.Y are set to '0'.

DESCR_SRC - Descriptor source

Table 40 M-DMA descriptor source

Bit	Name	Description
31:0	SRC_ADDR	Base address of source location

DESCR_DST - Descriptor destination

Table 41 M-DMA Descriptor destination

Bit	Name	Description
31:0	DST_ADDR	Base address of destination location

DEDESCR_X_SIZE - Descriptor X loop size

This register is not present for the single transfer descriptor type.

Table 42 M-DMA descriptor X loop size

Bit	Name	Description
15:0	X_COUNT	Number of iterations (minus 1) of the X loop (X_COUNT+1 is the number of single transfers in a 1D transfer). This field is an unsigned number in the range [0, 65535], representing 1 through 65536 iterations. For the memory copy descriptor type, this field specifies the number of transferred Bytes (minus 1). For the scatter descriptor type, this field specifies the number of (address, write data) initialization pairs (times 2, minus 1).

DESCR_X_INCR - Descriptor X loop increment

This register is not present for the single transfer, memory copy, and scatter descriptor types.

Table 43 M-DMA descriptor X loop increment

Bit	Name	Description
15:0	SRC_X_INCR	Specifies increment of source address for each X loop iteration (in multiples of SRC_TRANSFER_SIZE). This field is a signed number in the range [–32768, 32767]. If this field is '0', the source address is not incremented. This is useful for reading from RX FIFO structures.
31:16	DST_X_INCR	Specifies increment of destination address for each X loop iteration (in multiples of DST_TRANSFER_SIZE). This field is a signed number in the range [–32768, 32767]. If this field is 0, the destination address is not incremented. This is useful for writing to TX FIFO structures.

DEDESCR_Y_SIZE - Descriptor Y loop size

This register is not present for the single transfer, 1D transfer, memory copy, and scatter descriptor types.

Table 44 M-DMA descriptor y loop size

Bit	Name	Description
15:0	Y_COUNT	Number of iterations (minus 1) of the Y loop (X_COUNT+1) × (Y_COUNT+1) is the number of single transfers in a 2D transfer). This field is an unsigned number in the range [0, 65535], representing 1 through 65536 iterations.

DESCR_Y_INCR - Descriptor Y loop increment

This register is not present for the single transfer, 1D transfer, memory copy, and scatter descriptor types.

Table 45 M-DMA descriptor Y loop increment

Bit	Name	Description
15:0	SRC_Y_INCR	Specifies increment of source address for each Y loop iteration (in multiples of SRC_TRANSFER_SIZE). This field is a signed number in the range [–32768, 32767].
31:16	DST_Y_INCR	Specifies increment of destination address for each Y loop iteration (in multiples of DST_TRANSFER_SIZE). This field is a signed number in the range [–32768, 32767].

DESCR_NEXT_PTR - Descriptor next pointer

***Note**: For a single transfer descriptor type, this register is at offset 0x0c. For a 1D transfer descriptor type, this register is at offset 0x14. For a 2D transfer descriptor type, this register is at offset 0x1c. For a memory copy transfer descriptor type, this register is at offset 0x10. For a scatter transfer descriptor type, this register is at offset 0x0c.*

Table 46 M-DMA descriptor next pointer

Bit	Name	Description
31:2	ADDR	Address of next descriptor in the descriptor list. When this field is 0, this is the last descriptor in the descriptor list.

7.3 Registers

Table 47 P-DMA registers

Register	Name	Description
DWx_CTL0	Control register	This register provides P-DMA enable/disable control and ECC checking/injection for SRAM enable/disable control
DWx_STATUS0	Status register	This register provides status of the P-DMA controller
DWx_ACT_DESCR_CTL0	Active Descriptor Control register	This register provides the copy of DESCR_CTL field of the currently active descriptor
DWx_ACT_DESCR_SRC0	Active Descriptor Source register	This register provides the copy of DESCR_SRC field of the currently active descriptor.
DWx_ACT_DESCR_DST0	Active Descriptor Destination register	This register provides the copy of DESCR_DST field of the currently active descriptor.
DWx_ACT_DESCR_X_CTL0	Active Descriptor X Loop Control register	This register provides the copy of DESCR_X_CTL field of the currently active descriptor. If the currently active descriptor does not have X_CTL, this register provides undefined information.

(table continues...)

Table 47 (continued) P-DMA registers

Register	Name	Description
DWx_ACT_DESCR_Y_CTL0	Active Descriptor Y Loop Control register	This register provides the copy of DESCR_Y_CTL field of the currently active descriptor. If the currently active descriptor does not have Y_CTL, this register provides undefined information.
DWx_ACT_DESCR_NEXT_PTR0	Active Descriptor Next Pointer register	This register provides the copy of DESCR_ NEXT_PTR field of the currently active descriptor.
DWx_ACT_SRC0	Active Source register	This register provides the current address of source location. This location is not a copy of the source address in the descriptor, but provides a real time source address of the active transfer.
DWx_ACT_DST0	Active Destination register	This register provides the current address of destination location. This location is not a copy of the destination address in the descriptor, but provides a real time destination address of the active transfer.
DWx_ECC_CTL0	ECC Control register	This register provides to specify the word address where an error will be injected and ECC parity to use for ECC error injection.
DWx_CRC_CTL0	CRC Control register	This register provides to specify the bit order in which a data byte is processed (reversal is performed after XORing), and to specify whether the remainder is bit reversed (reversal is performed after XORing)
DWx_CRC_DATA_CTL0	CRC Data Control register	This register provides to specify a byte mask with which each data byte is XOR'd. The XOR is performed before data reversal.
DWx_CRC_POL_CTL0	CRC Polynomial Control register	This register provides to specify CRC polynomial. The polynomial is represented without the high order bit (this bit is always assumed '1'). The polynomial should be aligned/shifted such that the more significant bits (bit 31 and down) contain the polynomial and the less significant bits (bit 0 and up) contain padding '0's. Some frequently used polynomials: • CRC32: POLYNOMIAL is 0x04c11db7 (x^32 + x^26 + x^23 + x^22 + x^16 + x^12 + x^11 + x^10 + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1). • CRC16: POLYNOMIAL is 0x80050000 (x^16 + x^15 + x^2 + 1, shifted by 16 bit positions). • CRC16 CCITT: POLYNOMIAL is 0x10210000 (x^16 + x^12 + x^5 + 1, shifted by 16 bit positions).

(table continues...)

Table 47 (continued) P-DMA registers

Register	Name	Description
DWx_CRC_LFSR_CTL0	CRC LFSR Control register	This register provides the state of a 32-bit Linear Feedback Shift registers (LFSR) that is used to implement CRC. This register needs to be initialized by software to provide the CRC seed value. The seed value should be aligned such that the more significant bits (bit 31 and down) contain the seed value and the less significant bits (bit 0 and up) contain padding '0's. Note that software can write this field. This functionality can be to used prevent information leakage (through DWx_CRC_LFSR_CTL0 or DWx_CRC_REM_RESULT0).
DWx_CRC_REM_CTL0	CRC Remainder Control register	This register provides to specifies a mask with which the DWx_CRC_LFSR_CTL0 register is XOR'd to produce a remainder. The XOR is performed before remainder reversal.
DWx_CRC_REM_RESULT0	CRC Remainder Result register	This register provides the remainder value. The alignment of the remainder depends on DWx_CRC_REM_CTL0.REM_REVERSE. Note that this field is combinatorially derived from DWx_CRC_LFSR_CTL0.LFSR32, DWx_CRC_CTL0.REM_REVERSE and DWx_CRC_REM_CTL0.REM_XOR.
DWx_CH_STRUCTy_CH_CTL	Channel Control register	This register provides generic channel control information.
DWx_CH_STRUCTy_CH_STATUS	Channel Status register	This register provides channel status which are the sources of interrupt factors and pending state.
DWx_CH_STRUCTy_CH_IDX	Channel Current Indices register	This register provides the current X and Y indices of the channel into the current descriptor.
DWx_CH_STRUCTy_CH_CURR_PTR	Channel Current Descriptor Pointer register	This register provides the address of the memory location where the current descriptor is located.
DWx_CH_STRUCTy_INTR	Interrupt register	This register provides an interrupt request. Bit 0 is set ‘1’ when interrupt event (completion or error) is detected. Software can clear this by writing to this bit.
DWx_CH_STRUCTy_INTR_SET	Interrupt Set register	This register provides interrupt setting. Software can write 1 to this register to set the corresponding DMAC_CHx_INTR register. When read, this register reflects the DWx_CH_STRUCTy_INTR register.

(table continues...)

Table 47 (continued) P-DMA registers

Register	Name	Description
DWx_CH_STRUCTy_INTR_MASK	Interrupt Mask register	This register provides interrupt mask setting. The corresponding interrupt is enabled by writing 1 to this register.
DWx_CH_STRUCTy_INTR_MASKED	Interrupt Masked register	This register provides interrupt masked. When read, this register reflects a bit-wise AND between the DWx_CH_STRUCTy_INTR and DWx_CH_STRUCTy_INTR_MASK fields.
DWx_CH_STRUCTy_SRAM_DATA0	SRAM Data 0 register	DWx_CH_STRUCTy_SRAM_DATA0 and DWx_CH_STRUCTy_SRAM_DATA1 are provided for ECC fault injection functionality.
DWx_CH_STRUCTy_SRAM_DATA1	SRAM data 1 register	DWx_CH_STRUCTy_SRAM_DATA0 and DWx_CH_STRUCTy_SRAM_DATA1 are provided for ECC fault injection functionality.
DWx_CH_STRUCTy_TR_CMD	Software Trigger register	When written with '1', a trigger is generated which sets 'trigger pending' (only if the channel is enabled). A read always returns a '0'.

***Note**: In DWx_CH_STRUCTy, 'x' signifies the DW/P-DMA instance and 'y' signifies the channel number.*

Table 48 M-DMA registers

Register	Name	Description
DMAC_CTL	Control register	This register provides M-DMA enable/disable control
DMAC_ACTIVE	Active Channels register	This register provides active channels
DMAC_CHx_CTL	Channel Control register	This register provides generic channel control information.
DMAC_CHx_IDX	Channel Current Indices register	This register provides the current X and Y indices of the channel into the current descriptor.
DMAC_CHx_SRC	Channel Current Source Address register	This register provides the current address of source location.
DMAC_CHx_DST	Channel Current Destination Address register	This register provides the current address of destination location.
DMAC_CHx_CURR	Channel Current Descriptor Pointer register	This register provides the address of the memory location where the current descriptor is located. When this field is 0, there is no valid descriptor.

(table continues...)

Table 48 (continued) M-DMA registers

Register	Name	Description
DMAC_CHx_TR_CMD	Software Trigger register	When written with 1, a trigger is generated which sets 'trigger pending' (only if the channel is enabled). A read always returns a 0.
DMAC_CHx_DESCR_STATUS	Channel Descriptor Status register	This register provides the validity of other DMAC_CHx_DESCR registers.
DMAC_CHx_DESCR_CTL	Channel Descriptor Control register	This register provides the copy of DESCR_CTL field of the currently active descriptor.
DMAC_CHx_DESCR_SRC	Channel Descriptor Source register	This register provides the copy of DESCR_SRC field of the currently active descriptor.
DMAC_CHx_DESCR_DST	Channel Descriptor Destination register	This register provides the copy of DESCR_DST field of the currently active descriptor.
DMAC_CHx_DESCR_X_SIZE	Channel Descriptor X size register	This register provides the copy of DESCR_X_SIZE field of the currently active descriptor.
DMAC_CHx_DESCR_X_INCR	Channel Descriptor X Increment register	This register provides the copy of DESCR_X_INCR field of the currently active descriptor.
DMAC_CHx_DESCR_Y_SIZE	Channel Descriptor Y size register	This register provides the copy of DESCR_Y_SIZE field of the currently active descriptor.
DMAC_CHx_DESCR_Y_INCR	Channel Descriptor Y Increment register	This register provides the copy of DESCR_Y_INCR field of the currently active descriptor.
DMAC_CHx_DESCR_NEXT	Channel Descriptor Next Pointer register	This register provides the copy of DESCR_NEXT_PTR field of the currently active descriptor.
DMAC_CHx_INTR	Interrupt register	This register provides an interrupt request. There are eight possible causes that can generate an interrupt. These causes are encoded in bits 0 to 7. Software can clear these by writing to these bits.
DMAC_CHx_INTR_SET	Interrupt Set register	This register provides interrupt setting. There are eight bits (same as DMAC_CHx_INTR) and software can write 1 to any of these bits to set the corresponding INTR bit. When read, this register reflects the DMAC_CHx_INTR register.
DMAC_CHx_INTR_MASK	Interrupt Mask register	This register provides interrupt mask setting for corresponding field in DMAC_CHx_INTR register. There are eight bits (same as DMAC_CHx_INTR) and they can be selectively enabled by writing 1 to the corresponding bits.

(table continues...)

Table 48 (continued) M-DMA registers

Register	Name	Description
DMAC_CHx_INTR_MASKED	Interrupt Masked register	When read, this register reflects a bitwise AND between the corresponding DMAC_CHx_INTR and DMAC_CHx_INTR_MASK fields.

The register access size and the initial value are described in the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual.

***Note**: In DMAC_CHx, 'x' signifies the DMAC/M-DMA instance.*

8 Code flash

Code flash is a flash memory used to store programs. Code flash is a part of Infineon' eCT Flash, which is an embedded flash targeted for use in industrial applications. A common usage is as code storage for user application execution and local data storage/update for MCU-based systems in an industrial environment. The eCT flash also includes work flash, which is the flash memory to store data; for more details, see the Work flash.

8.1 Features

This section lists the features of code flash.

- Optional memory size: 4 MB and 8 MB
- Programming and erasing functions
- ECC function: 64b + 8b
- Erase sector size of 32 KB for large sector and 8 KB for small sector
- Program size: 64b, 256b, and 4096b
- Supports single bank and dual bank modes
- Supports reading while programming/erasing
- Endurance of 1 k
- Retention of 20 years

Refer to the device datasheet for more information on the erase and program times.

8.2 Configuration

8.2.1 Block diagram

Figure 36 illustrates the position of code flash.

eCT flash, which contains code flash is a part of the CPU subsystem. The Cortex®-M7 cores can access code flash via AXI. and Cortex®-M0+ core can access code flash via AHB. The CPU subsystem also has other subsystems connected with the AHB, such as DMA and Crypto.

The SROM APIs are designed for use with Arm® Cortex®-M0+ (CM0+) on XMC7000 devices. The SROM library includes APIs for flash programming and testing. The SROM APIs are executed within the Arm® CM0+ IRQ0/1 exception generated using the IPC structures.

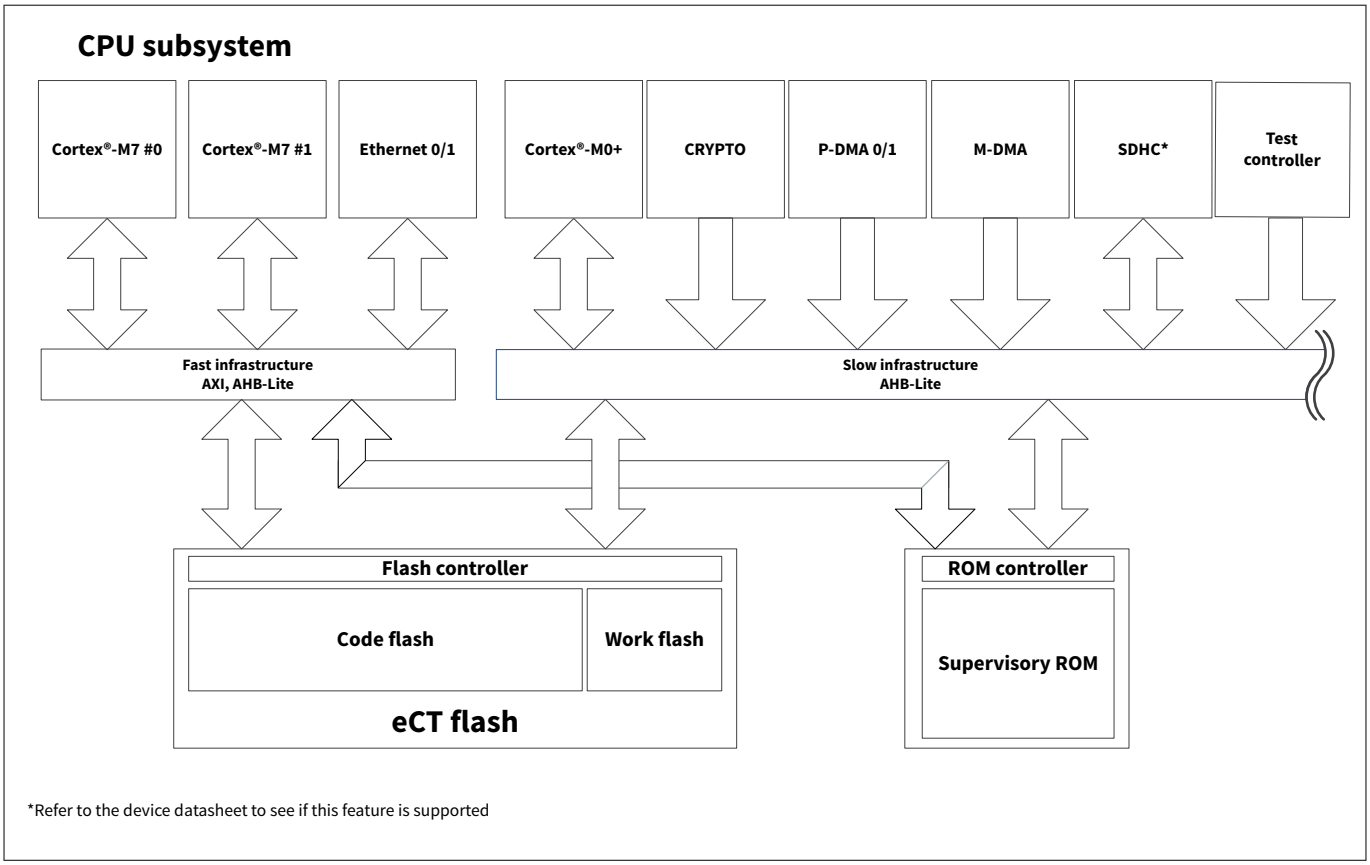


Figure 36 Position of code flash

8.2.2 Flash controller

The flash controller has multiple AHB-Lite interfaces and AXI bus interfaces:

- An AXI bus interface in the fast clock domain for Ethernet MAC 0/1
- An AHB-Lite bus interface in the slow clock domain for the CM0+ CPU.
- An AHB-Lite bus interface in the slow clock domain for Crypto.
- An AHB-Lite bus interface in the slow clock domain for P-DMA0/1.
- An AHB-Lite bus interface in the slow clock domain for DMA controller.
- An AHB-Lite bus interface in the slow clock domain for SDHC.

Note that each master has a dedicated AHB-Lite bus interface. This is unlike the ROM and SRAM controllers, where the slow bus masters are combined in the slow bus infrastructure.

This micro-architecture decision is driven by the difference in data width of the bus infrastructure (32-bit) and the flash memory (32-bit, 64-bit, 128-bit, or 256-bit): an AHB-Lite bus transfer is for a maximum of 32 bits, whereas a flash memory access always provides 32, 64, 128, or 256 bits. As flash memory accesses typically have wait states, it is beneficial to buffer or cache the complete flash memory data, rather than just selecting the requested 32 bits and discarding the rest of the flash memory data. Buffering or caching improves flash controller performance if bus transfers have temporal or spatial locality, as some bus transfers can be served from the buffer or cache (without wait states), rather than requiring a flash memory access (with wait states). However, there is no temporal or spatial locality between bus transfers from different bus masters. Therefore, a dedicated buffer or cache is required for each bus master. Hence, the dedicated AHB-Lite bus interfaces.

Typically, Crypto, DataWire, and DMA controller transfers have more locality than CPU bus transfers. The former are typically sequential in nature, whereas the latter are less sequential due to jump/branch instructions. In addition, CPU performance is more important than Crypto, DataWire, or DMA controller performance.

Therefore, Crypto, DataWire, or DMA controller transfers are supported through buffers only, whereas the CPU transfers are supported through a cache and a buffer. CPU interfaces support a cache for main interface flash memory data and a buffer for work interface memory data.

Other interfaces support a buffer for main interface flash memory data and a buffer for work interface memory data.

Figure 37 gives an overview of the flash controller micro-architecture.

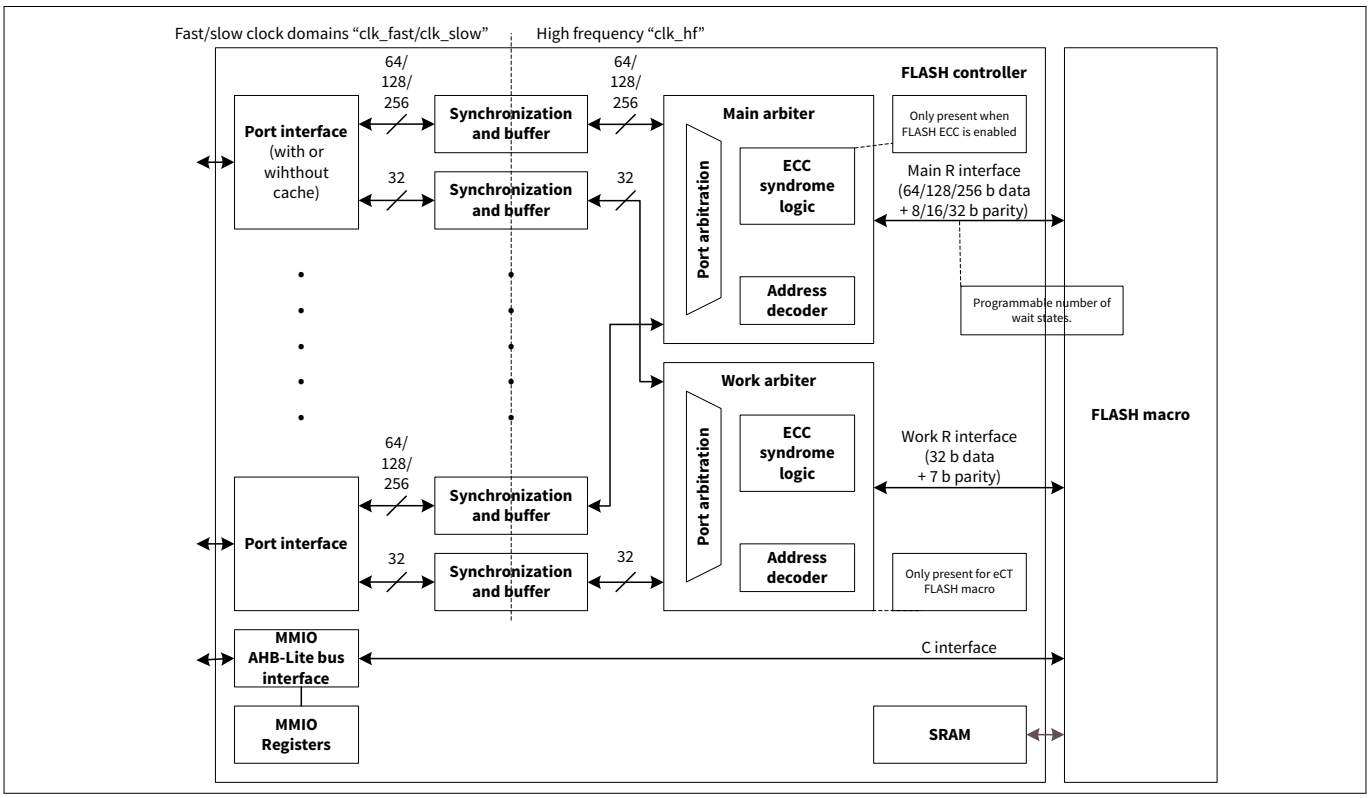


Figure 37 Flash controller

8.2.2.1 Bus error

The flash controller generates an AHB-Lite/AXI bus error under the following conditions:

- A flash macro write access.
- A flash macro read access to a logical bank that is currently being programmed/erased.
- A read access to a memory hole in the logical flash memory region. A memory hole is defined as a flash memory region address to a location that is not occupied by the code region, work region, or supervisory region.
- Non-correctable ECC error resulting from read access.

The error responses due to 2, 3, and 4 above can be suppressed by setting FLASHC_FLASH_CTL.MAIN_ERR_SILENT.

Table 49 Flash Main Error Silent Register

Register	Bit field and bit name	Description
FLASHC_FLASH_CTL	MAIN_ERR_SILENT	Specifies bus transfer behavior for a non-recoverable error on the flash macro main interface. 0: Bus transfer has a bus error 1: Bus transfer does not have a bus error; that is, the error is silent.

The errors dues to 2 and 3 above for read accesses from CPU masters are captured in FLASHC_CM0_STATUS/ FLASHC_CM7_0_STATUS/FLASHC_CM7_1_STATUS Registers.

Table 50 Flash CM0+/7_0/7_1 main status Register

Register	Bit field and bit name	Description
FLASHC_CM0_STATUS	MAIN_INTERNAL_ERR	Specifies/registers the occurrence of a flash macro main interface internal error (typically the result of a read access while a program erase operation is ongoing) as a result of a CM0+ access. Software clears this field to '0'. Hardware sets this field to '1' on a flash macro main interface internal error. Typically, software reads this field after a code section to detect the occurrence of an error. ***Note**: This field is independent of FLASHC_FLASH_CTL.MAIN_ERR_SILENT.*
FLASHC_CM7_0_STATUS	MAIN_INTERNAL_ERR	See FLASHC_CM0_STATUS.MAIN_INTERNAL_ERROR
FLASHC_CM7_1_STATUS	MAIN_INTERNAL_ERR	See FLASHC_CM0_STATUS.MAIN_INTERNAL_ERROR

8.2.2.2 Wait cycle count

If CLK_MEM is higher than the maximum operating frequency of the flash memory, it is necessary to insert wait cycles when accessing the flash memory by setting an appropriate value in FLASHC_FLASH_CTL.MAIN_WS register.

Users can set MAIN_WS according to the followings:

- FLASHC_FLASH_CTL.MAIN_WS = 0 for CLK_MEM ≤ 100 MHz
- FLASHC_FLASH_CTL.MAIN_WS = 1 for 100 MHz < CLK_MEM ≤ Fmem_max
- Fmem_max refers to the maximum frequency of CLK_MEM

A cache miss will cause more than six FLASHC_FLASH_CTL.MAIN_WS wait states inserted for CPU main flash access.

Table 51 Flash Main Wait Status Register

Register	Bit field and bit name	Description
FLASHC_FLASH_CTL	MAIN_WS	Flash macro main interface wait states: 0: 0 wait states … 15: 15 wait states

8.2.2.3 Power modes

The flash controller provides active functionality. In deep sleep power mode, the following are retained:

- Retention Registers
- Cache tag structure: Valid and Tags Registers
- Cache LRU structure
- Cache data structure: four SRAMs

Note that buffer information (in the AHB-Lite buffer interfaces and synchronization logic) are not retained. Losing buffer information after deep sleep transition has limited performance impact.

8.2.2.4 CM0+ CPU cache

***Note**: The cache in this flash macro is only for CM0+ core and not for CM7 cores because CM7 cores have their own I and D caches.*

The cache has the following features:

- 8 KB read-only capacity. This capacity provides a good hit rate for a range of benchmarks.
- Four-way set associative with an LRU replacement scheme. A four-way associative cache design provides a better hit rate than a direct mapped cache design at the same cache capacity.
- Sequential cache design. The cache tag functionality is performed before the cache data access. A sequential cache design has lower power consumption than a parallel cache design.
- 256 B line/sector, with thirty-two 8 B, sixteen 16 B, or eight 32 B subsectors each. For an 8 KB capacity, this results in a total of 32 lines distributed over eight sets. The subsector design allows for low overhead tag information, as the 16 subsectors in a line/sector share the tag and only have dedicated valid bits.

For each read transfer, the cache tag structure is evaluated before the cache data structure is accessed. The subsector design results in a relatively low number of 32 lines. The 32 associated tags are implemented in flip flops. The cache data structure is implemented using SRAM memory.

Read transfers that “hit” are processed by the cache. Read transfers that “miss” result in a flash controller access.

Each cache set has an associated 6-bit LRU field, which keeps track of the access history (from least recently used to most recently used) of the lines in the set.

Each cache line has an associated cache tag. The cache tag identifies the location of the line in system memory.

- The address bits that identify a byte in a cache line are not part of the cache tag (byte address bits 7 down to 0).
- The address bits that identify a cache set are not part of the cache tag (byte address bits 10 and 8).
- The address bits that are not part of the flash memory address (byte address bits 31 down to 27) are not part of the tag.

The above omissions of address bits result in small tags. As a result, the cache tag structure can be evaluated quickly.

In addition, the cache tag includes 16 valid bits – one valid bit for each subsector in the cache line.

Figure 38 gives an overview of the cache design.

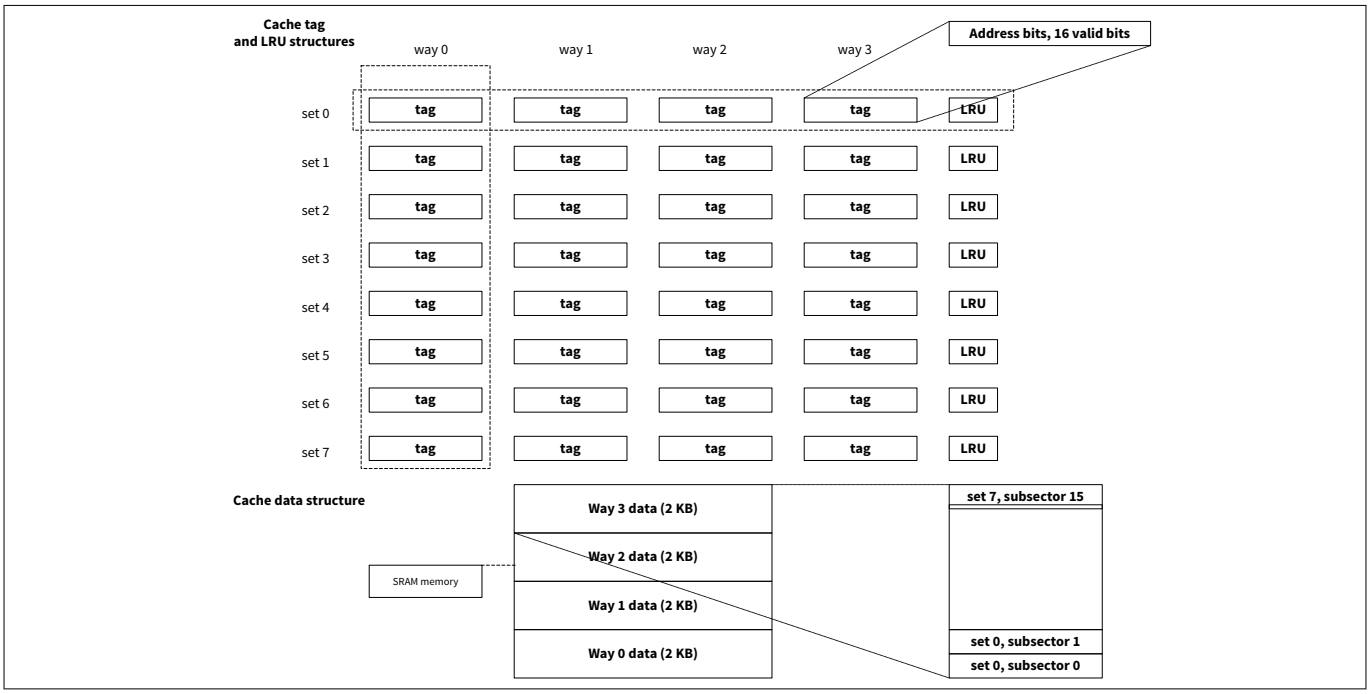


Figure 38 Cache

A cache “miss” results in a 16 B (subsector) refill. The cache data structure is updated with 16B of refilled data. Two cases are considered:

- The refilled data is a subsector of a resident cache line. Here, the data is refilled to the cache used by the resident cache line. The subsector's valid field is set to '1' (the valid fields of all other subsectors in the cache line remain unchanged).
- The refilled data is not a subsector of a resident cache line. Here, the data is refilled to the cache identified by the LRU scheme. The cache line address bits are updated, and the subsector's valid field is set to '1' (the valid fields of all other subsectors in the cache line are set to '0'). Note that this case replaces a resident cache line.

The cache has an LRU replacement scheme. Each cache set has an associated 6-bit LRU field:

- LRU[5]: '1' when Way 0 is less recently used than way 1, otherwise '0'.
- LRU[4]: '1' when Way 0 is less recently used than way 2, otherwise '0'.
- LRU[3]: '1' when Way 0 is less recently used than way 3, otherwise '0'.
- LRU[2]: '1' when Way 1 is less recently used than way 2, otherwise '0'.
- LRU[1]: '1' when Way 1 is less recently used than way 3, otherwise '0'.
- LRU[0]: '1' when Way 2 is less recently used than way 3, otherwise '0'.

Although six bits allow for 2^6 = 64-bit patterns, only 4×3×2×1 = 24-bit patterns are legal LRU representations. The LRU set information is reset to all '1' or 0b111111, representing a set in which Way 0 is less recently used than Way 1, which is less recently used than Way 2, which is less recently used than Way 3. In this case, the line in Way 0 is replaced when a new line is brought into the set. A line is made the most recently used line of its set, when it is brought into the set, or when its line data is used because of an AHB-Lite data transfer request.

Users can enable/disable the cache through CM0_CA_CTL.CA_EN.

Table 52 Flash Cache Enable Registers

Register	Bit field and bit name	Description
FLASHC_CM0_CA_CTL0	CA_EN	Cache enable: 0: Disabled 1: Enabled

When the cache is disabled, the cache tag valid bits are reset to '0's and the cache LRU information is set to '1's (making way 0 the LRU way and way 3 the MRU way).

The cache supports prefetching through FLASHC_CM0_CA_CTL0.PREF_EN.

Table 53 Flash Cache Prefetch Enable Registers

Register	Bit field and bit name	Description
FLASHC_CM0_CA_CTL0	PREF_EN	Prefetch enable: 0: Disabled 1: Enabled

If prefetch is enabled, a cache miss results in a 16 B (subsector) refill for the missing data and a 16 B prefetch for the next sequential data (independent of whether this data is already in the cache). The data of the 16 B prefetch is stored in a temporary buffer and only copied into the cache when a future read transfer “misses” in the cache and requires the buffered data.

For debug purposes, the tag and 16 valid bits of a cache line are readable through registers. The LRU information of a cache set is readable through registers.

8.2.2.5 Code flash ECC

The flash controller supports error correcting code (ECC) for the flash and cache SRAM memories. It can be enabled or disabled using the FLASHC_FLASH_CTL.MAIN_ECC_EN Register field.

Table 54 Flash ECC Enable Registers

Register	Bit field and bit name	Description
FLASHC_FLASH_CTL	MAIN_ECC_EN	Enable ECC checking for flash main interface: 0: Disabled. No correctable or non-correctable faults are reported. 1: Enabled

Figure 39 shows an overview of the data path of the flash ECC.

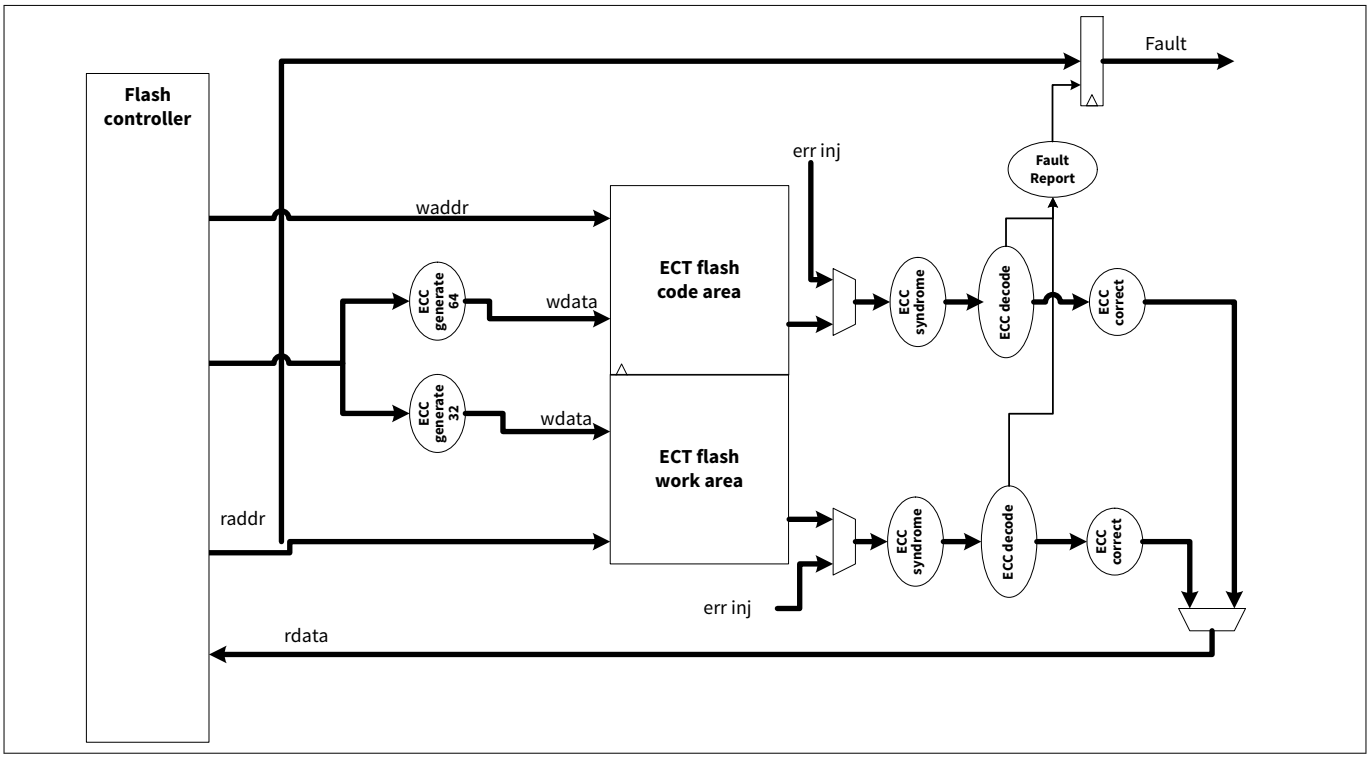


Figure 39 Data path overview for flash ECC

ECC protection is added to the flash for functional safety. The ECC implements a single error correction, dual error detection (SECDED) scheme. In the flash code, 64 bits of data are covered by eight ECC bits.

Single-bit error correction is done in-line, without the need to stall the data returning to the flash controller. Adding the delay to correct single-bit errors to the read path will not have a significant effect on performance, due to flash cache. Flash read delay is long enough and ECC time will not be a significant portion of the read delay.

Programming code flash

Code flash is 64 bits wide, and supports the 64-bit, 256-bit, and 4096-bit program.

For a 64-bit or larger program:

1. The first data input is stored in a write buffer and not written to flash.
2. When the last data input arrives, ECC is calculated, including the buffered data; 64-bit data + 8-bit ECC are programmed to flash.

ECC (single-bit errors)

When the ECC logic detects a single-bit error, the error bit can be corrected. The error correction is in-line, so corrected data will be returned with no delay. The fault is reported through the regular fault structure.

ECC uncorrectable errors

If the ECC logic detects that the data has more than one bit wrong then the data cannot be corrected. In this case, the flash does not return the data to the CPU, and instead returns a bus error response. The fault handler is responsible for recovering from uncorrectable errors.

Fault reporting

Both the correctable and non-correctable ECC errors are reported to the central fault structure in the same way.

All data correction and recovery are left to the ISR. There is no hardware support for writing corrected data back to flash.

Error injection

Error injection is done through FLASHC_FLASH_CTL. MAIN/WORK_ECC_INJ_EN and FLASHC_ECC_CTL.PARITY/WORD_ADDR Register fields.

Table 55 Flash ECC Error Injection Control Registers

Register	Bit field and bit name	Description
FLASHC_FLASH_CTL	MAIN_ECC_INJ_EN	Enable error injection for flash main interface. 0: Disabled 1: Enabled When enabled, the parity bit (FLASHC_ECC_CTL.PARITY [31:24]) is used to load from the FLASHC_ECC_CTL.WORD_ADDR word address.
FLASHC_ECC_CTL	WORD_ADDR	Specifies the word address where an error will be injected. For flash main interface ECC, WORD_ADDR is device address A [26:3]. Device address A is defined as follows: A[31:27] = b'00010 A[26:3] = WORD_ADDER A[2:0] = b'000 On a flash main interface read and when FLASHC_FLASH_CTL.MAIN_ECC_INJ_EN bit is '1', PARITY replaces the flash macro parity.
FLASHC_ECC_CTL	PARITY	Specifies the ECC parity to use for ECC error injection at WORD_ADDR. For flash main interface ECC, the 8-bit parity (PARITY) is for a 64-bit word.

When error injection is enabled, the read address is compared to the device address A. If they are equal, the data read from flash is replaced with the parity register value.

It allows testing of the error recovery routines without continuous interrupts, as every flash read causes an error.

8.2.2.6 Software generating code flash ECC

This section describes an algorithm to generate the correct ECC parity value with software. Note that this algorithm is not implemented in the hardware. Because the actual algorithm is optimized for hardware performance, it is different from the software algorithm described in this section.

"Value" in this algorithm represents the code flash 64-bit data value.

```
CW = 0x0000_0000_0000_0108_0000_0000_0000_0000 | Value
ECC_P0 = 0x01bf_bb75_be3a_72dc_4484_4a88_952a_ad5b
ECC_P1 = 0x02df_76f9_dd99_b971_1108_9311_26b3_366d
ECC_P2 = 0x04ef_cf9f_9ad5_ce97_0611_1c22_38c3_c78e
ECC_P3 = 0x08f7_ecf6_ed67_4e6c_9821_e043_c0fc_07f0
ECC_P4 = 0x10fb_7baf_6ba6_b5a6_e03e_007c_00ff_f800
ECC_P5 = 0x20fd_b7ce_f36c_ab5b_ffc0_007f_ff00_0000
ECC_P6 = 0x40fe_dd7b_74db_55ab_ffff_ff80_0000_0000
ECC_P7 = 0x807f_0000_07ff_ffff_d442_2584_4ba6_5cb7

parity[0] = ^ (CW & ECC_P0)
parity[1] = ^ (CW & ECC_P1)
...
parity[7] = ^ (CW & ECC_P7)
```

***Note**: '^' means reduction XOR. For example, ^(4'b0011) = 0^0^1^1.*

8.2.2.7 Cache ECC

The flash controller supports Error Correcting Code (ECC) for the cache SRAM memories. It can be enabled or disabled through the FLASHC_CM0_CA_CTL0.RAM_ECC_EN Register fields.

***Note**: The cache controller does not generate an AHB-Lite bus error even if uncorrectable errors were detected.*

Table 56 Flash Cache ECC Enable Registers

Register	Bit field and bit name	Description
FLASHC_CM0_CA_CTL0	RAM_ECC_EN	Enable ECC checking for cache accesses: '0': Disabled '1': Enabled

Error injection

The cache SRAM memory ECC uses 7-bit SECDED parity for each 32-bit data. The cache SRAM ECC supports the error injection through FLASHC_CM0_CA_CTL0.RAM_ECC_INJ_EN and FLASHC_ECC_CTL.PARITY/WORD_ADDR register fields: on a fetch of flash memory data to the cache, the parity for a specific 32-bit word can be injected.

Table 57 Flash Cache ECC Error Injection Control Registers

Register	Bit field and bit name	Description
FLASHC_CM0_CA_CTL0	RAM_ECC_INJ_EN	Enable error injection for cache. '0': Disabled '1': Enabled When enabled, the parity (FLASHC_ECC_CTL.PARITY) is used when a refill is done from the flash macro to the FLASHC_ECC_CTL.WORD_ADDR word address.

(table continues...)

Table 57 (continued) Flash Cache ECC Error Injection Control Registers

Register	Bit field and bit name	Description
FLASHC_ECC_CTL	WORD_ADDR	Specifies the word address where an error will be injected. For cache SRAM ECC, WORD_ADDR is device address A [25:2]. Device address A is defined as follows: A[31:26] = b'000100 A[25:2] = WORD_ADDR A[1:0] = b'00 On a read from the code flash and CM0_CA_CTL.RAM_ECC_INJ_EN bit is '1', the parity (PARITY [6:0]) is injected and stored in the cache.
FLASHC_ECC_CTL	PARITY	Specifies the ECC parity to use for ECC error injection at WORD_ADDR. For cache SRAM ECC, the 7-bit parity is for a 32-bit word. The least significant 7 bits of PARITY will represent the 7-bit parity and the remaining parity bits are ignored.

8.2.2.8 Software generating cache ECC

This section describes an algorithm to generate the correct ECC parity value with software. Note that this algorithm is not implemented in the hardware. Because the actual algorithm is optimized for hardware performance, it is different from the software algorithm described in this section.

“Value” in this algorithm represents the code flash 32-bit data value to be fetched to the cache.

```
CW = 0x0000_0000_0000_0000 | Value
ECC_P0 = 0x037f_36db_2254_2aab
ECC_P1 = 0x05bd_eb5a_4499_4d35
ECC_P2 = 0x09dd_dcee_08e2_71c6
ECC_P3 = 0x11ee_bba9_8f03_81f8
ECC_P4 = 0x21f6_d775_f003_fe00
ECC_P5 = 0x41fb_6db4_fffc_0000
ECC_P6 = 0x8103_fff8_112c_965f

parity[0] = ^ (CW & ECC_P0)
parity[1] = ^ (CW & ECC_P1)
...
parity[6] = ^ (CW & ECC_P6)
```

***Note**: ‘^’ means reduction XOR. For example, ^(4'b0011) = 0^0^1^1.*

8.2.3 Flash geometry

8.2.3.1 Interface, regions, and type of use

eCT flash is divided into work flash and code flash.

The top sectors in code flash are assigned as supervisory region and other sectors are assigned as code region. All sectors in work flash are assigned as work region.

The supervisory area is used to store trim parameters, system configuration parameters, protection and security settings, boot scripts, and other Infineon proprietary information. Read access to this region is permitted, but program/erase access is prohibited. Code region is the memory field to store program code flash. Work region is the memory field to store data.

Note that although supervisory region is located in code flash and it is contiguous with code region physically, the memory address of supervisory region is separated from code region. Work region is located between them as shown in Figure 40.

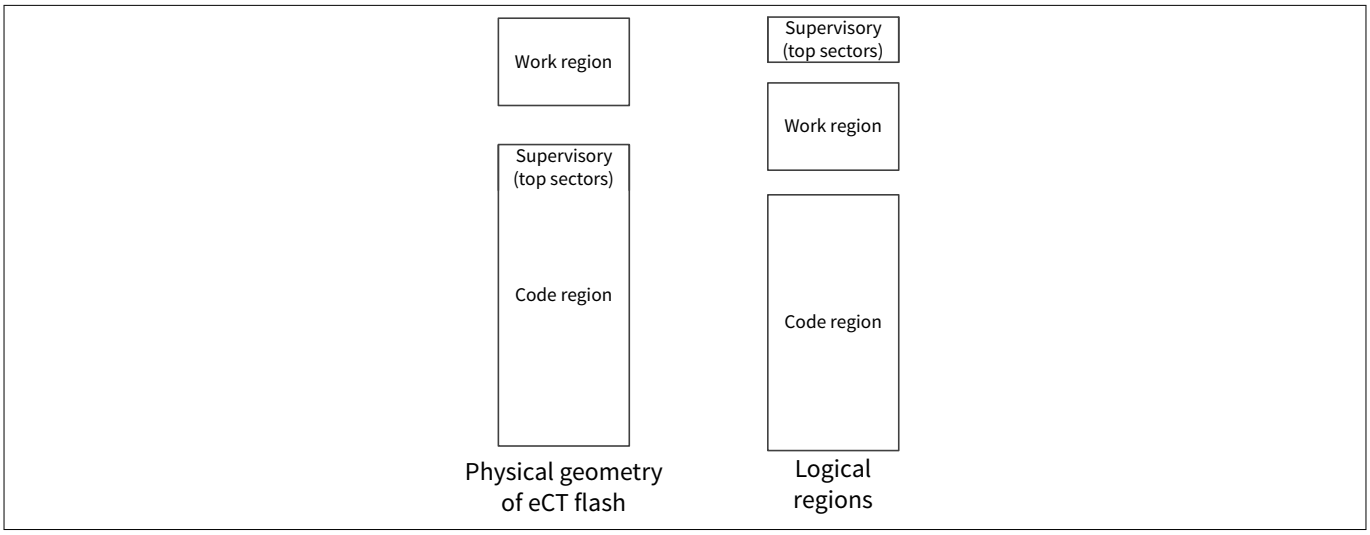


Figure 40 Regions of eCT flash

8.2.3.2 Geometries

eCT code sectors are composed of some memory units.

- Program word: This is the unit of program. It is the smallest unit of code flash, including 64 bits for data and 8 bits for ECC.
- Read word: This is the unit of read. It is composed of four units of program word.
- Page: This is composed of 16 units of read word.
- Erase sector: This is the unit of erase, which has the following types:
  - Large sector – composed of 64 pages.
  - Small sector – composed of 16 pages.

Figure 41 shows the geometries for code flash.

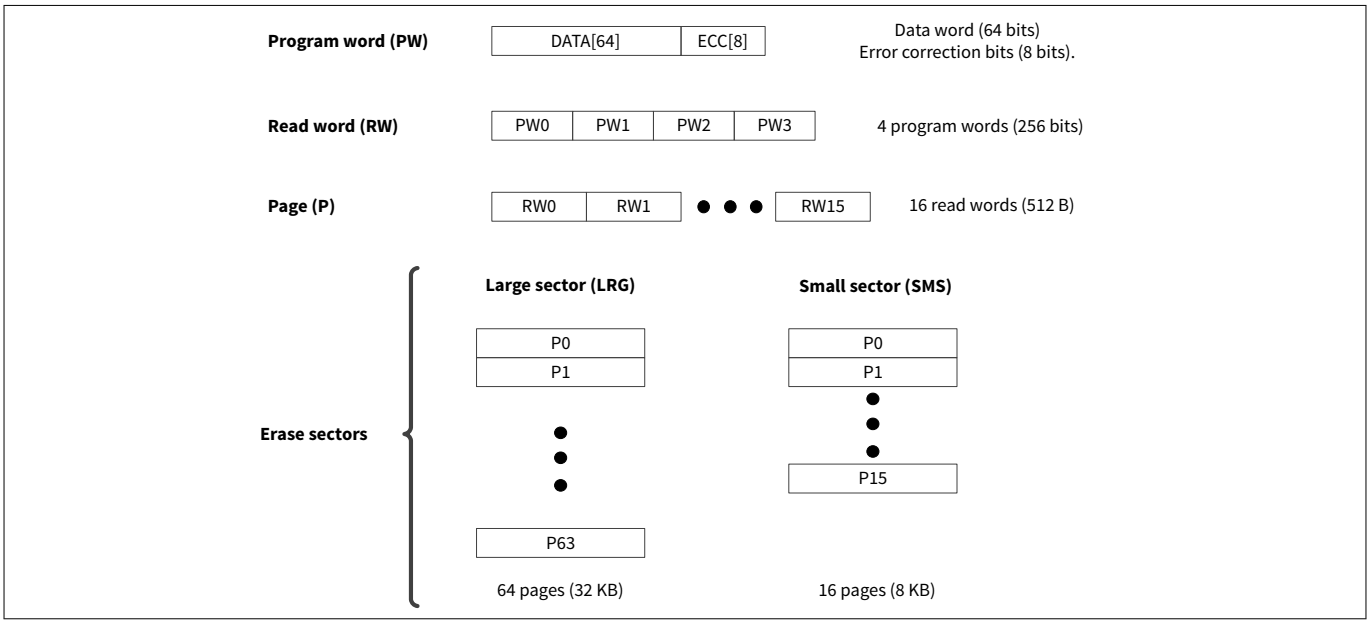


Figure 41 Code flash sector organization

Figure 42 shows the code flash arrays for each memory size. Note that upper two large sectors belong to supervisory region and do not count for code programming.

***Note**: "LRG" refers to large sector and SMS refers to small sector.*

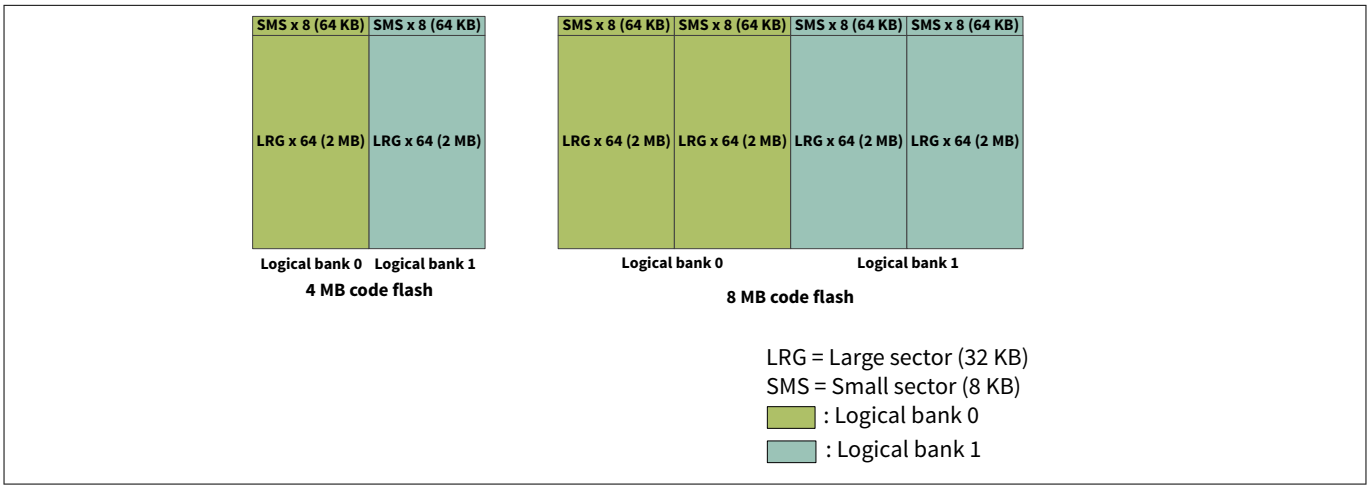


Figure 42 Code flash array organization

8.2.3.3 Logical bank

This flash memory controller has the dual bank mode feature. When using dual bank mode, flash memory region is split into two half banks. One is called logical bank 0 and the other is called logical bank 1. Flash memory always has two logical banks regardless of its size. Figure 42 shows an illustration of the logical bank. See Over-the-air (OTA) support for details about dual bank mode.

8.2.4 Over-the-air (OTA) support

OTA indicates that the flash macro supports a Read While Write operation on the same flash (code or work). OTA is possible on a logical bank resolution. This means a write can be done on one logical bank and a read can be done from any of the other logical banks in the non-write logical bank. If the read is done from the same logical bank, it will result in an error. In addition, a parallel read from the non-accessed logical bank can be performed.

8.2.4.1 Dual bank mode and remap functionality

The main flash region supports dual bank mode. The user can select the mode through FLASHC_FLASH_CTL.MAIN_BANK_MODE.

Table 58 Flash Main Bank Mode Register

Register	Bit field and bit name	Description
FLASHC_FLASH_CTL	MAIN_BANK_MODE	Specifies bank mode of flash macro main array. 0: Single bank mode 1: Dual bank mode

This is to support OTA updates of the software image in flash memory. For example, the CPU executes from a current software image in the lower sectors while the higher sectors are programmed with a new software image. When the CPU reboots, the user code changes the MAIN_MAP field, such that the CPU executed from the new image is on the higher sectors.

The hardware remap functionality only affects the read flash region access path; it does not affect the write/program flash access path. The device SROM flash management APIs will perform all necessary address conversions; users do not have to consider this read/write address mismatch.

These address maps are configurable to support bank swapping as follows:

- When configuring single bank mode, the entire code and supervisory logical regions are mapped as a single contiguous address region, starting with all large sectors, followed by all small sectors.
- When configuring dual bank mode, these logical regions are split into two halves each, and each half is presented as a separate address region. Furthermore, these halves can be swapped, to support same-location firmware upgrades.
  - Choosing mapping A will present the first half in the lower region and the second half in the upper region.
  - Choosing mapping B will present the first half in the upper region and the second half in the lower region.

Users can select mapping mode through FLASHC_FLASH_CTL.MAIN_MAP.

Table 59 Flash Main Remap Register

Register	Bit field and bit name	Description
FLASHC_FLASH_CTL	MAIN_MAP	Specifies remapping of flash macro main region. 0: Mapping A 1: Mapping B This field is only used when MAIN_BANK_MODE is '1' (dual bank mode).

Address mappings for each of the six supported code flash densities are shown in the following sections.

8.2.4.2 Address mapping for 4 MB memory

The code region has 126 large sectors of 32 KB and 16 small sectors of 8 KB.



Figure 43 Code flash memory mapping (4 MB)

8.2.4.3 Address mapping for 8 MB memory

The code region has 254 large sectors of 32 KB and 32 small sectors of 8 KB.



Figure 44 Code flash memory mapping (8 MB)

8.3 Operation

Typically, APIs that are pre-installed in the SROM are used to operate the eCT Flash.

8.3.1 SROM APIs

See SROM API library for details. To execute the SROM APIs, it is recommended to use the core M0+ through inter-processor communication. See the Inter-processor communication for details.

ROM APIs related to code flash operation is listed in Table 60.

Table 60 SROM APIs for flash operation

SROM API	Description
Program row	Programs the addressed FLASH page
Erase all	Erases all FLASH
Erase sector	Erases the addressed FLASH sector
Erase suspend	Suspends ongoing erase operation
Erase resume	Resumes an erase suspend operation

Note:
- *Reprogramming previously programmed words is not allowed without first erasing the sector. If the data value to be reprogrammed is the same as the value of programmed data words, reprogramming is permitted.*
- *The flash state will be unknown if reset/power-down occurs during program/erase. Erase the area because it may contain garbage data.*

8.4 Registers

The following register map shows the various register definitions and its functionality.

Table 61 FLASHC Registers

Offset	Width	Name	Description
0x0000	32	FLASHC_FLASH_CTL	Control
0x0004	32	FLASHC_FLASH_PWR_CTL	Flash power control
0x0008	32	FLASHC_FLASH_CMD	Command
0x02a0	32	FLASHC_ECC_CTL	ECC control
0x0400	32	FLASHC_CM0_CA_CTL0	CM0+ cache control
0x0404	32	FLASHC_CM0_CA_CTL1	CM0+ cache control
0x0408	32	FLASHC_CM0_CA_CTL2	CM0+ cache control
0x0440	32	FLASHC_CM0_CA_STATUS0	CM0+ cache status 0
0x0444	32	FLASHC_CM0_CA_STATUS1	CM0+ cache status 1
0x0448	32	FLASHC_CM0_CA_STATUS2	CM0+ cache status 2
0x0460	32	FLASHC_CM0_STATUS	CM0+ interface status
0x04e0	32	FLASHC_CM7_0_STATUS	CM7 #0 interface status

(table continues...)

Table 61 (continued) FLASHC Registers

Offset	Width	Name	Description
0x0560	32	FLASHC_CM7_1_STATUS	CM7 #1 interface status
0x0580	32	FLASHC_CRYPTO_BUFF_CTL	Cryptography buffer control
0x0600	32	FLASHC_DW0_BUFF_CTL	Datawire 0 buffer control
0x0680	32	FLASHC_DW1_BUFF_CTL	Datawire 1 buffer control
0x0700	32	FLASHC_DMAC_BUFF_CTL	DMA controller buffer control
0x0780	32	FLASHC_SLOW0_MS_BUFF_CTL	Slow external master 0 buffer control

Table 62 FM_CTL_ETC Registers

Offset	Width	Name	Description
0x0400	32	FLASHC_MAIN_FLASH_SAFETY	Main (code) flash security enable
0x0404	32	FLASHC_STATUS	Status read from flash macro
0x0500	32	FLASHC_WORK_FLASH_SAFETY	Work flash security enable

9 Work flash

Work flash is a flash memory used to store data. Work flash is a part of Infineon eCT flash, which is an embedded flash targeted for use in industrial applications. A common usage is as local data storage/update for MCU-based systems in an industrial environment. The eCT flash also includes code flash, which is the flash memory to store programs; for more details, see the Code flash.

9.1 Features

This section lists the features of work flash.

- Optional memory size: 256 KB
- Programming and erasing functions
- ECC function: 32 b + 7 b
- Erase sector size is 2 KB for large sector and 128 B for small sector
- Program size: 32 b
- Supports single bank and dual bank modes
- Supports reading while programming/erasing
- Supports differential sensing architecture.
- Endurance of 250 k
- Retention of 10 years

Refer to the device datasheet for more information on the erase and program times.

9.2 Configuration

9.2.1 Block diagram

Figure 45 illustrates the position of work flash.

eCT Flash, which contains work flash is a part of the CPU subsystem. The Cortex®-M7 cores can access work flash via AXI, and Cortex®-M0+ core can access work flash via AHB. The CPU subsystem also has other subsystems connected with the AHB, such as DMA and Crypto.

The SROM APIs are designed for use with Arm® Cortex®-M0+ (CM0+) on XMC7000 devices. The SROM library includes APIs for flash programming and testing. The SROM APIs are executed within the Arm® CM0+ IRQ0/1 exception generated using the IPC structures.

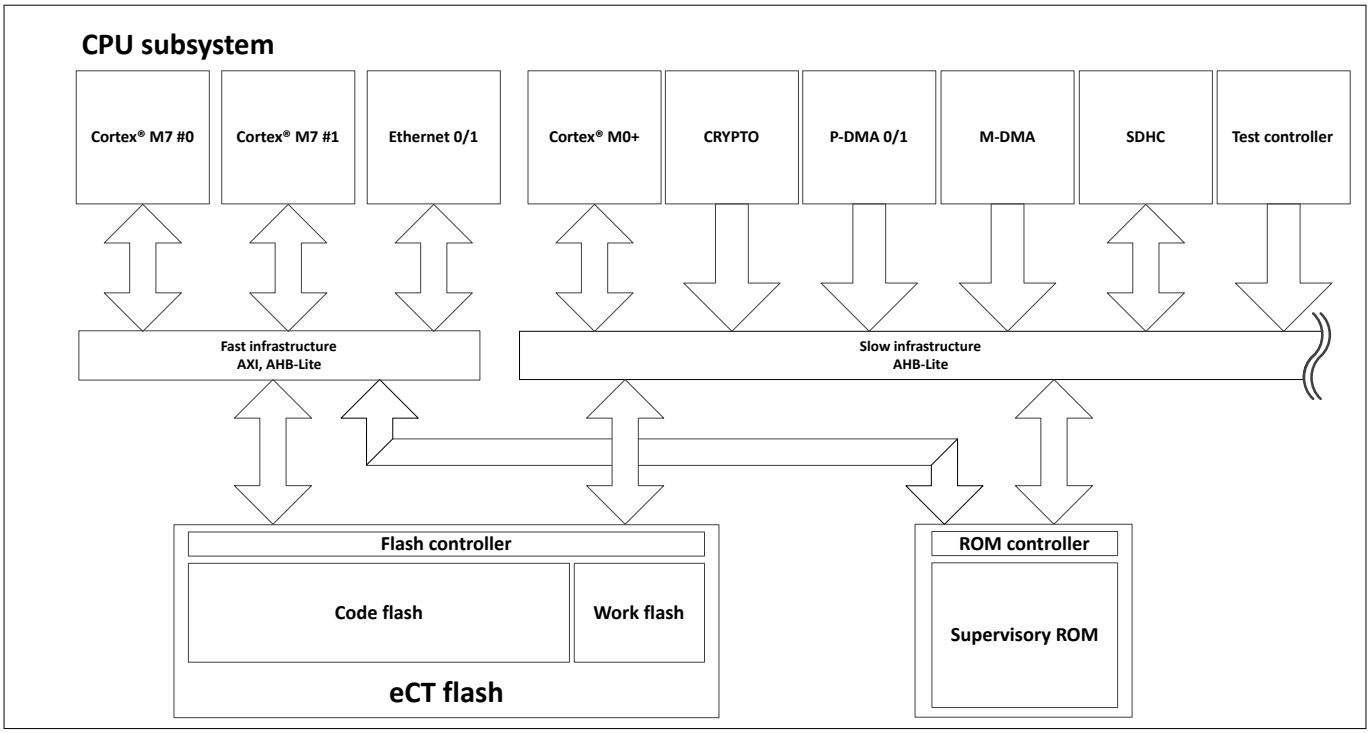


Figure 45 Position of work flash

9.2.2 Flash controller

Refer to Flash controller.

9.2.2.1 Bus error

The flash controller generates an AHB-Lite/AXI bus error under the following conditions:

- A flash macro write access.
- A flash macro read access to a logical bank that is currently being programmed/erased.
- A read access to a memory hole in the logical flash memory region. A memory hole is defined as a flash memory region address to a location that is not occupied by the code region, work region, or supervisory region.
- Non-correctable ECC error resulting from read access.

The error responses due to 2, 3, and 4 above can be suppressed by setting FLASHC_FLASH_CTL.WORK_ERR_SILENT.

Table 63 Flash Work Error Silent register

Register	Bit field and bit name	Description
FLASHC_FLASH_CTL	WORK_ERR_SILENT	Specifies bus transfer behavior for a non-recoverable error on the flash macro work interface. 0: Bus transfer has a bus error 1: Bus transfer does not have a bus error; that is, the error is silent.

The errors due to 2 and 3 for read accesses from CPU masters are captured in the FLASHC_CM0_STATUS/FLASHC_CM7_0_STATUS/FLASHC_CM7_1_STATUS registers.

Table 64 Flash CM0+/7_0/7_1 Work Status register

Register	Bit field and bit name	Description
FLASHC_CM0_STATUS	WORK_INTERNAL_ERR	Specifies the occurrence of a flash macro work interface internal error (typically the result of a read access while a program erase operation is ongoing) as a result of a CM0+ access. Software clears this field to '0'. Hardware sets this field to '1' on a flash macro work interface internal error. Typically, software reads this field after a work section to detect the occurrence of an error. ***Note**: This field is independent of FLASHC_FLASH_CTL.WORK_ERR_SILENT.*
FLASHC_CM7_0_STATUS	WORK_INTERNAL_ERR	See FLASHC_CM0_STATUS.WORK_INTERNAL_ERROR
FLASHC_CM7_1_STATUS	WORK_INTERNAL_ERR	See FLASHC_CM0_STATUS.WORK_INTERNAL_ERROR

9.2.2.2 Work flash ECC

The flash controller supports error correcting code (ECC) for the work flash. It can be enabled or disabled using the FLASHC_FLASH_CTL.WORK_ECC_EN register field.

Table 65 Flash ECC Enable registers

Register	Bit field and bit name	Description
FLASHC_FLASH_CTL	WORK_ECC_EN	Enable ECC checking for flash work interface: 0: Disabled. No correctable or non-correctable faults are reported. 1: Enabled

Refer to Figure 39 for an overview of the flash ECC data path.

ECC protection is added to the flash for functional safety. The ECC implements a single error correction, dual error detection (SECDED) scheme. The flash work area has 32-bit data, covered by seven ECC bits.

- ECC (single-bit errors)

Refer to ECC (single-bit errors) for details.

- ECC uncorrectable errors

Refer to ECC uncorrectable errors for details.

- Fault reporting

Refer to Fault reporting for details.

- Error injection

Error injection is done through the WORK_ECC_INJ_EN and FLASHC_ECC_CTL.PARITY/WORD_ADDR Register fields.

Table 66 Flash ECC Error Injection Control registers

Register	Bit field and bit name	Description
FLASHC_FLASH_CTL	WORK_ECC_INJ_EN	Enable error injection for flash work interface. 0: Disabled 1: Enabled When enabled, the parity bit (FLASHC_ECC_CTL.PARITY) is used to load from the FLASHC_ECC_CTL.WORD_ADDR word address.
FLASHC_ECC_CTL	WORD_ADDR	Specifies the word address where an error will be injected. For flash work interface ECC, WORD_ADDR is device address A [25:2]. Device address A is defined as follows: A[31:26] = b'000101 A[25:2] = WORD_ADDR A[1:0] = b'00 On a flash work interface read and when FLASHC_FLASH_CTL.WORK_ECC_INJ_EN bit is '1', PARITY replaces the flash macro parity.
FLASHC_ECC_CTL	PARITY	Specifies the ECC parity to use for ECC error injection at WORD_ADDR. For flash work interface ECC, the 7-bit parity is for a 32-bit word. The least significant 7 bits of PARITY will represent the 7-bit parity and the remaining parity bits are ignored.

When error injection is enabled, the read address is compared to device address A. If they are equal, the data read from flash is replaced with the parity register value.

It allows testing of the error recovery routines without continuous interrupts, as every flash read causes an error.

9.2.2.3 Software generating work flash ECC

This section describes an algorithm to generate the correct ECC parity value with software. Note that this algorithm is not implemented in the hardware. Because the actual algorithm is optimized for hardware performance, it is different from the software algorithm described in this section.

“Value” in the algorithm represents work flash 32-bit data value.

```
CW = 0x0000_0007_0000_0000 | Value
ECC_P0 = 0x037f_36db_2254_2aab
ECC_P1 = 0x05bd_eb5a_4499_4d35
ECC_P2 = 0x09dd_dcee_08e2_71c6
ECC_P3 = 0x11ee_bba9_8f03_81f8
ECC_P4 = 0x21f6_d775_f003_fe00
ECC_P5 = 0x41fb_6db4_fffc_0000
ECC_P6 = 0x8103_fff8_112c_965f

parity[0] = ^ (CW & ECC_P0)
parity[1] = ^ (CW & ECC_P1)
...
parity[6] = ^ (CW & ECC_P6)
```

***Note**: ‘^’ means reduction XOR; for example, ^(4'b0011) = 0^0^1^1.*

9.2.3 Flash geometry

9.2.3.1 Interface, regions, and type of use

eCT flash is divided into work flash and code flash.

The top sectors in code flash are assigned as supervisory region and other sectors are assigned as code region. All sectors in work flash are assigned as work region.

The supervisory area is used to store trim parameters, system configuration parameters, protection and security settings, boot scripts, and other Infineon proprietary information. Read access to this region is permitted, but program/erase access is prohibited. Code region is the memory field to store program code. Work region is the memory field to store data.

Note that although supervisory region is located in code flash and it is contiguous with code region physically, the memory address of supervisory region is separated from the code region. Work region is located between them as shown in Figure 46.

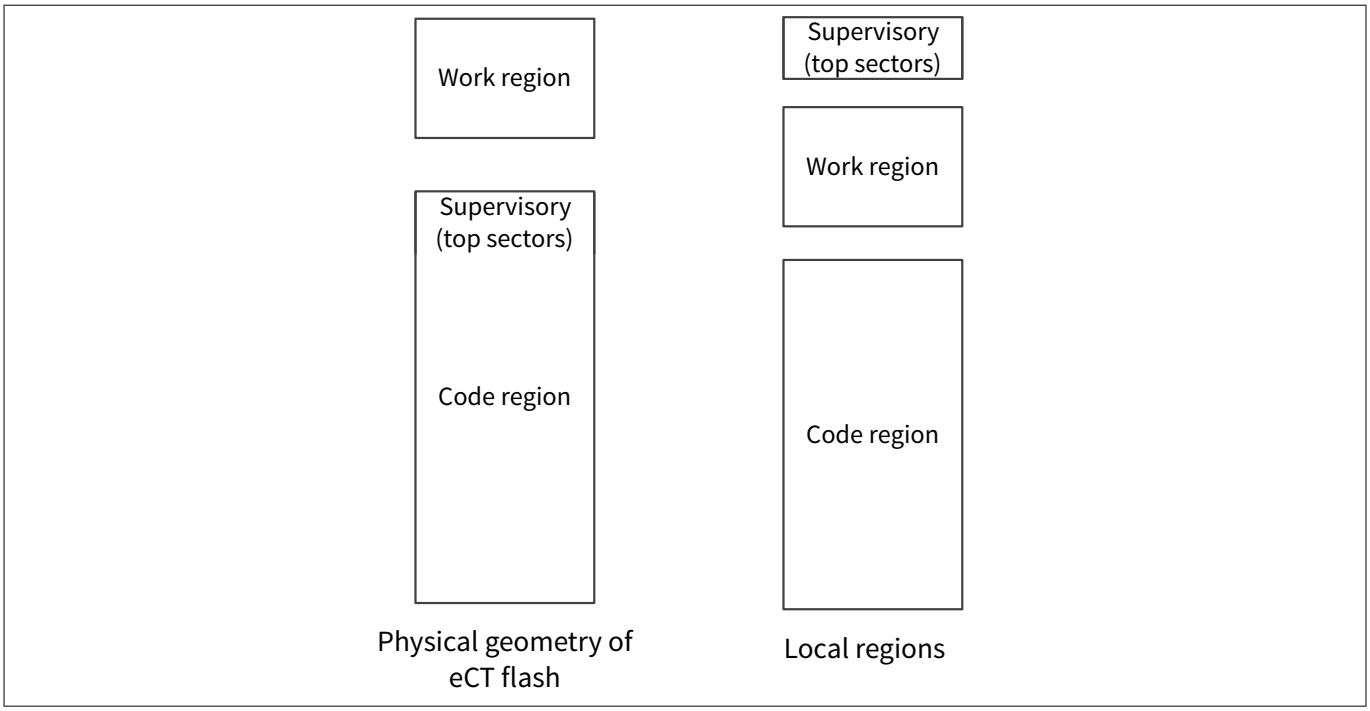


Figure 46 Regions of eCT flash

9.2.3.2 Geometries

eCT work sectors are composed of some memory units.

- Word: This is the unit of data. It is the smallest unit of work flash, including 32 bits for data and 7 bits for ECC.
- Page: This is composed of 16 units of Word (64 B, 624 bits).
- Erase sector: This is the unit of erase, which has the following types:
  - Large sector composed of 32 pages (2 KB)
  - Small sector composed of two pages (128 B)

Figure 47 shows the geometries for work flash.

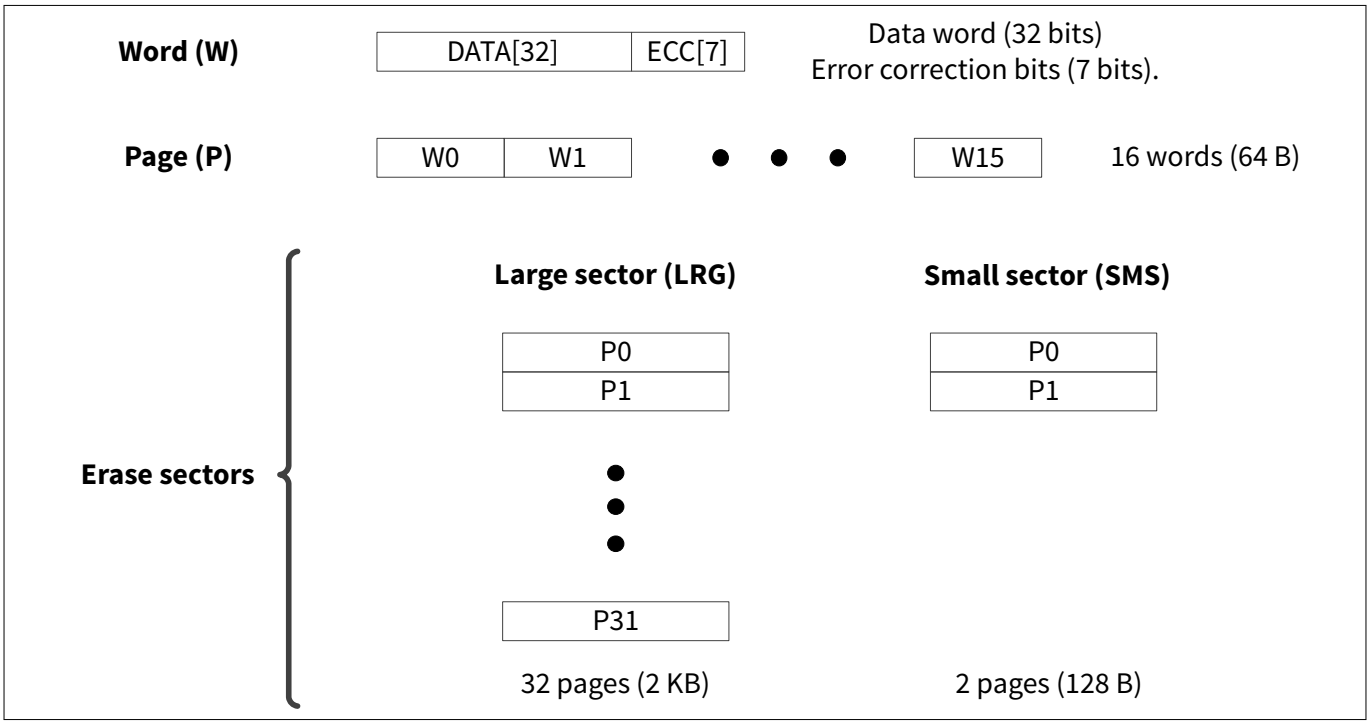


Figure 47 Work flash sector organization

Figure 48 shows the work flash arrays for each memory size. “LRG” stands for large sector and “SMS’ stands for small sector.

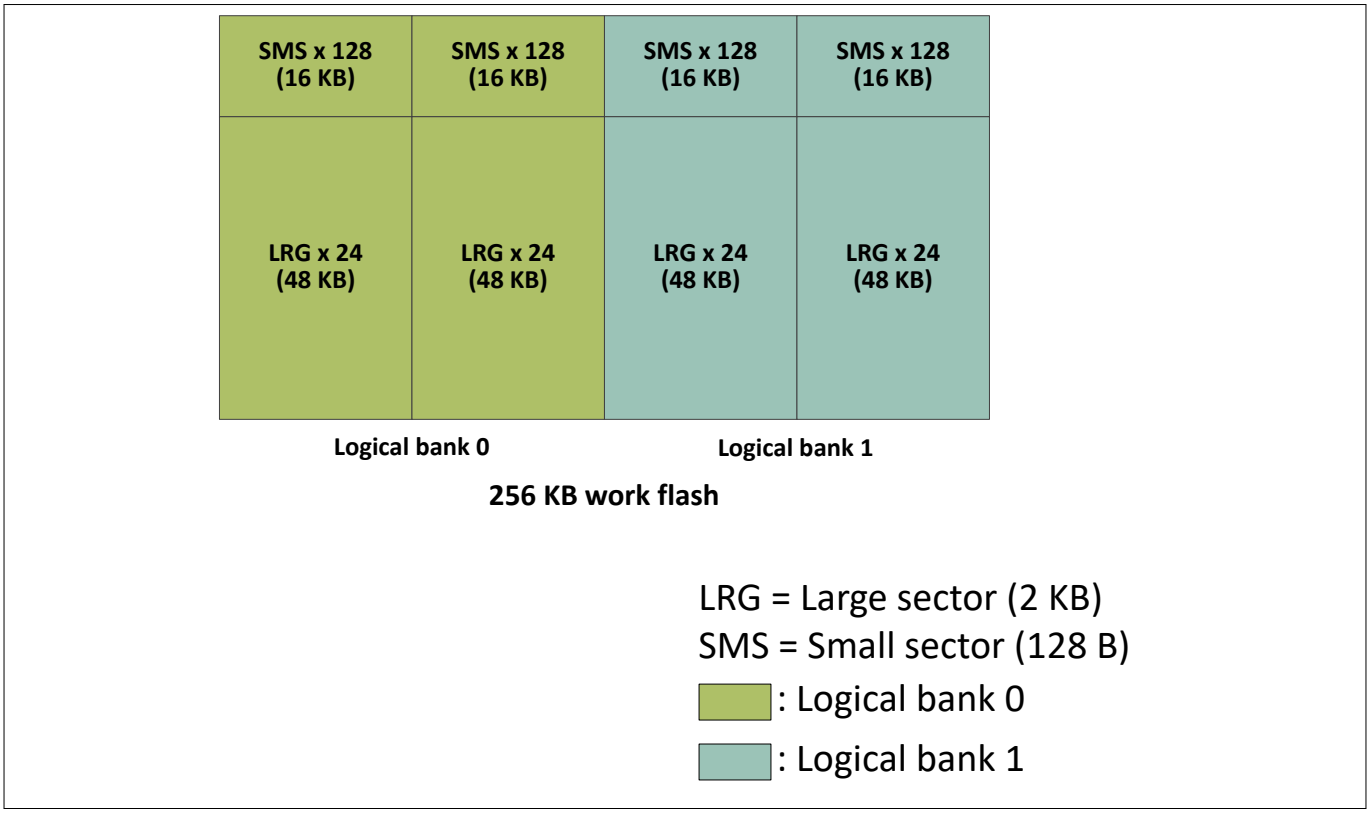


Figure 48 Work flash array organization

9.2.3.3 Logical bank

This flash memory controller has the dual bank mode feature. When using dual bank mode, flash memory region is split into two half banks. One is called logical bank 0 and the other is called logical bank 1. Flash memory always has two logical banks regardless of its size. Figure 48 shows an illustration of the logical bank. See Over-the-air (OTA) support for details about dual bank mode.

9.2.4 Over-the-air (OTA) support

In OTA, the flash macro supports a read-while-write operation on the same flash (that is, code or work). OTA is possible on a logical bank resolution. This means a write can be done on one logical bank and a read can be done from any of the other logical banks in the non-write logical bank. In case the read is done from the same logical bank, it will result in an error. In addition, a parallel read from the non-accessed logical bank can be performed.

9.2.4.1 Dual bank mode and remap functionality

The work flash region supports dual bank mode. This mode can be selected using FLASHC_FLASH_CTL.WORK_BANK_MODE.

Table 67 Flash Work Bank Mode register

Register	Bit field and bit name	Description
FLASHC_FLASH_CTL	WORK_BANK_MODE	Specifies bank mode of flash macro work array. 0: Single bank mode 1: Dual bank mode

The hardware remap functionality only affects the read flash region access path; it does not affect the write/program flash access path. The device SROM flash management APIs will perform all necessary address conversions; users do not have to consider this read/write address mismatch.

These address maps are configurable to support bank swapping as follows:

- When configuring single bank mode, the entire work region is mapped as a single contiguous address region, starting with all large sectors, followed by all small sectors.
- When configuring dual bank mode, this logical region is split into two halves, and each half is presented as a separate address region. Furthermore, these halves can be swapped to support same-location firmware upgrades.
  - Mapping A will present the first half in the lower region and the second half in the upper region.
  - Mapping B will present the first half in the upper region and the second half in the lower region.

Users can select the mapping mode using FLASHC_FLASH_CTL.WORK_MAP.

Table 68 Flash Work Remap register

Register	Bit field and bit name	Description
FLASHC_FLASH_CTL	WORK_MAP	Specifies remapping of flash macro work region. 0: Mapping A 1: Mapping B This field is only used when WORK_BANK_MODE is '1' (dual bank mode).

9.2.5 Address map of work flash

9.2.5.1 Address mapping for 128 KB memory

The work region has 48 large sectors of 2 KB and 256 small sectors of 128 B.

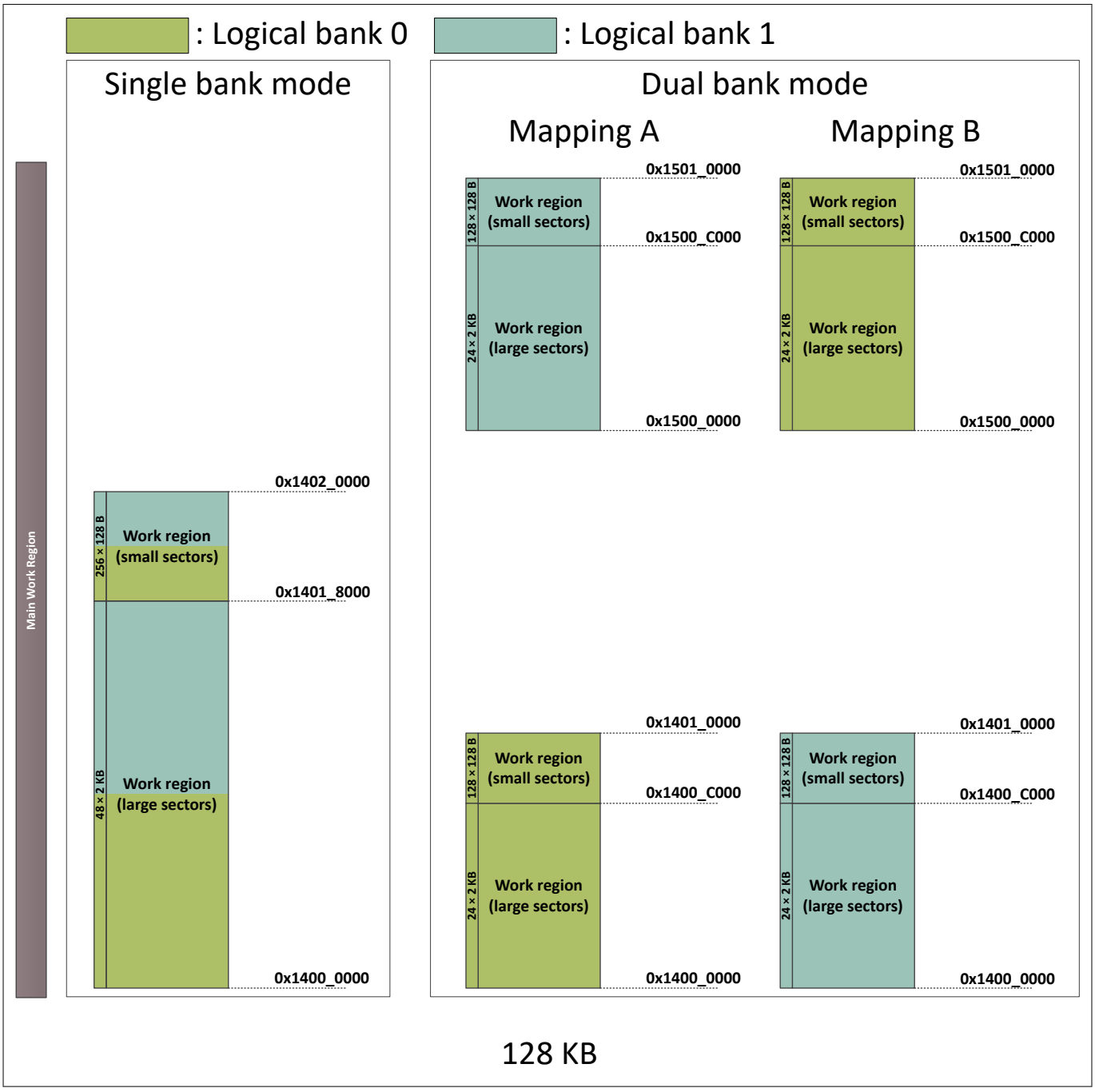


Figure 49 Work flash memory mapping (128 KB)

9.2.5.2 Address mapping for 256 KB memory

The work region has 96 large sectors of 2 KB and 512 small sectors of 128 B.

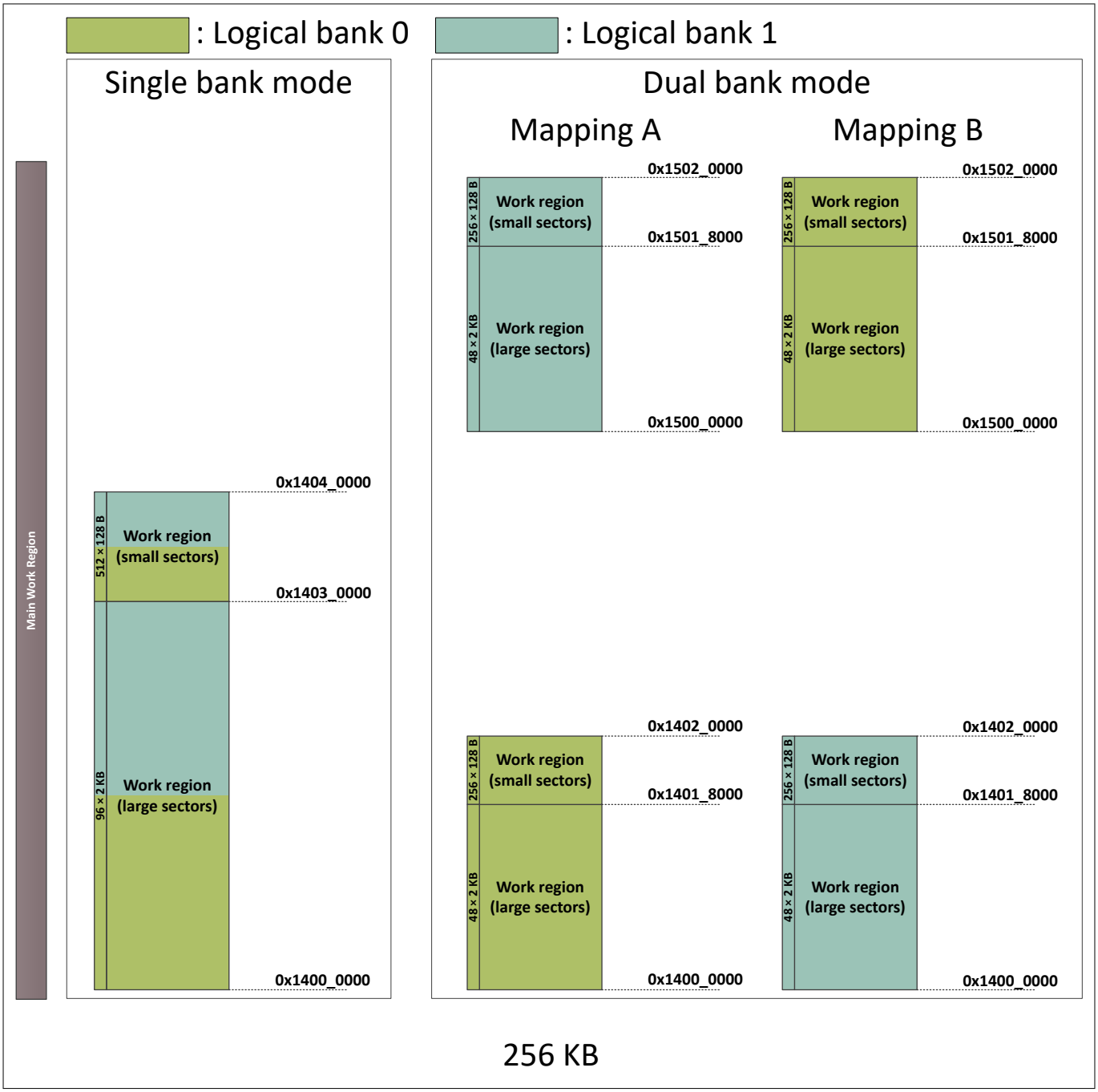


Figure 50 Work flash memory mapping (256 KB)

9.3 Operation

Typically APIs that are pre-installed in the SROM are used to operate the eCT flash. This section provides a brief summary of SROM APIs.

9.3.1 Read

There are some read restrictions due to the nature of differential flash. Normal usage of work flash is as follows:

1. Erase entire sector
2. Program words
3. Read

***Note**:*

- *Reading a word that is still in the erased state will result in random (spurious) data.*
  - *Caution: This reading will cause a ECC error.*
  - *Measures: Call the blank check SROM API before reading work flash to check whether the area is in the programmed or erased state.*
- *Work flash is always read 64-bit wide via AXI.*
  - *Caution: A read of 32-bit unit data on work flash results in 64-bit access. If the adjacent 32-bit data has not been programmed, it will result in an unexpected ECC error.*
  - *Measures: Take one of the following measures:*
    1. *Use DMA (M-DMA or P-DMA) that reads via AHB, which has a 32-bit width.*
    2. *Program data into the work flash – always aligned to 64 bits and by 64-bit units.*
- *ECC error can be notified to only one CPU via fault structure.*
  - *Caution: When multiple cores have to read from work flash, all the cores, except one core, cannot be notified about the ECC error*
  - *Measures: Take one of the following measures:*
    1. *Use DMA (M-DMA or P-DMA) to read from work flash. If a non-correctable error occurs during the DMA transmission, it will be detected and informed through one of the following DMA registers:*
       a. *M-DMA: Source bus error bit of the Interrupt register (DMAC_CHx_INTR.SRC_BUS_ERROR = 1).*
       b. *P-DMA: Interrupt cause bit of the Status register (DWx_CH_STRUCTy_CH_STATUS.INTR_CAUSE = 2). DMA does not detect correctable ECC errors.*
    2. *Assign one CPU core for non-correctable ECC error handling. This CPU core informs about the error caused.*
    3. *Set non-correctable ECC error action to reset (may not be acceptable depending to the application).*

9.3.2 SROM APIs

Refer to SROM API library for details. To execute the following SROM APIs, it is recommended to use the core M0+ through inter-processor communication. See the Inter-processor communication for details.

SROM APIs related to work flash operation are listed in Table 69.

Table 69 SROM APIs for flash operation

SROM API	Description
Program row	Programs the addressed flash page
Erase all	Erases all flash
Erase sector	Erases the addressed flash sector
Erase suspend	Suspends ongoing erase operation
Erase resume	Resumes an erase suspend operation
Blank check	Performs blank check on the addressed work flash

***Note**:*

- *Reprogramming previously programmed words is not allowed without first erasing the sector. If the data value to be reprogrammed is the same as the value of programmed data words, reprogramming is permitted.*
- *The flash state will be unknown if reset/power-down occurs during program/erase. Because it may contain garbage data, run blank check; if it is not blank, erase that area.*

9.4 Registers

The following register map shows various register definitions and its functionality.

Table 70 Registers

Offset	Width	Name	Description
0x0000	32	FLASHC_FLASH_CTL	Control
0x0004	32	FLASHC_FLASH_PWR_CTL	Flash power control
0x0008	32	FLASHC_FLASH_CMD	Command
0x02a0	32	FLASHC_ECC_CTL	ECC control
0x0400	32	FLASHC_CM0_CA_CTL0	CM0+ cache control
0x0404	32	FLASHC_CM0_CA_CTL1	CM0+ cache control
0x0408	32	FLASHC_CM0_CA_CTL2	CM0+ cache control
0x0440	32	FLASHC_CM0_CA_STATUS0	CM0+ cache status 0
0x0444	32	FLASHC_CM0_CA_STATUS1	CM0+ cache status 1
0x0448	32	FLASHC_CM0_CA_STATUS2	CM0+ cache status 2
0x0460	32	FLASHC_CM0_STATUS	CM0+ interface status
0x04e0	32	FLASHC_CM7_0_STATUS	CM7 #0 interface status
0x0560	32	FLASHC_CM7_1_STATUS	CM7 #1 interface status
0x0580	32	FLASHC_CRYPTO_BUFF_CTL	Cryptography buffer control

(table continues...)

Table 70 (continued) Registers

Offset	Width	Name	Description
0x0600	32	FLASHC_DW0_BUFF_CTL	Datawire 0 buffer control
0x0680	32	FLASHC_DW1_BUFF_CTL	Datawire 1 buffer control
0x0700	32	FLASHC_DMAC_BUFF_CTL	DMA controller buffer control
0x0780	32	FLASHC_SLOW0_MS_BUFF_CTL	Slow external master 0 buffer control

Table 71 FM_CTL_ETC registers

Offset	Width	Name	Description
0x0400	32	FLASHC_MAIN_FLASH_SAFETY	Main (code) flash security enable
0x0404	32	FLASHC_STATUS	Status read from flash macro
0x0500	32	FLASHC_WORK_FLASH_SAFETY	Work flash security enable

10 SRAM interface

SRAM controllers are implemented in the XMC7000 family device for the on-chip SRAM memory interface. RAMs are accessible by several masters connected to the fast and slow infrastructures: CPUs, peripheral-DMA (P-DMA), memory-DMA (MDMA), Crypto, and external masters. CPUs can also execute code out from these SRAMs.

10.1 Features

This section lists the features of SRAM controller.

- Optional memory size: up to 768 KB, 1024 KB
- AXI bus interfaces:
  - In the fast clock domain for the CM7 CPUs
- AHB-Lite bus interface:
  - In the slow clock domain for all bus masters (CM0+ CPU, CRYPTO, P-DMA, M-DMA, debug interface, and optional external bus master). The slow bus infrastructure combines the bus masters in the slow clock domain.
- Programmable wait states
- ECC function
  - Single-bit error correction and double-bit error detection (SECDED)
  - ECC error injection
- RAM retention function
- RAM power up delay control
  - Setting the power stabilization wait after switching on the SRAM power domain.

***Note**: The first 2 KB of SRAM is reserved and is not available for users. The first 32 KB block of SRAM0 should be in the enabled or retained state in active, LP active, sleep, LP sleep, and deep sleep modes.*

***Note**: The SRAM region from (SRAM size minus 6 KB) to (SRAM size minus 2 KB) and the SRAM region of the first word of the last 2 KB are used by Infineon firmware during boot operation. Therefore, this region is available to the user; however, data retention across resets is not guaranteed in this area, because it can be overwritten by Infineon boot firmware.*

10.2 Configuration

10.2.1 Block diagram

The SRAM controller has a 64-bit wide interface to SRAM memory. Figure 51 gives an overview of the SRAM controller.

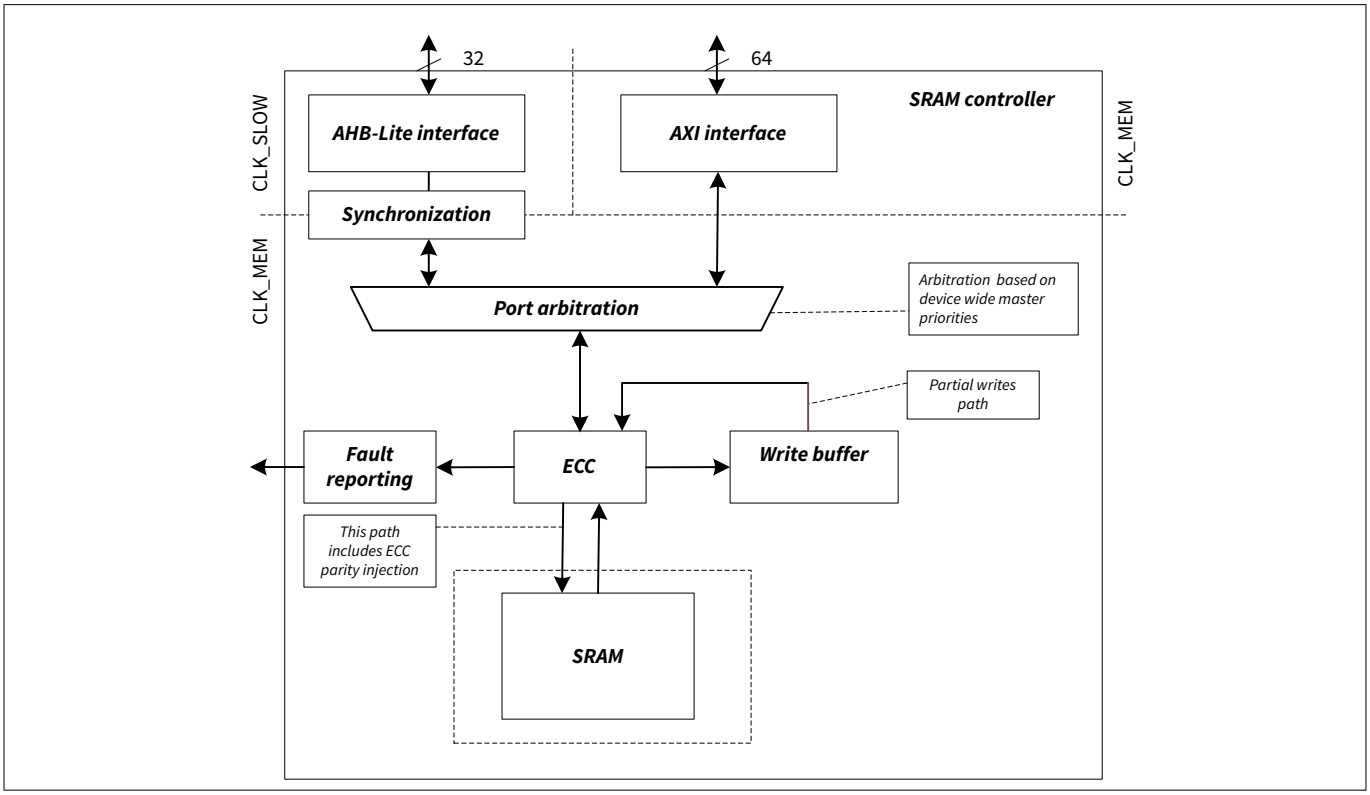


Figure 51 SRAM controller

The SRAM controller has one AXI interface and one AHB-Lite interfaces that connect to the AXI and AHB-Lite infrastructures. The AHB-Lite interface is connected to a synchronization component that translates between the interface clock (CLK_SLOW) and the high-frequency clock (CLK_MEM).

Arbitration is performed on the transfers from the two ports (AHB-Lite interface and AXI interface). Arbitration uses device wide, bus master specific priorities, and round-robin based acceptance within the same priority group. Therefore, although two bus interfaces are provided, one AHB-Lite or AXI transfer is accepted by the port arbitration component.

The SRAM controller supports Error Correcting Code (ECC) for SRAMs. This functionality can be disabled or enabled (CPUSS_RAMx_CTL0.ECC_EN). Initial value of CPUSS_RAMx_CTL0.ECC_EN is ‘1’ (ECC enabled).

- If ECC functionality is disabled (CPUSS_RAMx_CTL0.ECC_EN = '0'), Read and write transfers originating from the AHB-Lite or AXI interface are accessing SRAM directly. Note, that when ECC is disabled, no parity information is written to the RAM. All data sizes can be written to the SRAM by a single access.
- If ECC functionality is enabled (CPUSS_RAMx_CTL0.ECC_EN = '1'), an AHB-Lite or AXI transfer is translated into one or multiple SRAM accesses. Furthermore, ECC, and write buffer components are used to implement the desired functionality. Read data accesses use parity information stored beside each 64-bit word in the SRAM to correct single bit errors or to detect double bit errors. Write data accesses need to generate parity information for each 64-bit word in the SRAM. Writing full 64-bit size can be done by a single access. Writing smaller data sizes are translated into read accesses from the SRAM and combined (previous read and new partial write data) are stored into a write buffer. Such a partial write operation causes two accesses to the SRAM. Note, that partial write can achieve only half of the possible memory bandwidth.

Typically, the pending writes from the write buffer are executed when the SRAM is not accessed by the AHB-Lite or AXI ports.

10.2.2 Wait states

The SRAM controller supports programmable wait states. Dedicated wait states are provided for the fast and slow AHB-Lite bus interfaces. The programmable wait states represent the number of CLK_MEM cycles for a read path through the SRAM memory in either the fast domain (CM7 CPUs and optional AXI masters) or slow domain (CM0+ CPU, P-DMA, M-DMA, etc.).

The SRAM controller supports wait states in the range from 0 to 3. The number of wait states is expressed in CLK_MEM clock cycles.

The fast clock domain timing is closed at a higher frequency than the slow clock domain. The required number of fast wait states (CPUSS_RAMx_CTL0.FAST_WS) should be less than or equal to the required number of slow wait states (CPUSS_RAMx_CTL0.SLOW_WS).

Timing should be constrained so that minimum wait cycles required are as follows.

- CPUSS_RAMx_CTL0.FAST_WS = 0
- CPUSS_RAMx_CTL0.SLOW_WS = 1 (CLK_MEM ≤ 200 MHz, default)
- CPUSS_RAMx_CTL0.SLOW_WS = 0 (CLK_MEM ≤ 100 MHz)

As the wait states are represented in CLK_MEM cycles, the wait states do not have to be reprogrammed when the fast clock domain frequency (CLK_FAST_0 or CLK_FAST_1) or slow clock domain frequency (CLK_SLOW) is changed. However, it may be necessary to reprogram the wait states when CLK_MEM is changed.

10.2.3 Operation

The following describes the SRAM controller with ECC functionality enabled.

SRAM accesses originate from one of the following paths:

- AHB-Lite or AXI transfers.
- Write buffer requests. If ECC functionality is disabled, this path is not used.
- SRAM repair requests. If ECC functionality is disabled, this path is not used.

The AHB-Lite and AXI transfers are the origin for all SRAM accesses; the write buffer and SRAM repair requests result from AHB-Lite and AXI transfers. The SRAM controller differentiates between the following three types of AHB-Lite and AXI transfers:

- AHB-Lite and AXI read transfers.
- 64-bit AXI write transfers.
- 8-bit, 16-bit, and 32-bit AHB-Lite or AXI write transfers (also referred to as partial AHB-Lite or AXI write transfers).

Each type is described in more detail here.

AHB-Lite and AXI read transfers:

An AHB-Lite or AXI read transfer is translated into an SRAM read access using the ECC syndrome logic. The ECC syndrome logic corrects recoverable errors. If the read address matches in the write buffer, the SRAM has stale data and the write buffer provides the requested read data.

The ECC syndrome logic reports recoverable and non-recoverable errors to the fault reporting component in the SRAM controller.

A corrected, recoverable error requires an update of the SRAM: the SRAM address needs to be written/repaired with the corrected code word. This automatic repair functionality is enabled when CPUSS_RAMx_CTL0.ECC_AUTO_CORRECT is '1'.

64-bit AXI write transfers:

A 64-bit AXI write transfer is translated into an SRAM write access, using the ECC parity logic. If the write address matches in the write buffer, the matching write buffer entries have stale data and these entries are invalidated.

Partial AHB-Lite and AXI write transfers:

A partial AHB-Lite or AXI write transfer is translated into an SRAM read access and an SRAM write access. The SRAM read access is the direct result of the partial write transfer and the SRAM write access is the result of a write buffer request. A partial write transfer requires an SRAM read access to retrieve the “missing” data bytes from the SRAM. If the read address matches in the write buffer, the SRAM has stale data and the write buffer provides the requested read data. The requested read data is merged with the partial write data to provide a complete 64-bit data word. The address and the merged write data are written to the write buffer. A future write buffer request results in a SRAM write access with the merged write data.

Only the partial AHB-Lite and AXI write transfers of data size less or equal 32-bit (dependency on data size dependency on data size) use the write buffer.

10.2.4 Write buffer

The write buffer is a temporary holding station for future SRAM write accesses.

The buffer allows SRAM write accesses to be postponed. This allows for more performance critical AHB-Lite or AXI requests to “overtake” write buffer requests. Memory consistency is guaranteed by matching the SRAM access address with the write buffer entries' addresses: a matching SRAM read access uses the read merge component and a matching SRAM write access invalidates the matching write buffer entries.

When the write buffer is full, an entry needs to be freed to accommodate future partial AHB-Lite or AXI write transfers. Therefore, a full write buffer raises the priority of the write buffer request path.

The state of the write buffer is reflected by CPUSS_RAMx_STATUS.WB_EMPTY. The write buffer is not retained in deep sleep power mode. Therefore, when transitioning to system deep sleep power mode, the write buffer should be empty.

Note that this requirement is typically met, because a transition to deep sleep power mode also requires that there are no outstanding AHB-Lite or AXI transfers. If there are no outstanding AHB-Lite or AXI transfers, the write buffer gets SRAM access.

10.3 ECC details

The SRAM controller supports ECC. Specifically, it supports a hamming code with an additional parity bit. This code supports single error correction, double error detection (SECDED). The ECC is applied to the SRAM data and SRAM address.

- The ECC corrects single-bit errors in an SRAM code word (stored in SRAM memory).
- The ECC detects single-bit and double-bit errors in an SRAM code word and the SRAM address.

The SRAM controller does not generate AHB-Lite or AXI bus errors. In the case of an ECC error, a correctable error is corrected on the fly and a non-correctable error is communicated through the fault reporting structure.

Note that the initial value of SRAM is undefined. Therefore, SRAM should be initialized before reading or partial writing to prevent unintentional ECC faults. For initialization, the CPUSS_RAMx_CTL0.ECC_CHECK_DIS bit can be used. When this bit is set, ECC check, notification for fault reporting, and ECC correction are disabled. Set the CPUSS_RAMx_CTL0.ECC_CHECK_DIS bit only for initialization.

10.3.1 ECC parity generation for SRAM write accesses

For 64-bit AXI write bus transfers, only a single SRAM write access is required. For 8-bit, 16-bit, and 32-bit AHB-Lite and AXI write bus transfers, an additional SRAM read access precedes the SRAM write access to retrieve the “missing” data bytes. These missing bytes are required to construct the complete 64-bit data word. The 8-bit parity is calculated over the complete 64-bit data word.

10.3.2 ECC syndrome generation for SRAM read accesses

For read accesses, the syndrome specifies one of the following:

- No error is detected. The SRAM 64-bit data word can be used as the result for an AHB-Lite or AXI read bus transfer.
- A single error is detected in the data word. This error is recoverable. The syndrome specifies the bit error location. The correction process inverts the bit value at the error location. The corrected data word is used as the result for an AHB-Lite or AXI read bus transfer. An additional SRAM write access is required to update the SRAM code word in case ECC_AUTO_CORRECT feature is enabled.
- A single error is detected in the 8-bit parity. This error is recoverable. An additional SRAM write access is required to update the SRAM code word with the correct parity.
- A single error is detected in the word address. This error is non-recoverable.
- A double error is detected. This error is non-recoverable.

For AHB-Lite or AXI read bus transfers, typically only a single SRAM read access is required. However, when a recoverable error is detected, an additional SRAM write access is required for the ECC_AUTO_CORRECT feature. Recoverable errors are communicated through the fault reporting structure.

Note that when a non-recoverable error is detected, the data word that is used as the result for an AHB-Lite or AXI bus transfer is incorrect, but no AHB-Lite or AXI bus error is generated. Non-recoverable errors are communicated through the fault reporting structure.

The fault reporting structure supports two types of SRAM controller faults:

- Correctable ECC faults
- Non-correctable ECC faults

For both fault types, the same information is captured by the fault reporting structure:

- SRAM word address
- SRAM syndrome

Note that the SRAM code word (8-bit parity and 64-bit data word) are not captured.

Note, that fault reporting can capture only a certain rate of fault events. It cannot be guaranteed that each of the ECC faults can be captured in case of multiple ECC errors occur short after each other.

10.3.3 ECC error injection

The fault reporting structure for ECC faults can be verified through an SRAM controller ECC parity injection mechanism. This mechanism functions as follows:

- ECC injection is enabled through CPUSS_RAMx_CTL0.ECC_INJ_EN (for SRAM controller x).
- A word address is specified by CPUSS_ECC_CTL.WORD_ADDR[23:0] (CPUSS_ECC_CTL.WORD_ADDR = (0x00FFFFFF & (RAM_TEST_ADDRESS>>2))).
- A 8-bit parity is specified by CPUSS_ECC_CTL.PARITY[7:0].

When a write transfer to the specified word address is performed, the ECC parity generation uses the specified 8-bit parity, rather than the calculated parity. The data still originates from the bus transfer. Any access size can be used to inject parity. Note that parity injection invalidates the write buffer for this word address. If only a part of 64-bit data is written and consistency should be maintained, CPUSS_RAMx_STATUS.WB_EMPTY=1 should be checked before.

10.3.4 ECC parity generation by software

To inject the ECC error for fault generation, ECC parity must be generated by software. Follow this procedure to generate 8-bit ECC parity.

```
CODEWORD_SW[127:0] = {128 {1'b0}};
CODEWORD_SW[63:0] = ACTUALWORD[63:0];
ADDR_WIDTH = log2(RAM_SIZE)
CODEWORD_SW[ADDR_WIDTH+60:64] = ADDR[ADDR_WIDTH-1:3];
```

***Note**: RAM_SIZE is size of RAMx, where 'x' is the RAM unit number.*

```
ECC_P0_SW =
128b00000001_10111111_10111011_01110101_10111110_00111010_01110010_11011100_01000100_10000100_01
001010_10001000_10010101_00101010_10101101_01011011;
ECC_P1_SW =
128b00000010_11011111_01110110_11111001_11011101_10011001_10111001_01110001_00010001_00001000_10
010011_00010001_00100110_10110011_00110110_01101101;
ECC_P2_SW
=128b00000100_11101111_11001111_10011111_10011010_11010101_11001110_10010111_00000110_00010001_0
0011100_00100010_00111000_11000011_11000111_10001110;
ECC_P3_SW =
128b00001000_11110111_11101100_11110110_11101101_01100111_01001110_01101100_10011000_00100001_11
100000_01000011_11000000_11111100_00000111_11110000;
ECC_P4_SW =
128b00010000_11111011_01111011_10101111_01101011_10100110_10110101_10100110_11100000_00111110_00
000000_01111100_00000000_11111111_11111000_00000000;
ECC_P5_SW =
128b00100000_11111101_10110111_11001110_11110011_01101100_10101011_01011011_11111111_11000000_00
000000_01111111_11111111_00000000_00000000_00000000;
ECC_P6_SW =
128b01000000_11111110_11011101_01111011_01110100_11011011_01010101_10101011_11111111_11111111_11
111111_10000000_00000000_00000000_00000000_00000000;
ECC_P7_SW =
128b10000000_01111111_00000000_00000000_00000111_11111111_11111111_11111111_11010100_01000010_00
100101_10000100_01001011_10100110_01011100_10110111;
```

As shown here, reduction XOR of the ANDed result of CODEWORD_SW[127:0] and respective ECC constants will give a single parity bit.

```
parity[0] = ^ (CW_SW[127:0] & ECC_P0_SW)
parity[1] = ^ (CW_SW[127:0] & ECC_P1_SW)
…
parity[7] = ^ (CW_SW[127:0] & ECC_P7_SW)
```

Parity[6:0] gives seven bits parity for 32 bits ACTUALWORD[127:0].

10.4 RAM retention configuration

This section covers the steps for transitioning to the RAM retention mode in XMC7000. The registers in Table 72 and Table 73 are used. The write buffer should be empty (CPUSS_RAMx_STATUS.WB_EMPTY = 1), when the mode is set to RETAINED. Depending on the reset cause, the SRAM is not retained even when an orderly shutdown took place earlier. An "orderly shutdown" corresponds to emptying the SRAM write buffer. See Table 119 in for reset causes, with which SRAM stays retained.

Also, the SRAM write buffer is not retained in Deep Sleep power mode. Therefore, the SRAM write buffer execution must be finished by emptying the SRAM write buffer before entering the power save mode.

Table 72 Power Control register

Register	Bit field	Bit value	Mode	Description
CPUSS_RAM0_PWR_MACRO_CTLy for SRAM#0 [1)] CPUSS_RAMx_PWR_CTL for SRAM other than RAM#0	PWR_MODE	0	OFF	Switch SRAM OFF
		2	RETAINED	Put SRAM in retained mode
		3 (Default)	ENABLED	Switch SRAM ON
	VECTKEYSTAT[2)]	0xfa05	–	Register key (to prevent accidental writes). • Should be written with a 0x05fa key value for the write to take effect. • Always reads as 0xfa05.

1) SRAM#0 can be fully retained or retained in increments of 32-KB sectors. SRAM unit other than RAM#0 can be retained as a whole unit.
2) VECTKEYSTAT must be written at the same time as PWR_MODE. These registers should be written as the complete 32-bit data.

Table 73 RAM Status register

Register	Bit field	Description
CPUSS_RAMx_STATUS	WB_EMPTY	Write buffer empty: '0': Write buffer not empty '1': Write buffer empty

As mentioned earlier, when transitioning to Deep Sleep mode, the write buffer (CPUSS_RAMx_STATUS.WB_EMPTY = 1) should be empty.

1. Checking the CPUSS_RAMx_STATUS.WB_EMPTY Register and wait until the WB_EMPTY bit becomes '1'
2. When WB_EMPTY bit becomes 1, set the retained mode to the CPUSS_RAM0_PWR_MACRO_CTLy or CPUSS_RAMx_PWR_CTL register
3. Transfer to deep sleep mode or issues the software reset
4. When returning from dDeep Sleep, it is necessary to set to enable mode before using RAM

***Note**: SRAM0_PWR_MACRO_CTL0.PWR_MODE must be set to ENABLE or RETAINED in Active, LP Active, Sleep, LP Sleep, and Deep Sleep modes.*

10.5 Registers

Table 74 List of Registers

Registers name	Name	Description
CPUSS_RAMx_CTL0	RAMx Control Register	Specify the operation of the RAMx controller.
CPUSS_RAMx_STATUS	RAMx Status Register	Indicates RAMx controller status.
CPUSS_RAM0_PWR_MACRO_CTLy	RAM0 Power Control Register	These registers control the system SRAM 0 power states of a single macro. System SRAM 0 consists of up to sixteen 32 KB macros. Each macro is a single power partition and is controlled through a dedicated control field in one of these registers.
CPUSS_RAMx_PWR _CTL	RAMx Power Control Register	This register controls the system SRAMx power states. System SRAMx consists of a single power partition.
CPUSS_RAM_PWR_DELAY_CTL	RAM Power Up Delay Control Register	Number clock cycles delay needed after power domain power up.
CPUSS_ECC_CTL	ECC Control Register	Specifies the word address and ECC parity where an error will be injected.

***Note**: The 'x' in the register name denotes the SRAM memory unit number. The “y” in the register name denotes the SRAM0 memory macro number. Refer to the device datasheet for the specifications.*

11 BootROM

System boot is defined as the process of obtaining, validating, and starting the product firmware. XMC7000 has embedded ROM, and flash performs its entire boot process in software. The main function of the boot process is to configure the system (apply trims and wounding information, and configure access and protection settings according to the product life-cycle stage), authenticate the application, and transfer control to the application.

11.1 Features

The BootROM of XMC7000 supports the following features:

- After any type of reset, the boot code starts execution from ROM on the CM0+.
- The boot process consists of two parts: ROM boot process and flash boot process.
- See the Flash boot for more details.
- The ROM boot code applies life-cycle stage and protection state.
- The ROM boot code validates the integrity of the flash boot process before starting it.

11.2 ROM controller

The XMC7000 series has a supervisory ROM that contains BootROM code and SROM APIs. This section gives a brief overview of the ROM controller.

The ROM controller has two AHB-Lite bus interfaces:

- An AHB-Lite bus interface in the slow clock domain for all bus masters in the slow clock domain (CM0+ CPU, P-DMA, M-DMA, and so on). The slow bus infrastructure combines the bus masters in the slow clock domain.

11.2.1 Wait states

The ROM controller supports programmable wait states, which is defined at SLOW_WS[1:0] and FAST_WS[1:0] in the CPUSS_ROM_CLT Register. Dedicated wait states are provided for the fast and slow AHB-Lite bus interfaces. The programmable wait states represent the number of CLK_MEM cycles for a read path through the SROM in either the fast or slow domain.

The ROM controller supports wait states in the range 0 to 3. The number of wait states is expressed in CLK_MEM clock cycles. The application that changes CLK_MEM must set the ROM wait states corresponding to the new target of CLK_MEM frequency before CLK_MEM is raised.

- The wait states for the slow clock domain are:
  - CPUSS_ROM_CTL.SLOW_WS = ‘0’ for (CLK_MEM ≤ 100 MHz)
  - CPUSS_ROM_CTL.SLOW_WS = ‘1’ for (CLK_MEM > 100 MHz) and (CLK_MEM ≤ CLK_MEM Max)
- The wait state for the fast clock domain is:
  - CPUSS_ROM_CTL.FAST_WS = ‘0’ up to CLK_MEM Max

11.3 ROM boot process

11.3.1 Life-cycle stages and protection states

Life-cycle stages are governed by eFuse and are irreversible. A powered device also has a volatile protection state that reflects its life cycle stage. The protection state is determined on boot and defined by the value of the CPUSS_PROTECTION Register. For more details, see the Device security.

11.3.2 Multicore boot

XMC7000 starts up with all cores, except CM0+, in reset. The ROM boot process executes on M0+. Its main purpose is to start an M0+ flash boot process.

When other CPUs start, they enter the ROM boot process. The ROM boot process will recognize this and jump directly to the code pointed by CPUSS.CM7_0/CM7_1_VECTOR_TABLE_BASE for CM7_0/CM7_1 bypassing the full boot. The CPUSS_IDENTITY Register is used to determine which CPU is executing the ROM boot process.

11.3.3 “Secure Boot”

Before CM0+ executes the firmware in supervisory flash and the life-cycle stage is only SECURE and SECURE _WITH_DEBUG, it authenticates the flash boot code by comparing the pre-computed SECURE_HASH stored in eFuse with the generated one. Flash boot code will be executed only if it is found to be authentic; otherwise, boot code enters the DEAD protection state.

11.3.4 Protection setting

ROM boot reads the configurations of SMPU, PPU, and SWPU from SFlash and programs the protection units accordingly.

- DAP memory protection unit (MPU)

  This is used to restrict the access rights of DAP as indicated by NORMAL, SECURE, and DEAD access restrictions. The boot uses eight memory regions of MPU to implement the access restrictions.
- Shared memory protection unit (SMPU)

  These are used to implement access restrictions to memory such as ROM, Flash, and RAM. ROM/flash boot reads the SMPU configuration from SFlash and programs the corresponding SMPU Registers.
- Software protection unit (SWPU)

  These are used to implement access restrictions to flash (program/erase) and eFuse (read/write). There are 32 entries in SWPU. The SWPU is broken into two parts. The first part is stored in SFlash and implements the access restrictions related to PC1 and PCx. Here PC1 means protection context 1 and PCx means one of protection context {2, 3, …, 15}. See the Protection context for details. The second part is stored in SFlash and is used by the application for additional access restrictions specific to the application. ROM/flash boot reads the two parts of SWPU from SFlash and stores them in RAM.
- Peripheral protection unit (PPU)

  These are used to implement access restrictions to Peripheral Registers. Only a subset of the PPUs are required to enforce protection for PC1 and PCx and only these are stored in SFlash. Additional PPUs will be used by the application (not stored in SFlash) for additional access restrictions specific to the application.

See the Protection context chapter for details on SMPU, PPU, and SWPU.

11.3.4.1 SMPU configuration in SFlash

In SMPUs, address ranges will be chosen so that the access rights are well defined for both PC1 and PCx. The address ranges are also chosen such that the number of SMPUs required is minimized. One may require SMPUs with overlapping address ranges if the access rights for PC1 are different from the access rights for PCx. In that case, one may have to use the PC Match feature. For SMPUs, both the master and the slave registers are stored in SFlash.

SMPU31/15 and SMPU30/14 are configured during boot as follows:

- SMPU31/15 is configured to protect the first 2KB of SRAM such that only PC0 and PC1 canaccess it.
  - SMPU31/15 slave protection attribution ATT0 = 0x8A00037F
  - SMPU31/15 master protection attribution ATT0 = 0x8700FF49
- SMPU30/14 is configured to protect system partition of SROM such that it is accessible only by PC0 and PC1. User partition is accessible by all PC.
  - SMPU30/14 slave protection attribution ATT0 = 0x8A00037F
  - SMPU30/14 master protection attribution ATT0 = 0x8700FF49

***Note**: SMPU 31/30 or 15/14 usage by BootROM is device-specific, see the device datasheet for correct reference*

11.3.4.2 SWPU configuration in SFlash

As stated earlier, the SWPU is broken into two parts, which are stored in SFlash. The first part implements the access restrictions related to PC1 and PCx. The second part is used by the application for additional access restrictions specific to the application. ROM/flash boot reads the two parts of SWPU from SFlash and stores them in RAM.

“Write” (program/erase/erase suspend/erase resume) access protection for flash and “read/write” access protection for eFuse are provided using software protection units. These protection units are divided into three groups. The first group of protection units is called FLASH_WRITE_PU, the second group is called FUSE_READ_PU, and the third group is called FUSE_WRITE_PU. Write access protection for flash is provided by FLASH_WRITE_PU. Read and write accesses to eFuse are provided by FUSE_READ_PU and FUSE_WRITE_PU, respectively.

The maximum number of FLASH_WRITE_PUs, FUSE_READ_PU, and FUSE_WRITE_PU are parameters initialized from SFlash. The value of this parameter is not expected to exceed 32 for FLASH_WRITE_PU and 8 for FUSE_READ_PU and FUSE_WRITE_PU.

A copy of the SWPU structures is stored in SFlash and during boot time the structures are read into RAM. The address range covered by each SWPU entry is fixed when the SWPU is stored in SFlash and cannot be updated in RAM. The integrity of SWPU entries in SFlash is checked by SECURE_HASH during secure boot.

By default, the number of each SWPU type implemented within the first part of SWPU, which cannot be modified by the user, is as follows:

- eFuse Write PU: N_FUSE_WRITE_PU = 1
- eFuse Read PU: N_FUSE_READ_PU = 1
- Flash Write PU: N_FLASH_WRITE_PU = 0

This means that the index of eFuse Write PU, eFuse Read PU, and Flash Write PU in the second part of SWPU, which the user can update, starts from 1, 1 and 0 respectively. These indices are necessary to call the WriteSWPU and ReadSWPU APIs.

11.3.4.3 PPU configuration in SFlash

Read and write protection associated with each PPU can be categorized into one of the four read/write classes.

The write classes are defined as follows

- Class I – Both PC1 and PCx have write attribute = 0. For example, both PC1 and PCx do not have write access to the CPUSS_PROTECTION Register. For the corresponding PPU, both PC1 and PCx must have the attributes “UW=0, PW=0, NS=1” for the master and slave registers.
- Class II – PC1 write attribute = 0 and PCx write attribute = 1. For example, PC1 does not have write access to CPUSS_AP_CTL, but PCx does. For the corresponding PPU, PC1 must have the attributes “UW=0, PW=0, NS=1” and PCx must have the attributes “UW=1, PW=1, NS=1” for the master and slave registers.
- Class III – PCx write attribute = 0. And PC1 write attribute = 1. For example, PCx does not have write access to EFUSE_MXS40.CTL, but PC1 does. For the corresponding PPU, PC1 must have the attributes “UW=1, PW=1, NS=1” and PCx must have the attributes “UW=0, PW=0, NS=1” for the master and slave registers.
- Class IV – PCx and PC1 have write attribute = 1. For example, both PC1 and PCx have write access to IPC_STRUCT1. For the corresponding PPU, both PC1 and PCx must have the attributes “UW=1, PW=1, NS=1” for the master and slave registers.

The read classes are defined as follows

- Class I – Both PC1 and PCx have read attribute = 0. For the corresponding PPU, both PC1 and PCx must have the attributes “UR=0, PR=0, NS=1” for the slave register.
- Class II – PC1 read attribute = 0 and PCx read attribute = 1. For the corresponding PPU, PC1 must have the attributes “UR=0, PR=0, NS=1” and PCx must have the attributes “UR=1, PR=1, NS=1” for the slave register.
- Class III – PCx read attribute = 0. And PC1 read attribute = 1. For the corresponding PPU, PC1 must have the attributes “UR=1, PR=1, NS=1” and PCx must have the attributes “UR=0, PR=0, NS=1” for the slave register.
- Class IV – PCx and PC1 have read attribute = 1. For the corresponding PPU, both PC1 and PCx must have the attributes “UR=1, PR=1, NS=1” for the slave register.

In general, the read and write classification for each PPU is stored in SFlash only if at least one of them is class I or III. However, other classes may also be stored in SFlash. The storing is done in factory. SFlash has an entry that points to the protection settings. The ROM boot reads this classification and configures PPU accordingly. The following table shows the SFlash representation of the write/read access restrictions for PPUs. Refer to the device datasheet for information about the protection unit PPU_ID for the corresponding PPU region. For example, neither PC1 or PCx can write, but they can both read the registers in the PERI_MS_PPU_FX_CPUSS_BOOT region. For those PPUs whose classifications are not stored in SFlash, the ROM boot will configure the PPUs for both read and write to the default class IV.

Table 75 SFlash representation of write/read access restrictions for each PPU

Name of fixed PPU	Access for PC > 0? (slave attributes)	Access for PC > 0? (master attributes)
PERI_MS_PPU_FX_CRYPTO_BOOT	PC1 - read only PCx - read only	PC1 - read only PCx - read only
PERI_MS_PPU_FX_CPUSS_BOOT	PC1 - read only PCx - read only	PC1 - read only PCx - read only
PERI_MS_PPU_FX_FLASHC_FlashMgmt	PC1 - full access PCx - No access	PC1 - read only PCx - read only
PERI_MS_PPU_FX_EFUSE_CTL	PC1 - full access PCx - read only	PC1 - read only PCx - read only

(table continues...)

Table 75 (continued) SFlash representation of write/read access restrictions for each PPU

Name of fixed PPU	Access for PC > 0? (slave attributes)	Access for PC > 0? (master attributes)
PERI_MS_PPU_FX_EFUSE_DATA	PC1 - full access PCx - No access	PC1 - read only PCx - read only
PERI_MS_PPU_FX_SRSS_SECURE	PC1 - read only PCx - read only	PC1 - read only PCx - read only
PERI_MS_PPU_FX_CRYPTO_MAIN	PC1 - full access PCx - full access	PC1 - full access PCx - full access
PERI_MS_PPU_FX_CRYPTO_CRYPTO	PC1 - full access PCx - full access	PC1 - full access PCx - full access
PERI_MS_PPU_FX_IPC_STRUCT0_IPC	PC - full access PCx - full access	PC1 - full access PCx - full access
PERI_MS_PPU_FX_IPC_STRUCT1_IPC	PC1 - full access PCx - full access	PC1 - full access PCx - full access
PERI_MS_PPU_FX_IPC_STRUCT2_IPC	PC1 - full access PCx - full access	PC1 - full access PCx - full access
PERI_MS_PPU_FX_FLASHC_DFT	PC1 - full access PCx - read only	PC1 - read only PCx - read only
PERI_MS_PPU_FX_BIST	PC1 - read only PCx - read only	PC1 - read only PCx - read only
PERI_MS_PPU_FX_CRYPTO_BUF	PC1 - full access PCx - full access	PC1 - full access PCx - full access

The following programmable PPUs are configured during boot. Note that for all programmable PPUs, PC other than PC0 can only modify SL_ATT or MS_ATT; SL_ADDR and SL_SIZE can be modified only in PC0. Therefore, all unused programmable PPUs, that is, PPUs that are not configured during the boot process, are not available to the user. See Protection unit for details.

- Programmable PPUs 0, 1, and 2 are used to protect the following area of eFuse such that it is not accessible to any PC other than PC0.

PPU ID	SL_ADDR	SL_SIZE
0	0x402c0840	4 bytes
1	0x402c0840	32 bytes
2	0x402c0860	8 bytes

- Programmable PPU 3 is used to protect the CRYPTO Register (SL_ADDR = 0x40100000, SL_SIZE = 4 bytes). During boot it is enabled and access is provided to all PCs.
- Programmable PPU 5 is used to protect the CLK_TRIM_ILO0_CTL Registers. This PPU is configured to allow write access to the registers for any PC except PC1 (SL_ADDR = 0x40263014, SL_SIZE= 4B).
- Programmable PPU 6 is used to protect the CLK_TRIM_ILO1_CTL Registers. This PPU is configured to allow write access to the registers for any PC except PC1 (SL_ADDR = 0x40263220, SL_SIZE= 4B).

- Programmable PPU 7 is used to protect the unused CRYPTO_MEM_BUFF region from 0x8000 to 0xFFFF offset. This region will be accessible only to PC0 and is present only in XMC7000 TVII-BE-1M rev. ** devices.
- In all other XMC7000 devices after this revision, programmable PPU 7 is used to protect the PWR_TRIM_HT_PWRSYS_CTL Register. This PPU is configured in flash boot to allow write access to the registers for any PC1 and read access to any PC.
- Programmable PPU 8 is used to protect the part of flash controller register region (SL_ADDR = 0x4024f050, SL_SIZE = 16 bytes) in SECURE life cycle, such that they are accessible only to PC0.
- Programmable PPU 9 is used to allow only PC2 access to FLASHC_ECC_CTL Registers of the DFT region, from 0x2a0 to 0x2bc offset as DFT region will be protected using a fixed PPU such that only PC0 has access.
- Programmable PPU 10 is used to provide access to EFUSE_SEQ_DEFAULT to all PCx (SL_ADDR = 0x402C0020, SL_SIZE = 4B)

Table 76 Programmable PPUs modifiability summary

Programmable PPU	Modifiable by PC0	Modifiable by PC1	Modifiable by PCx
PPU0	yes	yes	no
PPU1	yes	no	no
PPU2	yes	no	no
PPU3	yes	yes	yes
PPU5	yes	no	yes
PPU6	yes	no	yes
PPU7	yes	no	no
PPU8	yes	no	no
PPU9	yes	no	no[1)]
PPU10	yes	yes	yes

1) Programmable PPU9 can be modified in PC0 and PC2; it cannot be modified in other PCs.

11.3.4.4 Boot protection settings in SFlash

Figure 52 shows how the protection settings are stored in SFlash.

- Object size – Size of boot protection object in bytes.
- N_SMPU – Number of SMPU structures (starting form SMPU15) stored in this object.
  For example, N_SMPU = 4 indicates SMPU15, SMPU14, SMPU13, and SMPU12 are configured.
- SMPU15 – Contains SMPU region address and SMPU region attributes.
- N_PPU – Number of PPU structures stored in this object.
- PPU_ID, PPU config defines a PPU – PPU_ID is the PPU number (2 bytes) and the PPU config is described using 1 byte (4 bits for write class and 4 bits for read class).
- N_FLASH_WRITE_PU – number of FLASH_WRITE_PUs stored in this object. It is followed by the contents of the FLASH_WRITE_PUs.
- FLASH_WRITE_PU – Data structure of FLASH_WRITE_PU.
- N_FUSE_READ_PU – number of FUSE_READ_PUs stored in this object. It is followed by the contents of the FUSE_READ_PUs.
- FUSE_READ_PU – Data structure of FUSE_READ_PU.

- N_FUSE_WRITE_PU – number of FUSE_WRITE_PUs stored in this object. It is followed by the contents of the FUSE_WRITE_PUs.
- FUSE_WRITE_PU – Data structure of FUSE_WRITE_PU.

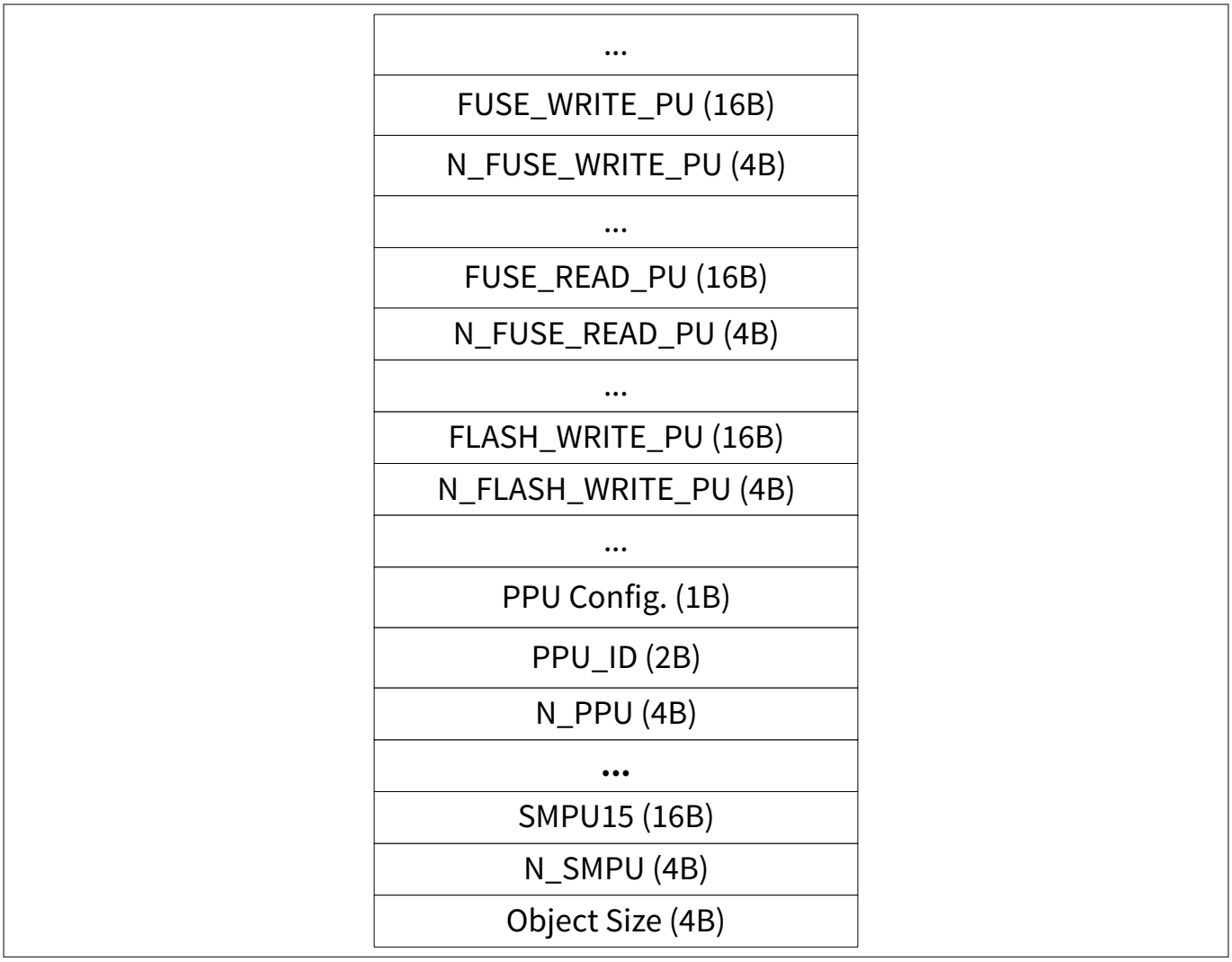


Figure 52 Boot protection settings in flash

11.3.4.5 Security enhancement PPU configuration in SFlash

This security enhancement PPU configuration is supported on the new flash boot version, indicated in the following table.

The Flash boot version can be read from address 0x17002018 (SFLASH_FLASH_BOOT_VERSION_LOW). See the device datasheet for details of boot time specifications.

Table 77 Flash boot version

	Flash boot version	Security Enhancement	Security Marker	Boot time spec
XMC7000	554 and later (new)	Supported	Yes	SID80A, SID80B, SID81A, SID81B

By programming the magic word to the security marker (TOC2_SECURITY_UPDATES_MARKER), Boot process configures the following PPUs for enhancement of security and safety.

Table 78 Security enhancement PPUs

Name of PPU	Protection start Address (SL_ADDR)	Size (SL_SIZE)	Access for PC > 0 (Slave attribute)	Access for PC > 0 (Master attribute)
Programmable PPU 11	0x40201000	32 bytes	PC1 - Full access PCx - Full access	PC1 - Full access PCx - Full access
Programmable PPU 12	0x402013c8	4 bytes	PC1 - Full access PCx - Full access	PC1 - Full access PCx - Full access
Programmable PPU 13	0x40201300	256 bytes	PC1 - Full access PCx - Full access	PC1 - Full access PCx - Full access
PERI_MS_PPU_FX_PERI_GR2_GROUP	0x40004050	4 bytes	PC1 - Read only PCx - Read only	PC1 - Read only PCx - Read only

Programmable PPUs 11 and 13 help separate HSM software and application software in combination with PERI_MS_PPU_FX_CPUSS_CM0. For example, programmable PPUs 11 and 13 are allowed access to application software, and PERI_MS_PPU_FX_CPUSS_CM0 is allowed access to HSM software. As a result, the CPUSS_AP_CTL Register is exclusively controlled by the HSM software while CPUSS_CM0_CLOCK_CTL and RAM0_PWR_CTL, RAM1_PWR_CTL can be controlled by the application software.

Programmable PPU 12 is used to protect the CPUSS_ECC_CTL Register. This register provides the ECC error insertion functionality. It is assumed that the ECC logic will be tested only during the startup of the device and the ECC error injection functionality is not required during the regular device operation once the startup is completed. It is recommended to disable the ECC error injection logic after the ECC test completion by blocking access to the ECC Error Injection Control Registers using this PPU configuration.

Accidental writing to PERI_GR2_SL_CTL Register can stop clock signals to the core MCU function blocks. PERI_MS_PPU_FX_PERI_GR2_GROUP protects PERI_GR2_SL_CTL from accidental write access.

Programmable PPUs 11, 12, and 13 are not configured and PERI_MS_PPU_FX_PERI_GR2_GROUP is default value.

These PPU configurations are the same as the previous flash boot version.

Security marker is part of TOC2. See the Flash boot chapter for the security marker location.

11.3.5 Debug and test access restrictions

Depending on the protection state (NORMAL, SECURE, or DEAD), the ROM boot process will enforce access restrictions on the debug access port (SWD/JTAG). See eFuse bits for access restriction field bit map. The ROM boot gets access enable bits that allow Debug Access Port (AP) for CM0+, CM7_0/CM7_1, and system from eFuse or SFlash.

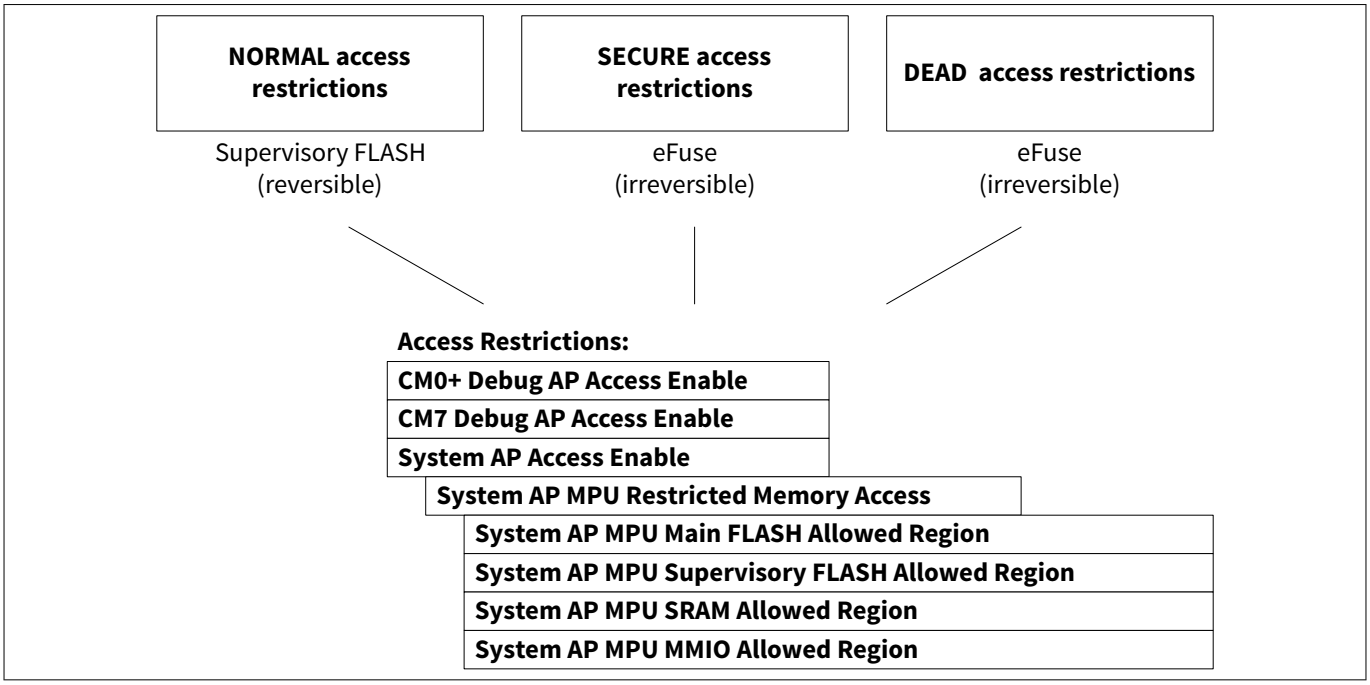


Figure 53 Debug access restrictions structures

- Three separate structures exist (two for SECURE and DEAD in eFuse and one for NORMAL in supervisory flash). All three structures have the same layout.
- NORMAL access restrictions are stored in SFlash and they can be updated unlike the access restrictions in SECURE and DEAD protection states.

11.3.6 SWD/JTAG initialization

When access restrictions prohibit use of the SWD/JTAG interface, the boot process does not access or change the SWD/JTAG pins in any way.

The customer firmware can, at any time, change the configuration of the SWD/JTAG pins to another mode, peripheral, configuration, or purpose.

To allow debugging of such applications, a 'listen window' is provided before starting customer firmware. The boot flash process will connect and enable the JTAG/SWD interface and wait for a specified time before starting application firmware. It is expected that application firmware checks the CPUSS_DP_STATUS.SWJ_CONNECTED bit and re-purposes the pins only when no SWD/JTAG connection is available.

11.3.7 Waking up from hibernate

Waking up from hibernate will result in system boot. The integrity checks on the SFlash trim values and SWD/JTAG connection delay (listen Window implemented by flash boot) is skipped when waking from hibernate.

11.3.8 ROM boot flow chart

Figure 54 shows the ROM boot flow chart.

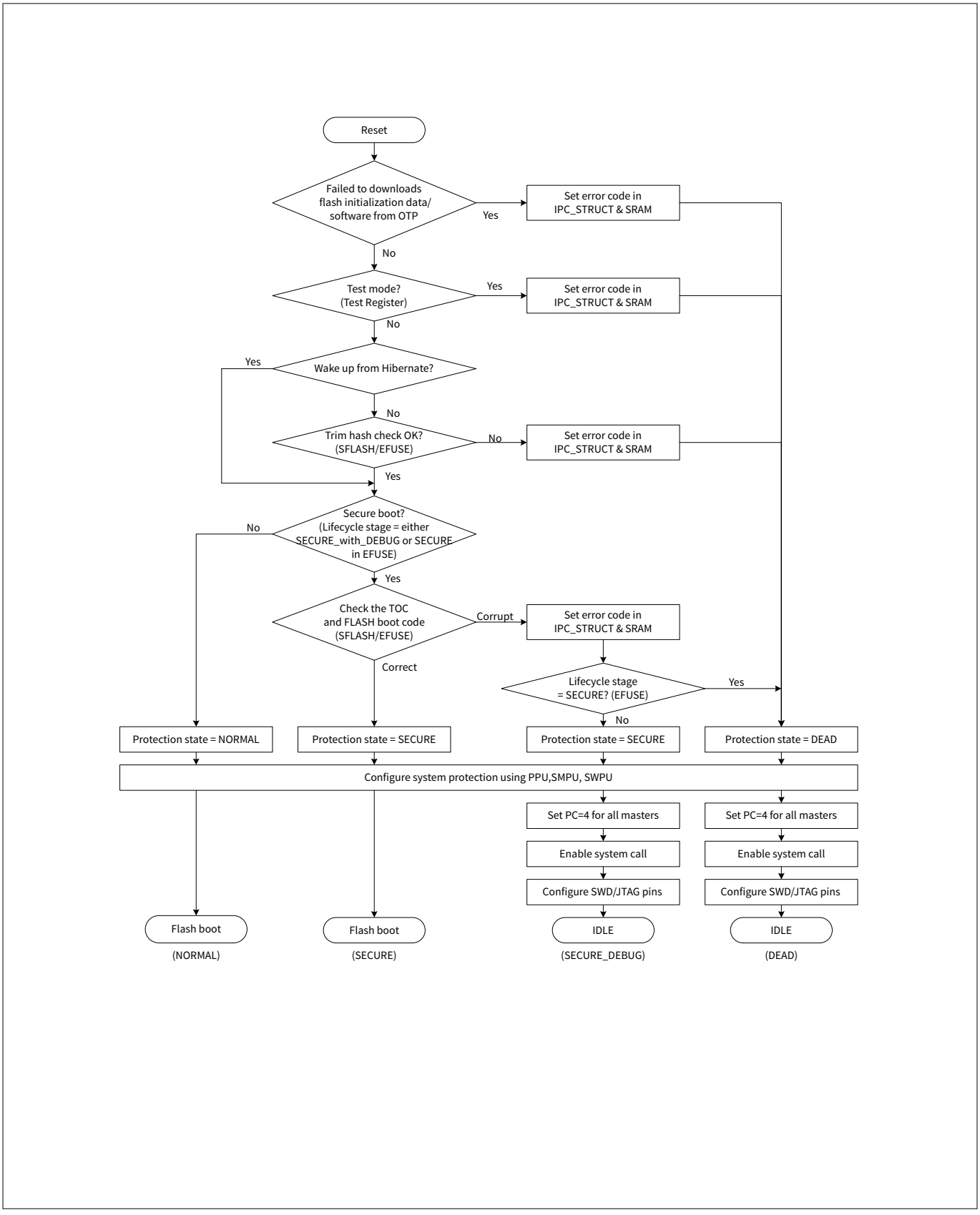


Figure 54 ROM boot flow chart

11.4 MMIO Registers and eFuse used by ROM boot

11.4.1 MMIO Registers

Table 79 MMIO Registers used by ROM boot

Register	Name	Description
CPUSS		
CPUSS_IDENTITY	Identity Register	Register that can be used to determine if the ROM boot process is executing on CM0+ or on CM7.
CPUSS_CM0_PC0_HANDLER	CM0+ Protection Context 0 Handler	Register that holds location of NMI vector.
CPUSS_CM0_VECTOR_TABLE_BASE	CM0+ Vector Table Base Register	Register that holds location of the vector table (SP and reset exception vector address are provided at offset 0x0 and 0x4) for the M0+ boot image to be used after CPU reset. Typically, this is the location of the Cortex®-M vector table in flash. This value must be set before issuing an M0+ CPU reset.
CPUSS_CM7_0/ CM7_1_VECTOR_TABLE_BASE	CM7_0/CM7_1 Vector Table Base Register	Register that holds location of the vector table (SP and reset exception vector address are provided at offset 0x0 and 0x4) for the CM7_0/CM7_1 boot image. Typically, this is the location of the Cortex®-M vector table in flash. This value must be set before releasing the CM7_0/CM7_1 from reset.
CPUSS_PROTECTION	Protection Status Register	Register that holds the current protection state. Can only be written to a different value according to the state diagram.
CPUSS_WOUNDING	Wounding Register	Register that indicates the amount of accessible flash and RAM array in this device.
CPUSS_AP_CTL	Access Port Control Register	Register that disables/enables any usage of CM0+, CM7_0/CM7_1, and system access port interface.
CPUSS_DP_STATUS	Debug Port Status Register	Register that indicates whether a SWD/JTAG connection is established.
SRSS		
TST_MODE	Test Mode Control Register	Register that indicates device is in a test mode. Setting this bit allows programming the flash before the control transfer to application.
PPU/SMPU/MPU/SWPU	–	Configures read/write access protection and flash program/erase protection. See the Protection unit for details.

(table continues...)

Table 79 (continued) MMIO Registers used by ROM boot

Register	Name	Description
IPC	–	Configures inter-process communication; used to implement system call interface. See the Inter-processor communication for details.

11.4.2 eFuse bits

Table 80 eFuse bits used by BootROM

Name[1)]	Bits	Description
SECURE access restrictions		
AP_CTL_CM0_DISABLE	1:0	Indicates that this device does not allow access to the CM0+ access port. 00 - Enable M0-AP 01 - Disable M0-AP 1x - Permanently disable M0-AP
AP_CTL_CM7_DISABLE	3:2	Indicates that this device does not allow access to the CM7 access port. 00 - Enable CM7-AP 01 - Disable CM7-AP 1x - Permanently Disable CM7-AP
AP_CTL_SYS_DISABLE	5:4	Indicates that this device does not allow access to the system access port. 00 - Enable SYS-AP 01 - Disable SYS-AP 1x - Permanently Disable SYS-AP
SYS_AP_MPU_ENABLE	6	Indicates that the MPU on the system access port must be programmed and locked according to the settings in the next six fields.
DIRECT_EXECUTE_DISABLE	7	Disables DirectExecute system call functionality (implemented in software). 0: DirectExecute API execution is allowed 1: DirectExecute API execution is not allowed

(table continues...)

Table 80 (continued) eFuse bits used by BootROM

Name[1)]	Bits	Description
FLASH_ALLOWED	10:8	This field indicates what portion of main flash is accessible through the system access port. Only a portion of flash starting at the bottom of the area is exposed. Encoding is as follows: 0: Entire region 1: 7/8th 2: 3/4th 3: 1/2 4: 1/4th 5: 1/8th 6: 1/16th 7: Nothing
SRAM_ALLOWED	13:11	This field indicates what portion of SRAM is accessible through the system access port. Only a portion of SRAM starting at the bottom of the area is exposed. Encoding is the same as FLASH_ALLOWED.
WORK_FLASH_ALLOWED	15:14	This field indicates what portion of work flash is accessible through the system access port. Only a portion of work flash starting at the bottom of the area is exposed. Encoding is as follows: 0: entire region 1: 1/2 2: 1/4th 3: Nothing
SFLASH_ALLOWED	17:16	This field indicates what portion of supervisory flash is accessible through the system access port. Only a portion of supervisory flash starting at the bottom of the area is exposed. Encoding is as follows: 0: Entire region 1: 1/2 2: 1/4th 3: Nothing
MMIO_ALLOWED	19:18	This field indicates what portion of the MMIO region is accessible through the system access port. Encoding is as follows: 0: All MMIO Registers 1: Only IPC MMIO Registers accessible (system calls) 2, 3: No MMIO access
DEAD access restrictions		
<Same as SECURE Access Restrictions>	-	The structure is identical to the one above but used when entering DEAD mode. It assumes that this structure is more restrictive than SECURE.
Critical Object Hash		
FACTORY_HASH	-	SHA-256 (upper 128 bits) that covers objects in TOC Part1. It is checked before transitioning to SECURE_WITH_DEBUG or SECURE.

(table continues...)

Table 80 (continued) eFuse bits used by BootROM

Name[1)]	Bits	Description
SECURE_HASH	-	SHA-256 that covers the flash boot image and other objects in TOC Part1 and Part2. Flash boot code is not started unless this value is correct.
SECURE_HASH_ZEROES	-	The number of bits that are '0' (fuses that are not blown) in the SHA-256. This guarantees that when a HASH is programmed, it cannot be changed into another valid HASH value.

1) See the device-specific datasheet to see whether a particular device feature is supported.

12 Interrupts

XMC7000 supports interrupts and exceptions on both Cortex®-M7 and Cortex®-M0+ cores. Interrupts refer to events generated by peripherals external to the CPU such as timers, serial communication block, and port pin signals. Exceptions refer to events generated by the CPU such as memory access faults and internal system timer events. Both interrupts and exceptions result in the current program flow being stopped and the exception handler or interrupt service routine (ISR) being executed by the CPU. Both Cortex®-M7 and Cortex®-M0+ cores provide their own unified exception vector table for both interrupt handlers/ISR and exception handlers.

12.1 Features

XMC7000 platform supports the following interrupt features:

- Supports up to 1023[1)] system interrupts
  - Eight Cortex®-M7 external interrupts and eight Cortex®-M7 internal (software only) interrupts. The CPU supports up to 240 interrupts, but only sixteen interrupts are used by the XMC7000 interrupt infrastructure. The eight external CPU interrupts support deep sleep (WIC) functionality.
  - Eight Cortex®-M0+ external interrupts and eight Cortex®-M0+ internal (software only) interrupts. The CPU supports up to 32 interrupts, but only sixteen interrupts are used by the XMC7000 interrupt infrastructure. The eight external CPU interrupts support deep sleep (WIC) functionality.
  - All the available system interrupt sources are usable in active power mode and can wake up from sleep power mode
  - A subset of available system interrupt sources capable of waking the device from deep sleep power mode
  - Four system interrupts can be mapped to each of the CPU NMI
- Nested vectored interrupt controller (NVIC) integrated with each CPU core, yielding low interrupt latency
- Wakeup interrupt controller (WIC) enabling interrupt detection (CPU wakeup) in deep sleep power mode
- Vector table may be placed in either flash or SRAM
- Configurable priority levels (eight levels for Cortex®-M7 and four levels for Cortex®-M0+) for each interrupt
- Level-triggered interrupt signals

1 For the list of system interrupts supported by the device variants, refer to Interrupt sources.

12.2 How it works

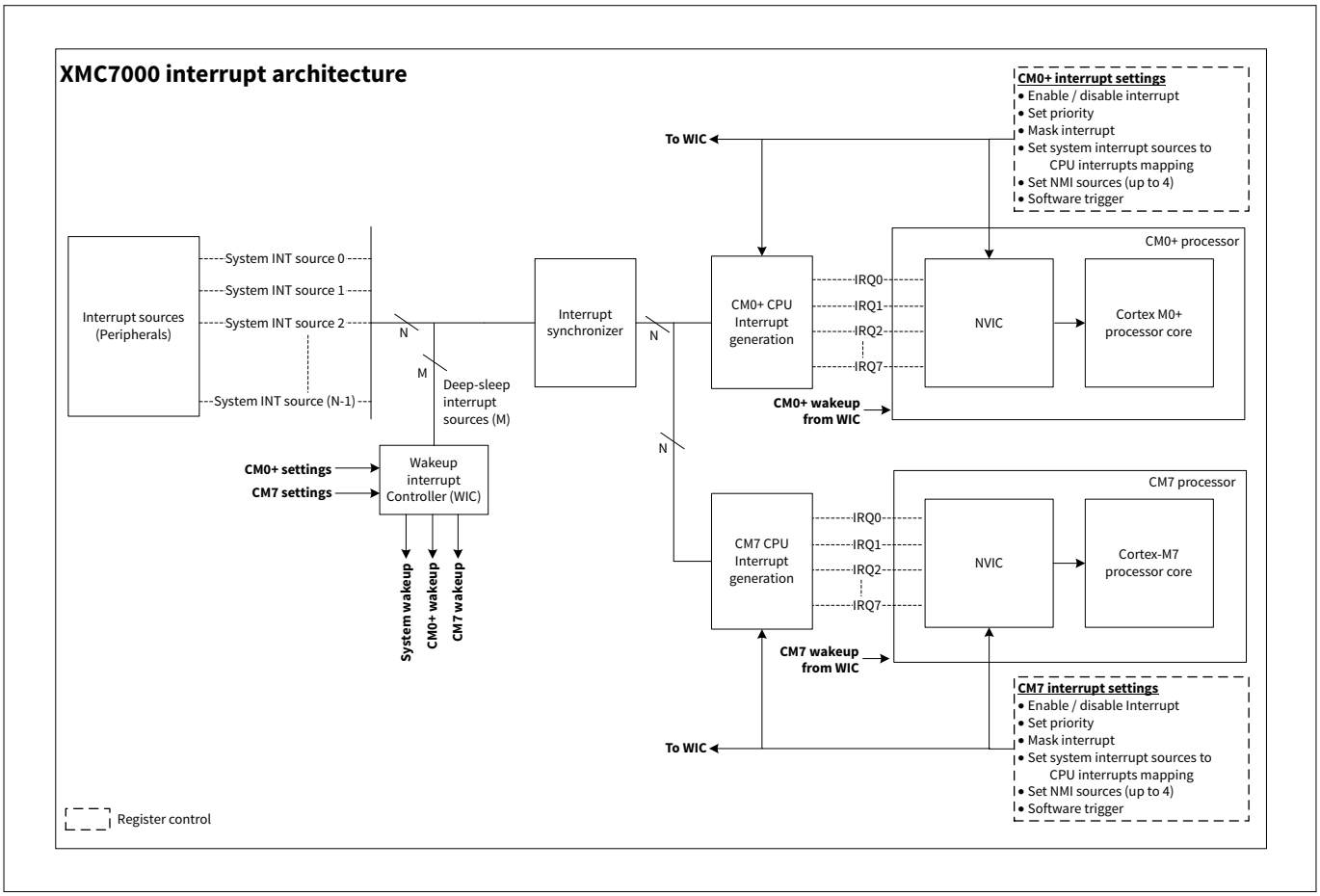


Figure 55 XMC7000 interrupts block diagram

Figure 55 shows the interrupt architecture in XMC7000. The 'N' system interrupts of XMC7000 are processed by the NVIC of the individual cores. The XMC7000 interrupt architecture uses eight CPU interrupts IRQ[7:0] out of the available CPU interrupts for each core. In the CM7 and CM0+ cores, the system interrupt source connection to a particular IRQn of the core is configurable and any of the 'N' system interrupts can be mapped to any of the IRQ[7:0] of each core. This ensures that all the system interrupts can be mapped onto any CPU interrupt simultaneously. The system interrupt to CPU interrupt mapping is independent for both CPUs. Refer to Interrupt sources for more details about the system interrupt to CPU interrupt mapping. The NVIC enables/ disables individual interrupt IRQs, priority resolution, and communication with the CPU core. Other exceptions such as NMI and HardFaults are not shown in Figure 55 because they are part of CPU core generated events, unlike interrupts, which are generated by peripherals external to the CPU.

In addition to the NVIC, XMC7000 supports a wakeup interrupt controller (WIC) and interrupt synchronizer block. The WIC provides detection of deep sleep interrupts in the deep sleep CPU power mode. Each CPU can individually be in deep sleep; the device is said to be in deep sleep only when all the CPUs are in deep sleep. Refer to the Device power modes for more details about the deep sleep power mode. Each CPU has independent WIC settings; that is, the interrupts capable of waking up the CPU is configurable independent of the other. However, the device exits deep sleep mode (system wakeup signal in Figure 55) as soon as one CPU wakes up. For the list of system interrupts capable of waking up the CPU from deep sleep power mode, refer to Interrupt sources. The interrupt synchronizer block synchronizes the interrupts to the CPU clock frequency as the peripheral interrupts can be asynchronous to the CPU clock frequency.

12.3 Interrupts and exceptions – operation

12.3.1 Interrupt/exception handling

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

1. Assuming that all the interrupt and exception signals are initially low (idle or inactive state) and the processor is executing the main code, a rising edge on any one of the signals is registered by the NVIC, if the interrupt or exception is enabled to be serviced by the CPU. The signal is now in a pending state waiting to be serviced by the CPU.
2. On detecting the 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 81 and Table 82. 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.

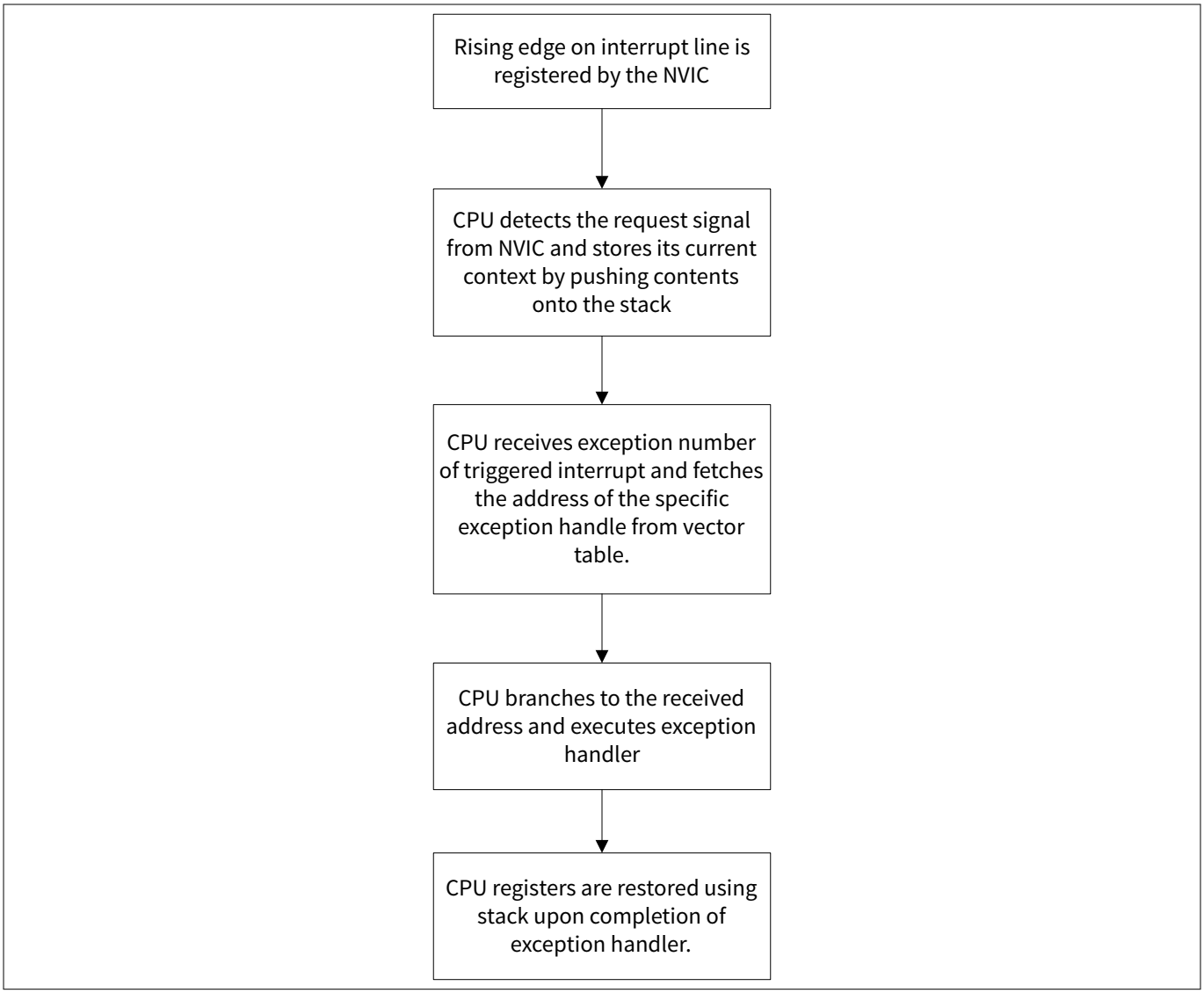


Figure 56 Interrupt handling when triggered

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 as interrupts. Each exception event has a unique exception number, which is used by the CPU to execute the appropriate exception handler.

***Note**: Because multiple system interrupts can be mapped on to the eight CPU interrupts (IRQ[7:0]), identification of system interrupts that triggered the CPU interrupt should be done in the CPU interrupt handler. This is described in* Interrupt sources*.*

12.3.2 Level interrupts

The CM0+ and CM7_0/CM7_1 NVICs support only level signals on the interrupt lines (IRQn). Pulse interrupts are not supported by XMC7000.

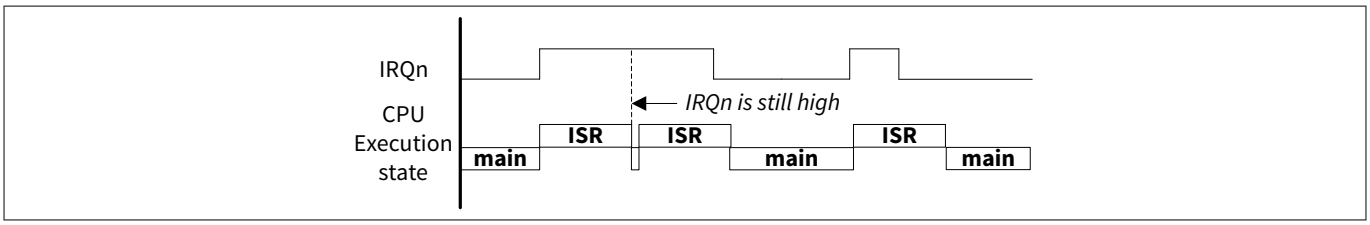


Figure 57 Level interrupts

Figure 57 shows the working of level interrupts. Assuming the interrupt signal is initially inactive (logic low), the following sequence of events explains the handling of level interrupts.

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.

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.

If the interrupt signal is still high after completing the ISR execution, it will be pending and the ISR is executed again. Figure 57 illustrates this for level triggered interrupts, where the ISR is executed as long as the interrupt signal is high.

12.3.3 Exception vector table

The exception vector tables (Table 81 and Table 82) store the entry point addresses for all exception handlers in Cortex®-M0+ and Cortex®-M7 cores. The CPU fetches the appropriate address based on the exception number.

Table 81 Cortex®-M0+ exception vector table

Exception number	Exception	Exception priority	Vector address
–	Initial stack pointer value	Not applicable (NA)	Start_Address = 0x0000 or CM0P_SCS_VTOR[1)]
1	Reset	–3, the highest priority	Start_Address + 0x04
2	Non maskable interrupt (NMI)	–2	Start_Address + 0x08

(table continues...)

Table 81 (continued) Cortex®-M0+ exception vector table

Exception number	Exception	Exception priority	Vector address
3	HardFault	–1	Start_Address + 0x0C
4-10	Reserved	NA	Start_Address + 0x10 to Start_Address + 0x28
11	Supervisory call (SVCall)	Configurable (0 - 3)	Start_Address + 0x2C
12-13	Reserved	NA	Start_Address + 0x30 to Start_Address + 0x34
14	PendSupervisory (PendSV)	Configurable (0 - 3)	Start_Address + 0x38
15	System timer (SysTick)	Configurable (0 - 3)	Start_Address + 0x3C
16	External interrupt (IRQ0)	Configurable (0 - 3)	Start_Address + 0x40
…	…	…	…
23	External interrupt (IRQ7)	Configurable (0 - 3)	Start_Address + 0x5C
24	Internal (software only) Interrupt (IRQ8)	Configurable (0 - 3)	Start_Address + 0x60
…	…	…	…
31	Internal (software only) Interrupt (IRQ15)	Configurable (0 - 3)	Start_Address + 0x7C

1) Start Address = 0x0000 on reset and is later modified by user code by updating the CM0P_SCS_VTOR register.

***Note**: Internal interrupts IRQ8–IRQ15 are not connected to any peripheral and can be triggered by software only.*

Table 82 Cortex®-M7 exception vector table

Exception number	Exception	Exception priority	Vector address
–	Initial stack pointer value	–	Start_Address = 0x0000 or CM7_0/ CM7_1_SCS_VTOR[1)]
1	Reset	–3, highest priority	Start _Address + 0x04
2	Non maskable interrupt (NMI)	–2	Start _Address + 0x08
3	HardFault	–1	Start _Address + 0x0C
4	Memory management fault	Configurable (0 – 7)	Start _Address + 0x10
5	Bus fault	Configurable (0 – 7)	Start _Address + 0x14
6	Usage fault	Configurable (0 – 7)	Start _Address + 0x18
7–10	Reserved	–	–
11	Supervisory call (SVCall)	Configurable (0 – 7)	Start _Address + 0x2C
12–13	Reserved	–	–

(table continues...)

Table 82 (continued) Cortex®-M7 exception vector table

Exception number	Exception	Exception priority	Vector address
14	Pend supervisory (PendSV)	Configurable (0 – 7)	Start _Address + 0x38
15	System tick timer (SysTick)	Configurable (0 – 7)	Start _Address + 0x3C
16	External interrupt (IRQ0)	Configurable (0 – 7)	Start _Address + 0x40
….	….	….	….
23	External interrupt (IRQ7)	Configurable (0 – 7)	Start _Address + 0x5C
24	Internal (software only) Interrupt (IRQ8)	Configurable (0 – 7)	Start _Address + 0x60
….	….	….	….
31	Internal (software only) Interrupt (IRQ15)	Configurable (0 – 7)	Start _Address + 0x7C

1) Start Address = 0x0000 on reset and is later modified by user code by updating the CM7_0/CM7_1_SCS_VTOR register.

***Note**: Internal interrupts IRQ8–IRQ15 are not connected to any peripheral and can be triggered by software only.*

In Table 81 and Table 82, the first word (four 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 XMC7000, both the vector tables can be configured to be located either in flash memory or SRAM. The Vector Table Offset register (VTOR) present as part of Cortex®-M0+ and Cortex®-M7 system control space registers configures the vector table offset from the base address (0x00000000). The CM0P_SCS_VTOR register sets the vector offset address for the CM0+ core and CM7_0/CM7_1_SCS_VTOR sets the offset for the CM7_0/CM7_1 core. The VTOR value determines whether the vector table is in flash memory or SRAM. Refer to the device specific datasheet for the address region of flash and SRAM memories. Note that the VTOR registers can be updated only in privilege CPU mode; refer to the Chip operational modes for details. 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 81 are not used, although they have addresses reserved for them in the vector table. The interrupt sources (exception numbers 16 to 23) are explained in Interrupt sources.

12.4 Exception sources

This section explains the different exception sources listed in Table 81 and Table 82 (exception numbers 1 to 15).

12.4.1 Reset exception

Device reset is treated as an exception in XMC7000. Reset exception is always enabled with a fixed priority of –3, the highest priority exception, in both the cores. When the device boots up, only the Cortex®-M0+ core is available. The CM0+ executes the ROM boot code and can enable Cortex®-M7 core from the application code. The reset exception of the CM0+ is tied to the device reset or startup. When the CM0+ releases the CM7 reset, the CM7_0/CM7_1 reset exception is executed. A device reset can occur due to multiple reasons, such as POR, external reset signal on XRES_L pin, or watchdog reset. When the device is reset, the initial boot code for configuring the device is executed by the CM0+ from 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 CM0+ code execution jumps to flash memory. Flash memory address 0x10000004 (Exception#1 in Table 81) stores the location of the startup code in flash memory. The CPU starts executing code out of this address. Note that the reset exception address in the SRAM vector table will never be used because the device comes out of reset with the flash vector table selected. The register configuration to select the SRAM vector table can be done only as part of the startup code in flash after the reset is de-asserted. Note that the reset exception flow for CM7 is the same as CM0+. However, CM7 execution begins only after CM0+ de-asserts the CM7_0/CM7_1 reset. Refer to Reset system for details about Reset and start-up.

12.4.2 Non-maskable interrupt exception

Non-maskable interrupt (NMI) is the highest priority exception next to reset. It is always enabled with a fixed priority of –2. Both the cores have their own NMI exception. There are two ways to trigger an NMI exception in a CPU core:

- **NMI exception from a system interrupt:** Both CM0+ and CM7 provide an option to trigger an NMI exception using four of the available system interrupts for each core. The NMI exception triggered due to the interrupt will execute the NMI handler pointed to by the active vector table. The four CPUSS_CMx_NMI_CTL registers per CPU select the system interrupt sources that can trigger the NMI from hardware.
- **NMI exception by setting NMIPENDSET bit (user NMI exception):** An NMI exception can be triggered in software by setting the NMIPENDSET bit in the interrupt control state registers (CM0P_SCS_ICSR and CM7_0/CM7_1_SCS_ICSR). Setting this bit will execute the NMI handler pointed to by the active vector table in the respective CPU cores.

12.4.3 HardFault exception

Both CM0+ and CM7_0/CM7_1 cores support 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 over 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 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.

12.4.4 Memory management fault exception

A memory management fault is an exception that occurs because of a memory protection-related fault. The fixed memory protection constraints determine this fault, for both instruction and data memory transactions.

This fault is always used to abort instruction accesses to execute never (XN) memory regions. The memory management fault is only supported by the CM7_0/CM7_1 core. The priority of the exception is configurable from 0 (highest) to 7 (lowest).

12.4.5 Bus fault exception

A bus fault is an exception that occurs because of a memory-related fault for an instruction or data memory transaction. This can be from an error detected on a bus in the memory system. The bus fault is only supported by the CM7_0/CM7_1 core. The priority of the exception is configurable from 0 (highest) to 7 (lowest).

12.4.6 Usage fault exception

A usage fault is an exception that occurs because of a fault related to instruction execution. This includes:

- An undefined instruction
- An illegal unaligned access
- Invalid state on instruction execution
- An error on exception return

The following can cause a usage fault when the core is configured to report them:

- An unaligned address on word and halfword memory access
- Division by zero.

The usage fault is only supported by the CM7_0/CM7_1 core. The priority of the exception is configurable from 0 (highest) to 7 (lowest).

12.4.7 Supervisor call (SVCall) exception

Both CM0+ and CM7_0/CM7_1 cores support 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.

The priority of a SVCall exception can be configured to a value between 0 and 3 for CM0+ and 0 to 7 for CM7_0/CM7_1 core by writing to the bit fields PRI_11 of the System Handler Priority register 2 (CM0P_SCS_SHPR2 and CM7_0/CM7_1_SCS_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_SCS_SHCSR and CM7_0/CM7_1_SCS_SHCSR) can be used to check or modify the pending status of the SVCall exception.

12.4.8 PendSV exception

Both CM0+ and CM7_0/CM7_1 cores support 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 similar to SVCall. The PendSV exception is triggered by setting the PENDSVSET bit in the Interrupt Control State register (CM0P_SCS_ICSR and CM7_0/CM7_1_SCS_ICSR). On setting this bit, the PendSV exception enters the pending state, and waits to be serviced by the CPU. The pending state of a PendSV exception can be cleared by setting the PENDSVCLR bit in the Interrupt Control State register. The priority of a PendSV exception can be configured to a value between 0 and 3 for CM0+ and 0 to 7 for CM7_0/CM7_1 by writing to the bit fields PRI_14 of the System Handler Priority register 3. See the Arm® v6-M architecture reference manual for more details.

12.4.9 SysTick exception

Both CM0+ and CM7_0/CM7_1 cores in XMC7000 support a system timer, referred to as SysTick, as part of their internal architecture. SysTick provides a simple, 24-bit decrementing counter for various timekeeping purposes such as an RTOS tick timer, high-speed alarm timer, or simple counter. The SysTick timer can be configured to generate an interrupt when its count value reaches zero, which is referred to as SysTick exception. The exception is enabled by setting the TICKINT bit in the SysTick Control and Status register (CM0P_SCS_SYST_CSR and CM7_0/CM7_1_SCS_SYST_CSR). The priority of a SysTick exception can be configured to a value between 0 and 3 for CM0+ and 0 to 7 for CM7_0/CM7_1 by writing to the bit fields PRI_15 of the System Handler Priority register 3 (SHPR3). The SysTick exception can always be generated in software by writing a one to the PENDSTSET bit in the Interrupt Control State register (ICSR). Similarly, the pending state of the SysTick exception can be cleared by writing a one to the PENDSTCLR bit in the ICSR.

***Note**: The SysTick clock source can be configured through SYSTICK_CTL register in the CPUSS.*

12.5 Interrupt sources

The XMC7000 family supports up to 1023 system interrupts from peripherals. However, the available system interrupts depend on the device variant. Check the device datasheet to know the list of system interrupt sources supported by the device variant.

The CM0+ CPU supports a maximum of 32 CPU interrupts (IRQ[31:0]) and the CM7_0/CM7_1 CPU supports a maximum of 240 CPU interrupts (IRQ[239:0]). To allow the support of up to 1023 system interrupts by the Cortex®-M7 and M0+ CPUs, an interrupt reduction functionality is used. The interrupt reduction functionality allows each system interrupt to be mapped onto one out of the eight external CPU interrupts (IRQ[7:0]). Multiple system interrupts can be mapped on the same CPU interrupt. Therefore, an active CPU interrupt may indicate one or multiple active system interrupts.

The interrupt controller logic is independent for each CPU and each system interrupt has an associated CM0/CM7_0/7_1_SYSTEM_INT_CTL register:

- CM0/CM7_0/7_1_SYSTEM_INT_CTL.CPU_INT_VALID configures if the system interrupt is enabled for the CPU.
- CM0/CM7_0/7_1_SYSTEM_INT_CTL.CPU_INT_IDX[2:0] configures on which CPU interrupt the system interrupt is mapped.

Typically, the CPU uses different priority levels for the different CPU interrupts and will map system interrupts to CPU interrupts accordingly (all system interrupts that are mapped on the same CPU interrupt have the same priority). In addition to the eight (external) hardware CPU interrupts (IRQ[7:0]), eight (internal) software CPU interrupts are supported (IRQ[15:8]).

As a result of the reduction functionality, multiple system interrupts share a CPU interrupt handler as provided by the CPU's VTOR table. Each CPU interrupt has an associated CM0/CM7_0/7_1_INT_STATUS register:

- CM0+/ CM7_0/CM7_1_INT_STATUS.SYSTEM_INT_VALID specifies if any system interrupt is active for the CPU interrupt.
- CM0+/ CM7_0/CM7_1_INT_STATUS.SYSTEM_INT_IDX[9:0] specifies the index (a number in the range [0, 1022]) of the lowest active system interrupt mapped to the corresponding CPU interrupt.

The CPU interrupt handler uses the SYSTEM_INT_IDX field to index a system interrupt lookup table and jumps to the system interrupt handler. The lookup table is typically located in one of the system memories. Note that

this scenario introduces a two step approach: a CPU interrupt handler followed by a system interrupt handler. The following code illustrates the approach:

```
void CM7_0/CM7_1_CpuIntr0_Handler (void)
{
    uint32_t    system_int_idx;
    SystemIntr_Handler handler;

    if(CPUSS_CM7_0/CM7_1_INT_STATUS[0].SYSTEM_INT_VALID)
    {
        system_int_idx = CPUSS_CM7_0/CM7_1_INT_STATUS[0].SYSTEM_INT_IDX;
        handler = SystemIntr_Table[system_int_idx];
        handler(); // jump to system interrupt handler
    }
    else
    {
        // Triggered by software or because software cleared a peripheral interrupt flag
        // but did not clear the Pending flag at NVIC
    }
}
…
void CM7_0/CM7_1_CpuIntr7_Handler (void)
{
    uint32_t      system_int_idx;
    SystemIntr_Handler handler;

    if(CPUSS_CM7_0/CM7_1_INT_STATUS[7].SYSTEM_INT_VALID)
    {
    system_int_idx = CPUSS_CM7_0/CM7_1_INT_STATUS[7].SYSTEM_INT_IDX;
    handler = SystemIntr_Table[system_int_idx];
    handler(); // jump to system interrupt handler
    }
    else
    {
        // Triggered by software or because software cleared a peripheral interrupt flag
        // but did not clear the Pending flag at NVIC
    }
}

void CM7_0/CM7_1_SystemIntr0_Handler (void)
{
        // Clear the peripheral interrupt request flag by register write
        // Read back the register, to ensure completion of register write access
        // Handle system interrupt 0.
}
…
void CM7_0/CM7_1_SystemIntr1022_Handler (void)
{
        // Clear the peripheral interrupt request flag by register write
        // Read back the register, to ensure completion of register write access
        // Handle system interrupt 1022.
}
```

The system interrupts include standard interrupts from the on-chip peripherals such as TCPWM, serial communication block, CSD block, watchdog, ADC and so on. The interrupt generated is usually the logical OR of the different peripheral states. The Peripheral Interrupt 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 Interrupt Status register be read in the ISR to clear the interrupt. If the Interrupt 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 for details on GPIO interrupts.

12.6 Exception priority

Exception priority is useful for exception arbitration when there are multiple exceptions that need to be serviced by the CPU. Both CM7_0/CM7_1 and CM0+ cores in XMC7000 provide flexibility in choosing priority values for different exceptions. All exceptions other than Reset, NMI, and HardFault can be assigned a configurable priority level. The Reset, NMI, and HardFault exceptions have a fixed priority of –3, –2, and –1 respectively. In XMC7000, 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 for Cortex®-M0+ and 0 to 7 for Cortex®-M7.

Both CM0+ and CM7_0/CM7_1 support 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 or lower than the active exception. The CPU resumes execution of the lower priority exception handler after servicing the higher priority exception. The CM0+ core in XMC7000 allows nesting of up to four exceptions; the CM7_0/CM7_1 core allows up to eight exceptions. When the CPU receives two or more exceptions requests of the same priority, the lowest exception number is serviced first.

The registers to configure the priority of exception numbers 1 to 15 are explained in Exception sources.

The priority of the eight CM0+ and eight CM7_0/CM7_1 interrupts can be configured by writing to the respective Interrupt Priority registers (CM0P_SCS_IPR and CM7_0/CM7_1_SCS_NVIC_IPR). This is a group of eight (CM0+) and sixty (CM7_0/CM7_1) 32-bit registers with each register storing the priority values of four interrupts, as given in Table 83 and Table 84.

Table 83 Interrupt Priority register bit definitions for Cortex®-M0+ (CM0P_SCS_IPR)

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

Table 84 Interrupt Priority register bit definitions for Cortex®-M7 (CM7_0/CM7_1_SCS_NVIC_IPR)

Bits	Name	Description
7:5	PRI_N0	Priority of interrupt number N
15:13	PRI_N1	Priority of interrupt number N+1
23:21	PRI_N2	Priority of interrupt number N+2
31:29	PRI_N3	Priority of interrupt number N+3

12.7 Enabling and disabling interrupts

The NVICs of both CM0+ and CM7_0/CM7_1 core provide registers to individually enable and disable the CPU interrupts in software. If an interrupt is not enabled, the NVIC will not process the interrupt requests on that

interrupt line. The Interrupt Set-Enable register (CM0P_SCS_ISER and CM7_0/CM7_1_SCS_NVIC_ISER) and the Interrupt Clear-Enable register (CM0P_SCS_ICER and CM7_0/CM7_1_SCS_NVIC_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 in CM0+. For CM7_0/CM7_1 core, there are eight ISER/ICER registers. These registers can also be read in software to get the enable status of the interrupts. Table 85 shows the register access properties for these two registers. Note that writing zero to these registers has no effect.

Table 85 Interrupt Enable/Disable registers

Register	Operation	Bit value	Comment
Interrupt Set Enable register	Write	1	To enable the interrupt
		0	No effect
	Read	1	Interrupt is enabled
		0	Interrupt is disabled
Interrupt Clear Enable register	Write	1	To disable the interrupt
		0	No effect
	Read	1	Interrupt is enabled
		0	Interrupt is disabled

The Interrupt Set Enable register (ISER) and Interrupt Clear-Enable register (ICER) registers are applicable only for the interrupts. 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.

The Priority Mask (PRIMASK) register in the CPUs (both CM0+ and CM7_0/CM7_1) 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 81. When the PRIMASK.PM bit 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 PRIMASK.PM bit is cleared.

12.8 Exception states

Each exception can be in one of the following states.

Table 86 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.

The Interrupt Control and State register (CM0P_SCS_ICSR and CM7_0/CM7_1_SCS_ICSR) contains status bits describing the various exceptions states.

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

12.8.1 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 eight (CM0+/CM7_0/CM7_1) interrupt lines by providing separate register bits to set and clear the pending states of the interrupts. The Interrupt Set-Pending registers (CM0P_SCS_ISPR and CM7_0/CM7_1_SCS_NVIC_ISPR) and the Interrupt Clear-Pending register (CM0P_SCS_ICPR and CM7_0/CM7_1_SCS_NVIC_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 87 shows the register access properties for these two registers. Note that writing zero to these registers has no effect.

Table 87 Interrupt Set Pending/Clear Pending registers

Register	Operation	Bit value	Comment
Interrupt Set-Pending Register (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 (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 ISER.

Note that the ISPR and ICPR are used only for the peripheral interrupts. 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.

12.9 Stack usage for exceptions

When the CPU executes the main code (in thread mode) and an exception request occurs, the CPU stores the state of its General-Purpose registers in the stack. It then starts executing the corresponding exception handler (in handler mode). The CPU pushes the contents of the eight 32-bit internal registers into the stack. These registers are the Program and Status Register (PSR), ReturnAddress, Link register (LR or R14), R12, R3, R2, R1, and R0. Both Cortex®-M7 and 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 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 CPU subsystem (CPUSS) for details.

12.10 Interrupts and low-power modes

XMC7000 allows device (CPU) wakeup from low-power modes when certain peripheral interrupt requests are generated. The WIC block generates a wakeup signal that causes the CPU 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) or wait for event (WFE) instructions executed by the CPU triggers the transition into sleep and deep sleep modes. Only the WFI instruction is meant for waking up using interrupts. The WFE instruction puts the CPU to sleep based on the status of an event bit and wakes up from an event signal, typically sent by the other CPU. The sequence to enter the different low-power modes is detailed in the Device power modes. Device low-power modes have two categories of interrupt sources:

- Interrupt sources that are available in the active, sleep, and deep sleep modes (see the Device power modes for the available sources)
- Interrupt sources that are available only in the active and sleep modes

12.11 Exception – initialization and configuration

This section discusses the steps to initialize and configure exceptions in XMC7000.

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 described in Exception vector table.

   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, as explained in earlier sections.
   - **a.** Configure the exception or interrupt source; this includes setting up the interrupt generation conditions. The register configuration depends on the specific exception required. Refer to the respective peripheral chapter to know more about the interrupt configuration supported by them.
   - **b.** Define the exception handler function and write the address of the function to the exception vector table. Table 81 gives the exception vector table format; the exception handler address should be written to the appropriate exception number entry in the table.
   - **c.** For system interrupts, define the system interrupt handler function, specify to which CPU interrupt the system interrupt is to be mapped in CM0/CM7_0/CM7_1_SYSTEM_INT_CTL.CPU_INT_IDX[2:0] and enable the system interrupt by setting CM0/CM7_0/CM7_1_SYSTEM_INT_CTL.CPU_INT_VALID. The CPU interrupt handler function should check the CM0/CM7_0/CM7_1_INT_STATUS.SYSTEM_INT_IDX[9:0] to determine the system interrupt that caused the interrupt and call the corresponding system interrupt handler function.
   - **d.** Set up the exception priority, as explained in Exception priority.
   - **e.** Enable the exception, as explained in Enabling and disabling interrupts.

12.12 Registers

Table 88 Register list

Register	Name	Description
CPUSS_CM7_0/ CM7_1_INTx_STATUS	CM7_0/CM7_1 CPU Interrupt Status register	The CPUSS_CM7_0/CM7_1_INT0_STATUS – CPUSS_CM7_0/CM7_1_INT7_STATUS registers provide the lowest CM7_0/CM7_1-activated system interrupt index for the eight external CPU interrupts.
CPUSS_CM7_0/ CM7_1_VECTOR_TABLE_BASE	CM7_0/CM7_1 Vector Table Base register	Address of CM7_0/CM7_1 vector table. This register is used for CM7_0/CM7_1 warm and cold boot purposes: the CM0+ CPU initializes the CM7_0/CM7_1_VECTOR_TABLE_BASE register and the CM7_0/CM7_1 boot code uses the register to initialize the CM7_0/CM7_1 Internal VTOR register.
CPUSS_CM7_0/CM7_1_NMI_CTLx	CM7_0/CM7_1 NMI Control register	The CPUSS_CM7_0/CM7_1_NMI_CTL0 – CPUSS_CM7_0/CM7_1_NMI_CTL3 registers allow connecting four system interrupts to the NMI. The four selected system interrupts are logically OR'd into a single CM7_0/CM7_1 NMI input.
CPUSS_CM0_INTx_STATUS	CM0+ CPU Interrupt Status register	The CPUSS_CM0_INT0_STATUS – CPUSS_CM0_INT7_STATUS registers provide the lowest CM0-activated system interrupt index for the eight external CPU interrupts.
CPUSS_CM0_VECTOR_TABLE_BASE	CM0+ Vector Table Base register	Address of CM0+ vector table. This register is used for CM0+ warm boot purposes: the CM0+ warm boot code uses the register to initialize the CM0+ Internal VTOR register.
CPUSS_CM0_NMI_CTLx	CM0+ NMI Control register	The CPUSS_CM0_NMI_CTL0 – CPUSS_CM0_NMI_CTL3 registers allow connecting four system interrupts to the CM0+ NMI. The four selected system interrupts are logically OR'd into a single CM0+ NMI input.
CPUSS_CM0_SYSTEM_INT_CTLx	CM0+ System Interrupt Control register	These registers are used to configure the mapping of system interrupt "x" to one of the eight external CM0+ CPU interrupts.
CPUSS_CM7_0/ CM7_1_SYSTEM_INT_CTLx	CM7_0/CM7_1 System Interrupt Control register	These registers are used to configure the mapping of system interrupt "x" to one of the eight external CM7_0/CM7_1 CPU interrupts.
CM0P_SCS_ISER[1)]	Cortex®-M0+ Interrupt Set-Enable register	The CM0P_SCS_ISER enables CM0+ external and internal (software only) interrupts, and shows which interrupts are enabled.
CM0P_SCS_ICER	Cortex®-M0+ Interrupt Clear Enable register	The CM0P_SCS_ICER disables CM0+ external and internal (software only) interrupts, and shows which interrupts are enabled.

(table continues...)

Table 88 (continued) Register list

Register	Name	Description
CM0P_SCS_ISPR	Cortex®-M0+ Interrupt Set-Pending register	The CM0P_SCS_ISPR forces CM0+ external and internal (software only) interrupts into the pending state, and shows which interrupts are pending.
CM0P_SCS_ICPR	Cortex®-M0+ Interrupt Clear-Pending register	The CM0P_SCS_ICPR removes the pending state from CM0+ external and internal (software only) interrupts, and shows which interrupts are pending.
CM0P_SCS_IPRx	Cortex®-M0+ Interrupt Priority registers	The CM0P_SCS_IPR registers allow to configure the priority for the CM0+ external and internal (software only) interrupts.
CM0P_SCS_ICSR	Cortex®-M0+ Interrupt Control and State register	The CM0P_SCS_ICSR provides: • A set-pending bit for the non-maskable interrupt (NMI) exception • Set-pending and clear-pending bits for the PendSV and SysTick exceptions The register also indicates: • The number of the highest priority pending exception
CM0P_SCS_VTOR	Cortex®-M0+ Vector Table Offset register	The CM0P_SCS_VTOR indicates the offset of the CM0+ vector table base address from memory address 0x00000000.
CM0P_SCS_AIRCR	Cortex®-M0+ Application Interrupt and Reset Control register	The CM0P_SCS_AIRCR provides endian status for CM0+ data accesses and reset control of the system.
CM0P_SCS_SHPR2	Cortex®-M0+ System Handler Priority register 2	The CM0P_SCS_SHPR2 allows to configure the priority for SVCall exception.
CM0P_SCS_SHPR3	Cortex®-M0+ System Handler Priority register 3	The CM0P_SCS_SHPR3 allows to configure the priority for SysTick and PendSV exceptions.
CM0P_SCS_SHCSR	Cortex®-M0+ System Handler Control and State register	The CM0P_SCS_SHCSR controls and provides the active and pending status of CM0+ exceptions.
CM7_0/CM7_1_SCS_NVIC_ISERx[2)]	CM7_0/CM7_1 Interrupt Set-Enable registers	The CM7_0/CM7_1_SCS_NVIC_ISER registers enable CM7_0/CM7_1 external and internal (software only) interrupts, and show which interrupts are enabled.

(table continues...)

Table 88 (continued) Register list

Register	Name	Description
CM7_0/CM7_1_SCS_NVIC_ICERx	CM7_0/CM7_1 Interrupt Clear Enable registers	The CM7_0/CM7_1_SCS_NVIC_ICER registers disable CM7_0/CM7_1 external and internal (software only) interrupts, and show which interrupts are enabled.
CM7_0/CM7_1_SCS_NVIC_ISPRx	CM7_0/CM7_1 Interrupt Set-Pending registers	The CM7_0/CM7_1_SCS_NVIC_ISPR registers force CM7_0/CM7_1 external and internal (software only) interrupts into the pending state, and show which interrupts are pending.
CM7_0/CM7_1_SCS_NVIC_ICPRx	CM7_0/CM7_1 Interrupt Clear-Pending registers	The CM7_0/CM7_1_SCS_NVIC_ICPR registers remove the pending state from CM7_0/CM7_1 external and internal (software only) interrupts, and shows which interrupts are pending.
CM7_0/CM7_1_SCS_NVIC_IABRx	CM7_0/CM7_1 Interrupt Active Bit registers	The CM7_0/CM7_1_SCS_NVIC_IABR registers indicate which CM7_0/CM7_1 external and internal (software only) interrupts are active.
CM7_0/CM7_1_SCS_NVIC_IPRx	CM7_0/CM7_1 Interrupt Priority registers	The CM7_0/CM7_1_SCS_NVIC_IPR registers allow to configure the priority for the CM7_0/CM7_1 external and internal (software only) interrupts.
CM7_0/CM7_1_SCS_STIR	CM7_0/CM7_1 Software Triggered Interrupt register	The CM7_0/CM7_1_SCS_STIR allows to generate CM7_0/CM7_1 external and internal (software only) interrupts from software. This register has the same function as CM7_0/ CM7_1_SCS_NVIC_ISPR except that STIR can be configured to allow access by unprivileged software.
CM7_0/CM7_1_SCS_ICSR	CM7_0/CM7_1 Interrupt Control State register	The CM7_0/CM7_1_SCS_ICSR provides: • A set-pending bit for the NMI exception • Set-pending and clear-pending bits for the PendSV and SysTick exceptions This register also indicates: • The exception number of the exception being processed • Whether there are preempted active exceptions • The exception number of the highest priority pending exception • If any interrupts are pending
CM7_0/CM7_1_SCS_VTOR	CM7_0/CM7_1 Vector Table Offset register	The CM7_0/CM7_1_SCS_VTOR indicates the offset of the CM7_0/CM7_1 vector table base address from memory address 0x00000000.

(table continues...)

Table 88 (continued) Register list

Register	Name	Description
CM7_0/CM7_1_SCS_AIRCR	CM7_0/CM7_1 Application Interrupt and Reset Control register	The CM7_0/CM7_1_SCS_AIRCR provides priority grouping control for the exception model, endian status for data accesses of CM7_0/CM7_1, and reset control of the system.
CM7_0/CM7_1_SCS_SHPR1	CM7_0/CM7_1 System Handler Priority register 1	The CM7_0/CM7_1_SCS_SHPR1 allows to configure the priority for UsageFault, BusFault, and MemManage exceptions.
CM7_0/CM7_1_SCS_SHPR2	CM7_0/CM7_1 System Handler Priority register 2	The CM7_0/CM7_1_SCS_SHPR2 allows to configure the priority for SVCall exception.
CM7_0/CM7_1_SCS_SHPR3	CM7_0/CM7_1 System Handler Priority register 3	The CM7_0/CM7_1_SCS_SHPR3 allows to configure the priority for SysTick and PendSV exceptions.
CM7_0/CM7_1_SCS_SHCSR	CM7_0/CM7_1 System Handler Control and State register	The CM7_0/CM7_1_SCS_SHCSR enables the system handlers, and indicates: • The pending status of the BusFault, MemManage fault, and SVC exceptions • The active status of the system handlers
CPUSS_SYSTICK_CTL	SysTick Timer Control register	The CPUSS_SYSTICK_CTL register allows to configure the SysTick timer clock source, specify the clock source precision, and the number of clock source cycles that make up 10 ms. ***Note**: If an external clock source is configured using this register, the external clock frequency must be less than the CPU internal clock frequency.*

1) Refer to the Arm® Cortex®-M0+ TRM for details about this register.
2) Refer to the Arm® Cortex®-M7 TRM for details about this register.

13 Device security

XMC7000 offers several features to protect user designs from unauthorized access or copying. Selecting a secure life-cycle stage, enabling memory and peripheral protection, configuring flash write and eFuse read/write protection, and using hardware-based cryptography can provide a high level of security.

13.1 Features

The XMC7000 provides the following device security features:

- Nonvolatile and irreversible life-cycle stages that can limit program and debug access.
- Memory protection units (MPU), shared memory protection units (SMPU), and peripheral protection units (PPU) provide memory and peripheral protection, such as preventing unauthorized reading of sensitive data.
- Software protection units (SWPU) that define flash write (or erase) permissions and eFuse read and write permissions.
- A cryptographic function block that provides hardware-based encryption and decryption of data and code.

13.2 How it works

13.2.1 Life-cycle stages

XMC7000 devices have configurable, nonvolatile life-cycle stages. Life-cycle stages follow a strict, irreversible progression governed by invoking system management APIs that will change the one-time programmable (OTP) eFuse settings accordingly.

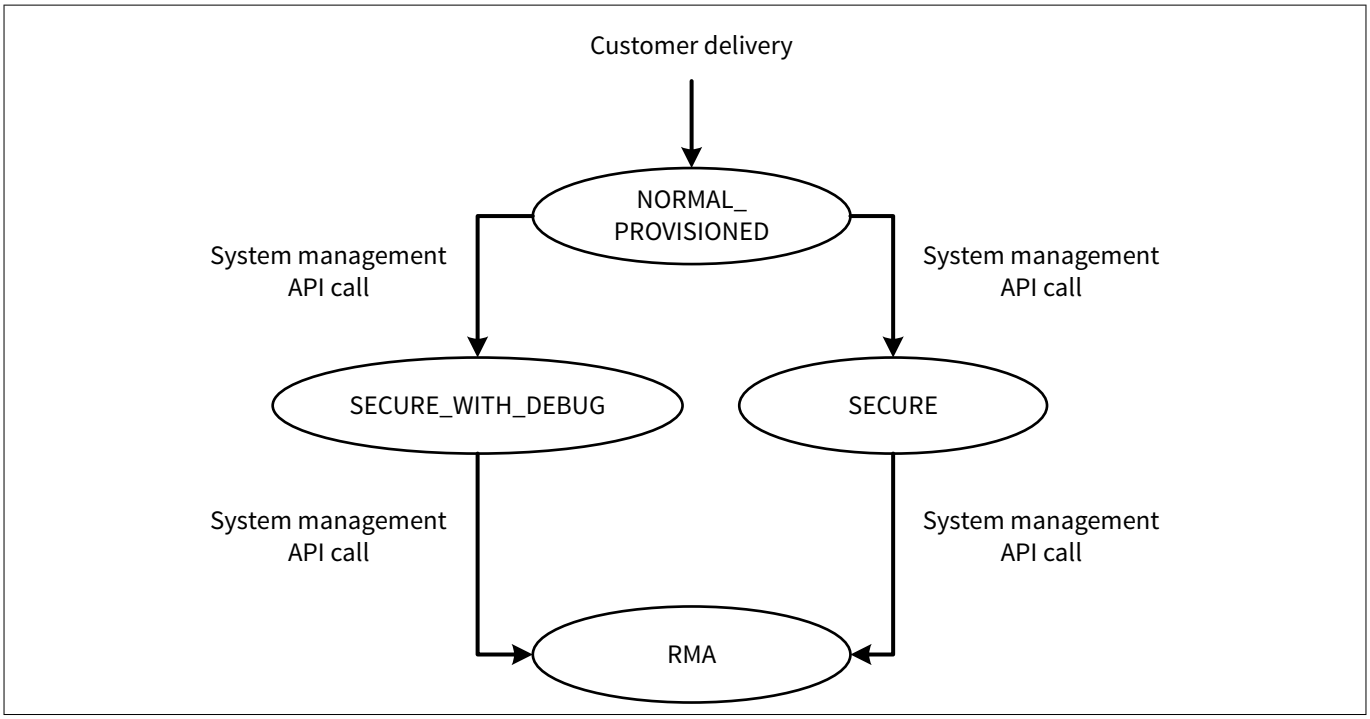


Figure 58 XMC7000 life cycle stage transitions

XMC7000 supports the following life-cycle stages:

- NORMAL_PROVISIONED – Customers receive parts in this stage.
- SECURE – You can secure the device in this stage after the application has been created and tested. A secure device will boot only when there is a successful authentication of its flash boot code and application code. Access restrictions in SECURE mode are controlled by eFuse settings.
- SECURE_WITH_DEBUG – This is similar to the SECURE life-cycle stage, except with NORMAL access restrictions applied to enable debugging, even if authentication fails. Devices that are in this stage are only used by developers and testers.
- RMA – Devices can be brought into this stage so that Infineon can perform a failure analysis. Sensitive data should be erased before transitioning to this life-cycle stage. The boot process will set access restrictions such that only the “Open for RMA” system management API call can be executed from outside; this requires a part-specific certificate provided by the customer.
- CORRUPTED (not shown in state diagram) – This stage is entered in case an error is detected when the boot process tries to determine the current life-cycle stage.

The current nonvolatile life-cycle stage as well as the volatile protection state can be retrieved with the SiliconID system call. The protection state is also available in the CPUSS_PROTECTION Register.

The following table shows the mapping of life-cycle stages to the protection states.

Table 89 Life-cycle stage mapping

Life-cycle stage	Protection state
VIRGIN[1)]	VIRGIN
SORT[1)]	
PROVISIONED[1)]	
RMA	
NORMAL[1)]	NORMAL
NORMAL_PROVISIONED	
SECURE	SECURE
SECURE_WITH_DEBUG	
Any of the above stages (on certain conditions)	DEAD
CORRUPTED	

1) These life-cycle stages are not applicable for final samples.

13.2.2 Memory and peripheral protection

The MPU, SMPU, and PPU can be used to restrict access to memory (RAM and flash) or peripheral address space. This can prevent unauthorized code or bus masters from reading/writing sensitive address areas.

For more details, see the Protection unit.

13.2.3 Flash write and eFuse read/write protection

XMC7000 devices include software protection units (SWPU), which define permissions for flash writing (or erasing) and eFuse reading and writing. This feature prevents malicious or inadvertent modification of flash or eFuse, or reading of sensitive eFuse data. In addition, unauthorized changes to the application are detected by the secure boot operation.

For more details, see the Protection unit.

13.2.4 Hardware-based cryptography

XMC7000 has a cryptographic block (Crypto) that provides hardware implementation and acceleration of cryptographic functions. It implements symmetric key encryption and decryption, hashing, message authentication, random number generation (pseudo and true), cyclic redundancy checking, and hardware acceleration of asymmetric cryptography. See the Cryptography block.

14 Chip operational modes

XMC7000 is capable of executing firmware in four different modes. These modes dictate execution from different locations in flash and ROM, with different levels of hardware privileges. Only three of these modes are used in end-applications; debug mode is used exclusively to debug designs during firmware development. This chapter gives an overview of the XMC7000 operational modes. The device power modes are explained in the Device power modes. These modes are independent of privileged and unprivileged access levels of Arm® Cortex® core.

The operational modes in XMC7000 are:

- Boot
- User
- Trusted
- Debug

14.1 Boot

In the boot mode the device is configured by instructions hard-coded in the device ROM and from supervisory flash. 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 so that the boot firmware can set up the device for operation without being interrupted. During boot mode, hardware trim settings are loaded from flash to guarantee proper operation during power-up. After executing ROM boot code, supervisory flash boot code execution begins after flash boot authentication. ROM boot is the root of trust as it is immutable. Flash boot is more flexible and can be modified during the VIRGIN life cycle. However it is treated as an extension of ROM boot because it is authenticated by the ROM boot. After both ROM boot and flash boot, the device enters user mode and code execution from user flash begins. See the BootROM for the details of ROM boot and flash boot.

14.2 User

In the user mode normal user firmware from flash is executed. User mode cannot execute code from ROM. The boot process transfers control to this mode after it has completed its tasks. Then the user application starts from the default user application address. Both privileged and unprivileged access levels of Arm® Cortex® core can be executed in the user mode.

14.3 Trusted

Trusted mode allows execution of special subroutines that are stored in the device ROM. These subroutines cannot be modified by the user and are used to execute proprietary code that is not meant to be interrupted or observed. Debugging is not allowed in the trusted mode.

This mode is entered from user mode by executing the system call (ROM API code). Trusted ROM code can be executed only when the master is in protection context 1. Only the CM0+ (secure CPU) can attain protection context 1 upon a trusted interrupt handler entry. See the Device security for more details on protection contexts. Exit from this mode returns the device to user mode.

14.4 Debug

Debug mode allows observation of the XMC7000 operational parameters. This mode is used to debug firmware during development. The debug mode is entered when a debugger connects to the device during the acquire time window, which occurs during device reset. Debug mode allows IDEs to debug the firmware. This mode is available only on devices whose access restriction settings allow debugging. For NORMAL protection state, access restrictions settings are stored in the supervisory flash (SFlash). For DEAD and SECURE states, it is stored in eFuse. For more details on protection states, see the Device security. For more details on the debug interface, see the Program and debug interface.

15 Fault subsystem

The fault subsystem contains information about faults that occur in the system. The subsystem can cause a reset, give a pulse indication, or trigger another peripheral. The XMC7000 platform uses a centralized fault report structure. The centralized nature allows for a system-wide, consistent handling of faults, which simplifies software development as follows:

- Only a single fault interrupt handler is required
- The fault report structure provides the fault source and additional fault-specific information from a single set of memory mapped input/output (MMIO) registers; that is, no iterative search is required for the fault source and fault information
- All pending faults are available from a single set of MMIO registers

The fault subsystem captures faults related to, but not limited to:

- MPU/SMPU/PPU protection violations
- Peripheral-specific errors
- Memory controller specific errors, such as SRAM controller ECC errors, flash controller “read-while-program”, and ECC errors
- Processor tightly-coupled memory (TCM) ECC errors
- Timeout errors

Note that some of the above faults also result in errors on the bus infrastructure. These faults are communicated in two ways:

- As a bus error to the master of the faulting bus transfer
- As a fault in a fault report structure. This fault can be communicated as a fault interrupt to any processor in the system. This allows fault handling on a processor that is not the master of the faulting bus transfer. It is useful for faults that cause the master of the faulting transfer to become unresponsive or unreliable

The fault subsystem only captures faults. It does not take any action to correct it.

15.1 Fault report structure

Figure 59 gives an overview of the fault report structure.

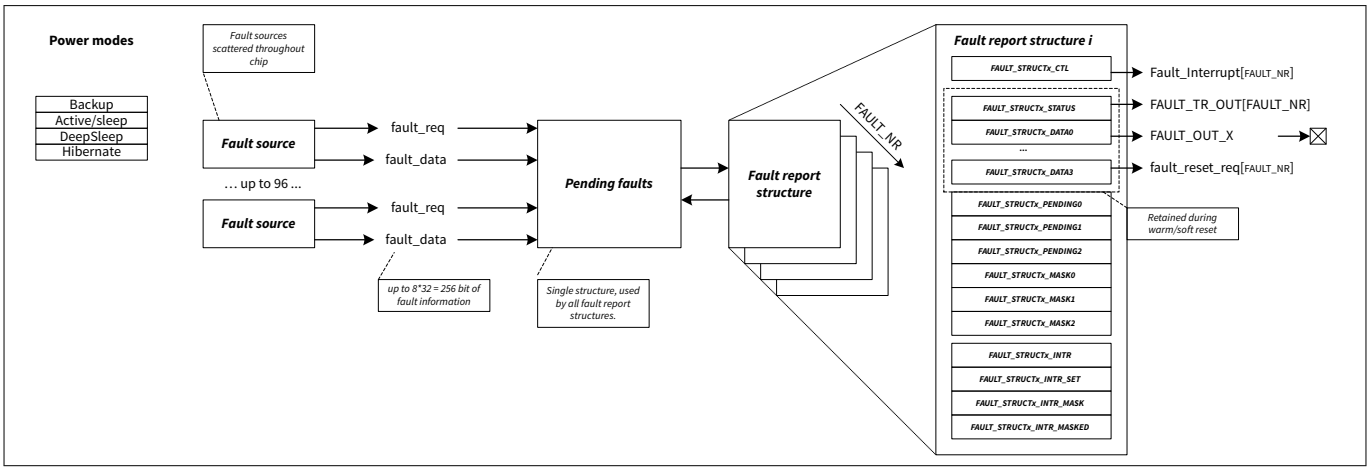


Figure 59 Fault reporting structure

Refer to the device datasheet for information about the number of fault report structures (FAULT_NR) supported. Each structure has a dedicated set of control and Status registers, and captures a single fault. The captured fault information includes:

- A validity bit field that indicates a fault is captured (FAULT_STRUCTx_STATUS.VALID).

- A fault index that identifies the fault source (FAULT_STRUCTx_STATUS.IDX).
- Additional fault information describing fault specifics (FAULT_STRUCTx_DATAy). This additional information is fault type-specific. Most fault types use only a few of the FAULT_STRUCTx_DATAy registers. For example, an MPU protection violation provides information on the violating bus address, bus master identifier, and bus access control information in only two FAULT_STRUCTx_DATAy registers.

In addition to the captured fault information, each fault report structure supports a signaling interface to notify the rest of the system of the captured fault. This interface supports the following:

- A fault interrupt (interrupts_fault). This interrupt is supported by the platform Interrupt registers: FAULT_STRUCTx_INTR, FAULT_STRUCTx_INTR_SET, FAULT_STRUCTx_INTR_MASK, and FAULT_STRUCTx_INTR_MASKED. Only a single interrupt cause is present: FAULT (indicating that a fault is detected). The FAULT_STRUCTx_INTR_MASK register provides a mask/enable for the cause. The interrupt cause is set to '1' when a fault is captured.
- A trigger (FAULT_TR_OUT[FAULT_NR]). An enabled trigger is activated (generating a two-cycle '1' pulse) when FAULT_STRUCTx_STATUS.VALID is set to '1'. The trigger is enabled by FAULT_STRUCTx_CTL.TR_EN. The trigger can be connected to a DMA controller, for example, which can transfer captured fault information from the fault report structure to memory and can clear the FAULT_STRUCTx_STATUS.VALID field. For failure analysis, a memory location that is retained during warm/soft reset is desirable.
- An output signal (FAULT_OUT_x, x = 0, 1, 2, 3). An enabled output signal is active/'1' when FAULT_STRUCTx_STATUS.VALID is '1'. The output signal is enabled by FAULT_STRUCTx_CTL.OUT_EN. It can be used to communicate non-recoverable faults, for example, to off-chip components (possibly resulting in a deice reset).
- A fault reset request (fault_reset_req). An enabled request is active/'1' when FAULT_STRUCTx_STATUS.VALID is '1'. The request is enabled by FAULT_STRUCTx_CTL.RESET_REQ_EN. The reset request feeds into the logic that generates a warm/soft reset.

The four different signaling interfaces provided have their own 'enable' functionality. Each enabled interface is activated when FAULT_STRUCTx_STATUS.VALID is '1'.

As the system resources subsystem (SRSS) has a single fault_reset_req input signal, the individual fault_reset_req[i] signals are combined (logical OR'd) into a single fault_reset_req signal.

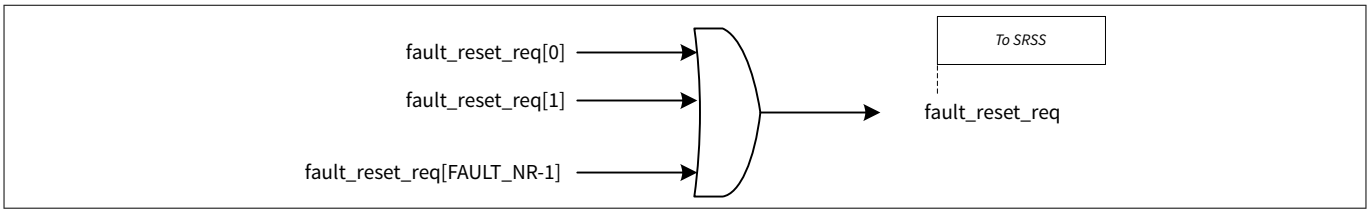


Figure 60 Fault reset

A central structure, shared by all fault report structures, keeps track of all pending faults in the system. The FAULT_STRUCTx_PENDINGy registers reflect which of the fault sources are pending. These registers provide a dedicated pending bit for up to 96 fault sources. The FAULT_STRUCTx_PENDINGy registers are mirrored in each of the fault report structures. The fault source numbering scheme follows the numbering scheme of FAULT_STRUCTx_STATUS.IDX.

The fault sources corresponding to a pending bit (which is set) are the ones that are not yet captured by any of the fault structures. When a pending fault is captured by a fault structure, the associated pending bit is cleared to '0'. Each fault report structure is selective in the faults it captures.FAULT_STRUCTx_MASKy reflect which

pending fault source is captured by a fault structure. These faults are referred to as “enabled” faults. The FAULT_STRUCTx_MASKy registers are unique to each fault structure. This allows for the following:

- One fault report structure is used to capture recoverable faults and another is used to capture non-recoverable faults. The former can be used to generate a fault interrupt and the latter can be used to activate a chip output signal or a reset request.
- Two fault report structures are used to capture the same faults. This first fault is captured by the structure with the lower index (for example, fault structure 0) and the second fault is captured by the structure with the higher index (for example, fault structure 1).

***Note**: FAULT_STRUCTx_STATUS.VALID bits are different for each of the fault structures. As an example, consider that the MCWDT lower threshold is linked to fault structure#0 and higher threshold is linked to Fault structure#1. Fault Ssructure#0 occurs first to give a warning; then, fault structure#1 occurs to trigger a reset.*

A fault structure only captures “enabled” faults when FAULT_STRUCTx_STATUS.VALID is ‘0’.

When a fault is captured, the hardware sets FAULT_STRUCTx_STATUS.VALID to ‘1’. In addition, the hardware clears the associated pending bit to ‘0’. When a fault structure is processed, the software (if the fault is processed by an interrupt handler) or a DMA transfer (if a triggered DMA transfer copied the captured fault information) should clear FAULT_STRUCTx_STATUS.VALID to ‘0’. Note that fault capturing does not consider FAULT_STRUCTx_INTR.FAULT:

- Fault capturing is only conditioned by FAULT_STRUCTx_STATUS.VALID being ‘0’.
- If an interrupt handler is used to process the fault structure, software should clear FAULT_STRUCTx_INTR.FAULT to ‘0’.

15.2 Fault and reset

As mentioned, a captured fault may result in a warm/soft reset. This type of reset brings regular MMIO registers to their default/reset state. This is not acceptable for the registers that capture fault information; for failure analysis, fault information should be retained during a warm/soft reset. Therefore, the FAULT_STRUCTx_STATUS and FAULT_STRUCTx_DATAy registers are connected to a cold reset. This illustrates another benefit of centralized fault report structures: only the centralized structure is connected to a cold reset. The multiple fault sources that are scattered throughout the system can use the regular reset, as a copy of the fault information is captured by the fault structure.

***Note**: When the fault is configured to trigger reset, then debugging of the configured fault structure is not possible.*

15.3 Fault and power modes

The fault report structure functionality is available only in active/sleep power modes (it is an active functionality):

- Deep sleep fault sources are not supported. These fault sources require dedicated solutions.
- The interfaces between the active fault sources and the centralized fault report structures is reset in deep sleep power mode. Note that the fault information is retained.

As the fault report structure is an active functionality, pending faults (in the FAULT_STRUCTx_PENDINGy registers) are not retained when transitioning to deep sleep power mode. This is acceptable, because the fault source itself is an active functionality.

For fault assignments, refer to the device specific datasheet.

15.4 Registers

Table 90 Fault subsystem register list

Symbol	Name	Description
FAULT_STRUCTx_CTL	Fault Control	This register is used to enable or disable the output trigger, I/O output signal, and reset request when a fault occurs.
FAULT_STRUCTx_STATUS	Fault Status	This register provides the fault source index and validity of data in the fault data registers.
FAULT_STRUCTx_DATAy	Fault Data	The data registers capture fault information.
FAULT_STRUCTx_PENDING0	Fault Pending 0	The FAULT_STRUCTx_PENDINGy registers specify pending (not captured) fault sources. The fault source for which data is captured in FAULT_STRUCTx_DATAy registers and is validated by FAULT_STRUCTx_STATUS.VALID and identified by FAULT_STRUCTx_STATUS.IDX is not included in this list of pending fault sources. When a fault source is captured, its corresponding bit field in FAULT_STRUCTx_PENDINGy is set to 0.
FAULT_STRUCTx_PENDING1	Fault Pending 1	
FAULT_STRUCTx_PENDING2	Fault Pending 2	
FAULT_STRUCTx_MASK0	Fault Mask 0	The FAULT_STRUCTx_MASKy registers specify “enables” for fault sources. Only “enabled” fault sources will be captured by this fault structure (and result in FAULT_STRUCTx_STATUS.VALID and FAULT_STRUCTx_INTR.FAULT being set to 1). When a fault source is captured, its corresponding bit field in FAULT_STRUCTx_PENDINGy is set to ‘0’.
FAULT_STRUCTx_MASK1	Fault Mask 1	
FAULT_STRUCTx_MASK2	Fault Mask 2	
FAULT_STRUCTx_INTR	Interrupt	This register sets the register bit when an enabled pending fault source is captured.
FAULT_STRUCTx_INTR_SET	Interrupt Set	This register sets the corresponding bits in the Interrupt Request register.
FAULT_STRUCTx_INTR_MASK	Interrupt Mask	Mask for Interrupt Request register.
FAULT_STRUCTx_INTR_MASKED	Interrupt Masked	Bitwise AND of Interrupt Request and Mask registers.

***Note**: In FAULT_STRUCTx, 'x' signifies the fault structure instance and 'y' in FAULT_STRUCTx_PENDINGy/ MASKy varies from 0 through 2 and FAULT_STRUCTx_DATAy varies 0 through 3.*

16 Power supply and monitoring

Top-level architecture

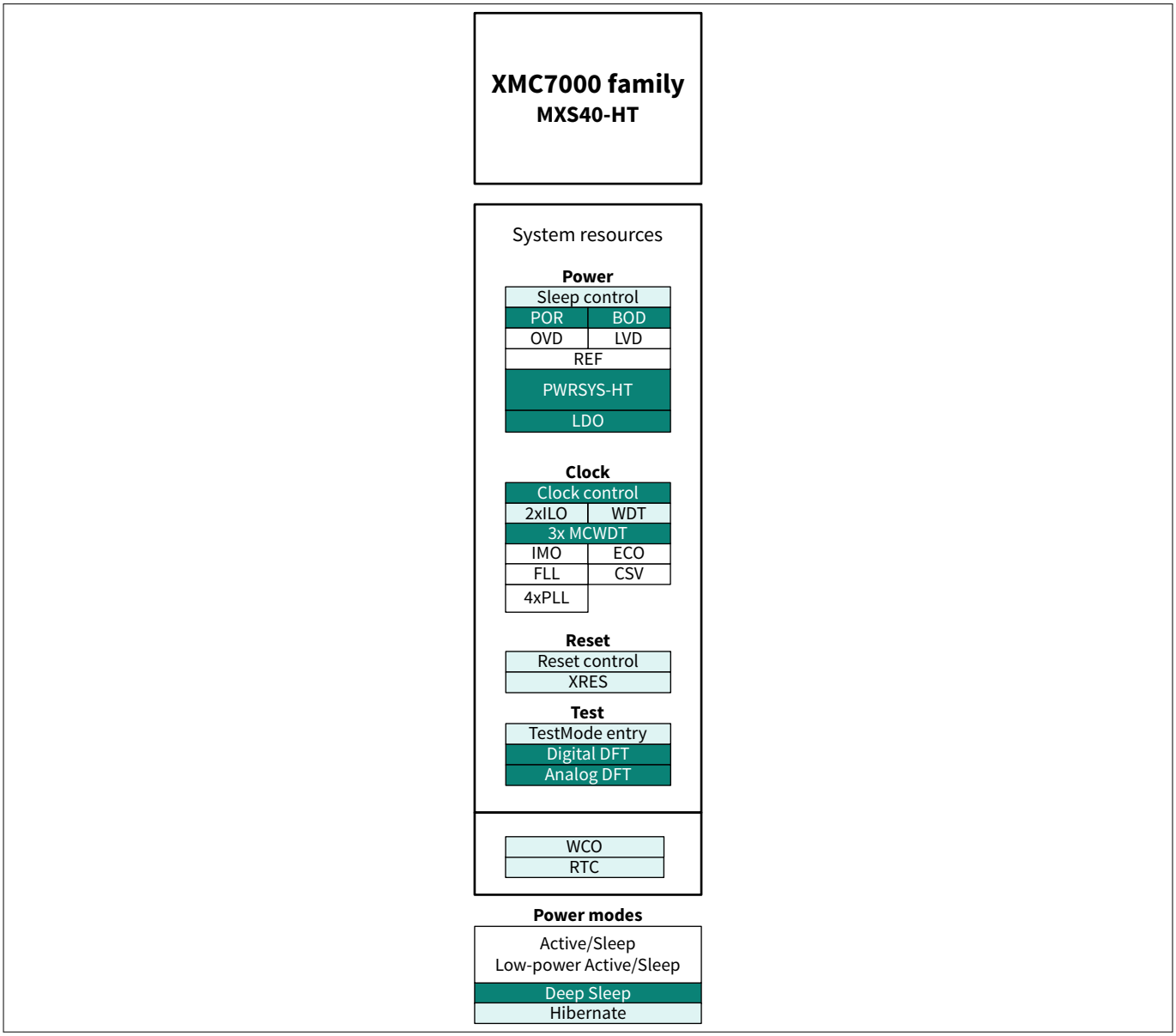


Figure 61 System-wide resources Block diagram

The XMC7000 family supports multiple power supply rails – V_{DDD}, V_{DDA}, V_{CCD}, and multiple V_{DDIO} rails. It integrates multiple regulators to power the blocks within the device in various power modes.

XMC7000 devices support power-on-reset (POR), brownout detection (BOD), over-voltage detection (OVD), over-current detection (OCD), and low-voltage detection (LVD) circuit for power supply monitoring and failure detection purposes. The low-voltage detection circuit can also be used as a high-voltage detection (HVD) circuit.

- POR provides a reset pulse during the V_{DDD} initial power ramp.
- BOD on V_{DDD} or V_{CCD} generates a reset if V_{DDD} or V_{CCD} voltage dips below the threshold voltage.
- BOD on V_{DDA} can generate a reset or a fault if V_{DDA} voltage dips below the threshold voltage.
- OVD on V_{DDD} or V_{CCD} generates a reset if V_{DDD} or V_{CCD} voltage goes above the threshold voltage.
- OVD on V_{DDA} can generate a reset or a fault if V_{DDA} voltage goes above the threshold voltage.

- OCD generates a reset if the load current of a regulator is over the regulator limit.
- LVD (HVD) can generate an interrupt or a fault whenever V_{DDD} voltage crosses the threshold in the configured direction.

16.1 Features

The features of the XMC7000 power supply subsystem are as follows:

- V_{DDD} power supply voltage range of 2.7 V to 5.5 V.
- Core supply rail (V_{CCD}).
- Independent multiple power supply rails (V_{DDD}, V_{DDA}, V_{CCD}, and multiple V_{DDIO} rails) for XMC7000 core peripherals.
- Multiple on-chip regulators.
  - Active regulator to power the MCU in active/sleep mode in case of low current consumption
  - Deep sleep regulator to power peripherals operating in deep sleep mode
  - High-current regulator to support higher current load by using an external pass transistor or by controlling for an external power management integrated circuit (PMIC) or low-dropout (LDO) regulator
- Low-voltage (V_{CCD}) and high-voltage (V_{DDD} and V_{DDA}) BOD circuits are available in all power modes except hibernate and XRES modes.
- Low-voltage (V_{CCD}) and high-voltage (V_{DDD} and V_{DDA}) OVD circuits are available in all power modes except hibernate and XRES modes.
- Two LVD circuits to monitor V_{DDD} for falling detection (LVD), rising detection (HVD), or both in all power modes except hibernate and XRES modes.
- OCD circuit to monitor V_{CCD} current in all power modes except hibernate and XRES modes. OCD is not monitored for PMIC.

16.2 Power supply

The regulators and supply pins/rails, shown in Figure 62, power various blocks inside the device. The availability of various supply rails/pins for an application will depend on the device package selected. See the device datasheet for details.

All the core regulators draw their input power from the V_{DDD} supply pin. V_{CCD} supply is used to power all active domains and deep sleep domains. From V_{CCD}, there are power domain switches that allow disabling active circuitry while leaving deep sleep circuitry connected to V_{CCD}. The hibernate domain does not implement any regulators and the peripherals available in that domain such as ILO operate directly from V_{DDD}.

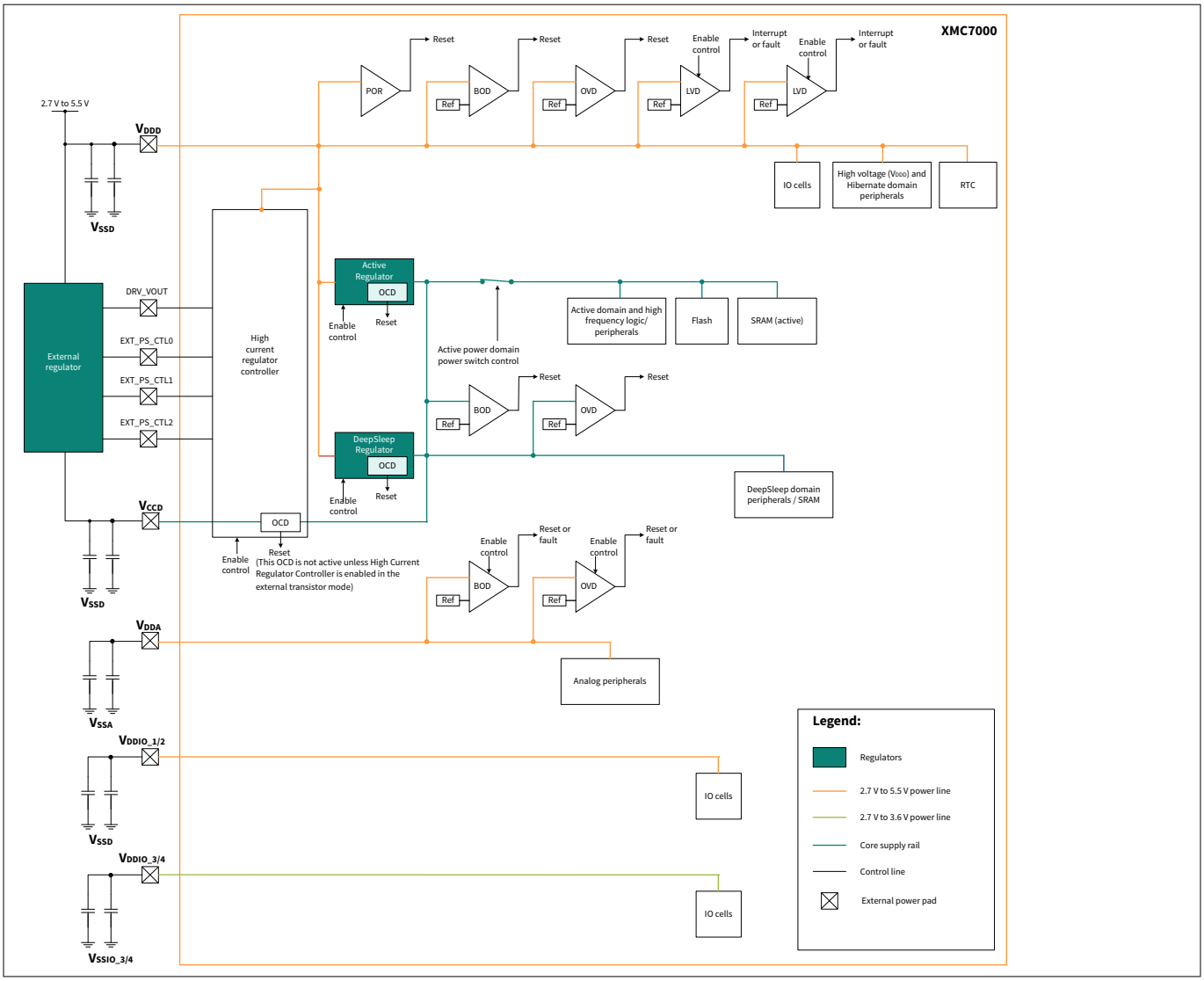


Figure 62 Power system block diagram

The I/O cells operate from V_{DDA} or multiple V_{DDIO} rails depending on the port they are located. V_{CCD} supply is used to drive logic inside the I/O cells from core peripherals. To know which I/Os operate from which supply see the I/O system.

16.2.1 Core regulators

The device includes the following core regulators to power peripherals and blocks in various power modes.

Note that in hibernate mode, all regulators are OFF and V_{CCD} is not driven. Hibernate related logic operates from V_{DDD} directly. For details, see the Device power modes.

Active regulator

The device includes a linear LDO regulator to power the active and sleep mode peripherals. This regulator generates the core voltage (V_{CCD}) from V_{DDD} during active and sleep modes. It is operational during device start up, when servicing interrupts that do not require the M7 CPUs to be fully active, or when the device is active at slow clock frequencies. The high-current regulator powers the MCU in case of high current consumption in active/sleep mode, which cannot be supplied by the active regulator.

Deep sleep regulator

In addition to the active regulator, the device includes a deep sleep regulator, which generates the core voltage (V_{CCD}) during deep sleep mode. The primary differences from the active regulator are that it has lower drive capability and consumes much less current.

High-current regulator

The high-current regulator controller supports higher load currents than the active regulator using an external pass transistor, or it can control an external PMIC or LDO.

XMC7000 starts with the active regulator, before switching to the high-current regulator.

16.2.2 Power pins and rails

Table 91 lists all the power supply pin names available in the device. The PCB must short all identically named pins externally with low-impedance connections (that is, either connect to a plane or use a wide top layer route between pins). And, all ground pins must be at same potential. For details, see the device datasheet.

Table 91 Supply pins

Supply pin	Ground pin	Power supply voltage range	Description
V_{DDD}	V_{SSD}	2.7 V to 5.5 V	Digital and I/O supply
V_{CCD}	V_{SSD}	1.1 V to 1.2 V	Core supply
V_{DDA}	V_{SSA}	2.7 V to 5.5 V	Analog supply, $V_{DDA} = V_{DDIO_2}$
V_{DDIO_1}	V_{SSD}	2.7 V to 5.5 V	I/O supply
V_{DDIO_2}	V_{SSD}	2.7 V to 5.5 V	I/O supply
V_{DDIO_3}	V_{SSIO_3}	2.7 V to 3.6 V	I/O supply
V_{DDIO_4}	V_{SSIO_4}	2.7 V to 3.6 V	I/O supply

16.2.3 Power sequencing requirements

V_{DDD}, multiple V_{DDIO}, and V_{DDA} do not have any sequencing limitation and can establish in any order. These supplies except V_{DDA} and V_{DDIO_2} are independent in voltage level. See the device datasheet for details of device operating conditions.

There are operating limits if a supply is not present:

- The part will not boot unless V_{DDD} is present
- V_{DDA} must be equal to V_{DDIO_2}
- A BOD can be configured by software to reset the part when V_{DDA} is not present

16.2.4 Power supply sources

XMC7000 offers power supply options that support a wide range of application voltages and requirements. The recommended V_{DDD} voltage range is 2.7 V to 5.5 V. If the application voltage is in this range, then XMC7000 (V_{DDD}) can be interfaced with any power supply voltage in the range of 2.7 V to 5.5 V. Other supply rails and pins such as V_{DDA} and multiple V_{DDIO} rails exist independent of V_{DDD} and V_{CCD}. See the device datasheet for details of device operating conditions.

16.2.5 Usage of high-current regulator controller

The high-current regulator controller (REGHC) is initially disabled and must be enabled with the REGHC_EN bit [31] of PWR_REGHC_CTL2 register. The active regulator supports the chip current until the high-current regulator controller is configured, enabled, operating, and ready. The current consumption of the full device must stay within the operation conditions of the active regulator.

The high-current regulator controller has two configurations. One is using an external pass transistor configuration (Figure 63) and the other is using an external PMIC or LDO device (Figure 65 or Figure 66) for driving much higher load currents. Table 92 lists the high-current regulator controller pins for each configuration. EXT_PS_CTL0/1/2 pins are shared with the GPIO function. Therefore, the GPIO function must be disabled (high-impedance mode with input buffer disabled same as the default state) by software before REGHC is enabled.

Table 92 High-current regulator controller pins

Pin name	External transistor mode		External PMIC mode	
	Direction	Description	Direction	Description
DRV_VOUT	OUT	Base of the pass transistor	Unused	–
EXT_PS_CTL0	IN	Positive terminal of the current sense resistor	IN	Power good input from PMIC
EXT_PS_CTL1	IN	Negative terminal of the current sense resistor	OUT	Enable output for PMIC
EXT_PS_CTL2	Unused	–	OUT (optional)	Reset threshold adjustment for some PMIC

The high-current regulator controller transitions when:

- Software switches between the active regulator and the high-current regulator controller.
- The high-current regulator controller is disabled by hardware for transitions to OFF and XRES states. Operation resumes with the active regulator when the reset condition is removed. Software can change back to the high-current regulator controller after the device reboots.
- A low-voltage reset can leave the device in an unintended state that requires software recovery. See Transitioning from active/sleep to reset.
- The following options are supported for deep sleep:
  - Hardware changes to the active regulator before entering deep sleep using PWR_REGHC_CTL4.REGHC_PMIC_DPSLP = 0. The device wakes up from deep sleep using the active regulator. After waking from deep sleep, hardware changes back to the high-current regulator controller.
  - Follow the sequence in Deep sleep entry/exit.
- PMIC is configured to operate during deep sleep using PWR_REGHC_CTL4.REGHC_PMIC_DPSLP = 1. Hardware does not change the power system settings when entering or exiting deep sleep.
- The high-current regulator controller is disabled by hardware for transitions to hibernate mode. When using and external PMIC, the high-current regulator controller tristates the PMIC enable output and a pull resistor on the PCB or within the PMIC disables the PMIC. The device wakes from hibernate using the active regulator. After wakeup, software can reconfigure the high-current regulator controller and change back to it.

***Notes**:*

1. *We recommend using system calls from CM0+ for setting up REGHC, instead of writing directly to the REGHC registers. See the Non-volatile memory programming for more details about system calls that can be used to set up REGHC.*
2. *Transitioning from the REGHC to the Active regulator is ignored when the debugger is connected.*

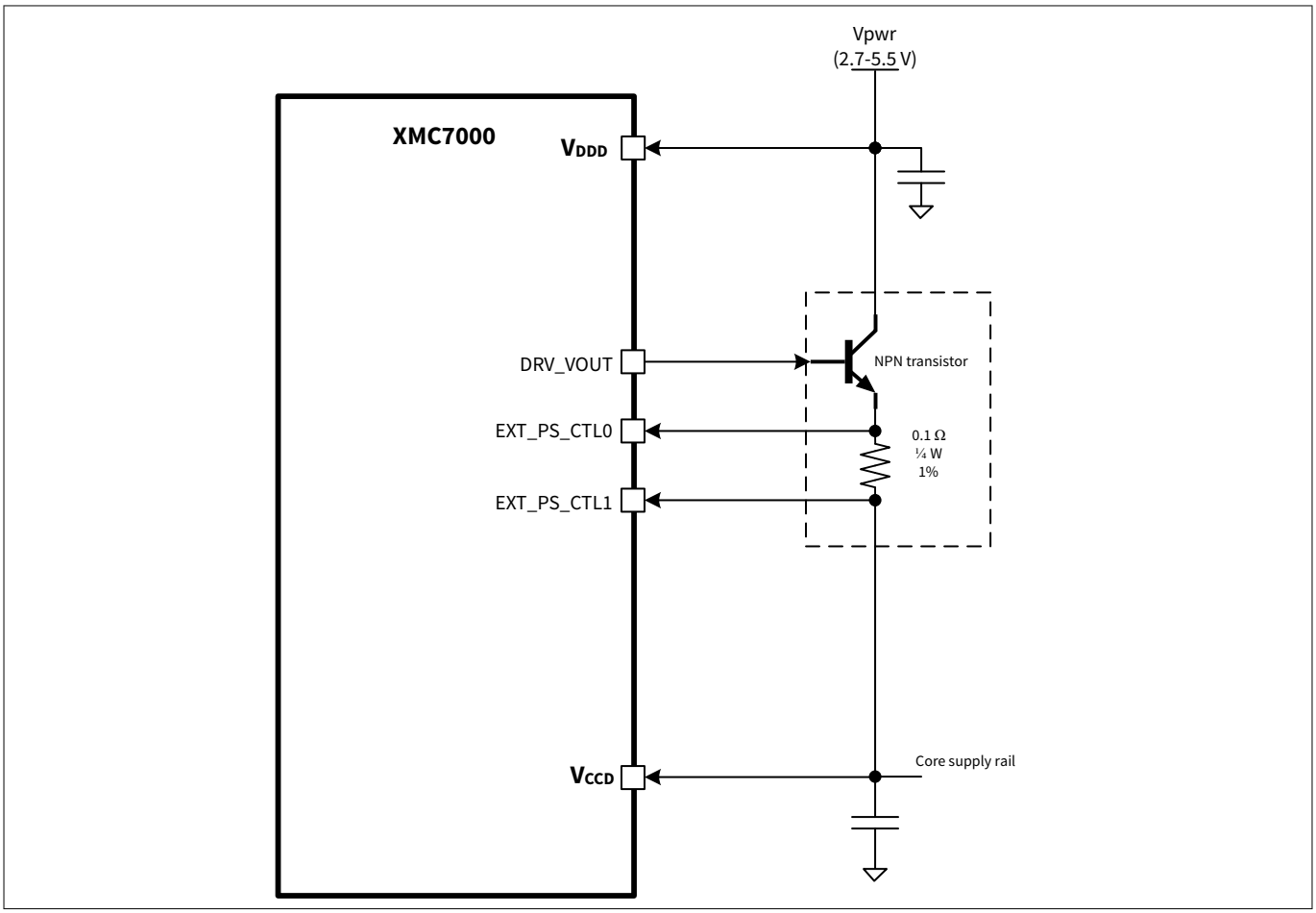


Figure 63 External pass transistor configuration

The high-current regulator controller has an error amplifier that controls the V_{CCD} voltage. The output of this error amplifier is sent to DRV_VOUT, which controls the external NPN transistor. This series of connections constitutes a feedback loop to control the V_{CCD} voltage. EXT_PS_CTL0 and EXT_PS_CTL1 are used for over-current detection and are not related to V_{CCD} voltage control.

For pass transistor configurations, a sense resistor (0.1 Ω 1%) is connected between the emitter of the external pass transistor and the V_{CCD}. Over-current is detected by detecting the voltage between the terminals of this current sense resistor with EXT_PS_CTL0 and EXT_PS_CTL1.

Table 93 External transistor requirement specification (type: NPN)

Parameter	Symbol	Min Value	Unit	Conditions
Static forward current transfer ratio	H_{FE}	100	–	I_C = 1 A, V_{CE} = 1 V
Collector-emitter voltage	V_{CEO}	10	V	–
Transition frequency	f_T	100	MHz	–

(table continues...)

Table 93 (continued) External transistor requirement specification (type: NPN)

Parameter	Symbol	Min Value	Unit	Conditions
Collector current	I_C	1	A	–
Collector power dissipation	P_{CD}	2	W	–

Do not exceed the rated temperature; it is necessary to estimate the junction temperature.
Loss of the pass transistor under the operating conditions is calculated using the following equation.

$$P_{Loss} = (V_{DDD_max} - V_{CCD_min}) \times I_{VCCD_max} \quad (1)$$

Where:
P_{Loss}: Loss of pass transistor (W)
V_{DDD_max}: Maximum V_{DDD} voltage (V)
V_{CCD_min}: Minimum V_{CCD} voltage (V)
I_{VCCD_max}: Maximum V_{CCD} load current (V)
The maximum power dissipation of the pass transistor must be greater than or equal to the above P_{Loss}.
The junction temperature of the pass transistor is calculated using the following equation:

$$T_J = P_{Loss} \times \theta_{JA} + T_A \quad (2)$$

Where:
T_J: Pass transistor junction temperature (°C)
T_A: Ambient temperature (°C)
P_{Loss}: Loss by pass transistor (W)
θ_{JA}: Thermal resistance from junction to ambient temperature (°C/W)
T_J must be below the rated temperature.
Table 94 shows examples of maximum θ_{JA}.

Table 94 Maximum θ_{JA} examples

$T_{A\,max}$ (°C)	V_{DDD}	$P_{Loss\,max}$ (W)	θ_{JA} (°C/W)
+125	5-V power rail	2.64	< 9.5
+105			< 17.0
+85			< 24.6
+125	3.3-V power rail	1.50	< 16.7
+105			< 30.0
+85			< 43.3

Conditions: V_{DDD} maximum voltage = 5.5 V/3.6 V, V_{CCD} minimum voltage = 1.10 V, V_{CCD} maximum load current = 0.6 A.

16.2.5.1 Transitioning from active regulator to high-current regulator controller with external transistor

To set up the high-current regulator controller with external transistor:

1. Confirm if the circuit board has populated the necessary external components for the high-current regulator controller, including a compatible transistor, capacitor, and current sense resistor.

This sequence is provided to customers as part of the ConfigureRegulator API:

2. Configure the high-current regulator controller. Writes in same registers can be done in same cycle:
   - **a.** Write PWR_REGHC_CTL.REGHC_MODE = 0 to configure the external transistor mode.
   - **b.** Write PWR_REGHC_CTL.REGHC_TRANS_USE_OCD = 1, depending on whether a current monitoring resistor is integrated on the PCB. Current monitoring is recommended.
   - **c.** Write PWR_REGHC_CTL.REGHC_VADJ to the required feedback setting. Add the signed offset in SFLASH SRSS_PWR_OFFSET.REGHC_TRANS_VADJ_OFFSET to account for die to die variation.
   - **d.** Write PWR_REGHC_CTL.REGHC_CONFIGURED = 1.

     After the high-current regulator controller is set up, it can be enabled without writing the setup again.

     This sequence is handled by the SwitchOverRegulators API. Call the API with blocking = 1 to ensure the transition occurs in the proper order:
   - **e.** Execute the system call (LoadRegulatorTrims) to change internal regulator trims.
   - **f.** Write PWR_REGHC_CTL2.REGHC_EN = 1.
   - **g.** Wait until PWR_REGHC_STATUS.REGHC_SEQ_BUSY = 0 and PWR_REGHC_STATUS.REGHC_ENABLED = 1. This should occur within 15 µs.
   - **h.** Again, execute the trim change system call.
   - **i.** The device is now operating on the high-current regulator controller with external transistor. Additional current load can be enabled.

16.2.5.2 Transitioning from high-current regulator controller with external transistor to active regulator

To transition from the high-current regulator controller with external transistor to the active regulator:

1. Reduce the current consumption to within the active regulator limit.

This part of the sequence is handled by the SwitchOverRegulators API. Call the API with blocking = 1 to ensure the transition occurs in the proper order.

2. Execute the system call (LoadRegulatorTrims) to change internal regulator trims.
3. Write PWR_REGHC_CTL2.REGHC_EN = 0.
4. Wait until PWR_REGHC_STATUS.REGHC_SEQ_BUSY = 0 and PWR_REGHC_STATUS.REGHC_ENABLED = 0. This should occur within 10 µs.
5. Again, execute the trim change system call.
6. The device is now operating on the active regulator.

Figure 64 shows the transitions between active regulator and external transistor.

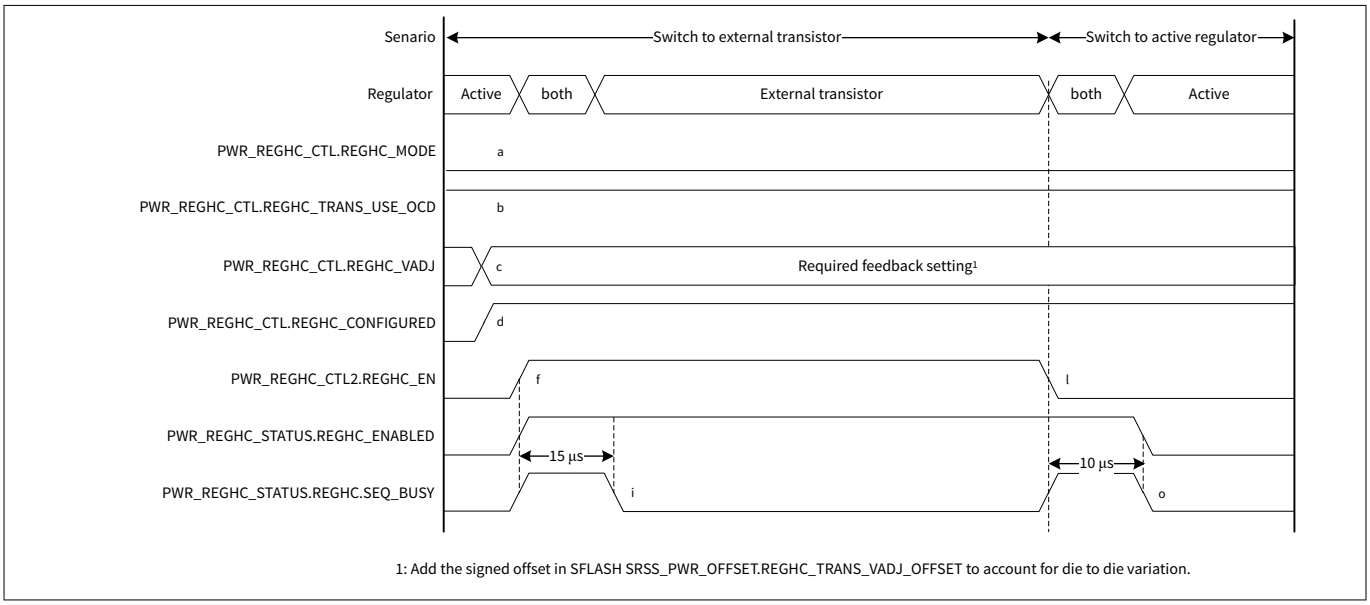


Figure 64 Transitions between active regulator and external transistor

Figure 65 shows how to connect to a PMIC that does not discharge its output.

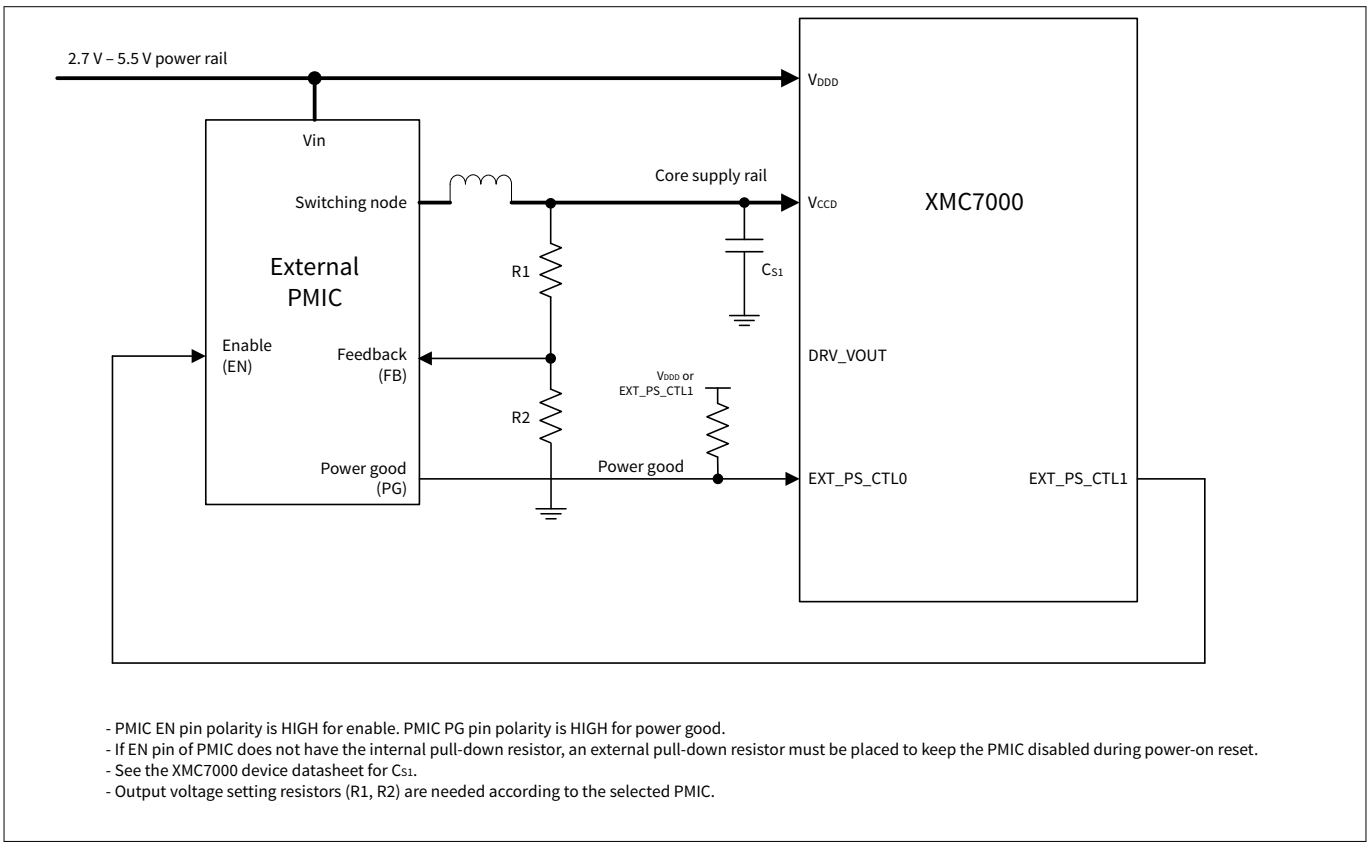
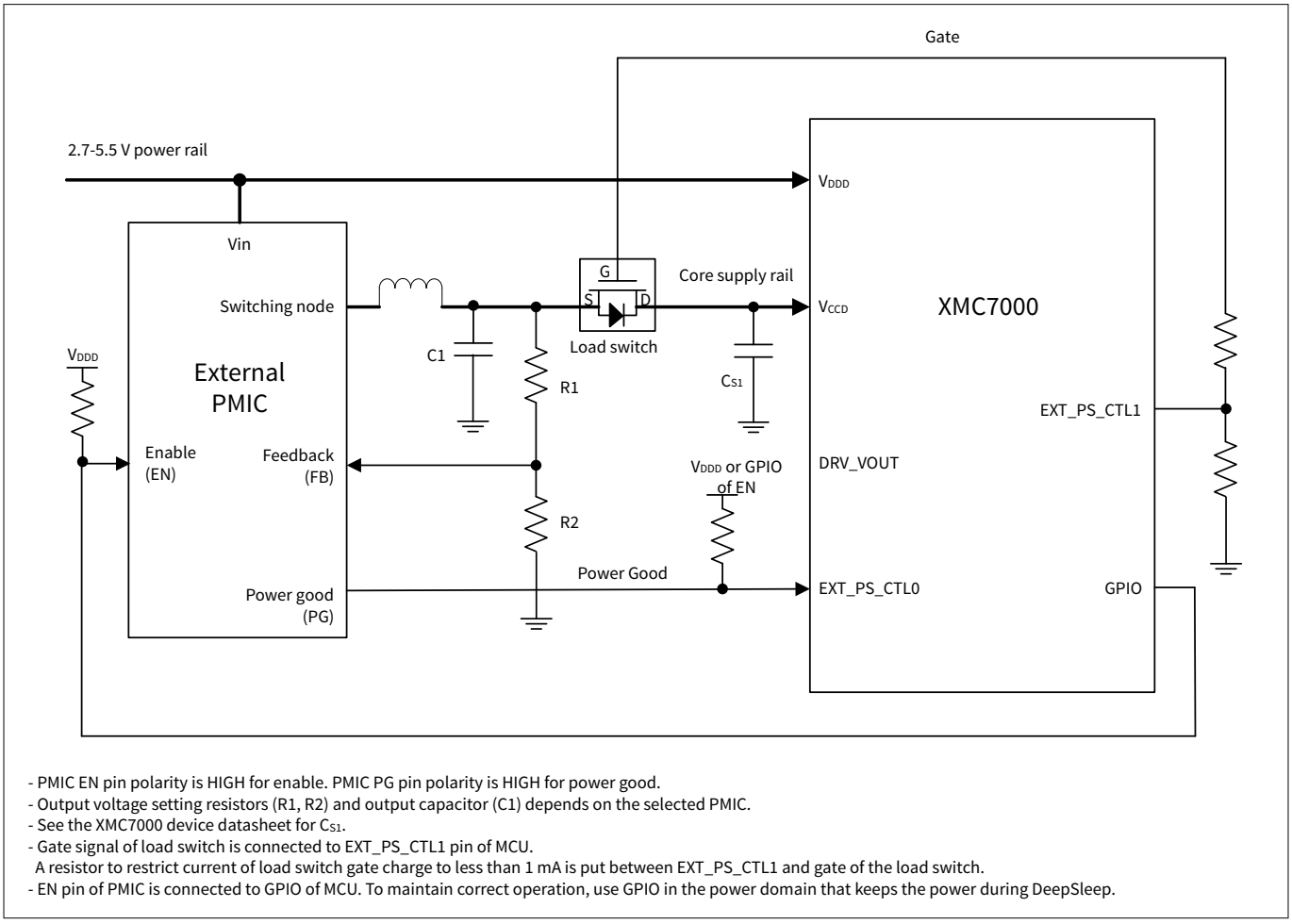


Figure 65 External PMIC/LDO configuration

Figure 66 shows how a load switch can be used for compatibility with a PMIC that discharges its output when disabled.



- PMIC EN pin polarity is HIGH for enable. PMIC PG pin polarity is HIGH for power good.
- Output voltage setting resistors (R1, R2) and output capacitor (C1) depends on the selected PMIC.
- See the XMC7000 device datasheet for C_{S1}.
- Gate signal of load switch is connected to EXT_PS_CTL1 pin of MCU.
  A resistor to restrict current of load switch gate charge to less than 1 mA is put between EXT_PS_CTL1 and gate of the load switch.
- EN pin of PMIC is connected to GPIO of MCU. To maintain correct operation, use GPIO in the power domain that keeps the power during DeepSleep.

Figure 66 External PMIC with load switch configuration

16.2.5.3 Transitioning from active regulator to high-current regulator controller with external PMIC

To setup the high-current regulator controller with external PMIC:

1. Confirm if the circuit board has populated the necessary external components for the high-current regulator controller, including a compatible PMIC. When using an external PMIC, the PMIC is responsible for supervision of V_{CCD}.

This sequence is provided to customers as part of ConfigureRegulator API:

2. Configure the high-current regulator controller for PMIC operation. These fields can be written in same cycle in the PWR_REGHC_CTL register:
   a. Write PWR_REGHC_CTL.REGHC_MODE = 1 to configure the PMIC mode.
   b. Write PWR_REGHC_CTL.REGHC_PMIC_STATUS_INEN = 1, to enable the input path for PMIC status. Write PWR_REGHC_CTL.REGHC_PMIC_STATUS_POLARITY to the setting that indicates an error condition (depending on the polarity of the PMIC status output).
   c. Write PWR_REGHC_CTL.REGHC_PMIC_CTL_POLARITY to the setting that enables the PMIC (depending on polarity of PMIC enable input).

d. Customer option: Configure PWR.REGHC_CTL.REGHC_PMIC_USE_LINREG = 1 to keep the internal active regulator enabled for its supply supervision capability. If this feature is not desired, write PWR_REGHC_CTL.REGHC_PMIC_USE_LINREG = 0.

e. Customer option: Configure PWR_REGHC_CTL.REGHC_PMIC_USE_RADJ = 1 and PWR_REGHC_CTL.REGHC_PMIC_RADJ to generate a reset threshold for the PMIC. If this feature is not needed, write PWR_REGHC_CTL.REGHC_PMIC_USE_RADJ = 0. Reset threshold adjustment is offered to reduce the need for external components. If it is used, they must be set such that the PMIC can robustly supply V_{CCD}.

f. Customer option: Configure PWR_REGHC_CTL.REGHC_PMIC_STATUS_WAIT to give additional settling time after PMIC status input is initially correct, until the sequencer continues. This is written if PMIC needs more startup time.

3. PMIC configuration in the PWR_REGHC_CTL4 register:

a. Customer option: Configure REGHC_PMIC_DPSLP to specify the PMIC behavior during deep sleep. If using a PMIC with load switch, and the PMIC is configured to be always enabled on the PCB (not under programmable control), then configure REGHC_PMIC_DPSLP = 1.

b. Configure REGHC_PMIC_VADJ_DIS = 1 to reduce power consumption. This is especially useful to reduce deep sleep current, if REGHC_PMIC_DPSLP = 1.

4. Configure this set of fields in a separate write cycle from the previous set. This prevents a possible glitch on PMIC enable output, which can happen if the polarity and output enable are changed at the same time.

a. Write PWR_REGHC_CTL.REGHC_PMIC_CTL_OUTEN = 1

b. Write PWR_REGHC_CTL.REGHC_CONFIGURED = 1

After the high-current regulator controller is set up, it can be enabled without writing the setup again. If using a PMIC with load switch, ensure the PMIC is enabled and operating. (Set GPIO to '1' for enabling PMIC. Then, wait until PWR_REGHC_STATUS.REGHC_PMIC_STATUS_OK = 1.)

This part of the sequence is handled by the SwitchOverRegulators API. Call the API with blocking = 1 to ensure the transition occurs in the proper order:

c. Execute the system call (LoadRegulatorTrims) to change the internal regulator trims.

d. Write PWR_REGHC_CTL2.REGHC_EN = 1.

e. If blocking = 1, wait until PWR_REGHC_STATUS.REGHC_SEQ_BUSY = 0 and PWR_REGHC_STATUS.REGHC_ENABLED = 1. This delay depends strongly on the startup time of the PMIC, based on its status output and the value in PWR_REGHC_CTL.REGHC_PMIC_STATUS_WAIT.

f. Again, execute the trim change system call.

g. If PWR_REGHC_CTL.REGHC_PMIC_USE_LINREG = 0 and PWR_REGHC_CTL4.REGHC_PMIC_DPSLP = 1, then it may be possible to disable the deep sleep regulator by writing PWR_CTL2.DPSLP_REG_DIS = 1.This bit must not be set if it is later intended to switch back to the active regulator using the sequence in the next section. The API with blocking = 1 case assumes it is never intended to switch back, and it disables the deep sleep regulator for this register configuration. If the application wants the future ability to switch back to the active regulator, it must call the API with blocking = 0 and not write PWR_CTL2.DPSLP_REG_DIS (leave it 0).

h. The device is now operating on the high-current regulator controller with external PMIC. Additional current load can be enabled.

16.2.5.4 Transitioning from high-current regulator controller with external PMIC to active regulator

***Note**: This sequence cannot be used if the deep sleep regulator is disabled (PWR_CTL2.DPSLP_REG_DIS = 1 at any time). The deep sleep regulator cannot be re-enabled, and it is needed for the active regulator to operate. If the application wants to use this sequence, do not disable the deep sleep regulator in the transition to the PMIC.*

To transition from the high-current regulator controller with external PMIC to the active regulator:

1. Reduce the current consumption to within the active regulator limit.

This part of the sequence is handled by the SwitchOverRegulators API. If using a PMIC with load switch, call the API with blocking = 0. For all other cases, call the API with blocking = 1 to ensure the transition completes in the proper order:

2. Execute the system call (LoadRegulatorTrims) to change internal regulator trims.

3. Write PWR_REGHC_CTL2.REGHC_EN = 0.

4. If blocking = 1, wait until PWR_REGHC_STATUS.REGHC_SEQ_BUSY = 0 and PWR_REGHC_STATUS.PWR_REGHC_ENABLED = 0. This delay depends on how long is takes for the external PMIC to deassert its power good signal (that is, until PWR_REGHC_STATUS.REGHC_PMIC_STATUS_OK = 0).

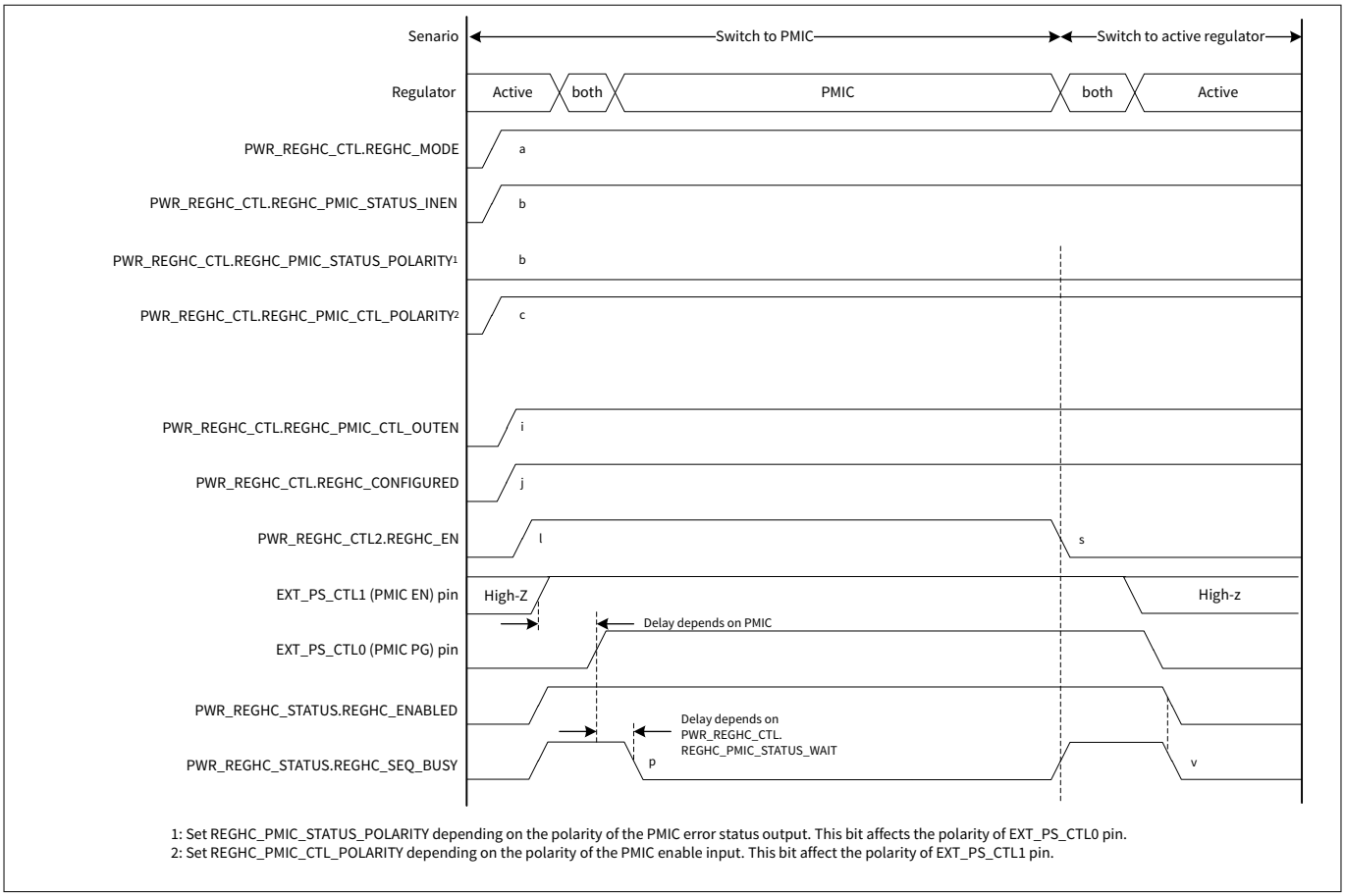
5. Again, execute the trim change system call.

6. The device is now operating on the active regulator.

If using a PMIC with load switch and PWR_REGHC_CTL4.REGHC_PMIC_DPSLP = 0, wait until the load switch is fully off and then disable the PMIC so it deasserts its power good signal and PWR_REGHC_STATUS.REGHC_PMIC_STATUS_OK = 0. This allows the internal state machine to complete.

If the SwitchOverRegulators API was called with blocking = 0, set GPIO as ‘0’ to disable PMIC after the GPIO that is shared with EXT_PS_CTL1 is 0 (load switch turned off). Later, wait for the transition to complete (PWR_REGHC_STATUS.REGHC_SEQ_BUSY = 0 and PWR_REGHC_STATUS.REGHC_ENABLED = 0).

Figure 67 shows an example of transitions between the active regulator and external PMIC, with REGHC_PMIC_CTL_POLARITY = 1 and REGHC_PMIC_STATUS_POLARITY = 0.



1: Set REGHC_PMIC_STATUS_POLARITY depending on the polarity of the PMIC error status output. This bit affects the polarity of EXT_PS_CTL0 pin.
2: Set REGHC_PMIC_CTL_POLARITY depending on the polarity of the PMIC enable input. This bit affect the polarity of EXT_PS_CTL1 pin.

Figure 67 Transitions between active regulator and external PMIC

16.2.5.5 Internal regulator configuration when using PMIC

To configure the high-current regulator controller (REGHC) to PMIC mode, set the active and deep sleep regulators to enable or disable.

When PMIC is enabled and the active regulator is in parallel, the OCD function of the active regulator is used to supply power from the PMIC. The OCD is the integrated part of the active regulator; it is enabled only when the active regulator is enabled. In power save modes such as deep sleep, the active regulator is OFF; therefore, the OCD is also off. V_{CCD} brownout, such as when the PMIC drops out of regulation range or cannot provide a fast load current increase, can be detected by the MCU. This is effective when the PMIC does not have an OCD function. This function is disabled when the active regulator is disabled.

The deep sleep regulator supplies the core supply only in deep sleep power mode. The register setting helps to decide if the PMIC or the deep sleep regulator supplies power to the MCU in deep sleep mode.

The following configurations are available depending on the system.

Table 95 Internal regulator configuration for PMIC case

Use case	Configuration	Active regulator (including internal OCD)	Deep sleep regulator (including internal OCD)
Use PMIC without own OCD feature. PMIC is disabled in deep sleep power mode.	OCD is enabled when supplying power from PMIC. Deep sleep regulator supplies power in deep sleep mode.	Enabled	Enabled
Use PMIC with own OCD feature. PMIC is disabled in deep sleep power mode.	OCD is disabled when supplying power from PMIC. Deep sleep regulator supplies power in deep sleep mode.	Disabled	Enabled
Use PMIC with own OCD feature. PMIC is enabled in deep sleep power mode.	OCD is disabled when supplying power from PMIC. PMIC supplies power in deep sleep mode.	Disabled	Enabled/disabled

***Note**: The internal OCD feature is for self-protection of the internal LDOs and the active and deep sleep regulators only. The PMIC without its own OCD feature can be used with the internal OCD feature, but this detection works outside the current path of PMIC.*

***Note**: When you set the deep sleep regulator to "Disabled" (PWR_CTL2.DPSLP_REG_DIS = 1), you cannot switch back from the PMIC to the internal regulator. If your system is required to switch back to the internal regulator again after handover to PMIC, deep sleep regulator should be enabled.*

16.2.5.6 Transitioning from active/sleep to reset

This section describes the reset behavior for SRSS, trimming, and user program parts.

For SRSS part, a reset results in the following behavior:

- For resets that do not reset the power system (low-voltage (LV) resets such as fault, internal system reset, MCWDT, or CSV), the high-current regulator controller (REGHC) settings are not changed. If REGHC is already operating, it continues to operate. The V_{CCD} rail continues uninterrupted.
- For resets that do reset the power system (high-voltage (HV) resets such as POR, BOD, OVD, OCD, WDT, hibernate wakeup, or XRES_L), it turns off REGHC and the device restarts with the active regulator.
- For the trimming part, the normal trim download overwrites the regulator targets with the internal settings. When this happens, the current load is within the active regulator limit so it does not risk false OCD if REGHC is also enabled.
- For the reset-recovery user software, it detects if the device is operating on REGHC and restores settings needed to increase the current level beyond the limit of the active regulator.

Figure 68 shows a simplified flow. The reset-recovery flow can be run after every reset.

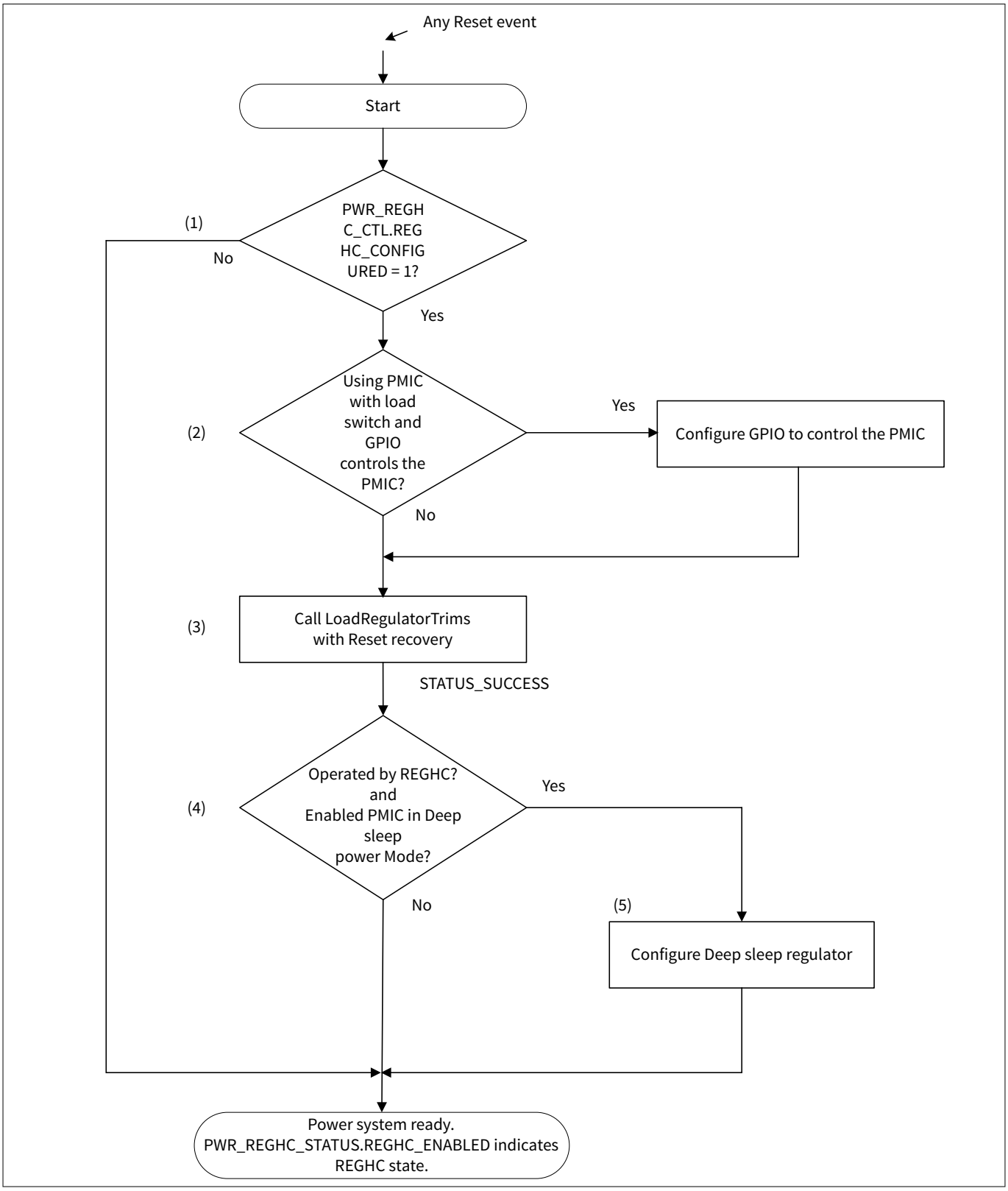


Figure 68 Reset recovery flow

1. Check for the REGHC configuration. When PWR_REGHC_CTL.REGHC_CONFIGURED is set to '1', go to (2). When PWR_REGHC_CTL.REGHC_CONFIGURED is set to '0', the power system is ready. PWR_REGHC_STATUS.REGHC_ENABLED indicates the REGHC state.

2. If you have the REGHC with load switch configuration, you need to configure GPIO for REGHC enable control. In this case, REGHC enable state before LV reset is reconfigured. The following is an example of GPIO reconfiguration:
   a. Write ~ (PWR_REGHC_CTL2.REGHC_EN ^ PWR_REGHC_CTL.REGHC_PMIC_CTL_POLARITY) to GPIO data register.
   b. Configure to GPIO output.
3. Call the LoadRegulatorTrims API with reset recovery.
4. When LoadRegulatorTrims returns STATUS_SUCCESS, go to (4).
5. LoadRegulatorTrims returns an error if the hardware state machine is still transitioning. This can happen if the reset occurred during the hardware sequence. If this case occurs, software may wait for the transition to complete (for example, a PMIC that is still turning on). If the transition does not complete within the PMIC enable time, it is recommended to reset the entire chip, including the power system, by using WDT or requesting an external system controller trigger XRES_L.
6. Check the REGHC operation and deep sleep regulator configuration. When PWR_REGHC_STATUS.REGHC_ENABLED is set to '1', PWR_REGHC_CTL.REGHC_PMIC_USE_LINREG is set to '0', and PWR_REGHC_CTL4.REGHC_PMIC_DPSLP is set to '1', go to (5).
7. Set PWR_CTL2.DPSLP_REG_DIS to '1'.

- After the flowchart completes, it is possible for the device to operate from the internal active regulator or from REGHC. If the device is operating from REGHC, it is indicated in PWR_REGHC_STATUS.REGHC_ENABLED.

16.2.5.7 Deep sleep entry/exit

When entering deep sleep, perform the following steps:

- If PWR_REGHC_CTL4.REGHC.PMIC_DPSLP = 1, deep sleep can be entered immediately with no other steps.
- Otherwise. reduce current within the active regulator limit.
- If a fast wake time is required, change from REGHC to active regulator before entering deep sleep. The device wakes with the same regulator that was operating before going to deep sleep, and active regulator wakeup is usually faster than PMIC startup time.
- Execute the system call (LoadRegulatorTrims) to update regulator targets for deep sleep entry.
- Enter deep sleep.
- When exiting deep sleep, perform the following steps:
- If operating from PMIC and PWR_REGHC_CTL4.REGHC_PMIC_DPSLP = 1, there are no special steps to be performed because the PMIC is already operating and the internal regulator settings are already correct.
- If operating from REGHC, because it was enabled by hardware during deep sleep wakeup:
  - Wait until PWR_REGHC_STATUS.REGHC_SEQ_BUSY = 0 and PWR_REGHC_STATUS.REGHC_ENABLED = 1.
  - Execute the system call (LoadRegulatorTrims) for DEEP SLEEP exit.
  - After the system call is completed, current will be increased.

16.3 Voltage monitoring

The XMC7000 family offers multiple voltage monitoring and supply failure protection options. This includes POR, BOD, OVD, LVD, OCD, and ADC monitoring. Table 96 lists the dedicated supply monitors in the device.

Table 96 Dedicated supply monitors

Monitor	Monitored supply	Number of trip points	Output	Available power mode
POR	V_{DDD}	1 (Fixed)	Reset	All power modes
BOD	V_{DDD}	2 (Programmable)	Reset	All power modes except hibernate and XRES modes
	V_{DDA}	2 (Programmable)	Reset, fault, or no action	
	V_{CCD}	1 (Fixed)	Reset	
OVD	V_{DDD}	2 (Programmable)	Reset	
	V_{DDA}	2 (Programmable)	Reset, fault, or no action	
	V_{CCD}	1 (Fixed)	Reset	
OCD[1)]	V_{CCD}	1 (Fixed)	Reset	
LVD	V_{DDD}	26 (Programmable)	Interrupt, fault, or no action	

1) XMC7000 does not have OCD for the PMIC.

16.3.1 Power-on-reset (POR)

POR circuits provide a reset pulse during the initial power ramp. POR circuits monitor only V_{DDD} voltage. See the device datasheet for details on the POR trip-point levels.

16.3.2 Brownout-detection (BOD)

The BOD circuit detects supply conditions below a threshold and applies reset to the device. XMC7000 offers three BOD circuits – BOD on V_{DDD}, BOD on V_{DDA}, and BOD on V_{CCD}. The system will not come out of RESET until V_{DDD} and V_{CCD} supplies are detected to be valid again. BOD on V_{DDA} is initially disabled and is configurable by software. There is no BOD support in hibernate and XRES modes. Applications that require BOD support should not use hibernate mode and should disable it. See the Device power modes for details.

16.3.2.1 BOD on V_{DDD}

The BOD on V_{DDD} supports two voltage levels (thresholds) to monitor < 2.7 V or < 3.0 V. The PWR_SSV_CTL.BODVDDD_VSEL bit selects the threshold levels of the BOD on V_{DDD}. The BOD on V_{DDD} cannot be disabled. For details on supported thresholds, see the device datasheet and the PWR_SSV_CTL register definition in the XMC7100 registers technical reference manual or XMC7200 registers technical reference manual. The PWR_SSV_STATUS.BODVDDD_OK bit indicates the status of the BOD on V_{DDD}. This will always read 1 (no brownout voltage detected), because a detected brownout will reset the chip.

16.3.2.2 BOD on V_{DDA}

The BOD on V_{DDA} supports two voltage levels (thresholds) to monitor < 2.7 V or < 3.0 V. The PWR_SSV_CTL.BODVDDA_VSEL bit selects the threshold levels of the BOD on V_{DDA}. The PWR_SSV_CTL.BODVDDA_ACTION bits can be used to select a reset, a fault or no action (default). The PWR_SSV_CTL.BODVDDA_ENABLE bit can be used to enable or disable (default) the BOD on V_{DDA}. However, it is not available unless V_{DDD} is present and valid. For details on supported the thresholds, see the device datasheet and the PWR_SSV_CTL register definition in the XMC7100 registers technical reference manual or XMC7200 registers technical reference manual. The PWR_SSV_STATUS.BODVDDA_OK bit indicates the status of the BOD on V_{DDA}.

16.3.2.3 BOD on V_{CCD}

The BOD on V_{CCD} cannot be disabled. The BOD on V_{CCD} is not as robust as the BOD on V_{DDD}/V_{DDA}. The limitation is because of the small voltage detection range available for this circuit on the minimum allowed V_{CCD}. For details on supported thresholds, see the device datasheet. The robust operation is possible with robust BOD on V_{DDD} and robust OCD on V_{CCD}, even without robust BOD on V_{CCD}. The input voltage to the regulator is robustly supervised by BOD on V_{DDD}. The load current of the active and deep sleep regulators are monitored for current that exceeds the regulator limit by OCD on V_{CCD}. Therefore, OCD monitors the operating conditions met when using the internal regulators. However, XMC7000 does not monitor the current of the PMIC. The PWR_SSV_STATUS.BODVCCD_OK bit indicates the status of the BOD on V_{CCD}. This will always read '1' (no brownout voltage detected), because a detected brownout will reset the chip.

16.3.3 Over-voltage detection (OVD)

XMC7000 offers three over-voltage detection circuits that monitor V_{CCD}, V_{DDD}, and V_{DDA} supply. Similar to the BOD circuit, the OVD circuit detects supply conditions above a threshold and applies a reset. As the name suggests, the OVD circuit maintains a device reset, if V_{CCD} or V_{DDD} supply stays higher than thresholds. The OVD circuit can generate a reset in all device power modes except the hibernate and XRES modes, provided the V_{DDD} and V_{DDA} supply ramp satisfies the datasheet maximum supply ramp limits in that mode. Applications that require OVD support should not use hibernate mode and should disable it. See the Device power modes for details.

16.3.3.1 OVD on V_{DDD}

The OVD on V_{DDD} supports two voltage levels (thresholds) to monitor > 5.5 V or > 5.0 V. The PWR_SSV_CTL.OVDVDDD_VSEL bit selects the threshold levels of the OVD on V_{DDD}. The OVD on V_{DDD} cannot be disabled. For details on supported thresholds, see the device datasheet and the PWR_SSV_CTL register definition in the XMC7100 registers technical reference manual or XMC7200 registers technical reference manual. The PWR_SSV_STATUS.OVDVDDD_OK bit indicates the status of the OVD on V_{DDD}. This will always read 1 (no overvoltage detected), because a detected overvoltage will reset the chip.

16.3.3.2 OVD on V_{DDA}

The OVD on V_{DDA} supports two voltage levels (thresholds) to monitor > 5.5 V or > 5.0 V. The PWR_SSV_CTL.OVDVDDA_VSEL bit selects the threshold levels of the OVD on V_{DDA}. The PWR_SSV_CTL.OVDVDDA_ACTION bits can be used to select a reset, a fault, or no action (default). The PWR_SSV_CTL.OVDVDDA_ENABLE bit can be used to enable or disable (default) the OVD on V_{DDA}. However, it is not available unless V_{DDD} is present and valid. For details on supported thresholds, see the device datasheet and the PWR_SSV_CTL register definition in the XMC7100 registers technical reference manual or XMC7200 registers technical reference manual. The PWR_SSV_STATUS.OVDVDDA_OK bit indicates the status of the OVD on V_{DDA}.

16.3.3.3 OVD on V_{CCD}

The OVD on V_{CCD} cannot be disabled. For details on supported thresholds, see the device datasheet. The PWR_SSV_STATUS.OVDVCCD_OK bit indicates the status of the OVD on V_{CCD}. This will always read 1 (no overvoltage detected), because a detected overvoltage will reset the chip.

16.3.4 Low-voltage-detection (LVD)

Two LVD circuits monitor external supply voltage (V_{DDD}) and detects depletion of the energy source. The LVD detectors generate an interrupt or a fault that causes the system to take preventive measures. The PWR_LVD_CTL/2.HVLVD1/2_ACTION bit can be used to select an interrupt or a fault.

These low-voltage detection circuits can also be used for high-voltage detection (HVD). They can each be configured as LVD (falling detection), HVD (rising detection), or both by the PWR_LVD_CTL/2.HVLVD1/2_EDGE_SEL bits. Each LVD supports up to 26 voltage levels (thresholds) to monitor between 2.8 V and 5.3 V. The PWR_LVD_CTL/2.HVLVD1/2_TRIPSEL_HT bits select the threshold levels of the HVLVD1/2. The LVD should be disabled before selecting the threshold. The PWR_LVD_CTL/2.HVLVD1/2_EN_HT bit can be used to enable or disable the HVLVD1/2. The LVD operates in active, sleep, and deep sleep modes. It does not operate in hibernate and XRES modes. To use HVLVD1/2 in deep sleep mode, the PWR_LVD_CTL/2.HVLVD1/2_DPSLP_EN_HT bit should be enabled.

Whenever the voltage level of the supply being monitored crosses the threshold, the LVD generates an interrupt or a fault. This interrupt status is available in the SRSS_INTR register. And the real-time status is available in the PWR_LVD_STATUS/2 register.

The SRSS_INTR register indicates a pending LVD interrupt. The SRSS_INTR_MASK register decides whether LVD interrupts are forwarded to the CPU.

For details on supported LVD thresholds, see the device datasheet and the PWR_LVD_CTL/2 register definition in the XMC7100 or XMC7200 registers TRM.

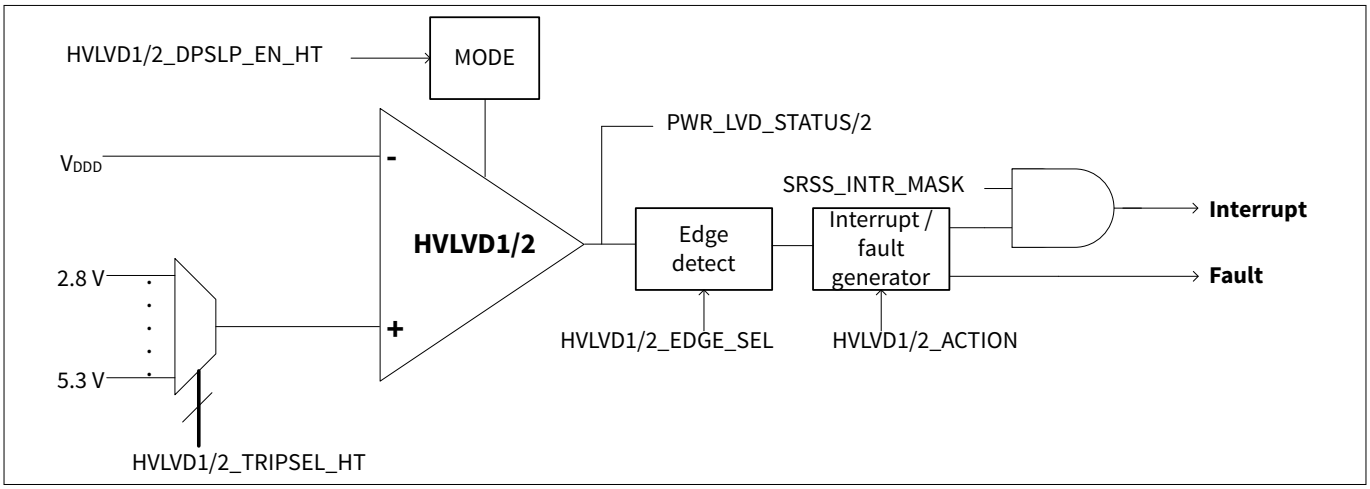


Figure 69 XMC7000 LVD block

16.3.5 Over-current detection

The OCD circuit monitors V_{CCD} current and detects if the load current of a regulator is higher than expected. If the current is over the regulator limit, the OCD circuit generates a reset to protect the device. OCD for the high-current regulator controller is not active unless the high-current regulator controller is enabled. For details on detection range, see the device datasheet. OCD operates in active, sleep, and deep sleep modes. Because the regulators are disabled in hibernate mode, the OCD circuit also does not operate. The PWR_SSV_STATUS register indicates the OCD status. XMC7000 does not have OCD for the PMIC. However, when PWR_REGHC_CTL.REGHC_PMIC_USE_LINREG = 1, MCU keeps the internal active regulator and its OCD enabled to improve supply supervision of V_{CCD} for the external PMIC mode. When using this feature, if the PMIC fails to keep V_{CCD} above the internal regulator target, then the internal regulator will attempt to recover V_{CCD}. If the regulator current is too high, the regulator triggers an OCD reset.

16.3.6 Voltage monitoring by ADC

In addition to the dedicated monitors, analog connections are provided to allow the ADC to monitor all high-voltage supplies and grounds. This is the only monitor capability provided for supplies without dedicated monitors (such as multiple V_{DDIO} rails).

To facilitate voltage monitoring by the ADC, a monitor switch in the power pad creates a connection between the power or ground pad and the AMUXBUS. The HSIOM_MONITOR_CTL_0 register controls the connectivity of

power/ground pads to either AMUXBUS_A or AMUXBUS_B respectively. For details, see Table 97. The power monitor cell can connect the power pad to AMUXBUS_A as shown in Figure 70. The ground monitor cell can connect ground pad to AMUXBUS_B. It is shown in Figure 71. For details on HSIOM_MONITOR_CTL_0 register, see the XMC7100 registers technical reference manual or XMC7200 registers technical reference manual.

The series resistor is intended to allow voltage division using a matching resistor in the ADC. This enables measuring supplies outside of the V_{DDA} (V_{REFH})/V_{SSA} (V_{REFL}) limits. For details on the ADC, see the Reference buffer.

Table 97 Relation between HSIOM_MONITOR_CTL_0 register and power/ground pads

HSIOM_MONITOR_CTL_0	Power/ ground pads	AMUXBUS	BGA-272 for XMC7200	BGA-272 for XMC7100	TEQFP-176	TEQFP-144	TEQFP-100
Bit 0	V_{DDD}	A	F8, H13, J13, K13, L13, N11	F8, H13, J13, K13, L13, N11	176	144	100
Bit 2					22	18	12
Bit 4					43	35	24
Bit 13					110	90	62
Bit 15					132	108	75
Bit 17					153	124	86
Bit 1	V_{SSD}	B	A1, A18, D9, G7, G12, H9, H10, H11, J9, J10, J11, J15, K9, K10, K11, M7, M12, R5, R14, V1, V18	A1, A18, D9, G7, G12, H9, H10, H11, J9, J10, J11, J15, K9, K10, K11, M7, M12, R5, R14, V1, V18, L9, L10	1	1	1
Bit 3					23	19	13
Bit 6					45	37	26
Bit 12					89	73	51
Bit 14					114	94	66
Bit 16					133	109	76
Bit 19					155	126	88
Bit 5	V_{DDIO_1}	A	F9, F10, F11	F9, F10, F11	44	36	25
Bit 18	V_{SSD_1} (BGA)	B	L11	L11	154	125	87
	V_{SSD} (TEQFP)						
Bit 7	V_{REFL}	B	K8	K8	76	62	41
Bit 8	V_{SSA}	B	L8	L8	77	63	42
Bit 9	V_{DDA}	A	L6	L6	78	64	43
Bit 10	V_{REFH}	A	K6	K6	79	65	44
Bit 11	V_{DDIO_2}	A	N8	N8, N9, N10	88	72	50
Bit 20	V_{DDIO_3}	A	H6, J6	H6, J6	–	–	–
Bit 21	V_{SSIO_3}	B	H8, J8	H8, J8	–	–	–
Bit 22	V_{DDIO_4}	A	N9, N10	–	–	–	–

(table continues...)

Table 97 (continued) Relation between HSIOM_MONITOR_CTL_0 register and power/ground pads

HSIOM_MONITOR_CTL_0	Power/ ground pads	AMUXBUS	BGA-272 for XMC7200	BGA-272 for XMC7100	TEQFP-176	TEQFP-144	TEQFP-100
Bit 24					–	–	–
Bit 23	V_{SSIO_4}	B	L9, L10	–	–	–	–
Bit 25					–	–	–

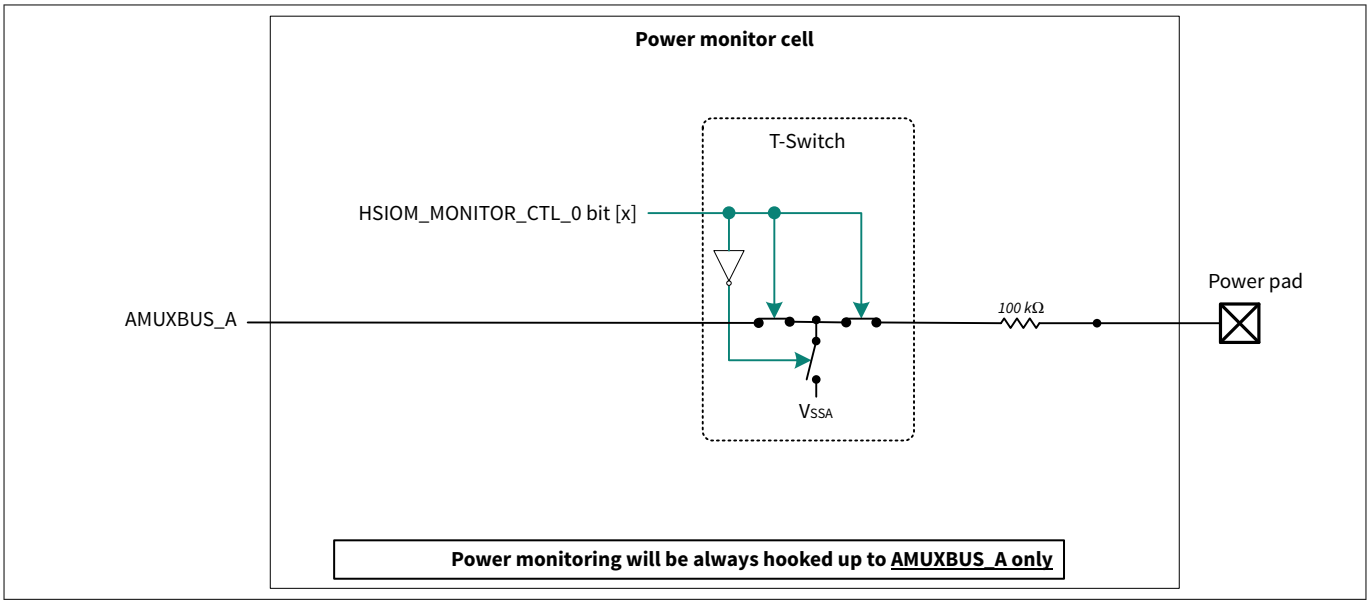


Figure 70 Power monitor cell

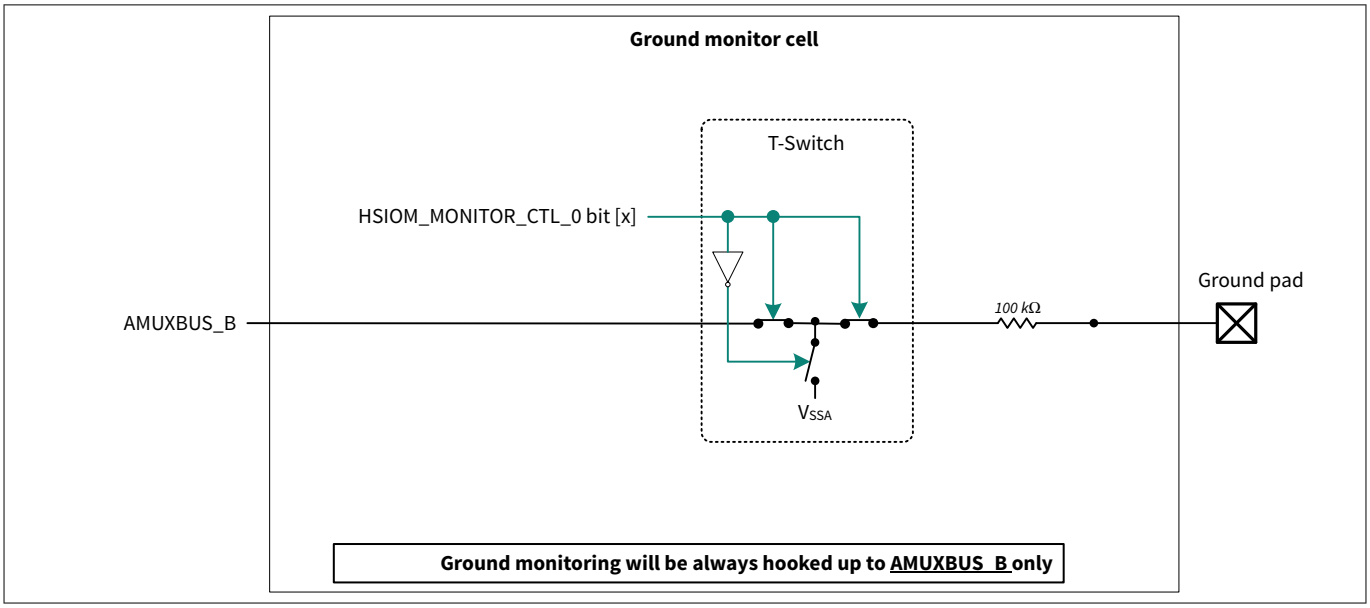


Figure 71 Ground monitor cell

16.4 Registers

Table 98 Register list

Register	Name	Description
PWR_LVD_CTL	High-Voltage/Low-Voltage Detector (HVLVD) Configuration register	This register shows the configuration bits for HVLVD1
PWR_LVD_CTL2	High-Voltage/Low-Voltage Detector (HVLVD) Configuration register 2	This register shows the configuration bits for HVLVD2
PWR_LVD_STATUS	High-Voltage/Low-Voltage Detector (HVLVD) Status register	This register shows the real time status for HVLVD1
PWR_LVD_STATUS2	High-Voltage/Low-Voltage Detector (HVLVD) Status register 2	This register shows the real time status for HVLVD2
PWR_SSV_CTL	Supply Supervision Control register	This register shows the controls for BOD and OVD
PWR_SSV_STATUS	Supply Supervision Status register	This register shows the status for BOD and OVD
SRSS_INTR	SRSS Interrupt register	This register shows interrupt requests from the SRSS peripheral.
SRSS_INTR_SET	SRSS Interrupt Set register	This register is used for firmware testing.
SRSS_INTR_MASK	SRSS Interrupt Mask register	This register controls forwarding of the interrupt to CPU.
SRSS_INTR_MASKED	SRSS Interrupt Masked register	This register shows the logical AND of the corresponding SRSS interrupt request (SRSS Interrupt register) and mask bits (SRSS Interrupt Mask register)
HSIOM_MONITOR_CTL_0	Power/Ground Monitor Cell Control 0 register	This register controls the connectivity of power/ground monitor cells to either AMUXBUS A or B respectively.
PWR_REGHC_CTL	High-Current Regulator Control register	This register shows the control for the high-current regulator controller.
PWR_REGHC_CTL2	High-Current Regulator Control register 2	This register shows the control for the high-current regulator controller.
PWR_REGHC_CTL4	High-Current Regulator Control register 4	This register shows the control for the high-current regulator controller.
PWR_REGHC_STATUS	High-Current Regulator Status register	This register shows the Status register for the high-current regulator controller.

17 Device power modes

The XMC7000 device can operate in different power modes that are intended to minimize the average power consumption in an application. The power modes supported by XMC7000 MCUs in the order of decreasing power consumption are:

- Active – all peripherals are available
- Low-power active (LPACTIVE) profile – Low-power profile of active mode where all peripherals including the CPU are available, but with limited capability
- Sleep – all peripherals except the CPU are available
- Low-power sleep (LPSLEEP) profile – Low-power profile of sleep mode where all peripherals except the CPU are available, but with limited capability
- Deep sleep – only low-frequency peripherals are available
- Hibernate – the device and I/O states are frozen and the device resets on wakeup
- XRES – the device enters this state when the XRES_L pin is asserted

Active, sleep, and deep sleep are standard Arm®-defined power modes supported by the Arm® CPUs and instruction set architecture (ISA). Hibernate mode is an additional low-power mode supported in XMC7000. LPACTIVE and LPSLEEP are similar to active and sleep modes, respectively; however, the high-current components are either frequency or current limited or turned off. Hibernate mode and XRES state are the lowest power mode/state that the XMC7000 device can be in. On wakeup from XRES or hibernate mode, the CPU and most peripherals go through a reset. Peripherals such as RTC or watchdog can be used during any of these power modes and also to trigger a transition to other active power modes.

17.1 Features

XMC7000 power modes have the following features:

- Software can use power modes to optimize power consumption in an application
- Low-power deep sleep mode with support for multiple wakeup sources and configurable amount of SRAM retention
- Ultra-low-power hibernate mode with wakeup from I/O and timer alarms

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

- Enabling and disabling clocks to peripherals
- Powering on/off clock sources
- Powering on/off peripherals and parts inside the device

17.2 Device power modes

Table 99 summarizes the power modes available in XMC7000 MCUs, their description, and details on entry and exit conditions.

Table 99 XMC7000 MCU power modes

Power mode	Description	Entry condition	Wakeup source	Wakeup action
Active	Primary mode of operation; all peripherals are available (programmable).	Wakeup from sleep/deep sleep modes, hibernate reset, or any other reset.	Not applicable	Not applicable

(table continues...)

Table 99 (continued) XMC7000 MCU power modes

Power mode	Description	Entry condition	Wakeup source	Wakeup action
Low-power active profile	A low-power profile of active mode; most peripherals are available with limited capabilities	Register write from active mode and wakeup from LPSLEEP/deep sleep modes.	Not applicable	Not applicable
Sleep	CPU is in sleep mode; all other peripherals are available.	Register write from active mode or wakeup from deep sleep through debugger	Any interrupt to CPU	Interrupt
Low-power sleep profile	A low-power profile of sleep mode; CPU is in sleep mode; most peripherals are available with limited capabilities.	Register write from LPACTIVE mode.	Any interrupt to CPU	Interrupt
Deep sleep	All high-frequency clocks and peripherals are turned off. Low-frequency clock (ILO) and low-power analog and digital peripherals are available for operation and as wakeup sources. SRAM can be retained (configurable).	Register write from active or LPACTIVE modes.	GPIO interrupt, event generators, SCB[1], watchdog timer, and RTC alarms[2] and debugger	Interrupt or debug
Hibernate	GPIO states are frozen; all high-frequency clocks and peripherals are switched off. Low-frequency clocks (32 kHz), WCO can function. Device resets on wakeup event.	Register write from active or LPACTIVE modes.	WAKEUP pins, RTC alarm, and watchdog timer	Hibernate reset

1) See the device-specific datasheet for the SCB-instance capable of waking up the device from Deep sleep mode.
2) RTC (along with optional WCO) is supplied with V_{DDD} and is available irrespective of the device power mode. RTC alarms are capable of waking up the device from any power mode.

17.2.1 Active and sleep modes

The active and sleep modes are the standard Arm®-defined power modes supported by both Cortex®-M7 and Cortex®-M0+ cores.

The device enters active mode upon any reset. In this mode, the CPU executes code along with all logic and memory powered. The firmware may decide to enable or disable specific peripherals and power domains depending on the application and power requirement. All the peripherals are available for use in active mode.

In sleep mode, the CPU clock is turned off and the CPU enters sleep. Note that in XMC7000, both Cortex®-M7 and Cortex®-M0+ support their own CPU sleep modes and each CPU can be in sleep, independent of the state of the other CPU. But the device is said to be in sleep mode, when both the cores are in sleep. All peripherals available in active mode are available in sleep mode. Any unmasked interrupt can wake up the CPU to active mode.

17.2.1.1 Low-power profiles - LPACTIVE and LPSLEEP

Low-power profiles are intended to reduce power consumption during active or sleep mode. They are software-controlled configurations to fine-tune current consumption. Power consumption can be reduced by controlling the following parameters:

- Reducing frequency either by selecting a slower source (such as IMO) or selecting a different output frequency (such as FLL and PLL), or dividing the clock using the pre-divider or PERI dividers.
- Disabling unnecessary clocks either at the source or by disabling the clock root muxes.
- Disabling unused circuitry such as low-voltage detection (LVD), which are used periodically to monitor external power supply source (such as battery).
- Disabling internal clock sources that are not generating a system clock. All clock sources are initially disabled, except the IMO. Note that some clock sources, such as the crystal oscillators (WCO and ECO) have relatively long startup times. Switching these circuits off and on may result in more overall current if the system must idle while they start up.
- Either Cortex®-M7 or Cortex®-M0+, or both can be put to the sleep state by controlling the clock provided to them. Cores can be put to either sleep or deep sleep states depending on the configurations.
- Firmware may allow disabling the flash macro. An unused macro can be disabled to reduce static current consumption; this can be done dynamically based on the application need to access a macro. Further, some code can be copied to SRAM and run from there, because reading from SRAM takes less current than reading from flash. In such a case, it may be possible to disable the flash macro. The current savings needs to be compared with the cost of re-enabling the macro and copying the code.

Examples of low-power profiles are as follows.

- LPACTIVE: Low-speed source clock (IMO), PLL/FLL off, Cortex®-M7 in sleep mode, and Cortex®-M0+ in active mode.
- LPSLEEP: Low-speed source clock (IMO), PLL/FLL off, Cortex®-M7 in deep sleep mode, and Cortex®-M0+ in sleep mode.

Using such configurations in combination with cyclic wakeup from deep sleep can help achieve low-power operation.

17.2.1.2 CM7 power domains

When the system is in active power mode, each CM7 CPU can be put in one of the following CM7 power modes shown in Table 100 using the CPUSS_CM7_x_PWR_CTL MMIO Register. See the XMC7100 registers technical reference manual or XMC7200 registers technical reference manual for the register description.

Table 100 CM7 power domains

Power mode	Description
ENABLED	Switch CM7 ON Power ON, clock ON
RESET	Reset CM7 Clock OFF, no retain and reset. ***Note**: CM7 CPU has the AIRCR.SYSRESETREQ register field, which allows the CM7 to reset the complete device.*

(table continues...)

Table 100 (continued) CM7 power domains

Power mode	Description
RETAINED	Put CM7 in retained mode. This can only become effective if CM7 is in deep sleep CPU power mode. Check the PWR_DONE flag to see if CM7 RETAINED state is reached. Power OFF, clock OFF, retain and no reset.
OFF	Switch CM7 OFF. Power OFF, clock OFF, no retain and reset.

***Note**: Accessing the CM7 ITCM/DTCM by other masters, at addresses CM7_0_ITCM_ADDR, CM7_0_DTCM_ADDR, CM7_1_ITCM_ADDR, and CM7_1_DTCM_ADDR is possible only when the power mode of the corresponding CM7 is enabled.*

Before changing the CM7 power mode from ENABLED to another mode, perform the following steps (accessing the TCM without these steps may cause the device to hang):

1. Disable access to the CM7 TCM by setting the CM7_0/1_CTL.TCMC_EN field to '0'.
2. Confirm that there are no outstanding accesses to the CM7 TCM from other bus masters by checking if the CM7_0/1_STATUS.TCMC_* fields are '0'. Repeat step 2 if necessary.
3. Now it is safe to change the CM7 power mode.

When the debugger is connected, OFF and RESET modes behave similarly. If the mode is changed from ENABLED to RESET/OFF of any CM7 core, the respective CM7 comes to the reset handler and starts execution from the Vector table base.

When the debugger is not connected, CM7 can be transitioned to ENABLED mode from either RESET or OFF, by configuring CPUSS_CM7_x_PWR_CTL.PWR_MODE to ENABLED.

It is recommended to use RESET when CM7 is intended to go through reset, and OFF when the intention is to save power by switching off CM7.

17.2.2 Deep sleep mode

In deep sleep mode, all the high-speed clock sources are off and high-speed peripherals are unusable. Low-speed clock sources and peripherals continue to operate, if configured and enabled by the firmware. In addition, peripherals that do not need a clock or receive clock from their external interface continue to operate, if configured for deep sleep operation. XMC7000 provides an option to configure the amount of SRAM, in blocks of 32 KB, to be retained during deep sleep.

Note that both Cortex®-M0+ and Cortex®-M7 can enter their local deep sleep mode independently. However, the entire device enters deep sleep mode only when both the CPUs are in the deep sleep state. The device can enter deep sleep mode after the following conditions are met.

- PWR_CTL.LPM_READY should read '1'. This ensures the device is ready to enter low-power modes. If the PWR_CTL.LPM_READY reads '0', then the device will enter normal CPU sleep instead of deep sleep until the bit is set, at which instant the device will automatically enter deep sleep mode, if requested.
- Both Cortex®-M0+ and Cortex®-M7 are in deep sleep. This is achieved by setting SCR.SLEEPDEEP of both Cortex®-M0+ (CM0P_SCS_SCR) and Cortex®-M7 (CM7_0/CM7_1_SCS_SCR).
- Debugger is not connected.

Refer to Debugger effect on device power modes for more information about how the debug session affects power mode transitions. Cortex®-M0+ must make sure that there are no pending flash memory transactions (write/erase operation) before going to deep sleep mode.

In this mode, the active mode regulator is turned off and a low-power deep sleep regulator supplies peripherals in deep sleep mode. Table 104 provides the list of resources available in deep sleep mode.

Interrupts from low-speed asynchronous or low-power analog peripherals can cause a CPU wakeup from deep sleep mode. A debug wakeup from deep sleep returns to sleep mode.

17.2.3 Hibernate mode

Hibernate mode is the lowest power mode of the device when external supplies are still present and XRES_L is deasserted. It is intended for applications in a dormant state. In this mode, both the active and deep sleep regulators are turned off and GPIO states must be frozen.

Hibernate mode is entered by performing three identical writes to the PWR_HIBERNATE Register. Each of these writes should have the UNLOCK code, set FREEZE, set hibernate commands, and load the other fields (TOKEN, POLARITY_HIBPIN, MASK_HIBPIN, MASK_HIBALARM, MASK_HIBWDT) as desired. The first write unlocks hibernate; the second freezes the I/Os; and the third enters hibernate mode. Unlike entry to deep sleep mode, active debug session cannot prevent transition to hibernate mode. Instead, after the device enters hibernate mode, debugger host will be disconnected.

Hibernate mode is exited by either generating a wakeup event or asserting XRES_L. A wakeup event can come from dedicated wakeup pins (up to two pins) with configurable polarity or through alarms from RTC or WDT wakeup event. All wakeup signals from GPIO pins or XRES_L are level-sensitive and must be held long enough for the hibernate bit to be cleared. Set the respective mask bits for alarm, WDT, or for external pins to wake up the device from hibernate mode. See the device datasheet for the supported number of pins that can wake up the device from hibernate mode.

***Note**: See the device datasheet for information about the number of wakeup pins supported. For unsupported pins, the respective MASK_HIBPIN bits must not be set to HIGH.*

The device goes through a reset (except RTC, Backup Registers, and Hibernate Registers) on wakeup and I/O pins must be unfrozen by firmware upon entering active mode. The PWR_HIBERNATE (except the hibernate bit [31]) register along with the PWR_HIB_DATA Register are retained through the hibernate wakeup sequence and can be used by the application to retain some content through the hibernate wakeup sequence. Note that these registers are reset by other reset events. On a hibernate wakeup event, PWR_HIBERNATE.HIBERNATE bit is cleared.

Asserting XRES_L in hibernate mode will lead to device reset; however, it is not the wakeup event. In this case, GPIOs will lose their frozen state and will be tristated.

Consider these restrictions while using hibernate mode:

- Supplies must be stable through hibernate mode
- Supplies must remain stable from 250 µs before entering hibernate mode until hibernate is fully entered. This allows the key writer to write all requested keys completely. Failure to observe this requirement can result in undefined behavior.
- The brownout detect (BOD) or overvoltage detect (OVD) blocks are not available in hibernate mode. As a result, the device will not recover from a brownout or overvoltage event in hibernate mode. If detection is needed, an external supervisor can be used to assert XRES_L in a brownout or overvoltage condition. Otherwise, it is recommended not to enter hibernate mode in applications that require brownout or overvoltage detection.

If these restrictions are unacceptable, accidental entry into hibernate mode can be prevented using the disable option – set PWR_HIBERNATE.HIBERNATE_DISABLE. Note that this bit is a write-once bit during execution and will be cleared on reset.

***Note**:*

- *SRAM cannot be retained in hibernate mode.*
- *The device has a separate clock domain, which can be ON, irrespective of the power modes mentioned earlier. This domain contains an RTC and WCO. The RTC provides an option to wake up the device from any of the low-power modes. It can be clocked by an external clock source such as WCO or by the internal low-speed oscillator (ILO0). This always-on domain is powered internally by V_{DDD}. It also offers a set of 32-bit Backup Registers (BACKUP_BREGx), which will retain the data through deep sleep and hibernate modes.*

17.2.4 Other operational states

In addition to the power modes discussed in the previous sections, there are two other states the device can be in – XRES and OFF state. You do not need a firmware action to enter these states or an interrupt or wakeup event to exit them. The device may be in these states if it is not in any of the modes described earlier.

17.2.4.1 XRES/OFF state

XRES is the device state when an external reset (XRES_L pin) is applied. XRES is not a power mode. During the XRES state, all the components in the device are powered down and I/Os are tristated keeping the power consumption to a minimum. The OFF state simply represents the device state with no power or insufficient power applied. The XRES and OFF states are discussed for completeness of all possible states the device can be in.

17.2.4.2 Reset

Reset is an intermediate state while the device starts.

17.3 Power mode transitions

Figure 72 shows various states the device can be in along with possible power mode transition paths. The transitions are described in detail in later sections.

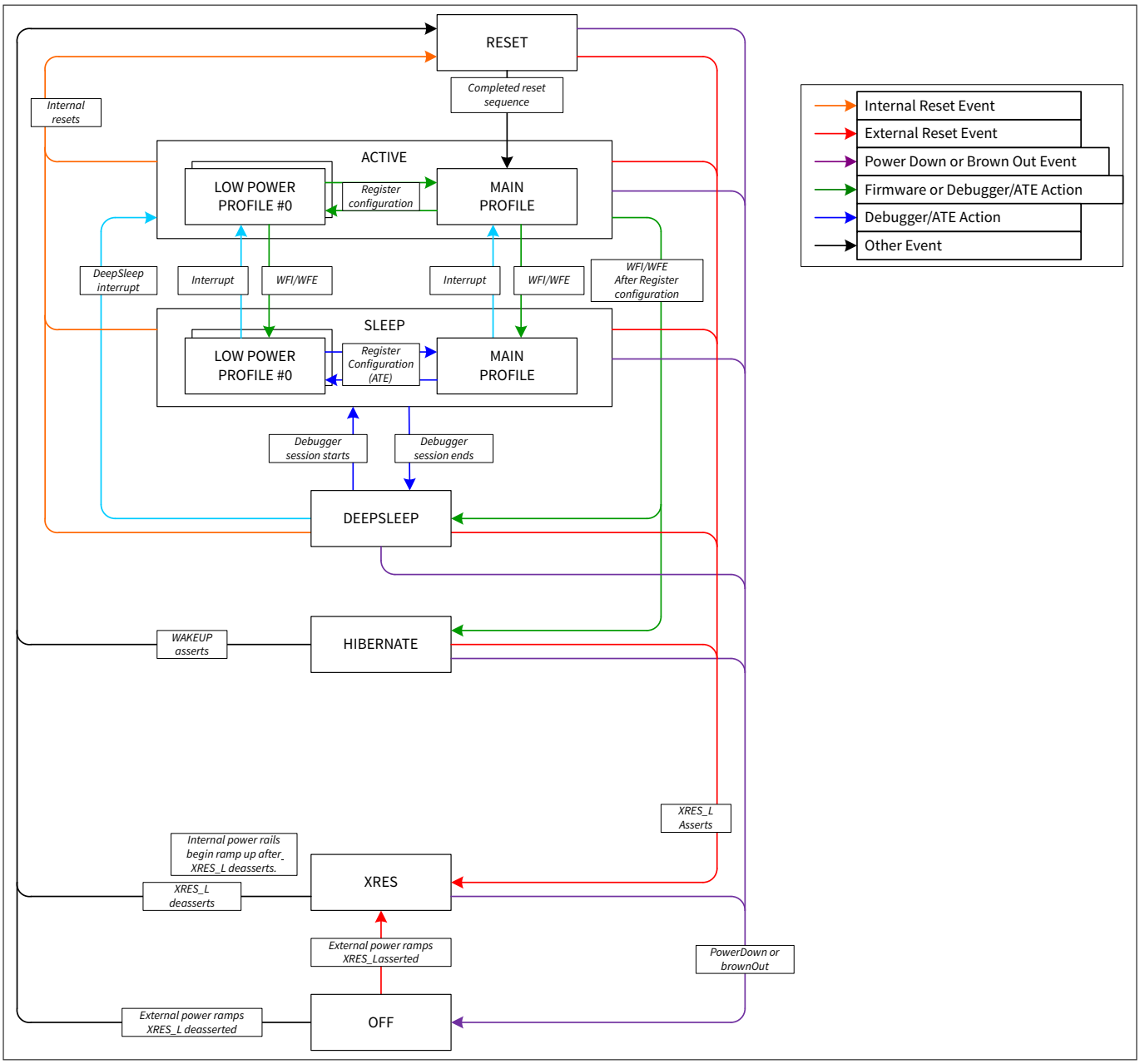


Figure 72 Power mode transition

***Note**: Most power mode transitions are implemented atomically and are not interruptible. The exceptions to this are the removal of external power, assertion of XRES_L, and a reset that occurs during deep sleep mode (for example, watchdog timer); these will cause an immediate transition to OFF, XRES, and RESET states respectively. Any reset returns the system to active, after executing the appropriate reset sequence.*

17.3.1 Power-up transitions

Table 101 summarizes various power-up transitions, their type, triggers, and actions.

Table 101 Power mode transition

Initial state/ mode	Final state/ mode	Type	Trigger	Actions
OFF	XRES	External	Power rail (V_{DDD}) ramps up above POR voltage level with XRES_L pin asserted.	**1.** All high-voltage logic such as SRSS, WDT, BOD/POR, hibernate control,and IMO are reset. Low-voltage logic is powered OFF.
OFF	Reset	External	Power rail (V_{DDD}) ramps up above POR and BOD voltage level with XRES_L pin deasserted.	**1.** All high-voltage logic is reset **2.** Low-voltage (internal active and deep sleep mode) regulators and references are ramped up. **3.** All low-voltage logic (operating from internal regulators) such as CPUs, high-speed peripherals, MCWDT, and low-speed peripherals are reset. **4.** IMO clock is started.
XRES	Reset	External	XRES_L pin is deasserted with V_{DDD} present and above POR and BOD level.	**1.** Low-voltage regulators and references are ramped up. **2.** All low-voltage logic is reset **3.** IMO clock is started.
Reset	Active	Internal	Reset sequence completes. This transition can also be caused by internal resets or hibernate wakeup event.	**1.** Clock is released to the system. **2.** System reset is deasserted. **3.** CPU starts execution.

17.3.2 Low-power mode transition

Table 102 shows the sequence from low-power mode to active mode.

Table 102 Low-power mode transitions

Initial state/ mode	Final state/ mode	Type	Trigger	Actions
Active	Sleep	Internal	Firmware action **1.** Clear SCR.SLEEPDEEP for both Cortex®-M0+ (CM0P_SCS_SCR) and Cortex®-M7 (CM7_0/CM7_1_SCS_SCR). **2.** Optionally, set SCR.SLEEPONEXIT if the CPU runs only on interrupts. When this bit is set, the CPU will not return to application code after the WFI/WFE instruction is executed. The CPU will wake up on any enabled interrupt or event and will enter sleep/deep sleep mode as soon as it exits the interrupt or services the event. **3.** Optionally, set SCR.SEVONPEND if the application needs to wake up the CPU from any pending interrupt. If this bit is set, any interrupt that enters a pending state will wake up the CPU. **4.** Execute WFI/WFE instruction on both CPUs.	**1.** CPU clocks are gated OFF. **2.** CPU waits for an interrupt or event to wake it up.

(table continues...)

Table 102 (continued) Low-power mode transitions

Initial state/ mode	Final state/ mode	Type	Trigger	Actions
Active	Deep Sleep	Internal	Firmware action If any FLL/PLL operates with the ECO as its reference clock, change the clock to either ECO direct or IMO direct or IMO with FLL/PLL before entering Deep Sleep. Perform these steps to enter Deep Sleep mode (PWR_CTL.LPM_READY should read '1' before performing these steps): **1.** Set SCR.SLEEPDEEP for both Cortex®-M0+ (CM0P_SCS_SCR) and Cortex®-M7 (CM7_0/CM7_1_SCS_SCR). **2.** Optionally, set SCR.SLEEPONEXIT if the CPU runs only on interrupts. When this bit is set, the CPU will not return to application code after the WFI/WFE instruction is executed. The CPU will wake up on any enabled interrupt or event and will enter sleep/deep sleep mode as soon as it exits the interrupt or services the event. **3.** Optionally, set SCR.SEVONPEND if the application needs to wake up the CPU from any pending interrupt. If this bit is set, any interrupt that enters a pending state will wake up the CPU. **4.** Execute WFI/WFE instruction on both CPUs. ***Notes:*** ***1.*** *Executing this sequence before the low-power mode is ready ((PWR_CTL.LPM_READY==1) will make the transition first to sleep mode. The device state will automatically move to deep sleep when PWR_CTL.LPM_READY is set.* ***2.*** *Make sure that any write transfer made before executing the WFI instruction is followed by the read access to the same memory location. This ensures that the write operation is successful.*	**1.** CPU enters low-power mode. **2.** High-frequency clocks are shut down. **3.** I/O cells associated with deep sleep-enabled blocks will be functional; the remaining I/Os and their configurations will be frozen automatically.[1)] **4.** Retention is enabled and non-retention logic is reset. **5.** Active regulator is disabled and Deep Sleep regulator takes over.

(table continues...)

Table 102 (continued) Low-power mode transitions

Initial state/ mode	Final state/ mode	Type	Trigger	Actions
Active	Hibernate	Internal	Firmware action **1.** Set PWR_HIBERNATE.TOKEN (optional) and PWR_HIB_DATA Register to some application-specific branching data that can be used on a wakeup event from hibernate mode. **2.** Set PWR_HIBERNATE.UNLOCK to 0x3A, this ungates writes to PWR_HIBERNATE.FREEZE and PWR_HIBERNATE.HIBERNATE bits. **3.** Configure wakeup pins polarity (PWR_HIBERNATE.POLARITY_HIBPIN), wakeup pins mask (PWR_HIBERNATE.MASK_HIBPIN), wakeup alarm mask (PWR_HIBERNATE.HIBALARM), and watchdog interrupt mask (PWR_HIBERNATE.MASK_HIBWDT) based on the application requirement. **4.** Set PWR_HIBERNATE.FREEZE to freeze the I/O pins. **5.** Set PWR_HIBERNATE.HIBERNATE to enter hibernate mode. **6.** Read the PWR_HIBERNATE Register to make sure that the write has taken effect. **7.** Execute WFI instruction on both CPUs. ***Note**: Transition to HIBERNATE mode can be canceled before setting PWR_HIBERNATE.HIBERNATE. To do so, clear PWR_HIBERNATE.FREEZE and UNLOCK to return to ACTIVE mode.* ***Note**: It is recommended to trigger hibernate mode atomically. This means, when entering the hibernate mode, disable all the interrupts and do a write operation on the PWR_Hibernate register.*	**1.** CPU enters low-power mode. **2.** Both high-frequency and low-frequency clocks except RTC are shut down. **3.** Pin output states and configurations are frozen. **4.** Both active and Deep Sleep regulators are powered down. The peripherals that are active in the hibernate domain operate directly out of V_{DDD}.

(table continues...)

Table 102 (continued) Low-power mode transitions

Initial state/ mode	Final state/ mode	Type	Trigger	Actions
			***Note**: Make sure that any write transfer made before executing the WFI instruction is followed by the read access to the same memory location. This ensures that the write operation is successful.*	
Sleep	Deep sleep	Internal	When the debugger is not connected and deep sleep mode is triggered, but PWR_CTL.LPM_READY==0, the device internally enters the sleep mode. The device will automatically transit to deep sleep when PWR_CTL.LPM_READY==1. If the debugger is connected and deep sleep mode is triggered by the firmware, the device will enter Deep Sleep only when the following conditions are met. **1.** PWR_CTL.LPM_READY==1 **2.** Debugger is disconnected	**1.** High-frequency clocks are shut down. **2.** I/O cells associated with deep sleep-enabled blocks will be functional; the remaining I/Os and their configurations will be frozen automatically. **3.** Retention is enabled and non-retention logic is reset. **4.** Active regulator is disabled and Deep Sleep regulator takes over.

1) If the port selects peripherals IP (except for CAN FD) and the port output value needs to be maintained after wakeup from Deep Sleep,set HSIOM_PRTx_PORT_SEL.IOy_SEL = 0 (GPIO) before Deep Sleep and set the required output value in GPIO configuration registers. After wakeup, change HSIOM_PRTx_PORT_SEL.IOy_SEL back to the peripheral IP.

17.3.3 Wakeup

Table 103 shows the sequence from low-power mode to active mode.

Table 103 Wakeup sequence

Initial state/ mode	Final state/ mode	Trigger source	Action
Sleep	Active	Any enabled interrupt	CPU exits sleep mode and executes the interrupt
Deep sleep	Active	Low-speed peripherals or interrupt from deep sleep peripheral	Device returns to the configuration it had while entering deep sleep mode. **1.** IMO/clocks enabled **2.** Non-retained state is reset **3.** GPIOs are unfrozen **4.** CPU exits low-power mode and executes interrupt

(table continues...)

Table 103 (continued) Wakeup sequence

Initial state/ mode	Final state/ mode	Trigger source	Action
Deep sleep	Sleep	Debug wakeup	**1.** Non-retained state is reset **2.** GPIOs are unfrozen **3.** High-frequency and low-frequency clocks are ON **4.** CPU remains in sleep
Hibernate	Active	RTC, WDT, wakeup from up to two pins	Hibernate wakeup is implemented as transition to active mode through reset. **1.** Low-voltage (internal active and deep sleep mode) regulators and references are ramped up **2.** All low-voltage logic is reset **3.** IMO clock starts **4.** CPU starts execution **5.** Software must unfreeze the I/Os by unlocking and clearing the PWR_HIBERNATE.FREEZE bit.

17.3.4 Internal reset transitions

When an internal reset occurs:

- I/O cells are disabled (excluding the PMIC control interface)
- Most low-voltage logic is reset. Exceptions include reference settings, regulator settings, reset cause registers, and fault logging system
- Most high-voltage logic is not reset, including hibernate peripherals, WDT, and RTC and BREG registers

When the device is in Active/Sleep mode and internal reset occurs, the reference and regulator settings are not changed; SRSS enables the IMO (if disabled) and makes the reset to active transition.

While the device is in Deep Sleep mode and internal reset occurs, then I/O cells are disabled (Hi-Z), most low-voltage logic is reset, retention is disabled, regulators are enabled, IMO starts, and the device enters active state.

17.3.5 Powering down/brownout/overvoltage

This transition occurs when power is partly or partially lost, and as a result one of the brownout or overvoltage detectors execute reset.

Note that the detectors are disabled in hibernate mode. If V_{DDD} is removed or becomes invalid in hibernate mode, then the system must restart with XRES_L applied. This is because the logic dependent on the V_{DDD} will slowly discharge and may become invalid.

17.3.6 Debugger effect on device power modes

The debugger uses non-AHB Registers through SWD and JTAG port to transition to/from debug mode. After the debugger connection is established with the device, CPUSS_DP_STATUS.SWJ_CONNECTED bit is set. Debugger connection is possible when the device is in active/sleep/deep sleep power modes but not when the device is in hibernate power mode.

Device will behave differently in certain cases when the debug connection is active. Some instances are as follows:

- Attempt to enter deep sleep mode results in a transition to sleep mode instead, with power and clocks unchanged
  - System power consumption is the same as sleep mode, which is higher than deep sleep mode.
  - Wakeup time will be the sleep wakeup time, which is shorter than a deep sleep wakeup time.
  - Non-retention Registers will not be reset upon wakeup and may lead to behavior that is different from its actual operation during a debug session.
- If the debugger is disconnected during active deep sleep request, SRSS will transition to deep sleep mode.
- A debugger connection request, when device is in deep sleep mode, will take the device to sleep mode. Mode transition from deep sleep to sleep is described in Wakeup.

An active debug session cannot prevent hibernate or OFF mode entry. When the device enters these modes, the debugger will be disconnected because the debug port is no longer powered.

17.4 Summary

Table 104 captures various device components and their availability during the device power modes/states.

Table 104 Resource available in different power modes/states

Component	Power modes/sates				
	Active/sleep	**Deep sleep**	**Hibernate**	**XRES**	**OFF**
Wakeup					
Wakeup event	Any interrupt	Deep sleep peripherals/ GPIOs	Dedicated wakeup pins[1)]	XRES	Power on
Wakeup action	Interrupt	Interrupt	Reset	Reset	Reset
Core function					
CPU	ON/sleep	Retention	OFF	OFF	OFF
SRAM	ON	Retention (opt)[2)]	OFF	OFF	OFF
Flash	ON	OFF	OFF	OFF	OFF
High speed clock (IMO, ECO, PLL)	ON	OFF	OFF	OFF	OFF
LVD	ON (opt)	Slow (opt)[3)]	OFF	OFF	OFF
ILO	ON	ON	ON	OFF	OFF
CSV	ON (opt)	ON (opt)	ON[4)]	OFF	OFF
Peripherals					
M_TTCAN	ON (opt)[5)]	Retention	OFF	OFF	OFF
WDT/MCWDT	ON (opt)	ON (opt)	WDT (opt)/ MCWDT off	OFF	OFF
ADC	ON (opt)	Retention	OFF	OFF	OFF
TCPWM	ON (opt)	OFF	OFF	OFF	OFF
SCB	ON (opt)	ON (opt)	OFF	OFF	OFF

(table continues...)

Table 104 (continued) Resource available in different power modes/states

Component	Power modes/sates				
	Active/sleep	**Deep sleep**	**Hibernate**	**XRES**	**OFF**
GPIO	ON	ON/Freeze	Freeze	Hi-Z	Hi-Z
Supplies and reset					
XRES_L	Deassert	Deassert	Deassert	Assert	Deassert
BOD	ON	Slow[6)]	OFF	OFF	OFF
POR	ON	ON	ON	ON	OFF
$V_{DD}/V_{DDD}/V_{DDA}$	ON	ON	ON	ON	OFF
Backup domain					
RTC	ON (opt)	ON (opt)	ON (opt)	ON (opt)	OFF
WCO	ON (opt)	ON (opt)	ON (opt)	ON (opt)	OFF
Backup Registers	ON	ON (opt)	ON (opt)	ON (opt)	OFF

1) See the device-specific document to find the supported number of pins to wake up the device.
2) Write buffers are not retained in Deep sleep mode.
3) See the device-specific document for LVD Slow (Deep sleep) specification.
4) See the device-specific document to check if CSV function is available in hibernate mode.
5) When all M_TTCAN channels in a group are powered down, message RAM will be powered off to save power.
6) See the device-specific document for the BOD Slow (Seep sleep) specification.

17.5 Registers

Table 105 Register list

Register	Name	Description
PWR_CTL	Power Control	Power Mode Status register shows the current state and device ready status
PWR_HIBERNATE	Hibernate Mode Register	Controls various hibernate mode entry/exit related options
PWR_HIB_DATA	Hibernate Mode Data Register	Data Register that is retained through a hibernate wakeup sequence
CM7_0/ CM7_1_SCS_SCR	Cortex®-M7 System Control Register	Controls the CPU level Sleep/Deep Sleep decisions on WFI/WFE instruction execution
CM0P_SCS_SCR	Cortex®-M0+ System Control Register	Controls the CPU level Sleep/Deep Sleep decisions on WFI/WFE instruction execution

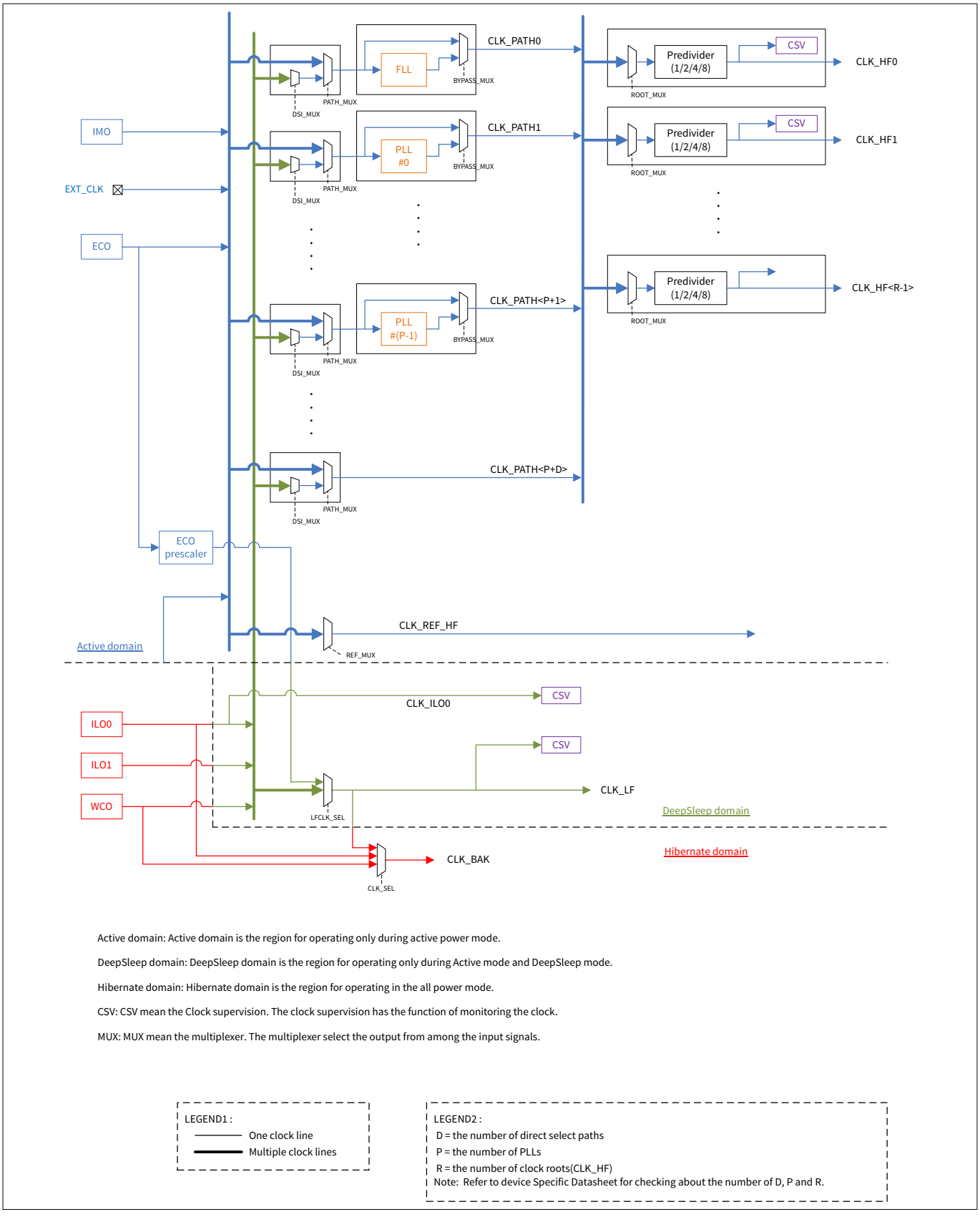
18 Clocking system

The XMC7000 family clocking system includes these resources:

- Three internal clock sources:
  - 8 MHz internal main oscillator (IMO)
  - Internal low-speed oscillators (ILO0/ILO1)
- Four external clock sources
  - External clock connected to one of two EXT_CLK inputs
  - External crystal oscillator (ECO)
  - External watch crystal oscillator (WCO)
- One frequency-locked loop (FLL)
- Several phase-locked loops (PLL)

18.1 Block diagram

Figure 73 gives a generic view of the clocking system in XMC7000 family devices.



Active domain: Active domain is the region for operating only during active power mode.

DeepSleep domain: DeepSleep domain is the region for operating only during Active mode and DeepSleep mode.

Hibernate domain: Hibernate domain is the region for operating in the all power mode.

CSV: CSV mean the Clock supervision. The clock supervision has the function of monitoring the clock.

MUX: MUX mean the multiplexer. The multiplexer select the output from among the input signals.

LEGEND1 :
One clock line
Multiple clock lines

LEGEND2 :
D = the number of direct select paths
P = the number of PLLs
R = the number of clock roots(CLK_HF)
Note: Refer to device Specific Datasheet for checking about the number of D, P and R.

Figure 73 Clocking system block diagram

18.2 Clock sources

18.2.1 Internal main oscillator (IMO)

The IMO is an accurate, high-speed internal (crystal-less) oscillator that produces a fixed frequency. Refer to the datasheet for the IMO frequency. The IMO output can be used by the PLL or FLL to generate a wide range of higher frequency clocks, or it can be used directly by the high-frequency root clocks. The IMO is enabled and disabled with CLK_IMO_CONFIG.ENABLE.

The IMO should not be disabled if it is the source of the clock path feeding high-frequency clock zero (CLK_HF0). CLK_HF0 is the source clock for the CPU. Therefore, if the IMO is in the source path of CLK_HF0, disabling the IMO disables the CPU.

The IMO is only available in active and sleep modes.

18.2.2 External crystal oscillator (ECO)

The XMC7000 family contains an oscillator to drive an external crystal. Refer to the datasheet for the frequency of ECO. This clock source is built using an oscillator circuit. The circuit employs an external crystal that needs to be populated on the external crystal pins of the XMC7000 device.

The ECO can be enabled using the CLK_ECO_CONFIG register fields.

18.2.2.1 ECO trimming

The ECO supports a wide variety of crystals and ceramic resonators with the nominal frequency range specification described in the datasheet. The crystal manufacturer typically provides numerical values for parameters, namely the maximum drive level (D_L), the equivalent series resistance (ESR), the ideal shunt capacitance (C_0) and the parallel load capacitance (C_L). These parameters can be used to calculate the transconductance (g_m) and the maximum peak oscillation voltage across the crystal (V_P).

The formula of V_P is as follows. ECO does not support V_P less than 0.3 V.

$$\text{Max peak value: } Vp = \frac{\sqrt{\frac{D_L}{2ESR}}}{\pi f(C_0 + C_L)} \tag{3}$$

The formula of Transconductance (g_m) is as follows. The ECO block can deliver a maximum Transconductance (g_m) of 17.6 mA/V.

$$\text{Transconductan ce: } g_m > 20 \times ESR \times (2\pi \times f)^2 \times (C_0 + C_L)^2 \tag{4}$$

The formula of Negative resistance (R_{neg}) is as follows. To guarantee crystal start up, negative resistance needs to be at least five times larger than ESR. The above gm is based on ESR * 5. If R_{neg} requires more than ESR * 10, double the gm value.

$$\text{Negative Resistance: } \left|R_{neg}\right| = \frac{g_m \times 4 \times C_L^2}{(2\pi \times f)^2 \times \left(4 \times C_L^2 + 4 \times C_L \times C_0\right)^2} \tag{5}$$

The ATRIM, WDTRIM, and FTRIM fields can be found in the CLK_ECO_CONFIG2 register. The ATRIM and WDTRIM settings control the trim for amplitude of the oscillator output. The FTRIM setting controls the filter used to prevent the third harmonic oscillation.

Amplitude trim (ATRIM) sets the crystal drive level when automatic gain control (AGC) is enabled (CLK_ECO_CONFIG.AGC_EN = 1). AGC must be enabled for V_P < 1.1 V and disabled for all other cases.

Watch dog trim (WDTRIM) sets the threshold on XO magnitude where the ECO block releases the clock to the system.

Filter trim (FTRIM) tunes the low-pass filter between the ECO_IN pin and the amplifier, which is used to prevent amplification of harmonics of the intended crystal frequency. The FTRIM value at 0x03 can be used; no other value is required.

Warning: ***The V_p setting is critical for reliable system performance. If the V_p settings are too large (or AGC is disabled), the crystal can be damaged or suffer premature aging due to excessive power dissipation. If the V_p settings are too small, the oscillation will be more susceptible to system noise.***

Based on the V_P value, the ATRIM, WDTRIM, and FTRIM values are set as shown in Table 106.

Table 106 ATRIM, WDTRIM, and FTRIM settings based on V_P

V_P [V]	AGC_EN[1)]	ATRIM	WDTRIM	FTRIM
0.5 ≤ V_P < 0.55	0x1	0x4	0x2	0x3
0.55 ≤ V_P < 0.60	0x1	0x5		
0.60 ≤ V_P < 0.65	0x1	0x6	0x3	
0.65 ≤ V_P < 0.70	0x1	0x7		
0.70 ≤ V_P < 0.75	0x1	0x8	0x4	
0.75 ≤ V_P < 0.80	0x1	0x9		
0.80 ≤ V_P < 0.85	0x1	0xA	0x5	
0.85 ≤ V_P < 0.90	0x1	0xB		
0.90 ≤ V_P < 0.95	0x1	0xC	0x6	
0.95 ≤ V_P < 1.00	0x1	0xD		
1.00 ≤ V_P < 1.05	0x1	0xE	0x7	
1.05 ≤ V_P < 1.10	0x1	0xF		
1.10 ≤ V_P	0x0	0x0-0xE[2)]		

1) If selected AGC_EN=0x0, the clock accuracy is improved compared to AGC_EN=0x1.
2) It is acceptable to select any value from 0x0 to 0xE.

The GTRIM sets up the trim for amplifier gain based on the calculated g_m, as shown in Table 107.

Table 107 GTRIM settings

g_m [mA/V]	GTRIM
0 ≤ g_m < 2.2	0x00
2.2 ≤ g_m < 4.4	0x01
4.4 ≤ g_m < 6.6	0x02
6.6 ≤ g_m < 8.8	0x03
8.8 ≤ g_m < 11	0x04

(table continues...)

Table 107 (continued) GTRIM settings

g_m [mA/V]	GTRIM
$11 \le g_m < 13.2$	0x05
$13.2 \le g_m < 15.4$	0x06
$15.4 \le g_m \le 17.6$	0x07

RTRIM should be oscillator feedback resistor, as shown in Table 108.

Table 108 RTRIM settings

Nominal Frequency f [MHz]	RTRIM
$28.6 < f$	0x00
$23.33 < f \le 28.6$	0x01
$16.5 < f \le 23.33$	0x02
$f \le 16.5$	0x03

First, set up the trim values based on Table 106 through Table 108 and then enable the ECO. After the ECO is enabled, the CLK_ECO_STATUS register can be checked to ensure it is ready.

18.2.3 External clock (EXT_CLK)

The external clock can be sourced from a signal on a designated I/O pin. This clock can be used as the source clock for either the PLL or FLL, or can be used directly by the high-frequency clocks.

When manually configuring a pin as the input to the EXT_CLK, the drive mode of the pin must be set to high impedance digital to enable the digital input buffer. See the I/O system for more details. Consult the device datasheet to determine the specific pin used for EXT_CLK.

The EXT_CLK function is bi-directional. See Clock generation for more details.

18.2.4 Internal low-speed oscillator (ILO)

The two ILO blocks operate with no external components and output a stable clock. Refer to the datasheet for the frequency of the two ILOs. The ILO block is relatively low power and low accuracy. It is available in all power modes. If the ILO is to remain active in hibernate mode, and across power-on-reset (POR) or brownout detect (BOD), CLK_ILO0_CONFIG.ILO0_BACKUP must be set.

The ILO blocks can be used as the clock source for:

- CLK_LF: CLK_LF in turn can be used as a source for the backup domain (CLK_BAK). CLK_BAK runs the real time clock (RTC). This can be useful if you do not wish to populate a WCO. Although the ILO is not suitable as an RTC due to its poor accuracy, it can be used as a HIBERNATE wakeup source using the wakeup alarm facility in the RTC. In this case, CLK_ILO0_CONFIG.ILO0_BACKUP must be set.
- DSI_Mux: While the ILOs are routable through the DSI_MUX, there are no supported use cases for doing so.
- ILO0 is the clock for the watchdog timer (WDT) and deep sleep CSV (clock supervision).
- ILO1 is used only if all of the following are true.
  - CLK_LF must be available in deep sleep mode (otherwise use ECO output).
  - Clock supervision of CLK_LF is necessary (otherwise, use ILO0).
  - WCO is not available (otherwise, use WCO).

The ILO0 and the ILO1 are enabled/disabled with CLK_ILO0_CONFIG.ENABLE and CLK_ILO1_CONFIG.ENABLE respectively. It is recommended to always leave ILO0 enabled as it is the source of the WDT.

User must ensure not to turn off the ILO0 (CLK_ILO0) before initiating any soft reset.

If the WDT is enabled, the only way to disable the ILO0 is to first clear WDT_CTL.WDT_LOCK and then clear CLK_ILO0_CONFIG.ENABLE. If the WDT_CTL.WDT_LOCK is set, any register write to disable the ILO0 will be ignored. Enabling the WDT will automatically enable the ILO0.

The MCU provides an opportunity to calibrate the ILOx by using the calibration counter, described in Clock calibration counters. For instance, the ECO can be used as a reference clock. This result can be used to determine how the ILOx needs to be adjusted. The ILO0 and ILO1 can be trimmed using the CLK_TRIM_ILO0_CTL and CLK_TRIM_ILO1_CTL registers.

18.2.5 Watch crystal oscillator (WCO)

The WCO is a highly accurate clock source. Refer to the datasheet for the frequency of WCO. It is the primary clock source for the RTC. The WCO can also be used as a source for CLK_LF.

The WCO can be enabled and disabled by setting BACKUP_CTL.WCO_EN for the backup domain. The WCO can also be bypassed and an external 32.768-kHz clock can be routed on a WCO output pin. This is done by setting BACKUP_CTL.WCO_BYPASS for the backup domain.

It is possible to improve the accuracy of the RTC by calibrating the WCO. WCO is routable through the DSI_MUX and can then be routed as the source of the FLL.

18.2.6 ECO prescaler

ECO prescaler divides the ECO and creates a clock that can be used with CLF_LF clock. This feature is only available during active and sleep mode. It cannot be used during deep sleep or hibernate modes.

The division function has a 10-bit integer divider and 8-bit fractional divider. This function is configured using the CLK_ECO_PRESCALE register.

18.3 Clock generation

This section explains phase locked loop (PLL) and frequency locked loop (FLL) implemented in the XMC7000 family.

XMC7000 family has two types of PLLs; PLL without SSCG and fractional operation, and PLL with SSCG and fractional operation. XMC7000 family has one type of FLL.

18.3.1 PLL without SSCG and fractional operation

Refer to the datasheet to identify where this PLL type is used. The datasheet also specifies the frequency range that can be input to the PLL and the frequency range that the PLL can output. This makes it possible to use the IMO or other clock to generate much higher clock frequencies for the rest of the system. Figure 74 shows the block diagram of a PLL without SSCG and fractional operation.

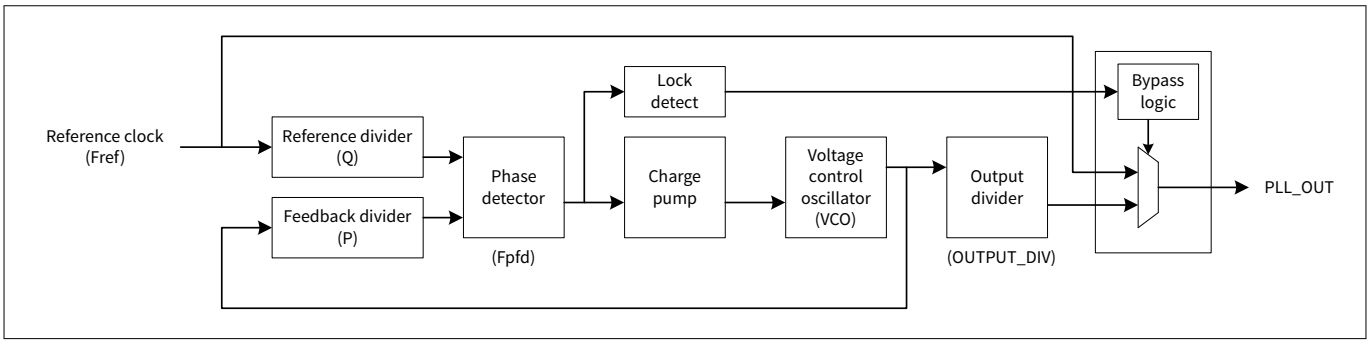


Figure 74 PLL without SSCG and fractional operation

The PLL is configured following these steps:

***Note**: Fref is the input frequency of the PLL; that is, the frequency of the input clock such as 8 MHz for the IMO.*

1. Determine the desired reference clock frequency (Fref) and desired output frequency (PLL_OUT). Calculate the reference (Q), feedback (P), and output (OUTPUT_DIV) dividers subject to the following constraints:
   a. PFD frequency (phase detector frequency). Fpfd = Fref / Q. There may be multiple reference divider values that meet this constraint.
   b. VCO frequency. VCO = Fpfd × P. There may be multiple feedback divider values that meet this constraint with different REFERENCE_DIV choices.
   c. Output frequency. PLL_OUT = VCO / OUTPUT_DIV. It may not be possible to get the exact desired frequency due to granularity; therefore, consider the frequency error of the two closest choices.
   d. Choose the best combination of divider parameters depending on the application.
2. Program the divider settings in the appropriate CLK_PLL_CONFIGx register. Do not enable the PLL on the same cycle as configuring the dividers. Do not change the divider settings while the PLL is enabled.
3. Enable the PLL (CLK_PLL_CONFIGx.ENABLE = 1). Wait at least 1 µs for PLL circuits to start.
4. Wait until the PLL is locked before using the output. By default, the PLL output is bypassed to its reference clock and will automatically switch to the PLL output when it is locked. This behavior can be changed using CLK_PLL_CONFIGx.BYPASS_SEL. The status of the PLL can be checked by reading CLK_PLL_STATUSx. This register contains a bit indicating the PLL has locked. It also contains a bit indicating if the PLL lost the lock status.

18.3.2 PLL with SSCG and fractional operation (400-MHz PLL)

Refer to the datasheet to identify where this PLL type is used. The datasheet also specifies the frequency range that can be input to the PLL and the frequency range that the PLL can output. This makes it possible to use the IMO or other clock to generate much higher clock frequencies for the rest of the system. Figure 75 shows the block diagram of a PLL with SSCG and fractional operation. This type of PLL is configured in the CLK_PLL400Mx_CONFIG register and the status is confirmed in the CLK_PLL400Mx_STATUS register.

The configuration of this PLL is the same as PLL without SSCG and fractional operation. See PLL without SSCG and fractional operation for details.

Note that you cannot operate SSCG and fractional operation together.

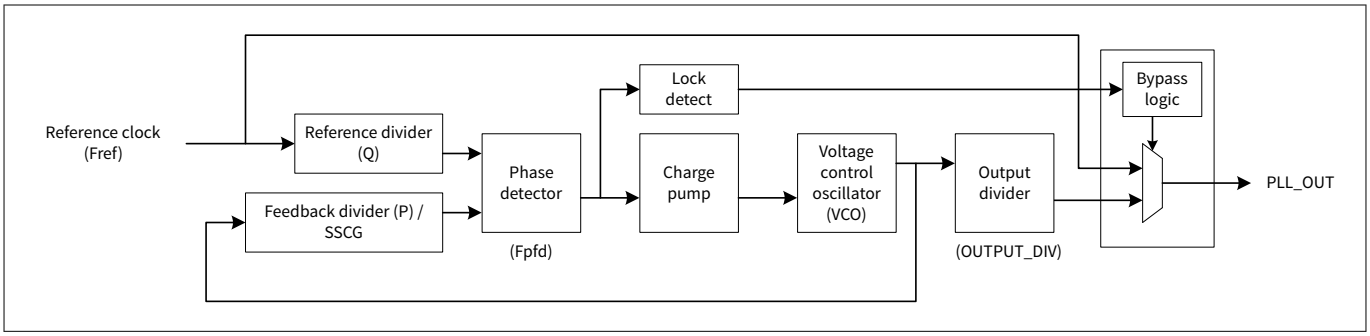


Figure 75 PLL with SSCG and fractional operation

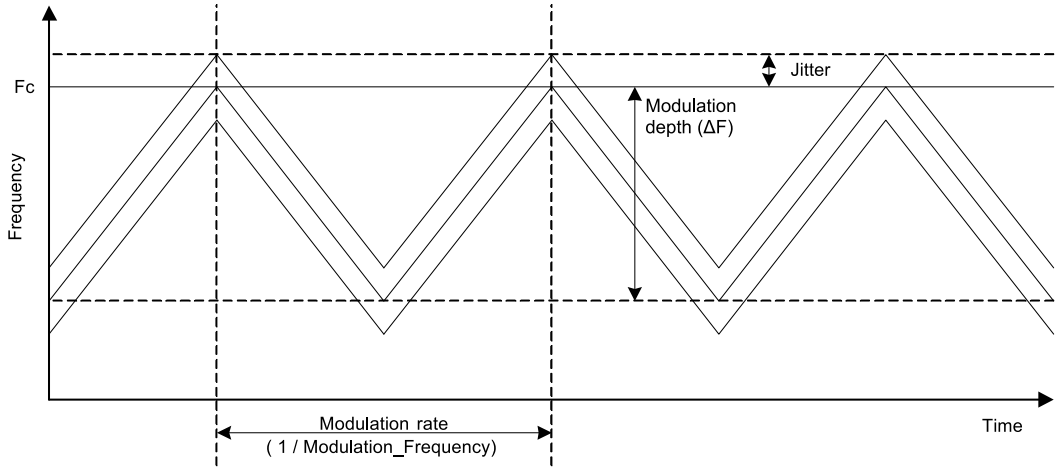
18.3.2.1 Spread spectrum clock generation (SSCG)

Spread spectrum clock generation (SSCG) is a method by which the energy contained in the narrow band of a clock source is spread over a wider band in a controlled manner, thus reducing the peak spectral amplitude of the fundamental and the harmonics to lower the radiated emission from the clock source. This is achieved by

modulating the clock frequency with a waveform. The configuration of the SSCG uses the CLK_PLL400Mx_CONFIG3 register.

Table 109 describes the three SSCG parameters:

Table 109 SSCG parameters

Parameters	Description
Modulation rate	Modulation rate (MR) is the rate (in Hz) at which the energy of the clock source is distributed over the band of frequencies around the output clock frequency. Modulation rate must be much lower than the source clock frequency, and must be above the audio frequency range.
Modulation depth	Modulation depth (also known as deviation) is the frequency range over which the clock changes while varying at the modulation rate. It is specified as a percentage (%), which is the ratio of the bandwidth of frequency excursion (ΔF) to the source clock frequency. This determines the amount of peak EMI reduction achievable. Generally, the larger the modulation depth, the greater the EMI reduction.
Modulation type	Modulation type (or spreading mode) specifies the relationship of the frequency deviation of the modulated clock relative to the non-modulated clock. For general SSCG, center, up, and down spread are available, but this PLL only supports down spread. Down spreading is where the maximum frequency of the spread spectrum clock is the same as that of the non-modulated clock. In a down-spread system, the output clock varies between (Fc − ΔF) and Fc at the modulation rate and following the modulation profile.  It can be represented as “Fout = Fc − ΔF”.

Fout = the modulated output clock frequency

Fc = the source or carrier frequency

ΔF = total frequency deviation (min to max)

18.3.2.2 Fractional operation

Fractional operation is a function that provides an output frequency that is a fractional multiple of the input frequency. When using fractional operation, the following formula holds.

PLL_OUT = (Fref / Q) × (P + Frac_div) / OUTPUT_DIV

The configuration of fractional operation uses the PLL400_CONFIG2 register. Frac_div is the value set by CLK_PLL400Mx_CONFIG2.FRAC_DIV divided by 2^{24}. While the fractional divider has 24 bits, accuracy is only guaranteed for the upper 21 bits.

18.3.3 Frequency locked loop (FLL)

The XMC7000 family device contains one FLL, which resides on CLK_PATH0. Refer to the datasheet for the frequency range that can be input to the FLL and the frequency range that the FLL can output. This makes it possible to use the IMO to generate much higher clock frequencies for the rest of the system. Figure 76 shows the block diagram of FLL.

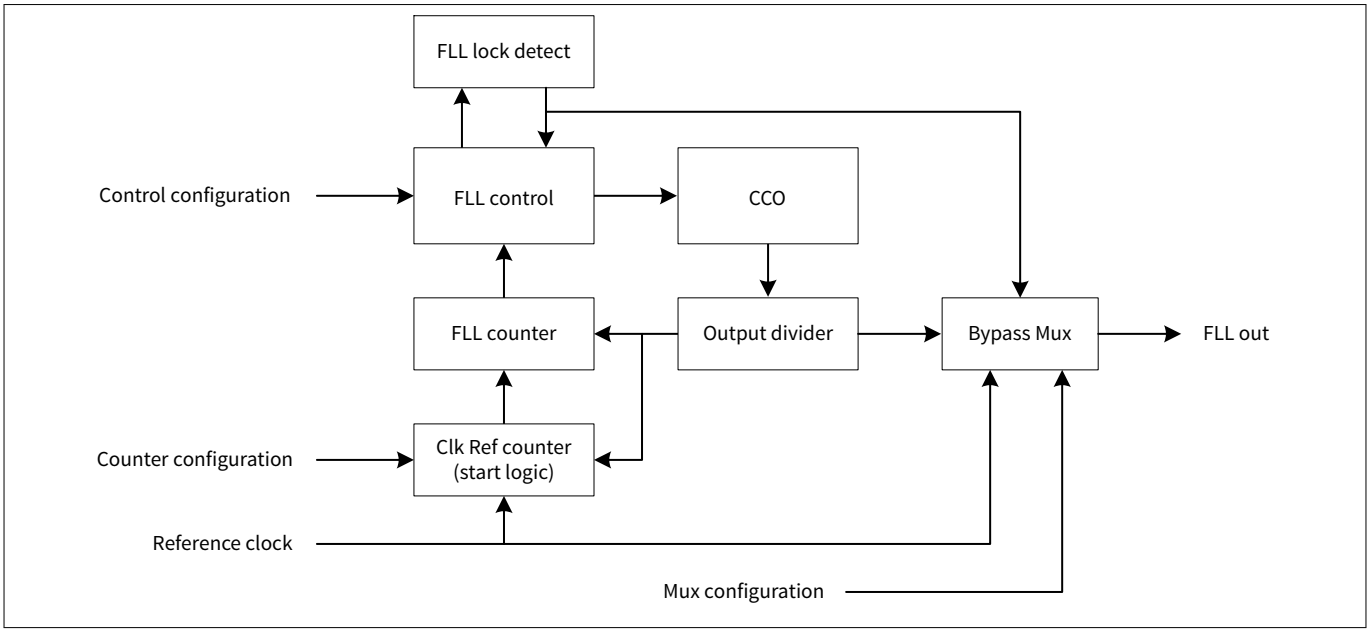


Figure 76 FLL block diagram

The FLL is similar in purpose to a PLL but is not equivalent; there are some differences:

- FLL can start up (lock) much faster than the PLL.
- It consumes less current than the PLL.
- FLL does not lock the phase. At the heart of the FLL is a current-controlled oscillator (CCO). The output frequency of this CCO is controlled by adjusting the trim of the CCO; this is done in hardware.
- FLL can produce a clock with good duty cycle through its divided clock output.
- FLL reference clock can be the WCO, IMO (8 MHz), or any other periodic clock source.

The CCO can output a stable frequency in the 48 MHz to 160 MHz range. This range is divided into five sub-ranges as shown by Table 110.

Table 110 CCO frequency range

CCO range	0	1	2	3	4
Fmin	48 MHz	64 MHz	85 MHz	113 MHz	150 MHz
Fmax	64 MHz	85 MHz	113 MHz	150 MHz	200 MHz

***Note**: The output of the CCO has an option to enable a divide by two or not. For this device, the divide by two must always be enabled. The output range of the FLL is shown in Table 110.*

Within each range, the CCO output is controlled via a 9-bit trim field. This trim field is updated via hardware based on the control algorithm described here.

A reference clock must be provided to the FLL. This reference clock is typically the IMO, but can be many different clock sources. The FLL compares the reference clock against the CCO clock to determine how to adjust the CCO trim. Specifically, the FLL will count the number of CCO clock cycles inside a specified window of reference clock cycles. The number of reference clock cycles to count is set by CLK_FLL_CONFIG2.FLL_REF_DIV.

After the CCO clocks are counted, they are compared against an ideal value and an error is calculated. The ideal value is programmed into CLK_FLL_CONFIG.FLL_MULT.

As an example, the reference clock is the IMO (8 MHz), the desired CCO frequency is 100 MHz, the value for CLK_FLL_CONFIG2.FLL_REF_DIV is set to 146. This means that the FLL will count the number of CCO clocks within 146 clock periods of the reference clock. In one clock cycle of the reference clock (IMO), there should be 100 / 8 = 12.5 clock cycles of the CCO. Multiply this number by 146 and the value of CLK_FLL_CONFIG.FLL_MULT should be 1825.

If the FLL counts a value different from 1825, it attempts to adjust the CCO such that it achieves 1825 the next time it counts. This is done by scaling the error term with CLK_FLL_CONFIG3.FLL_LF_IGAIN and CLK_FLL_CONFIG3.FLL_LF_PGAIN. Figure 77 shows how the error (err) term is multiplied by FLL_LF_IGAIN and FLL_LF_PGAIN and then summed with the current trim to produce a new trim value for the CCO. CLK_FLL_CONFIG4.CCO_LIMIT can be used to put an upper limit on the trim adjustment; this is not needed for most situations.

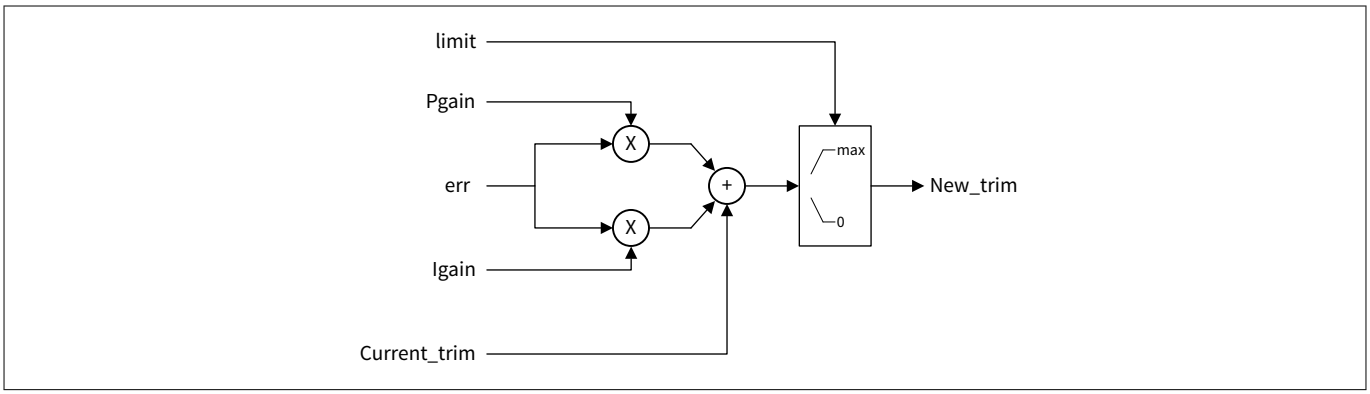


Figure 77 FLL error correction diagram

The FLL determines if it is “locked” by comparing the error term with CLK_FLL_CONFIG2.LOCK_TOL.

When the error is less than LOCK_TOL the FLL is considered locked.

After each adjustment to the trim the FLL can be programmed to wait a certain number of reference clocks before doing a new measurement. The number of reference clocks to wait is set in the CLK_FLL_CONFIG3.SETTLING_COUNT. It is recommended to set this such that the FLL waits ~1 µs before a new count. Therefore, if the 8 MHz IMO is used as the reference this field should be programmed to ‘8’.

When configuring the FLL there are two important factors that must be considered: lock time and accuracy. Accuracy is the closeness to the intended output frequency. These two numbers are inversely related to each other via the value of CLK_FLL_CONFIG2.FLL_REF_DIV.

Higher CLK_FLL_CONFIG2.FLL_REF_DIV values lead to higher accuracy, whereas lower CLK_FLL_CONFIG2.FLL_REF_DIV values lead to faster lock times.

In the example used previously the 8 MHz IMO was used as the reference, and the desired FLL output was 100 MHz. For that example, there are 12.5 CCO clocks in one reference clock. If the value for CLK_FLL_CONFIG2.FLL_REF_DIV is set to ‘1’ then CLK_FLL_CONFIG.FLL_MULT must be set to either ‘13’ or ‘12’. This will result in a CCO output of either 96 MHz or 104 MHz, and an error of 4 percent from the desired 100 MHz. Therefore, the best way to improve this is to increase CLK_FLL_CONFIG2.FLL_REF_DIV. However, the larger CLK_FLL_CONFIG2.FLL_REF_DIV is, the longer each measurement cycle takes, thus increasing the lock time. In this example, CLK_FLL_CONFIG2.FLL_REF_DIV was set to 146. This means each measurement cycle takes 146 × (1/8 MHz) = 18.25 µs, whereas when CLK_FLL_CONFIG2.FLL_REF_DIV is set to ‘1’, each measurement cycle takes 1 × (1/ 8 MHz) 0.125 µs.

Another issue with lower CLK_FLL_CONFIG2.FLL_REF_DIV values is that the minimum CLK_FLL_CONFIG2.LOCK_TOL is 1, so the output of the CCO can have an error of ± 1. In the example where CLK_FLL_CONFIG2.FLL_REF_DIV = 1 and CLK_FLL_CONFIG.FLL_MULT = 13, the MULT value can really be 12, 13, or 14 and still be locked.

This means the output of the FLL may vary between 96 and 112 MHz, which may not be desirable.

A choice must be made between faster lock times and more accurate FLL outputs. The biggest change is the value of REF_DIV. The CLK_FLL_STATUS register checks the status of FLL.

18.4 Clock trees

XMC7000 family clocks are distributed throughout the device, as shown in Figure 73. The clock trees are described in this section:

- Path clocks (CLK_PATH)
- High-frequency root clocks (CLK_HF)
- Low-frequency clock (CLK_LF)
- Timer clock (CLK_TIMER)

18.4.1 Path clocks

The XMC7000 family device has several clock paths: CLK_PATH0 contains the FLL, CLK_PATH1 to CLK_PATH<P+1> contains the PLLs, and CLK_PATH<P+D> is a direct connection to the high-frequency root clocks. Note that the FLL and PLL can be bypassed if they are not needed. These paths are the input sources for the high-frequency clock roots.

Each clock path has a mux to determine the source clock for that path. This configuration is done in the CLK_PATH_SELECTx register. Table 111 shows the clock path source selections.

Table 111 Clock path source selections

Name	Description
PATH_MUX	Selects the source for CLK_PATH 0: IMO 1: EXT_CLK 2: ECO 3: Reserved. Do not use. 4: DSI_MUX 5: Reserved 6-7: Reserved. Do not use.

The DSI mux is configured through the CLK_DSI_SELECTx register. Table 112 shows the DSI mux source selections.

Table 112 DSI mux source selections

Name	Description
DSI_MUX	Selects the source for DSI_MUX 0-15: Reserved. Do not use. 16: ILO0 17: WCO 18: Reserved. Do not use. 19: Reserved. Do not use. 20: ILO1 21-31: Reserved. Do not use.

18.4.2 High-frequency root clocks

XMC7000 family has several high-frequency root clocks (CLK_HF). Each CLK_HF has a particular destination on the device; refer to the datasheet for details.

Each high-frequency root clock has a mux to determine its source. This configuration is done in the CLK_ROOT_SELECTx register. The number of CLK_PATH depends on the device. Refer to the datasheet for details.

Each CLK_HF has a predivider, which is set in the CLK_ROOT_SELECTx register.

CLK_HF0 is always enabled because it is the source of the CPU clock. Other CLK_HF can be enabled or disabled using CLK_ROOT_SELECTx.ENABLE.

Table 113 CLK_HF divider selection

Name	Description
ROOT_DVI	Selects predivider value for the clock root 0: No Divider 1: Divide clock by 2 2: Divide clock by 4 3: Divide clock by 8

18.4.3 Low-frequency root clocks

The low-frequency clock (CLK_LF) in XMC7000 family has four input options: ILO0, WCO, ILO1, and ECO_Prescaler. CLK_LF is the source for the multi-counter watchdog timers (MCWDT), and can also be a source for the RTC. The source of CLK_LF is set in CLK_SELECT.LFCLK_SEL.

Table 114 LFCLK input selection bits LFCLK_SEL

Name	Description
LFCLK_SEL	LFCLK input clock selection 0: ILO0 1: WCO 2: Reserved. Do not use. 3: Reserved. Do not use. 4: ILO1 5: ECO Prescaler 6: Reserved 7: Reserved. Do not use.

18.4.4 Timer clock

The timer clock (CLK_TIMER) can be used as a clock source for the CPU SYSTICK timer. The source for CLK_TIMER can only be the IMO.

The CPU SYSTICK timer operation is configured by the CPUSS_SYSTICK_CTL register.

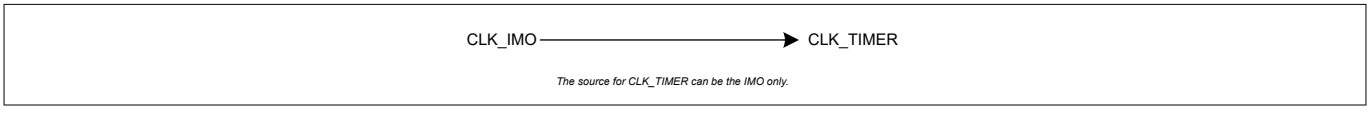


Figure 78 Connection to CLK_TIMER

Table 115 Timer clock selection

Name	Description
TIMER_SEL	Timer clock selection 0: CLK_IMO 1: CLK_HF0

18.4.5 Clock output function

The EXT_CLK terminal is bi-directional. The EXT_CLK terminal can input clock to XMC7000; it is also possible to output the XMC7000 internal clock. To use EXT_CLK as an input, configure the HSIOM to select EXT_CLK on a supported pin, and set that GPIO to the High Impedance digital drive mode. To use EXT_CLK as an output, configure the HSIOM to select EXT_CLK on a supported pin, and set that GPIO to a mode setting capable of driving the selected clock frequency.

The clock source available on EXT_CLK is CLK_HF3, which can choose any internal clock source including ECO. Note that CLK_HF3 is a shared resource; changing the predivider setting impacts all the connections.

18.5 CLK_HF distribution

XMC7000 has several CLK_HFs, which connect CLK_FAST_0, CLK_FAST_1, CLK_MEM, CLK_SLOW, CLK_PERI, CLK_GR and PCLK. CLK_FAST_0, CLK_FAST_1, CLK_MEM, CLK_SLOW, CLK_PERI and CLK_GR are source clocks of the CPU subsystem, SRSS, and some peripheral functions. Also, a part of CLK_HF connects to CSV. Refer to the datasheet for the connection relationship between CLK_HF and the source clocks.

18.5.1 CLK_MEM

CLK_MEM clocks the CPUSS fast infrastructure. This clock is a divided version of CLK_HF. The divider for this clock is set in the CPUSS_MEM_CLOCK_CTL register.

18.5.2 CLK_SLOW

CLK_SLOW is the source clock for the Cortex®-M0+. This clock is a divided version of CLK_MEM. The divider for this clock is set in CPUSS_SLOW_CLOCK_CTL.INT_DIV.

18.5.3 CLK_FAST_0 and CLK_FAST_1

CLK_FAST_x clocks the CM7. This clock is a divided version of CLK_HF1; the divider for this clock is set in the CPUSS_FAST_0_CLOCK_CTL and CPUSS_FAST_1_CLOCK_CTL registers.

18.5.4 CLK_PERI

CLK_PERI is the source clock for all programmable peripheral clock dividers. It is a divided version of CLK_HF. The divider for this clock is set in CPUSS_PERI_CLOCK_CTL.INT_DIV.

18.5.5 PCLK

PCLK is the source of peripheral functions: CAN FD, TCPWM, SCB, and SAR ADC. For details, Peripheral clock dividers.

PCLK are of different kinds; it connects to various peripheral functions. Refer to the peripheral clocks section in the datasheet for more information.

18.5.6 CLK_GR

CLK_GR is a clock input to peripheral functions. It is grouped by the clock gater. Each GR_CLK is divided by the source clock and generated. The divider for the clock is configured in PERI_GRx_CLOCK_CTL.INT_DIV. CLK_GR can be enabled or disabled. The configuration is configured using the PERI_GRx_SL_CTL register.

18.5.7 CLK_TRC_DBG

CLK_TRC_DBG is a clock used for debugging. This clock is generated by dividing a clock source. See the device datasheet for the clock source. Clock division uses the CPUSS_TRC_DBG_CLOCK_CTL register.

18.6 Peripheral clock dividers

XMC7000 family peripherals such as SCBs and TCPWMs require a clock. These peripherals can be clocked only by a peripheral clock divider.

The XMC7000 family has several peripheral clocks (PCLK) and peripheral clock dividers. Peripheral clock dividers can be 8-bit, 16-bit, 16.5-bit (16 integer bits and five fractional bits) and 24.5-bit (24 integer bits and five fractional bits). The output of any of these dividers can be routed to any peripheral. The divider also outputs a signal to divide (enable) the clock signal.

Refer to the datasheet for the number of dividers and assignment of peripheral clocks. Figure 79 shows the peripheral clock divider diagram.

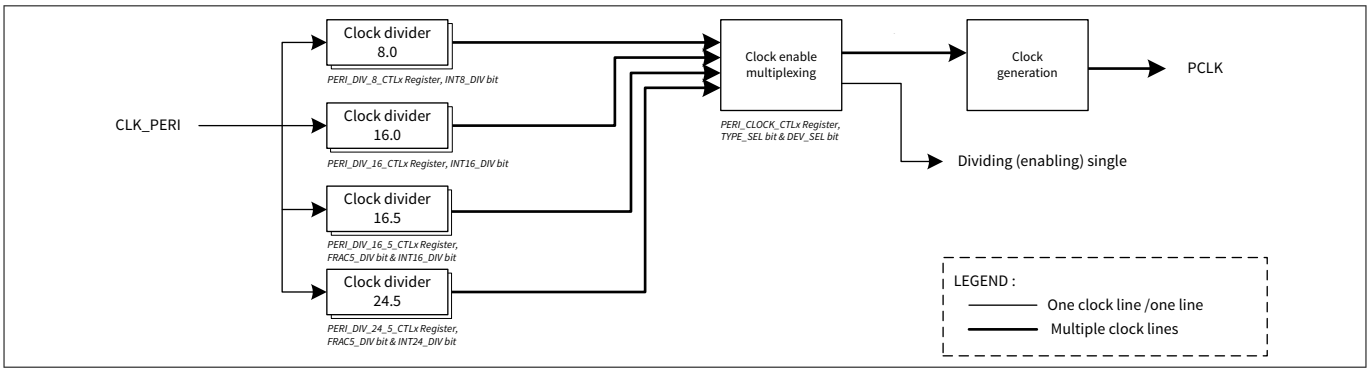


Figure 79 Peripheral clock divider

18.6.1 Fractional clock dividers

Fractional clock dividers allow the clock divisor to include a fraction of 0..31/32. For example, a 16.5-bit divider with an integer divide value of 3 generates a 16-MHz clock from a 48-MHz CLK_PERI. A 24.5-bit divider with an integer divide value of 4 generates a 12-MHz clock from a 48-MHz CLK_PERI. A 24.5-bit divider with an integer divide value of 3 and a fractional divider of 16 generates a 48 / (3 + 16/32) = 48 / 3.5 = 13.7 MHz clock from a 48-MHz CLK_PERI. Not all 13.7-MHz clock periods are equal in size; half of them will be three CLK_PERI cycles and half of them will be two CLK_PERI cycles.

Fractional dividers are useful when a high-precision clock is required (such as a UART/SPI serial interface). Fractional dividers are not used when a low jitter clock is required, because the clock periods have a jitter of one CLK_PERI cycle.

18.6.2 Peripheral clock divider configuration

The peripheral clock dividers are configured using registers from the Peripheral block; specifically PERI_PCLK_GRx_CLOCK_CTLy, PERI_PCLK_GRx_DIV_CMD, PERI_PCLK_GRx_DIV_8_CTLy, PERI_PCLK_GRx_DIV_16_CTLy, PERI_PCLK_GRx_DIV_16_5_CTLy, and PERI_PCLK_GRx_DIV_24_5_CTLy registers.

First the clock divider needs to be configured. This is done via the PERI_PCLK_GRx_DIV_8_CTLy, PERI_PCLK_GRx_DIV_16_CTLy PERI_PCLK_GRx_DIV_16_5_CTLy, and PERI_PCLK_GRx_DIV_24_5_CTLy registers. The number of each divider can be found in the Peripheral Clock Dividers section of the datasheet. Dividers not listed in “Divider Type” are not implemented. The divider selection is determined by the group number and the instance number. For example, use the PERI_PCLK_GR1_DIV_16_CTL3 register to configure the third 16-bit divider with Group 1.

After the divider is configured, use the PERI_PCLK_GRx_DIV_CMD register to enable the divider. This is done by setting the PERI_PCLK_GRx_DIV_CMD.DIV_SEL to the divider number you want to enable, and setting the PERI_PCLK_GRx_DIV_CMD.TYPE_SEL to the divider type. For example, if you wanted to enable the 0th 24.5-bit divider, write '0' to PERI_PCLK_GRx_DIV_CMD.DIV_SEL and '3' to PERI_PCLK_GRx_DIV_CMD.TYPE_SEL. If you wanted to enable the tenth 16-bit divider, write '10' to PERI_PCLK_GRx_DIV_CMD.DIV_SEL and '1' to PERI_PCLK_GRx_DIV_CMD.TYPE_SEL. See XMC7100 registers technical reference manual and XMC7200 registers technical reference manual for more details.

To connect a peripheral to a specific divider, the PERI_PCLK_GRx_CLOCK_CTLy register is used. There is a PERI_PCLK_GRx_CLOCK_CTLy register corresponding to each PCLK mentioned in the Peripheral Clock section of the datasheet. For example, to select the twelfth 16-bit divider for PCLK with group 1 and output 29, configure DIV_SEL to ‘12’ and TYPE_SEL to ‘1’ in the PERI_PCLK_GR1_CLOCK_CTL29 register. Also, the ‘Output’ order matches the order of y of the PERI_PCLK_GRx_CLOCK_CTLy register.

18.6.2.1 Phase aligning dividers

For specific use cases, it is required to generate clocks that are phase aligned. For example, consider the generation of two gated clocks at 24 and 12 MHz, both of which are derived from a 48-MHz CLK_PERI. If phase alignment is not considered, the generated gated clocks can appear as follows.

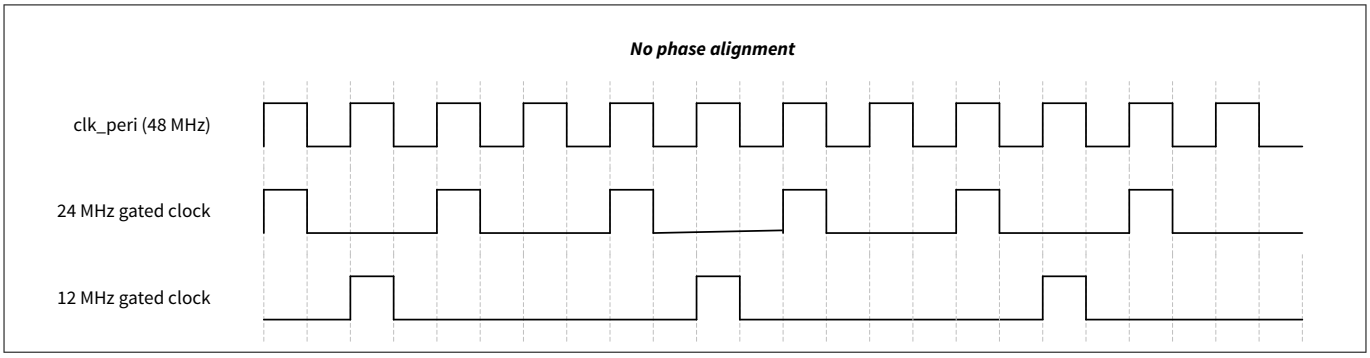


Figure 80 Non phase-aligned clock dividers

These clock signals may or may not be acceptable, depending on the logic functionality implemented on these two clocks. If the two clock domains communicate with each other, and the slower clock domain (12 MHz) assumes that each high/‘1’ pulse on its clock coincides with a high/‘1’ phase pulse in the higher clock domain (24 MHz), the phase misalignment is not acceptable. To address this, it is possible to have PCLK dividers produce clock signals that are phase-aligned with any of the other (enabled) clock dividers. Therefore, if (enabled) divider x is used to generate the 24-MHz clock, divider y can be phase-aligned to divider x and used to generate the 12 MHz clock. The generated gated clocks can appear as follows.

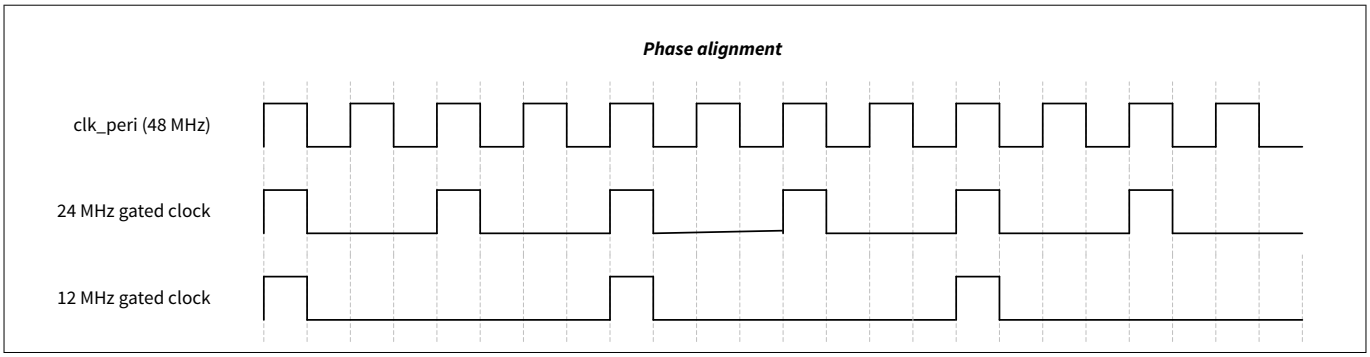


Figure 81 Phase-aligned clock dividers

Phase alignment also works for fractional divider values. If (enabled) divider x is used to generate the 38.4 MHz clock (divide by 1 8/32), divider y can be phase-aligned to divider x and used to generate the 19.2-MHz clock (divide by 2 16/32). The generated gated clocks can appear as follows.

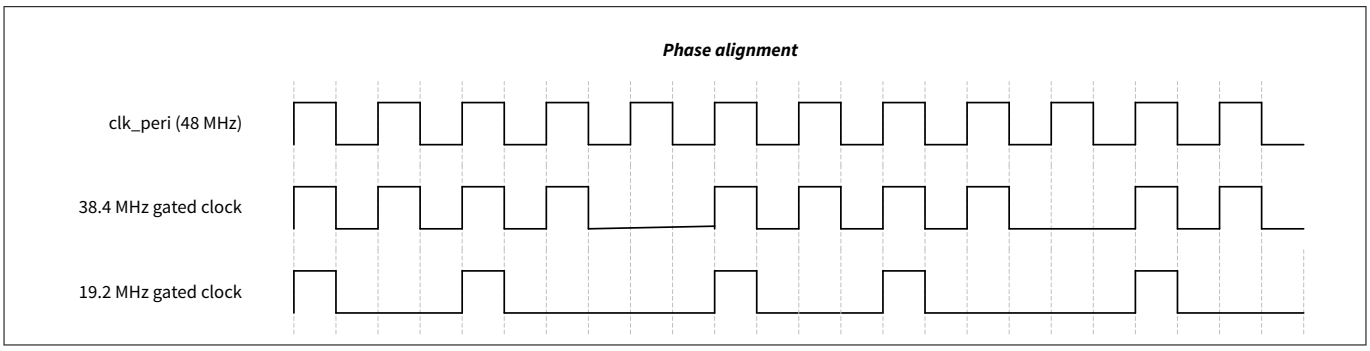


Figure 82 Phase-aligned fractional dividers

Divider phase alignment requires that the divider to which it is phase aligned is already enabled. This requires the dividers to be enabled in a specific order. This order can be represented by a divider dependency graph.

Phase alignment is implemented by controlling the start moment of the divider counters in hardware. When a divider is enabled, the divider counters are set to '0'. The divider counters will only start incrementing from '0' to the programmed integer and fractional divider values when the divider to which it is phase aligned has an integer counter value of '0'.

Note that the divider and clock multiplexer control register fields are all retained during deep sleep power mode. However, the divider counters that are used to implement the integer and fractional clock dividers are not. These counters are set to '0' during deep sleep power mode. Therefore, when transitioning from deep sleep to active power mode, all dividers (and clock signals) are enabled and phase-aligned by design.

Phase alignment is accomplished by setting the PERI_PCLK_GRx_DIV_CMD.PA_DIV_SEL and PERI_PCLK_GRx_DIV_CMD.PA_DIV_TYPE before enabling the clock. For example, to align the fourth 8-bit divider to the third 16-bit divider, set PERI_PCLK_GRx_DIV_CMD.DIV_SEL to '4', PERI_PCLK_GRx_DIV_CMD.TYPE_SEL to '0', PERI_PCLK_GRx_DIV_CMD.PA_DIV_SEL to '3', and PERI_PCLK_GRx_DIV_CMD.PA_TYPE_SEL to '1'.

18.7 Clock calibration counters

A feature of the clocking system in XMC7000 family is built-in hardware calibration counters. These counters can be used to compare the frequency of two clock sources against one another. The primary use case is to take a higher accuracy clock such as the ECO and use it to measure a lower accuracy clock such as the ILOx. The result of this measurement can then be used to trim the ILOx.

There are two counters: Calibration counter 1 is clocked off of calibration clock 1 (generally the high-accuracy clock) and it counts down; calibration counter 2 is clocked off of calibration clock 2 and it counts up. When

calibration counter 1 reaches 0, calibration counter 2 stops counting up and its value can be read. From that value the frequency of calibration clock 2 can be determined with the following equation.

For example, if calibration clock 1 = 8 MHz, counter 1 = 1000, and counter 2 = 5

Calibration clock 2 Frequency = (5/1000) × 8 MHz = 40 kHz.

Calibration clock 1 and calibration clock 2 are selected with the CLK_OUTPUT_FAST register. All clock sources are available as a source for these two clocks. CLK_OUTPUT_SLOW is also used to select the clock source.

Calibration counter 1 is programmed in CLK_CAL_CNT1. Calibration counter 2 can be read in CLK_CAL_CNT2.

When calibration counter 1 reaches 0, CLK_CAL_CNT1.CAL_COUNTER_DONE is set.

$$\text{CalibrationClock2frequency} = \frac{\text{Counter2FinalValue}}{\text{Counter1InitialValue}} \times \text{CalibrationClock1Frequency} \quad (6)$$

18.8 Clock supervision (CSV)

18.8.1 Overview

This section provides an overview of the clock supervision features.

- The CSV circuit checks whether the frequency of the monitored clock is within the allowed frequency window. If the monitored clock stops, or fails to start, it is detected as a low frequency.
- All CLK_HFs have the CSV. All CLK_HF CSVs use the same reference clock (CLK_REF_HF). The reference clock is a selection of one of the active clock sources in the CSV_REF_SEL register. Typically, the IMO is selected (default).
- Note that ILO0 is supervised both by CLK_LF and CLK_REF_HF, CLK_LF is needed for deep sleep supervision and CLK_REF_HF is needed for accuracy (while active).
- CSV_REF monitors CLK_REF_HF with CLK_ILO0.
- All CSV_HFs and CSV_REF are in the active domain.
- There are two CSVs (CSV_LF and CSV_ILO) in the deep sleep domain. CSV_LF is used to monitor the selected CLK_LF clock with ILO0. CSV_ILO is used to monitor ILO0 with CLK_LF. ILO1 is provided to enable clock supervision of ILO0 during deep sleep when the WCO is not being used.

Figure 83 gives an overview of the location of CSVs for the XMC7000.

Refer to the datasheet for details about the relationship between the monitored clock and reference clock for each CSV component.

18.8.2 CSV operation

The basic operation principle of the CSV circuit is as follows:

***Note**: Period is the monitored clock count. Target is the reference clock count. Their time periods are the same in an ideal situation.*

- The monitored clock generates a monitor event (period), and reference clock generates a lower and upper limit.
- The Monitor event is compared against a lower or upper limit
- An error is reported if Monitor event ≤ lower limit, or monitor event > upper limit

Figure 84 shows an example of the CSV operation signal.

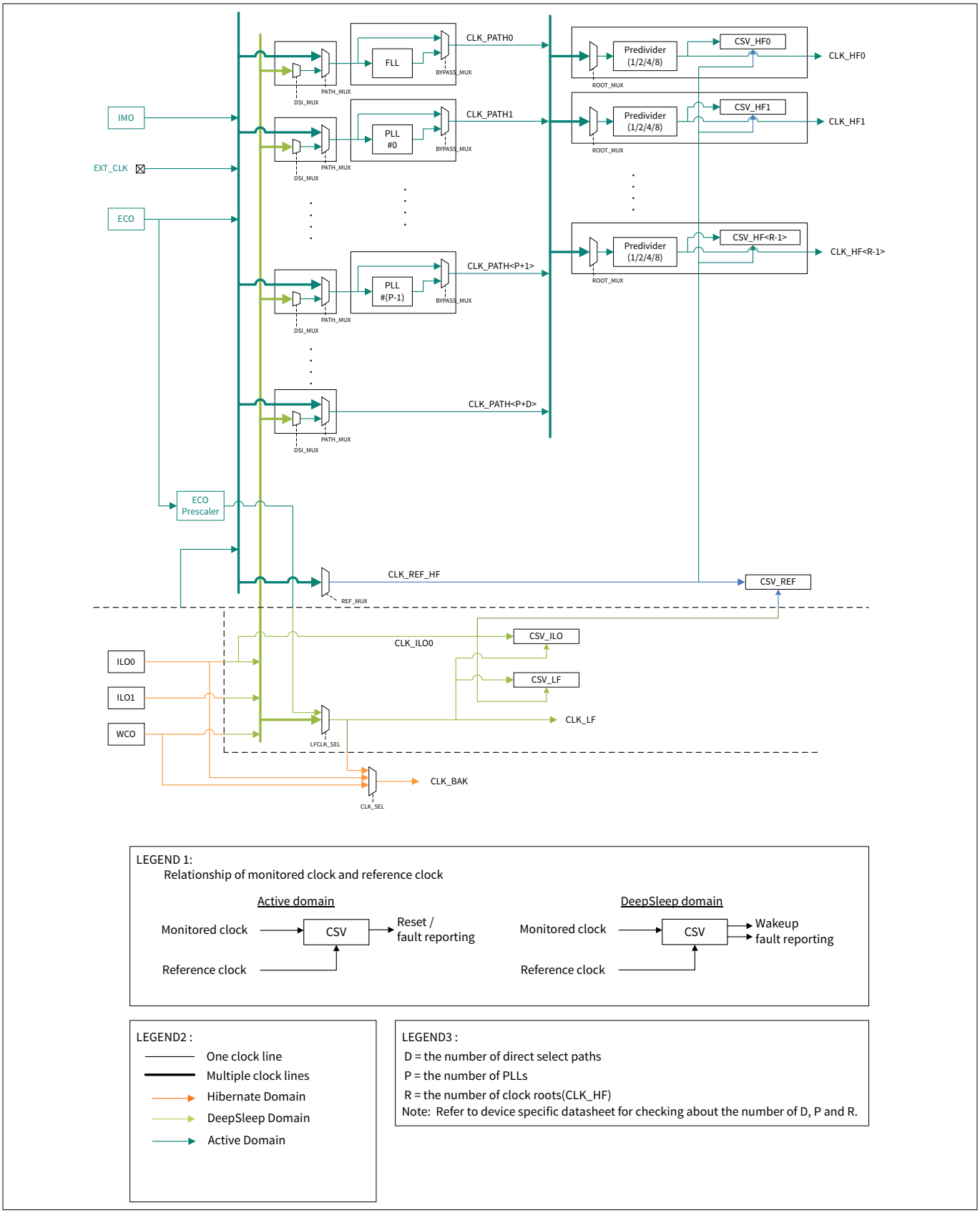
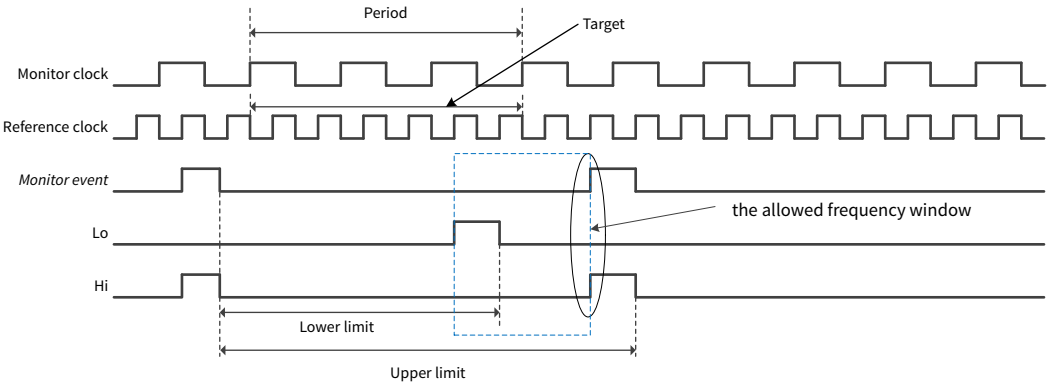
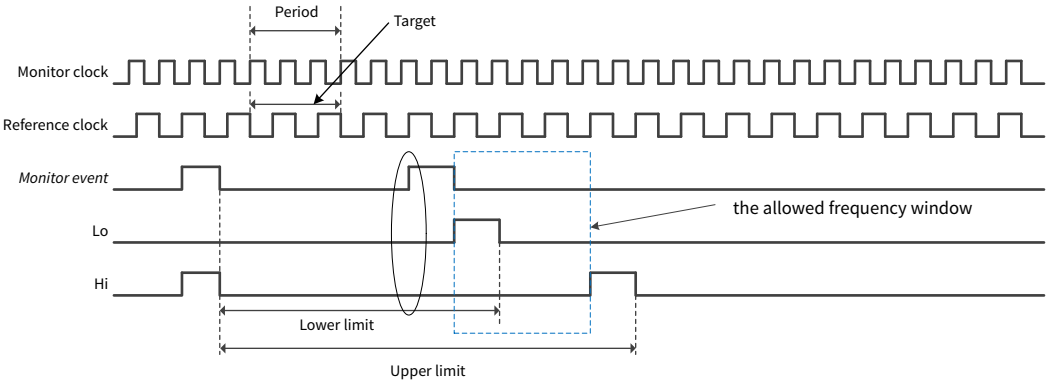


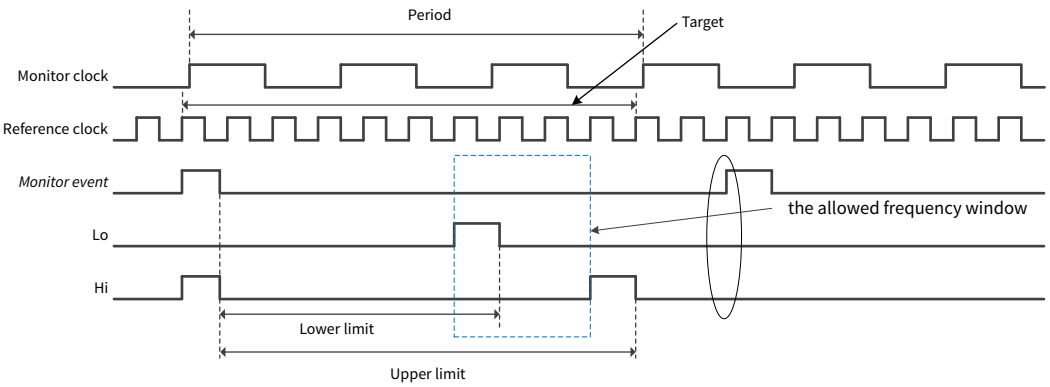
Figure 83 CSV diagram



(1) The error is not reported: Monitor event > Lower limit, and Monitor event <= Upper limit



(2) The error is reported: Monitor event <= Lower limit



(3) The error is reported: Monitor event > Upper limit

Figure 84 An example of CSV operation signal

Key points for CSV operation:

- Check the following parameters:
  - Reference clock frequency and tolerance (integer%)
  - Monitor clock frequency and the required tolerance (integer%)
    The required monitor clock tolerance should be equal to or larger than the reference clock tolerance.
- Determine “Target” with the following formula:
- Minimum Target = 200 / Reference clock tolerance

- For example, for a tolerance of one percent, the target must be at least 200. Increasing the target increases CSV accuracy and latency.
- Determine “Period” with the following formula:
- Period = Target / (Reference clock frequency / Monitor clock frequency)
- Determine Lower_limit and Upper_limit with the following formula:
  - Lower_limit = Period * ((Reference clock frequency * (1 – tolerance/100)) / (Monitor clock frequency * (1 + tolerance/100)))
  - Upper_limit = Period * ((Reference clock frequency * (1 + tolerance/100)) / (Monitor clock frequency * (1 – tolerance/100)))
- If the two clocks are asynchronous (typical) then there will be a one-cycle variation of Monitor Event periods.
- The frequency window needs to account for the maximum clock tolerance on both clocks.
- Lower_limit must be at least one less than Upper_limit.
- All CSVs are initially off and require configuration before enabling.
- The active domain CSVs are automatically stopped during deep sleep.
  - After wakeup they will automatically restart
  - Each CSV has a software programmable startup time. In the case of WCO, the CSV startup time is unused (STARTUP = 0). CSV should be enabled after WCO is started (BACKUP_STATUS.WCO_OK = 1).
- All active domain CSVs can either generate a reset or fault report.
  - The CLK_HF0 CSV must use reset because the fault structure runs on CLK_HF0.
  - All other CSVs should use a fault report to allow software to shut down.
  - A fault report will result in an interrupt.
- The deep sleep domain CSVs operate during active and deep sleep.
- deep sleep domain CSVs can only report faults (no reset option).
  - A CSV error detection will wake up the system (if needed), which enables fault reporting
  - The fault report will result in an interrupt (no direct interrupt from CSV)
- In some products, the hibernate domain has CSV_BAK on CLK_BAK.
  - CSV_BAK monitors CLK_BAK with ILO0. CSV_BAK can wake the device from HIBERNATE mode.
  - In Active/sleep/deep sleep it can generate a fault. If the device is in deep sleep, the fault causes a wakeup.
  - CSV_BAK is not implemented in some devices. Refer to the device-specific datasheet.
- All CSVs are enabled independent of the monitored clock; therefore:
  - Software should disable the CSV before stopping or reconfiguring the monitored clock (to avoid a false error detection)
  - The CSV needs to be reconfigured accordingly and restarted after the monitored clock is re-started
- The CSV_REF_SEL register elects a source to be used as the reference clock for CSV in the active domain. The registers to configure the CSV function are as follows. These registers can enable CSV, and can configure an action when CSV is activated.
  - CSV_HF_CSVx_REF_CTL
  - CSV_REF_CSVx_REF_CTL
  - CSV_LF_CSVx_REF_CTL
  - CSV_ILO_CSVx_REF_CTL
  - CSV_BAK_CSVx_REF_CTL

The following registers can configure upper limit and lower limit. Set the Lower_limit and Upper_limit as -1.

- CSV_HF_CSVx_REF_LIMIT
- CSV_REF_CSVx_REF_LIMIT
- CSV_LF_CSVx_REF_LIMIT
- CSV_ILO_CSVx_REF_LIMIT
- CSV_BAK_CSVx_REF_LIMIT

The following registers can configure PERIOD (time period). Set the Period as -1.

- CSV_HF_CSVx_MON_CTL
- CSV_REF_CSVx_MON_CTL
- CSV_LF_CSVx_MON_CTL
- CSV_ILO_CSVx_MON_CTL
- CSV_BAK_CSVx_REF_MON_CTL

- When CSV violation occurs, SRSS_CSV (0x18) is indicated in the IDX bit of the FAULT_STRUCT_STATUS register. The CSV fault report is captured in FAULT_STRUCT_DATA.

18.9 Registers

Registers related to the clock system are shown in Table 116, Table 117, and Table 118.

Table 116 Clock System registers in the PERI

Register	Name	Description
PERI_GRx_CLOCK_CTL	Divider Control register	This register configures division of CLK_GR.
PERI_GRx_SL_CTL	Slave Control	This register controls whether CLK_GR is enabled or disabled.
PERI_PCLK_GRx_CLOCK_CTLy	Divider Clock Control register	This register configures DIV_SEL and TYPE_SEL of PCLKs.
PERI_PCLK_GRx_DIV_8_CTLy	Divider Control register (for 8.0 divider)	This register controls the 8-bit divider.
PERI_PCLK_GRx_DIV_16_CTLy	Divider Control register (for 16.0 divider)	This register controls the 16-bit divider.
PERI_PCLK_GRx_DIV_16_5_CTLy	Divider Control register (for 16.5 divider)	This register controls the 16.5-bit divider.
PERI_PCLK_GRx_DIV_24_5_CTLy	Divider Control register (for 24.5 divider)	This register controls the 24.5-bit divider.
PERI_PCLK_GRx_DIV_CMD	Divider Command register	This register controls whether each divider is enabled or disabled.

Table 117 Clock System registers in the SRSS

Register	Name	Description
CLK_DSI_SELECTx	Clock DSI Select register	Configures DSI mux in the clock generation path. Each path has its own copy of this register. See Path clocks for DSI signal connectivity list.
CLK_OUTPUT_FAST	Fast Clock Output Select register	Two signals can be selected to enable comparison of clocks. Fast clock output is observable only in (LP)Active/(LP)Sleep and are clamped low during deep sleep.
CLK_OUTPUT_SLOW	Slow Clock Output Select register	Two signals can be selected to enable comparison of clocks. Slow clock output is observable in (LP)Active/(LP)Sleep/deep sleep.
CLK_CAL_CNT1	Clock Calibration Counter 1	This register is a calibration counter that counts down by CLK_FAST selected by the CLK_OUTPUT_FAST register.
CLK_CAL_CNT2	Clock Calibration Counter 2	This register is a calibration counter that is counted up by CLK_FAST selected by the CLK_OUTPUT_FAST register.
CLK_PATH_SELECTx	Clock Path Select register	Selects a source for clock path. The output of this mux can be used as the root of a clock tree. If there is a PLL on the path, this mux output is the PLL reference clock. The related PLL register contains a mux to select whether the clock path uses the PLL output or is bypassed to the PLL reference clock.
CLK_ROOT_SELECTx	Clock Root Select register	Selects a root for a high-frequency clock tree and DSI input. Each clock root has a copy of this register.
CLK_SELECT	Clock Selection register	Clock source selection register.
CLK_ILO0_CONFIG	ILO0 Configuration	Configuration register for ILO0.
CLK_ILO1_CONFIG	ILO1 Configuration	Configuration register for ILO1.
CLK_IMO_CONFIG	IMO Configuration	Internal high-speed R/C Oscillator Configuration register. Note that this oscillator comes up active on power up. The oscillator provides the primary system clock (HFCLK) on power up until firmware configures differently. This oscillator is also used before system start to count out power up delays.
CLK_ECO_CONFIG	ECO Configuration register	Internal High-Speed Oscillator Configuration register for the external-crystal.
CLK_ECO_PRESCALE	ECO Prescaler Configuration register	Fractional prescaler value to bring down the ECO frequency to 32768 Hz if used as CLK_LF. Do not divider settings while ECO prescaler is enabled or enabling.

(table continues...)

Table 117 (continued) Clock System registers in the SRSS

Register	Name	Description
CLK_ECO_STATUS	ECO Status register	Error and status indications.
CLK_FLL_CONFIG	FLL Configuration register	This register contains frequency lock loop (FLL) configuration. FLL circuit settings should not be changed while it is a selected clock (connected to logic). This prevents clock glitches that can crash the logic.
CLK_FLL_CONFIG2	FLL Configuration register2	–
CLK_FLL_CONFIG3	FLL Configuration register3	–
CLK_FLL_CONFIG4	FLL Configuration register4	–
CLK_FLL_STATUS	FLL Status register	This register indicates status for the FLL. This register is synchronized during an AHB read transaction. This causes a number of wait-states to be inserted in the transaction depending on the frequency ration between system and FLL frequency.
CLK_ECO_CONFIG2	ECO Configuration register 2	Internal High-Speed Oscillator Configuration register for the external-crystal.
CLK_PLL_CONFIGx	PLL Configuration register	This register contains PLL configuration. Each PLL has a copy of this register. PLL circuit settings should not be changed while it is a selected clock (connected to logic).
CLK_PLL_STATUSx	PLL Status register	This register indicates status for the PLL. Each PLL has a copy of this register.
CLK_ILO0_TRIM_CTL	ILO0 TRIM register	This register configures trim of ILO0 frequency.
CLK_ILO1_TRIM_CTL	ILO1 TRIM register	This register configures trim of ILO1 frequency.
CSV_REF_SEL	Select CSV Reference Clock register for active domain.	–
Active domain	Selects a source to be used as the reference clock for CSV in the active domain.	–
CSV_HF_CSVx_REF_CTL	Clock Supervision Reference Control register for CLK_HF	This register sets the control of CSV function.
CSV_REF_CSVx_REF_CTL	Clock Supervision Reference Control register for CLK_REF_LF	–
CSV_LF_CSVx_REF_CTL	Clock Supervision Reference Control register for CLK_LF	–
CSV_ILO_CSVx_REF_CTL	Clock Supervision Reference Control register for clk_ilo0	–

(table continues...)

Table 117 (continued) Clock System registers in the SRSS

Register	Name	Description
CSV_HF_CSVx_REF_LIMIT	Clock Supervision Reference Limits register for CLK_HF	This register sets LOWER and UPEER to be used for the CSV function.
CSV_REF_CSVx_REF_LIMIT	Clock Supervision Reference Limits register for CLK_REF_LF	–
CSV_LF_CSVx_REF_LIMIT	Clock Supervision Reference Limits register for CLK_LF	–
CSV_ILO_CSVx_REF_LIMIT	Clock Supervision Reference Limits register for clk_ilo0	–
CSV_HF_CSVx_MON_CTL	Clock Supervision Monitor Control register for CLK_HF	This register sets PERIOD to be used for the CSV function.
CSV_REF_CSVx_MON_CTL	Clock Supervision Monitor Control register for CLK_REF_LF	–
CSV_LF_CSVx_MON_CTL	Clock Supervision Monitor Control register for CLK_LF	–
CSV_ILO_CSVx_MON_CTL	Clock Supervision Monitor Control register for clk_ilo0	–
CLK_PLL400Mx_CONFIG	400-MHz PLL Configuration Register	This register contains PLL configuration for 400-MHz PLL. Each PLL 400 has a copy of this register. Configuration settings should not be changed while it is a selected clock (connected to logic). This prevents clock glitches that can crash the logic.
CLK_PLL400Mx_CONFIG2	400-MHz PLL Configuration Register2	This register configures the fractional divider.
CLK_PLL400Mx_CONFIG3	400-MHz PLL Configuration Register3	This register configures the SSCG.
CLK_PLL400Mx_STATUS	400-MHz PLL Status Register	This register indicates status for a PLL400. Each PLL400 has a copy of this register.
CPUSS_FAST_0_CLOCK_CTL	Fast 0 Clock Control Register	This register configures CLK_FAST_0.
CPUSS_FAST_1_CLOCK_CTL	Fast 1 Clock Control Register	This register configures CLK_FAST_0.
CPUSS_SLOW_CLOCK_CTL	Slow Clock Control Register	This register configures CLK_SLOW.
CPUSS_PERI_CLOCK_CTL	Peripheral Interconnect Clock Control Register	This register configures CLK_PERI.
CPUSS_MEM_CLOCK_CTL	Memory Clock Control Register	This register configures CLK_MEM.
CPUSS_TRC_DBG_CLOCK_CTL	Trace and Debug Clock Control Register	This register configures CLK_TRC_DBG.

Table 118 Clock System Registers in the BACKUP

Register	Name	Description
BACKUP_CTL	BACKUP Control Register	This register controls a function in the backup domain.
CSV_BAK_CSVx_REF_CTL	Clock Supervision Reference Control for CLK_BAK	This register sets the control of CSV function.
CSV_BAK_CSVx_REF_LIMIT	Clock Supervision Reference Limits for CLK_BAK	This register sets LOWER and UPPER to be used for the CSV function.
CSV_BAK_CSVx_MON_CTL	Clock Supervision Monitor Control for CLK_BAK	This register sets PERIOD to be used for the CSV function.

19 Reset system

XMC7000 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. Resets have a broad scope and are generally aligned with power domains and global power modes. The resets described in this chapter cause a reboot that ends in active mode. Some blocks may have local resets that are described in their respective chapters. Reset assertion is asynchronously propogated and reset deassertion is synchronized to each clock domain where it is used. XMC7000 also contains hardware to record which reset occurs.

The fault manager and processors can work together to reset parts of the device. The fault manager converts a fault into a high-priority interrupt (such as NMI) to give the processor an opportunity to return to a safe state, such as halting memory writes and releasing peripherals. The processor can then trigger its own local reset or a system reset. This allows recovery from faults generated by safety circuits (clock supervision, supply supervision, and multi-counter watchdog timer). These circuits can also generate their own direct reset for cases when the fault manager itself is unresponsive or possibly corrupted.

XMC7000 has these reset sources:

- Power-on reset (POR) to hold the device in reset while the power supply is below the level required for initialization of startup circuits.
- Brownout detection reset (BOD) to reset the device if the power supply falls below the device specifications during normal operation.
- Over-voltage detection reset
- Over-current detection reset of the active or deep sleep regulator
- External reset (XRES_L) to reset the device using an external input
- Watchdog resets of the basic watchdog timer (WDT) and the multi-counter watchdog timers (MCWDT) to reset the device if the firmware execution fails to periodically service the watchdog timer
- Internal system reset to reset the device on demand using firmware
- Fault detection resets to reset the device if certain faults occur
- Clock-supervision resets to reset the device when clock-related errors occur
- Hibernate wakeup resets most logic to bring the device out of the hibernate low-power mode

19.1 Reset sources

The following sections provide a description of the reset sources available in XMC7000. Table 119 shows the mapping of reset sources to the corresponding destinations that are affected by a reset event.

Table 119 Reset cause distribution

Reset Class[1]	Reset Cause	Affected areas (resetting / clearing)											
		Other HV reset cause flags cleared in RES_CAUSE Register	Other LV reset cause flags cleared in RES_CAUSE Register	Data Registers in FAULT Structures	Debug Unit	Hibernate Registers	RTC	CM0+	Application Core	SRAM retention	GPIO Pins[2] [3]	REGHC / PMIC Controller and its Pins [4] [5]	Reset Release Time to Active Mode[6]
HV	POR	x	x	x	x	x	x	x	x	No	x	x	Long
HV	XRES_L[3]	-	x	x	x	x	-	x	x	No	x	x	Long_XRES
HV	BOD	-	x	x	x	x	-	x	x	No	x	x	Long
HV	OVD	-	x	x	x	x	-	x	x	No	x	x	Long
HV	OCD	-	x	x	x	x	-	x	x	No	x	x	Long
HV	HIB WAKEUP	-	x	x	x	-	-	x	x	No	[7]	x	Long
HV	WDT	-	x	x	x	x	-	x	x	No	x	x	Long
LV	MCWDT	-	-	-	[8]	-	-	x	x	[9]	x	-	Short
LV	AIRCR.SYSRESETREQ [10]	-	-	-	-	-	-	x	x	[9]	x	-	Short
LV	CDBGRSTREQ	-	-	-	x	-	-	x	x	[9]	x	-	Short
LV	FAULT	-	-	-	[8]	-	-	x	x	[9]	x	-	Short
LV	CSV HF	-	-	-	-	-	-	x	x	[11]	x	-	Short
LV	CSV REF	-	-	-	-	-	-	x	x	No	x	-	Short

1) Reset cause monitors are part of different domains. HV: High-Voltage (VDDD) ; LV: Low-Voltage (VCCD)
2) Pins enter reset state: output high-z, input buffer disabled. JTAG / SWD pins must be taken with care, as they could be modified by the boot ROM configuration. Following pins are considered separately, if they are actively used part of device specific function: WCO and LPECO Pins for RTC and REGHC/PMIC controller pins.
3) During XRES_L assertion JTAG pins are active for boundary scan.
4) Int. core voltage regulator / REGHC / PMIC controller disabled and therefore resulting in longer reset release time to recharge external smoothing capacitor for VCCD. In case of REGHC / PMIC controller the dedicated Pins are reset (high-z / input buffer disabled).
5) See the device-specific document to check if REGHC / PMIC controller is supported.
6) passing power modes triggered by reset cause. Please compare with power mode transition diagram in device power mode chapter.
   - Long: Reset Cause -> OFF -> RESET -> ACTIVE
   - Long_XRES: Reset Cause -> OFF -> XRES -> ACTIVE
   - Short: Reset Cause -> RESET -> ACTIVE
7) To freeze IO Pins, FREEZE command must be done, if needed.
8) Reset occurs if the source triggers during DeepSleep.
9) Yes, if there is an orderly shutdown of the RAM.

10) AIRCR.SYSRESETREQ is software reset.
11) Yes if there is an orderly shutdown of the RAM and the CSV reset is not from CSV_HF0.

***Note**: The SRAM region of the last 6 KB is used by the Infineon firmware during boot operation. Therefore, this region is available to the user; however, data retention across resets is not guaranteed in this area because it can be overwritten by the Infineon boot firmware. See RAM retention configuration for details.*

19.1.1 Power-on reset (POR)

The POR keeps the system in a reset state during power-up. POR holds the device in reset until the supply voltage V_{DDD} reaches a sufficient level to initialize the startup circuits. The POR activates automatically at power-up. All other circuits are disabled until POR releases. See the Power supply and monitoring for more details.

19.1.2 Brownout detection (BOD) reset

The BOD circuits monitor the device digital voltage supply V_{DDD}, device analog voltage supply V_{DDA}, and internally-generated supply voltage V_{CCD} and generate a reset if they fall below their voltage threshold. The device stays in reset until all brownout detectors release. This also occurs during an initial power ramp, but is not recorded as brownout reset. See the Power supply and monitoring for more details.

19.1.3 Overvoltage detection (OVD) reset

The overvoltage circuits monitor the device digital voltage supply V_{DDD}, device analog voltage supply V_{DDA}, and internally-generated supply voltage V_{CCD} and generate a reset if they rise above their voltage threshold. The device stays in reset until all overvoltage detectors release. See the Power supply and monitoring for more details.

19.1.4 Overcurrent reset

The overcurrents of the internally-generated supply voltage V_{CCD} are detected and cause a reset. The observation is done in active and deep sleep power modes. See the Power supply and monitoring for more details. For devices supporting high-current regulator controller (REGHC), the same supervision of V_{CCD} is done to detect an over-current event within the active regulator in case it is enabled.

19.1.5 External reset

The external reset (XRES_L) is a reset triggered by an external signal that causes immediate system reset when asserted. The XRES_L pin is active low – a logic ‘1’ on the pin has no effect and a logic ‘0’ causes reset. The pin is pulled to logic ‘1’ inside the device. XRES_L is available as a dedicated pin. For the detailed pinout, refer to the pinout section of the device datasheet.

The XRES_L pin holds the device in reset as long as the pin input is ‘0’. When the pin is released (changed to logic ‘1’), the device goes through a normal boot sequence. The logical thresholds for XRES_L and other electrical characteristics are listed in the electrical specifications section of the device datasheet. XRES_L is available in all power modes.

19.1.6 Watchdog timer reset (WDT)

The watchdog timer reset causes a reset if the WDT or MCWDTs are not serviced by the firmware within a specified time limit or it is serviced too early in case of window mode.

For details, see the Watchdog timer.

19.1.7 Internal system reset

The internal system reset is a mechanism that allows software running on any of the CPUs or a connected debugger to request a system reset. The Cortex®-M0+ and Cortex®-M7 application Interrupt and Reset Control Registers (CM0P_SCS_AIRCR and CM7_0/CM7_1_SCS_AIRCR, respectively) can request a reset by writing a '1' to the SYSRESETREQ bit of the respective registers.

Note that a value of 0x5FA should be written to the VECTKEY field of the AIRCR Register before setting the SYSRESETREQ bit; otherwise, the processor ignores the write. See the CPU subsystem (CPUSS) for details.

19.1.8 Fault detection reset

The fault reporting structures in XMC7000 can be configured to request a reset for user-configurable faults, such as uncorrectable ECC errors or protection violations.

XMC7000 does not support direct handling of faults during deep sleep mode. If a fault occurs during deep sleep, it wakes the device and then triggers the active mode fault detection reset. The deep sleep mode fault detection reset is not used in XMC7000, so it cannot be set by a fault. Both bits remain set until cleared by firmware or until a POR reset.

Faults generated by clock supervision and MCWDTs can indicate that the fault system may also have failed. These circuits can be configured to directly cause a reset.

19.1.9 Clock-supervision reset

Clock-supervision logic initiates a reset when a monitored clock stops or is outside the configured relationship to a reference clock. Clock supervisors on the high-frequency clocks (HFCLKn) and the CSV reference clock supervisor can generate resets. Clock supervisors for the low-frequency clocks cannot trigger a direct reset, because the fault system and processor can safely convert these faults into resets.

The fault manager and processor clocks are derived from HFCLK0. It is recommended to configure the HFCLK0 CSV to generate a reset or fault-then-reset.

For more information on clocks, see the Clocking system.

19.1.10 Hibernate wakeup reset

Hibernate wakeup reset occurs when a wakeup source triggers an exit from hibernate mode. The device returns to the active power mode and the processors reboot. See the Device power modes for details on hibernate mode and the available wakeup sources.

TOKEN is an 8-bit field in the PWR_HIBERNATE Register that is retained through a hibernate-wakeup sequence. The firmware can use this bit field to differentiate hibernate wakeup from a general reset event, such as XRES_L or POR. Similarly, the PWR_HIB_DATAx Register retains its contents through a hibernate wakeup sequence.

19.1.11 PMIC reset

For devices supporting a PMIC controller, the power good signal of an external PMIC device – connected to the PMIC_STATUS input pin – can issue a PMIC reset.

The reset cause factor is reflected in RESET_PMIC bit [26] in the RES_CAUSE Register. See the Power supply and monitoring for more details.

19.2 Identifying reset sources

When the device comes out of reset, it is often useful to know the cause of the reset. Reset causes are recorded in the RES_CAUSE and RES_CAUSE2 Registers. The bits in these registers are set on the occurrence of the corresponding reset and remain set until cleared by the firmware or a POR reset.

An internal reset that occurred due to hibernate wakeup can be detected by examining the TOKEN field in the PWR_HIBERNATE Register as described previously. Hibernate exit caused by an XRES_L is recorded as an external reset. The reset causes in the RES_CAUSE and RES_CAUSE2 Registers are shown in Table 120.

After identifying and evaluating the reset cause, clear the RESET_CAUSE and RESET_CAUSE2 Register. This procedure is required to capture the next reset cause.

Table 120 Reset cause bits to detect reset source

Register [Bit_Pos]	Bit field	Description
RES_CAUSE [0]	RESET_WDT	A basic WDT reset.
RES_CAUSE [1]	RESET_ACT_FAULT	Fault logging system requested a reset from its active logic.
RES_CAUSE [2]	RESET_DPSLP_FAULT	Fault logging system requested a reset from its deep sleep logic.
RES_CAUSE [3]	RESET_TC_DBGRESET	Test controller or debugger asserted reset. Only resets debug domain.
RES_CAUSE [4]	RESET_SOFT	A CPU requested a system reset through its SYSRESETREQ.
RES_CAUSE [5]	RESET_MCWDT0	MCWDT reset #0.
RES_CAUSE [6]	RESET_MCWDT1	MCWDT reset #1.
RES_CAUSE [7]	RESET_MCWDT2	MCWDT reset #2.
RES_CAUSE [8]	RESET_MCWDT3	MCWDT reset #3.
RES_CAUSE [16]	RESET_XRES	External XRES_L pin is asserted.
RES_CAUSE [17]	RESET_BODVDDD	External V_{DDD} supply crossed the brownout limit.
RES_CAUSE [18]	RESET_BODVDDA	External V_{DDA} supply crossed the brownout limit.
RES_CAUSE [19]	RESET_BODVCCD	External V_{CCD} supply crossed the brownout limit.
RES_CAUSE [20]	RESET_OVDVDDD	Overvoltage detection on the external V_{DDD} supply.
RES_CAUSE [21]	RESET_OVDVDDA	Overvoltage detection on the external V_{DDA} supply.
RES_CAUSE [22]	RESET_OVDVCCD	Overvoltage detection on the internal core V_{CCD} supply.
RES_CAUSE [23]	RESET_OCD_ACT_LINREG	Overcurrent detection on the internal V_{CCD} supply when supplied by the active power mode linear regulator.
RES_CAUSE [24]	RESET_OCD_DPSLP_LINREG	Overcurrent detection on the internal V_{CCD} supply when supplied by the deep sleep power mode linear regulator.
RES_CAUSE [25]	RESET_OCD_REGHC	Overcurrent detection from high-current regulator controller (REGHC)
RES_CAUSE [26]	RESET_PMIC	Detection of PMIC status RESET
RES_CAUSE[30]	RESET_PORVDDD	Indicator that a POR occurred. This is a high-voltage cause bit, and hardware clears the other bits when this bit is set. It does not block further recording of other high-voltage causes.

(table continues...)

Table 120 (continued) Reset cause bits to detect reset source

Register [Bit_Pos]	Bit field	Description
RES_CAUSE2 [15:0]	RESET_CSV_HF	Clock supervision logic requested a reset due to loss or frequency violation of a high-frequency clock. Each bit index K corresponds to a HFCLK<K>.
RES_CAUSE2 [16]	RESET_CSV_REF	Clock supervision logic requested a reset due to loss or frequency violation of the reference clock source that is used to monitor other high-frequency clock sources.

For more information, see the RES_CAUSE and RES_CAUSE2 Registers in the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual.

19.3 Registers

Table 121 Reset system register list

Register	Name	Description
RES_CAUSE	Reset Cause Observation Register	Indicates the cause for the latest reset(s) that occurred in the system.
RES_CAUSE2	Reset Cause Observation Register 2	Indicates the cause for the latest reset(s) that occurred in the system.
PWR_HIBERNATE	Hibernate Mode Register	This register controls entry/exit from hibernate power mode.
PWR_HIB_DATAx	Hibernate Data Register	This register retains its contents through hibernate wakeup reset. 'x' signifies the number of such DATA Registers. Refer to the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual for more information.
CM7_0/ CM7_1_SCS_AIRCR	Application Interrupt and Reset Control Register of Cortex®-M7	Application Interrupt and Reset Control Register specific to Cortex®-M7
CM0P_SCS_AIRCR	Application Interrupt and Reset Control Register of Cortex®-M0+	Application Interrupt and Reset Control Register specific to Cortex®-M0+

Check the device datasheet to see if the feature is supported

20 Watchdog timer

The watchdog timer (WDT) in XMC7000 includes a hardware timer that automatically resets the device in the event of an unexpected firmware execution path. It uses LFCLK (ILO0, ILO1, WCO, or ECO) as the input clock. The WDT, if enabled, must be serviced periodically in firmware to avoid a reset. Otherwise, the timer will elapse and generate a device reset. In the window function mode, the WDT can generate a reset if it is serviced too early or not serviced at all before a timeout is reached. In addition, the WDT can be used as an interrupt source or a wakeup source in low-power modes.

XMC7000 includes one 32-bit free-running basic WDT supporting window mode and up to four multi-counter watchdog timers (MCWDT). Each MCWDT includes three subcounters – two 16-bit timers supporting window mode and one 32-bit free-running timer. All counters with window mode functionality can generate a reset and a WARN interrupt; the MCWDT counters can also generate a FAULT condition. Each MCWDT is independent; it is recommended to assign one MCWDT per processor.

20.1 Features

The XMC7000 watchdog timer supports the following features:

- One 32-bit free-running basic WDT with:
  - ILO0 as the input clock source
  - Programmable early threshold, warning threshold, and timeout threshold
  - Device reset generation if not serviced within a configurable interval
  - Warning threshold generates an interrupt to request servicing
  - Interrupt/wakeup generation in active, sleep, deep sleep, and hibernate power modes
- Up to four MCWDTs, each supporting:
  - LFCLK (ILO0, ILO1, WCO, or ECO) as the input clock source
  - Fault and device reset generation if not serviced within a configurable interval
  - Periodic interrupt/wakeup generation in active, sleep, and deep sleep power modes
  - Three independent counters: two 16-bit counters and one 32-bit counter
  - Warning threshold generates an interrupt to request servicing
- Both watchdog timer types support:
  - Window mode
  - Running and freezing timers during deep sleep mode
  - Debug

20.2 Block diagram

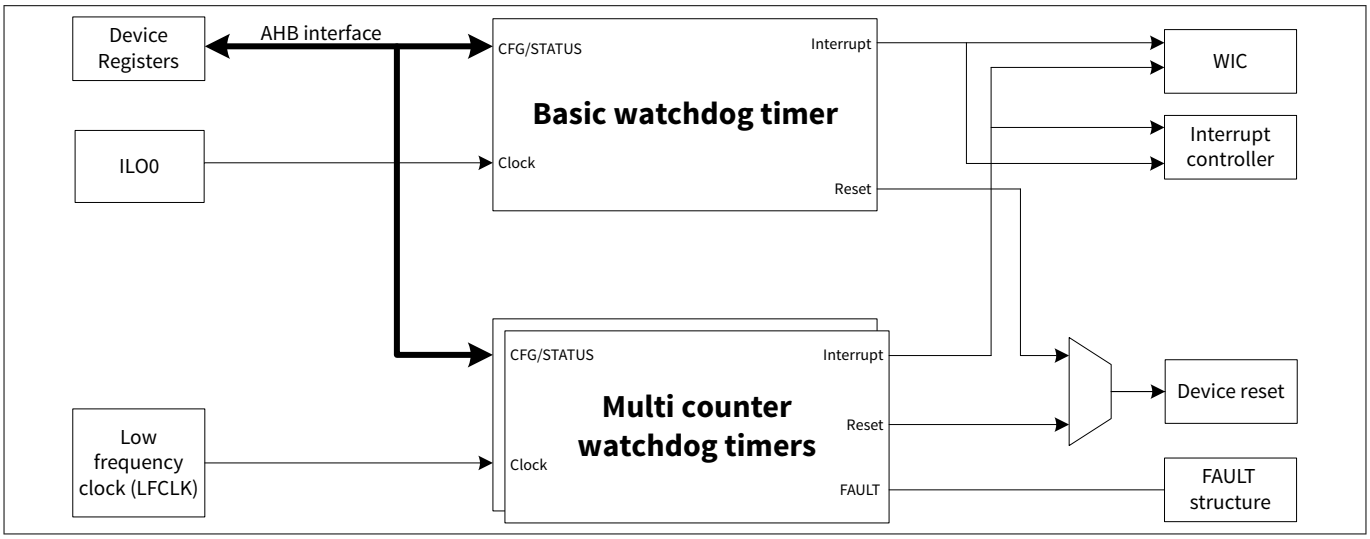


Figure 85 Watchdog timer block diagram

20.3 Basic watchdog timer

20.3.1 Overview

The WDT is a free-running up-counter with programmable limit values, a maximum of 32-bit resolution, and a clock from the ILO0. Servicing the watchdog clears and restarts the counter at zero.

The WDT can be configured to act on different counter limits where a reset is triggered if the watchdog is not serviced before the upper limit. In the window mode, a reset is triggered if the servicing occurs before the lower limit is reached. The warning limit triggers an interrupt to request servicing. Each of these actions can be activated independently. The WDT is enabled and Specific registers are locked by default. An unlocking sequence is required to prevent accidental accesses. The WDT operates in active, sleep, deep sleep, and hibernate modes. Hibernate mode operation is possible because the logic and clock source are powered by the external high-voltage supply (V_{DDD}). After a WDT reset the device returns to active mode.

Figure 86 shows the functional overview of the WDT.

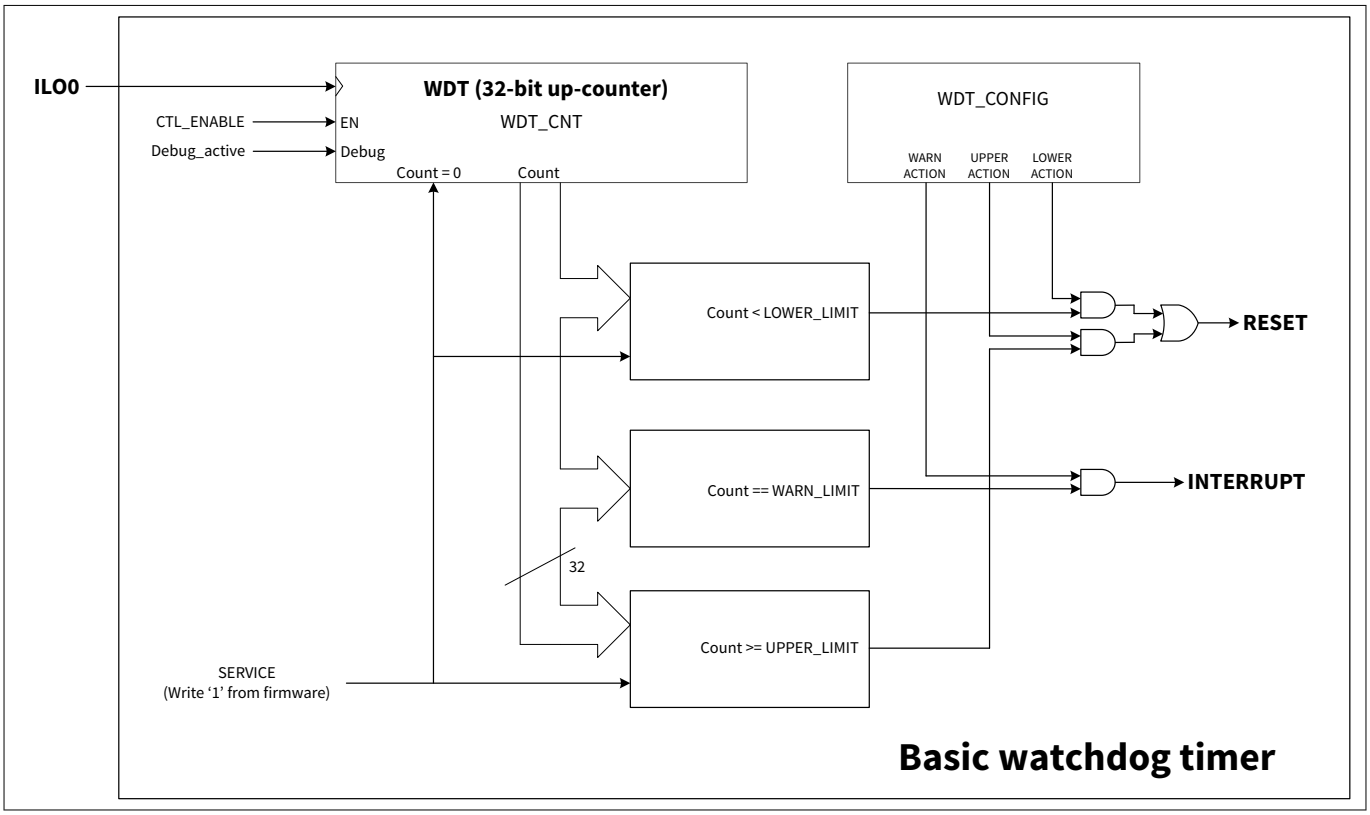


Figure 86 Basic WDT functional diagram

When enabled, the WDT counts up on each rising edge of the ILO0 clock. When the counter value (WDT_CNT register) equals the warning threshold value stored in WDT_WARN_LIMIT [31:0], an interrupt is generated if the WARN_ACTION [8] bit is set to '1' in the WDT_CONFIG register. The warn event will not reset the WDT counter and the WDT continues counting until it reaches the timeout threshold value stored in UPPER_LIMIT [31:0]; it generates a reset if the UPPER_ACTION [4] bit is set to '1' in the WDT_CONFIG register. If no action is taken on the upper threshold, the counter increments to the 32-bit boundary and then wraps around to '0' and counts up. In the window mode, an early threshold stored in LOWER_LIMIT [31:0] can be used if the LOWER_ACTION [0] bit is set to '1' in the WDT_CONFIG register for generating a reset if the counter is serviced too early. The watchdog counter is serviced by the SERVICE [0] bit in the WDT_SERVICE register. If this bit is set to '1' the watchdog counter is set to zero.

The WDT [0] bit in the WDT_INTR register is set whenever the WDT counter matches with the WARN_LIMIT and an interrupt is requested by the CPU. This interrupt must be cleared by writing a '1' to the same bit (WDT bit of WDT_INTR). Clearing the interrupt does not reset the watchdog counter.

The WDT can be enabled or disabled using the ENABLE [31] bit of the WDT_CTL register. The actual status of the counter is indicated by the ENABLED [0] bit of the WDT_CTL register.

The WDT provides a mechanism to lock WDT Configuration registers. The WDT_LOCK bits [1:0] control the lock status of WDT-related registers. These are special bits, which can enable the lock in a single write (WDT_LOCK = 3); to release the lock, two different write accesses are required (WDT_LOCK = 1 to clear WDT_LOCK [0] and WDT_LOCK = 2 to clear WDT_LOCK [1]). When the WDT_LOCK bits are not equal to '0' the write accesses to the CTL, LOWER_LIMIT, WARN_LIMIT, UPPER_LIMIT, CNT, and SERVICE registers are prohibited. Note that this field is two bits to force multiple writes only. It represents only a single write protect signal protecting all those registers at the same time. WDT will lock and enable on any reset. This field is not retained during deep sleep or hibernate mode, so the WDT will be locked after wakeup from these modes.

***Note**: The lock mechanism is an additional safety opportunity, which requires to unlock/lock the SERVICE register when servicing each watchdog counter. Alternatively, the WDT registers can also be protected by the PPU, which allows to keep the WDT registers unlocked.*

When the watchdog counter is disabled and unlocked, the count value can be written for verification and debugging purposes. Software writes are always ignored when the counter is enabled.

Table 122 explains various registers and bit fields used to configure and use the WDT.

Table 122 Basic watchdog timer configuration options

Register [Bit_Pos]	Bit name	Description
WDT_CTL[31]	ENABLE	Enable or disable the watchdog counter 0: Counter is disabled (not clocked) 1: Counter is enabled (counting up)
WDT_CTL[0]	ENABLED	Indicates actual state of watchdog
WDT_LOCK[1:0]	WDT_LOCK	Prohibits writing control and Configuration registers related to this MCWDT when not equal to '0' 0: No effect 1: Clear bit 0 2: Clear bit 1 3: Set both bit 0 and 1 (lock enabled)
WDT_CNT[31:0]	CNT	Current value of WDT counter
WDT_LOWER_LIMIT[31:0]	LOWER_LIMIT	Lower limit for watchdog
WDT_UPPER_LIMIT[31:0]	UPPER_LIMIT	Upper limit for watchdog
WDT_WARN_LIMIT[31:0]	WARN_LIMIT	Warn limit for watchdog
WDT_CONFIG[0]	LOWER_ACTION	Action taken if this watchdog is serviced before LOWER_LIMIT is reached 0: Do nothing 1: Trigger a reset
WDT_CONFIG[4]	UPPER_ACTION	Action taken if this watchdog is not serviced before UPPER_LIMIT is reached 0: Do nothing 1: Trigger a reset
WDT_CONFIG[8]	WARN_ACTION	Action taken when the count value reaches WARN_LIMIT 0: Do nothing 1: Trigger an interrupt
WDT_CONFIG[12]	AUTO_SERVICE	Automatically service when the count value reaches WARN_LIMIT. This allows creation of a periodic interrupt if this counter is not needed as a watchdog.

(table continues...)

Table 122 (continued) Basic watchdog timer configuration options

Register [Bit_Pos]	Bit name	Description
WDT_CONFIG[28]	DEBUG_TRIGGER_EN	Enables the trigger input for the WDT to pause the counter in debug mode. 0: Pauses the counter when a debug probe is connected. 1: Pauses the counter when a debug probe is connected and the trigger input is high.
WDT_CONFIG[29]	DPSLP_PAUSE	Pauses/runs this counter when the system is in deep sleep 0: Counter behaves normally during deep sleep 1: Counter pauses during deep sleep
WDT_CONFIG[30]	HIB_PAUSE	Pauses/runs this counter when the system is in hibernate 0: Counter behaves normally during hibernate 1: Counter pauses during hibernate
WDT_CONFIG[31]	DEBUG_RUN	Pauses/runs this counter while a debugger is connected 0: Counter pauses according to DEBUG_TRIGGER_EN configuration 1: Counter runs normally when debugger connected
WDT_INTR[0]	WDT	WDT interrupt request. This bit is set as configured by WDT action and limits. The WDT interrupt is cleared by writing a '1' to this bit.
WDT_INTR_SET[0]	WDT	WDT Interrupt Set register. Can be used to set interrupts for firmware testing.
WDT_INTR_MASK[0]	WDT	Mask for the WDT interrupt 0: WDT interrupt is masked to CPU 1: WDT interrupt is not masked to CPU
WDT_INTR_MASKED[0]	WDT	Logical AND of corresponding request and mask bits

***Note**: WDT configuration registers are in a separate protection region from the register used to service it. The protection regions are handled by the peripheral protection unit (PPU). Refer to the CPU subsystem (CPUSS) for more information.*

20.3.2 Watchdog reset

A watchdog is typically used to protect the device against firmware/system crashes or faults. When the WDT is used to protect against system crashes, the WDT counter should be cleared by writing a '1' to the SERVICE [0] bit in the SERVICE register from a portion of the code that is not directly associated with the WDT interrupt. Otherwise, even if the main function of the firmware crashes or is in an endless loop, the WDT interrupt vector can still be intact and feed the WDT periodically.

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

- Configure the UPPER_LIMIT such that firmware is able to reset the watchdog at least once during the period, even along the longest firmware delay path
- In window mode, configure the LOWER_LIMIT to serve the watchdog counter not too early, even along the shortest firmware delay path.
- Reset (feed) the watchdog for clearing the counter regularly in the main body of the firmware code by setting the SERVICE [0] bit to '1' in SERVICE register.

It is not recommended to reset the watchdog counter in the WDT interrupt service routine (ISR), if WDT is being used as a reset source to protect the system against crashes. If necessary, use the warning interrupt to set a flag in the ISR. Local processing loops can observe that flag and break out of their loop. This allows the main loop to reach the servicing code (and clear the flag for the next pass through the main loop).

Recommended steps to use WDT as a reset source are as follows:

1. Write UPPER_LIMIT value to define the timeout period for reset generation. Set UPPER_ACTION [4] bit to '1' in the WDT_CONFIG register to enable a reset trigger when the watchdog counter reaches the UPPER_LIMIT.
2. If required, write the WARN_LIMIT to generate an interrupt before reaching the UPPER_LIMIT threshold. Do not use the ISR to feed the WDT; instead, use this interrupt to indicate that there is a firmware delay path, which is already critical. Use a warn level that is close enough to the UPPER_LIMIT but consider also the delay to handle the ISR and return to your main body functions for serving the watchdog counter. Set WARN_ACTION [8] bit to '1' in the WDT_CONFIG register to enable a watchdog warn interrupt when the watchdog counter matches with the WARN_LIMIT.
3. In window mode, define an adequate LOWER_LIMIT, which cannot be violated by the shortest firmware delay path. Set the LOWER_ACTION [0] bit to '1' in the WDT_CONFIG register to enable a reset trigger when the watchdog counter is serviced before the counter reaches the LOWER_LIMIT.
4. Set the WDT [0] bit in the WDT_INTR register to clear any pending WDT interrupt.
5. Set the ENABLE [31] bit in the CLK_ILO0_CONFIG register to enable the ILO0 clock.
6. Enable the WDT by setting the ENABLE [31] bit in the WDT_CTL register.
7. In the firmware, write '1' to the SERVICE [0] bit in the SERVICE register to feed (reset) the watchdog.
8. Lock the WDT configuration by writing '3' to the WDT_LOCK bits.

Figure 87 shows all scenarios of the WDT operation while LOWER_ACTION, WARN_ACTION, and UPPER_ACTION are enabled.

- Counter is serviced between LOWER_LIMIT and WARN_LIMIT: This is the regular behavior of the WDT. No WARN interrupt is issued and no RESET is done
- Counter is serviced between WARN_LIMIT and UPPER_LIMIT: The service is done late, a WARN interrupt is issued but no RESET is done.
- Counter is not serviced at all: WARN interrupt is issued but the SERVICE bit is not set. When the counter reaches the UPPER_LIMIT a reset is executed.
- Counter is serviced before the LOWER_LIMIT is reached: The counter is serviced too early; a reset is executed because the counter is cleared outside of the window.

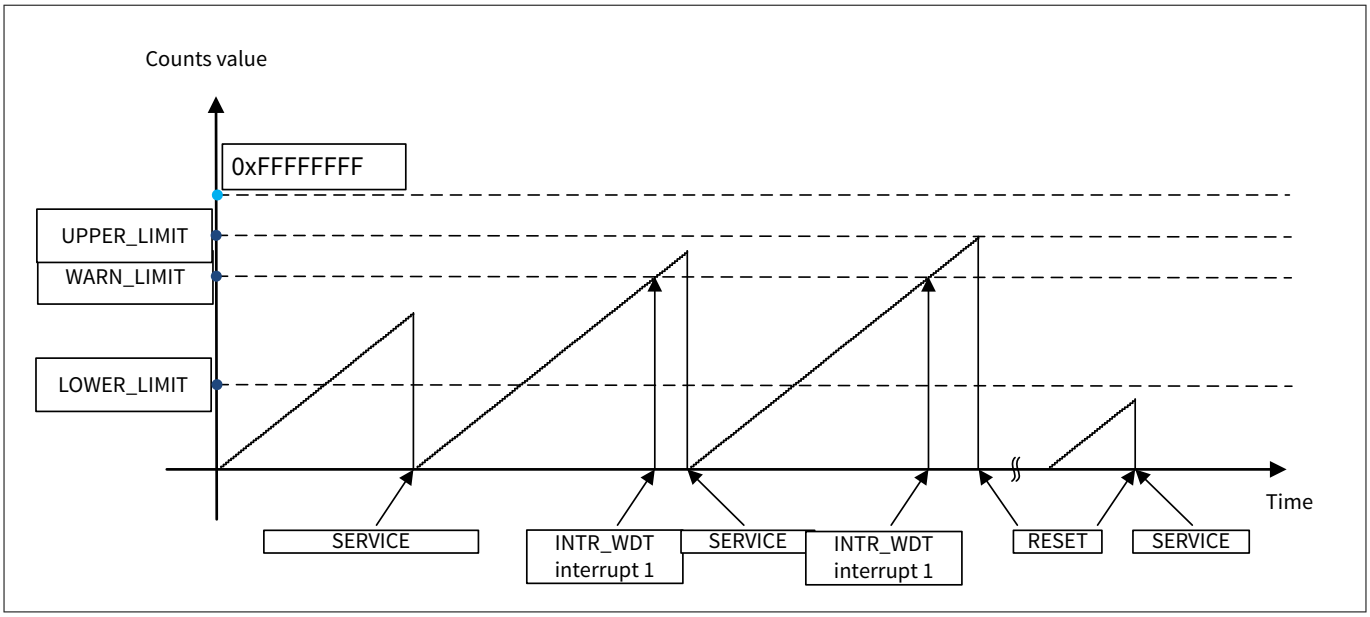


Figure 87 WDT counter operation in window mode

***Note**: This figure illustrates the different scenarios with or without servicing the watchdog counter. It does not consider the WDT configuration, especially after a reset.*

20.3.3 Watchdog interrupt

In addition to generating a device reset, the WDT can be used to generate interrupts. The watchdog counter can send interrupt requests to the CPU in active power modes and to the wakeup interrupt controller (WIC) in sleep and deep sleep power modes. In addition, the watchdog is capable of waking up the device from hibernate power mode. It works as follows:

- Active mode: In this mode, the WDT can send the interrupt to the CPU. The CPU acknowledges the interrupt request and executes the ISR. The interrupt must be cleared after entering the ISR in firmware.
- Sleep or deep sleep mode: In these modes, 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.
- Hibernate mode: In this mode, the entire device except a few peripherals (such as WDT) are powered down. Any interrupt to wake up the device in this mode results in a device reset. Hence, there is no interrupt service routine or mechanism associated with this mode.

For more details on device power modes, see the Device power modes. Because of its free-running nature, it is not recommended to use the WDT for periodic interrupt generation. The MCWDT counters can be used to generate periodic interrupts. If absolutely required, follow these steps to use the WDT as a periodic interrupt generator:

1. Write the WARN_LIMIT to set the interrupt period. If the WDT is not serviced, the counter will continue to count up until the maximum counter level of 0xFFFFFFFF is reached and then the counter starts from zero.
2. Set the WARN_ACTION [8] bit to ‘1’ in the WDT_CONFIG register to enable a watchdog warn interrupt when the watchdog counter matches with the WARN_LIMIT.
3. Set the WDT [0] bit in the WDT_INTR register to clear any pending WDT interrupt.
4. Enable the WDT interrupt to CPU by setting the WDT [0] bit in the WDT_INTR_MASK register.

5. Enable SRSS interrupt to the CPU by configuring the appropriate ISER register, see the Interrupts for details.
6. In the ISR, clear the WDT interrupt; if required, clear the watchdog timer by writing '1' to the SERVICE [0] bit in the SERVICE register. Servicing the WDT allows to generate various interrupt periods, which can be defined by the WARN_LIMIT. Alternatively, set the AUTO_SERVICE[12] bit to '1' in the CONFIG register to automatically service the WDT when the count value reaches WARN_LIMIT.

Waking up from deep sleep mode requires to execute an unlock sequence by writing the value '1' to the WDT_LOCK [1:0] bits in the WDT_LOCK register followed by writing '2' to the same bit field.

Figure 88 shows the behavior of the WDT counter in interrupt mode. LOWER and UPPER actions are disabled. An interrupt is issued each time the counter matches the WARN_LIMIT and continuous to count up to the 32-bit maximum value. The interrupt period is calculated by 2^{32} x ILO0 clock cycles. The WDT does not provide an automatic counter clear function; therefore, the counter must be cleared manually by writing '1' to the SERVICE[0] bit in the WDT_SERVICE register.

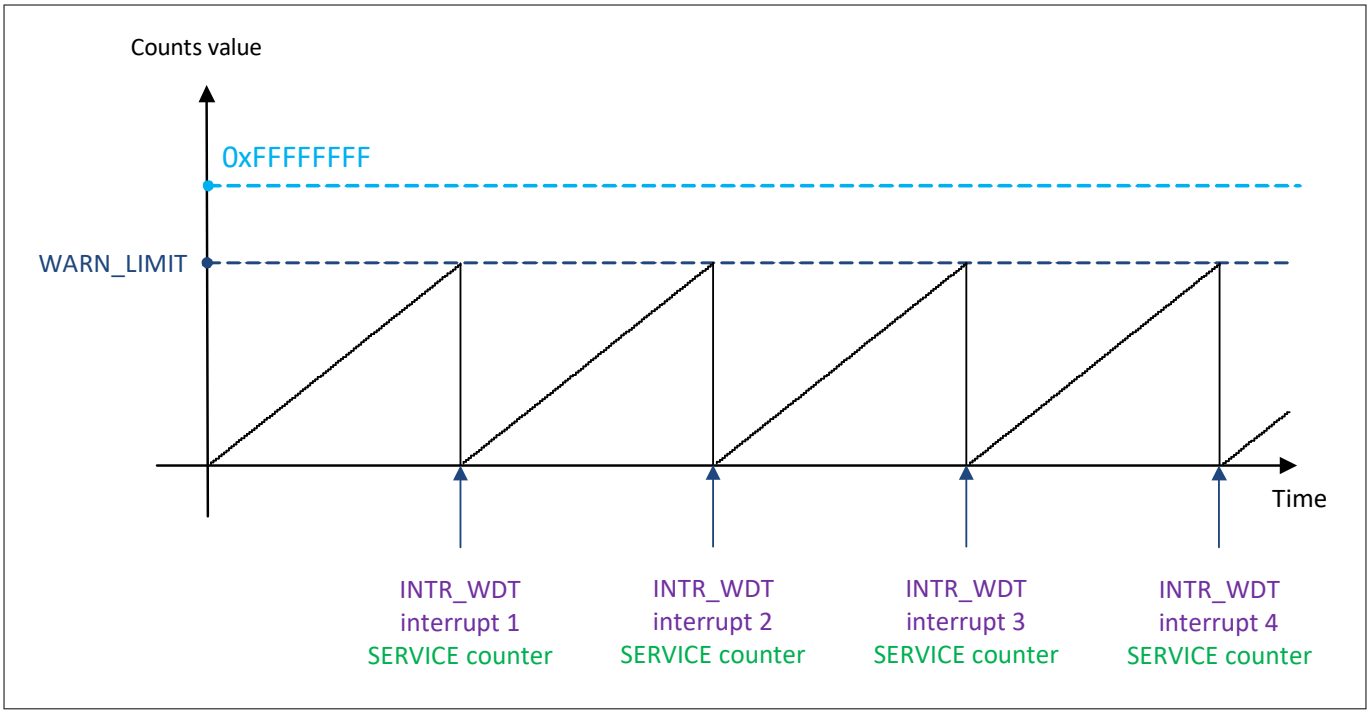


Figure 88 WDT counter operation with WARN interrupt only

20.4 Multi-counter watchdog timer

20.4.1 Overview

Figure 89 shows the functional overview of a single MCWDT block. Depending on the device, XMC7000 includes up to four MCWDT blocks. Each MCWDT block includes two 16-bit counters, subcounter 0 (MCWDTx_CNT0) and subcounter 1 (MCWDTx_CNT1), which include the same window and threshold concept described for the WDT, and one 32-bit counter, subcounter 2 (MCWDTx_CTR2_CNT). Cascading of these counters is not supported. These counters work independently.

The Subcounters 0 and 1 have the ability to generate a FAULT when the MCWDTx_CTRy_LOWER_LIMIT or MCWDTx_CTRy_UPPER_LIMIT is violated. The fault structure can convert this to an interrupt (such as a high-priority NMI) that gives the processor an opportunity to return to a safe state, such as halting memory writes and releasing peripherals. It can then clear the fault and trigger its own local reset. If the fault is not cleared within a fixed number of LFCLK cycles, MCWDT will trigger a system-wide reset as a failsafe.

***Note**: If a single MCWDT triggers additional fault actions while transferring fault data to the fault manager, then the fault manager receives only the first action in the fault data. The additional overlapping fault actions do not cause another fault report and are lost. Faults are transferred correctly, if they occur when the MCWDT is not in the middle of transferring another fault. Faults generated by a different MCWDT are not affected.*

A missed MCWDTx_CTRy_CONFIG.UPPER_ACTION fault can be detected during MCWDT fault processing. For counter values of both subcounter 0 and subcounter 1, check whether the condition CNT ≥ MCWDTx_CTRy_UPPER_LIMIT is valid. There is no known method to detect a missed MCWDTx_CTRy_CONFIG.LOWER_ACTION fault.

The 32-bit counter can only generate interrupt after a programmed bit position toggles.

All the counters are clocked by LFCLK and operate in active, sleep, and deep sleep modes. Hibernate mode is not supported. After a MCWDT reset, the chip is recovered to active mode. Servicing a counter clears and restarts the related counter at zero. The MCWDT is disabled and unlocked by default.

***Note**: Because XMC7000 includes two CPUs (Cortex® -M0+ and Cortex® -M7), it is recommended to associate one MCWDT block to only one CPU during runtime. Although both the MCWDT blocks are available to both CPUs, a single MCWDT is not intended to be used by multiple CPUs simultaneously.*

register protection is handled by putting the subcounter 0 and subcounter 1 configuration registers in a protection region. A separate protection region is used for the registers related to servicing and subcounter 2.

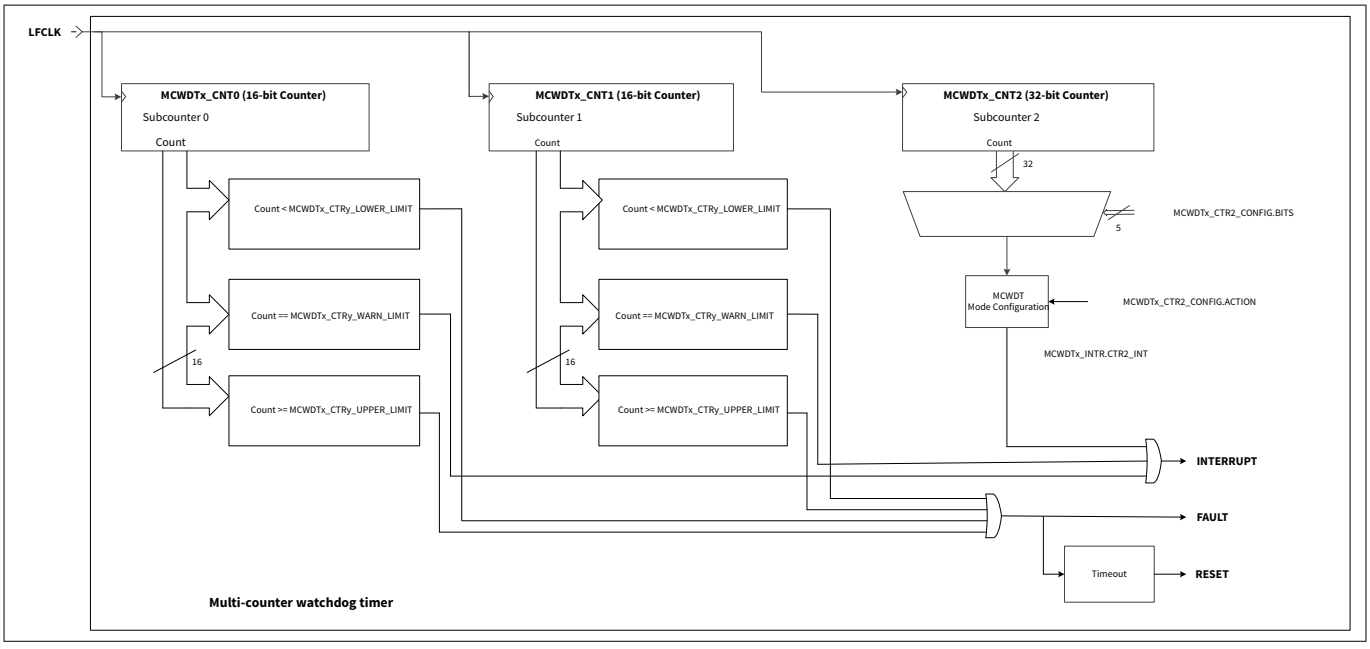


Figure 89 Multi counter watchdog timer functional diagram

20.4.2 How it works

20.4.2.1 Subcounter 0/1 operation

Subcounter 0 (MCWDTx_CNT0) and subcounter 1 (MCWDTx_CNT1) are independent 16-bit up-counters. If enabled, they can count up on each rising edge of the LFCLK clock. ILO0, ILO1, WCO, or ECO can be configured as a clock source. See the Clocking system.

When a counter value (MCWDTx_CTRy_CNT[2)] register) equals the warning threshold value stored in MCWDTx_CTRy_WARN_LIMIT [15:0], an interrupt is generated if the MCWDTx_CTRy_CONFIG.WARN_ACTION [8]

bit is set to ‘1’ in the MCWDTx_CTRy_CONFIG register. Both counters can be cleared automatically by each warn event when the MCWDTx_CTRy_CONFIG.AUTO_SERVICE [12] bit in the MCWDTx_CTRy_CONFIG register is set to ‘1’ when both MCWDTx_CTRy_CONFIG.UPPER_ACTION==NOTHING && MCWDTx_CTRy_CONFIG.LOWER_ACTION==NOTHING. This allows creation of a periodic interrupt if this counter is not needed as a watchdog. The CTR0_INT [0] or CTR1_INT [1] bits in the MCWDTx_INTR register are set whenever the corresponding MCWDT counter matches with the related WARN_LIMIT [15:0] and an interrupt occurs. This interrupt must be cleared by writing a '1' to the same bit.

If no automatic service is enabled the match event will keep the MCWDT counting until it reaches the timeout threshold value stored in MCWDTx_CTRy_UPPER_LIMIT [15:0]; this generates a FAULT if the MCWDTx_CTRy_CONFIG.UPPER_ACTION [5:4] bits are set to ‘1’ in the MCWDTx_CTRy_CONFIG register. In window mode, an early threshold stored in MCWDTx_CTRy_LOWER_LIMIT [15:0] can be used if the MCWDTx_CTRy_CONFIG.LOWER_ACTION [1:0] bit is set to ‘1’ in the MCWDTx_CTRy_CONFIG register to generate a FAULT if the counter is serviced too early.

All four faults for each counter (early and timeout for each 16-bit subcounter) are combined into a single fault triggered, so the fault structure can record the correct fault cause. The fault structure can convert this to an interrupt (such as a high-priority NMI) that gives the processor an opportunity to return to a safe state, such as halting memory writes and releasing peripherals. It can then clear the FAULT and trigger its own local reset. If the FAULT is not cleared within a fixed number of LFCLK cycles, MCWDT will trigger a system-wide reset as a failsafe if the value ‘2’ is written to MCWDTx_CTRy_CONFIG.LOWER_ACTION [1:0] or MCWDTx_CTRy_CONFIG.UPPER_ACTION [5:4] bits.

***Note**: MCWDT does not report overlapping faults generated by other subcounter actions. If a single MCWDT triggers additional fault action(s) while it is transferring fault data to the fault manager, then the fault manager receives only the first action in the fault data. The additional overlapping fault action(s) do not cause another fault report and are lost. Faults are transferred properly if they occur when the MCWDT is not transferring another fault. When processing an MCWDT fault, a missed MCWDTx_CTRy_CONFIG.UPPER_ACTION fault can be detected. For each subcounter 0 and 1, if MCWDTx_CTRy_CONFIG.UPPER_ACTION is configured for FAULT or FAULT_THEN_RESET, and CNT ≥ MCWDTx_CTRy_UPPER_LIMIT, then process it as a fault even if it is not present in the fault data. There is no known method to detect a missed MCWDTx_CTRy_CONFIG.LOWER_ACTION fault.*

If no action is taken on the upper threshold the counter increments up to the 16-bit boundary at which point, it wraps around to 0 and counts up.

The watchdog counters are serviced by dedicated service bits. CTR0_SERVICE [0] bit is related to subcounter 0 and CTR1_SERVICE [1] is related to subcounter 1. Both bits are located in the MCWDTx_SERVICE register. If this bit is set to ‘1’ the watchdog counter is set to zero.

***Note**: When the software writes the MCWDT SERVICE bit in the MCWDTx_SERVICE register just before updating a limit register in an enabled MCWDT counter, the limit update may take effect before the service clears the counter. The new limit may trigger actions when they are compared to the uncleared counter value. For example, this can happen if the value in the MCWDTx_CTRy_LOWER_LIMIT register is changed to a value smaller than the existing CNT value. An unexpected fault or reset can occur during update, depending on the MCWDT configuration. To avoid this issue, make sure that a pending service is completed by waiting until the SERVICE bit value is read ‘0’ before writing the limit registers. It can take up to three LFCLK cycles for the service to complete.*

Subcounter 0 and Subcounter 1 can be enabled or disabled using the ENABLE [31] bit of the MCWDTx_CTRy_CTL register. The actual status of the counter is indicated by the ENABLED [0] bit of the MCWDTx_CTRy_CTL register.

[2] Subcounter 0 and subcounter 1 have its own register sets. For simplification MCWDTx prefix is used for both register sets, MCWDT0 and MCWDT1. In all cases when subcounter 2 is used, MCWDT2 register set is mentioned.

Both subcounters have the same mechanism to lock the MCWDT Configuration registers as provided by the basic WDT. When the MCWDT_LOCK[1:0] bits in the MCWDTx_LOCK register are not equal to '0' the write access to the MCWDTx_CTRy_CTL, MCWDTx_CTRy_LOWER_LIMIT, MCWDTx_CTRy_UPPER_LIMIT,MCWDTx_CTRy_WARN_LIMIT, MCWDTx_CTRy_CONFIG, MCWDTx_CTRy_CNT, and MCWDTx_SERVICE registers is prohibited. MCWDT will be unlocked and disabled on any reset.

***Note**: The lock mechanism is an additional safety opportunity, which requires to unlock/lock the SERVICE register when servicing each watchdog counter. Alternatively, the MCWDT registers can also be protected by the PPU, which allows to keep these registers unlocked.*

When the watchdog counter is disabled and unlocked, the count value can be written for verification and debugging purposes. Software writes are always ignored when the counter is enabled.

Figure 90 shows the operation of the 16-bit subcounters. If the MCWDTx_CTRy_CONFIG.WARN_ACTION is activated, the counter can be used for interrupt generation exclusively. The counter continues to increment after the counter value matches the WARN_LIMIT until the 16-bit maximum value is reached. Then the counter restarts at zero. Note that the interrupt period is fixed in this use case. For various interrupt timing the counter must be serviced regularly.

To clear the counter manually within an ISR, the CTR0_SERVICE[0] or CTR1_SERVICE[1] bit in the MCWDTx_SERVICE register should be set to '1'. Alternatively, the counter can be serviced automatically by setting MCWDTx_CTRy_CONFIG.AUTO_SERVICE[12] bit in the CONFIG register to '1'.

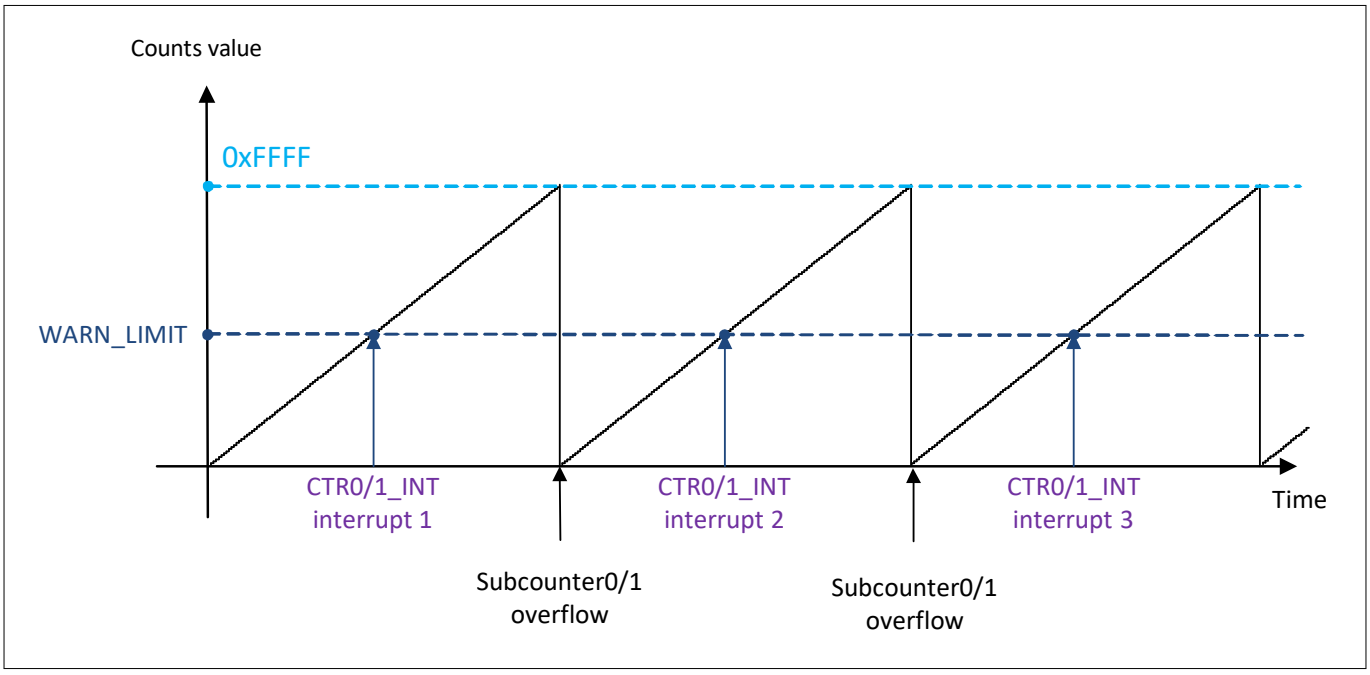


Figure 90 Subcounter 0/1 operation with WARN interrupt only (MCWDTx_CTRy_CONFIG.AUTO_SERVICE = 0)

Figure 91 illustrates the interrupt mode with enabled automatic service (MCWDTx_CTRy_CONFIG.AUTO_SERVICE = 1). Whenever the counter matches the WARN_LIMIT value, an interrupt is issued and the counter is restarted with zero.

***Note**: The MCWDTx_CTRy_CONFIG.AUTO_SERVICE bit is ignored when either MCWDTx_CTRy_CONFIG.LOWER_ACTION or MCWDTx_CTRy_CONFIG.UPPER_ACTION is enabled.*

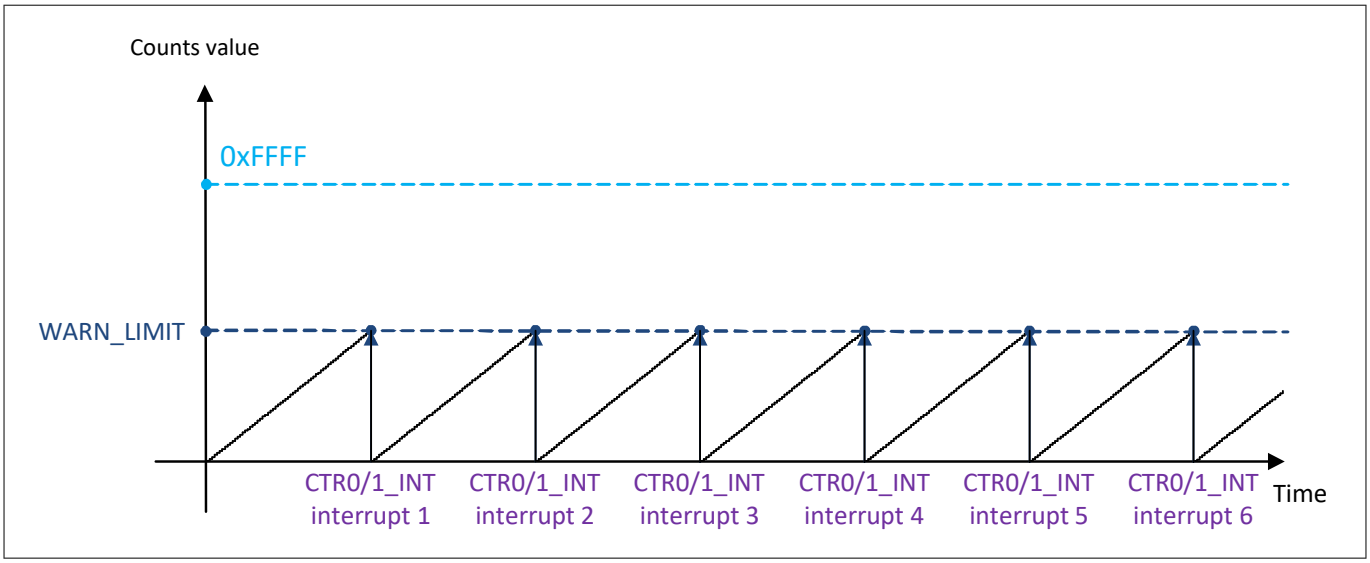


Figure 91 Subcounter 0/1 operation with WARN interrupt only (MCWDTx_CTRy_CONFIG.AUTO_SERVICE = 1)

In Figure 92 the window mode is shown when FAULT_THEN_RESET function is enabled. Four scenarios can happen while MCWDTx_CTRy_CONFIG.LOWER_ACTION, MCWDTx_CTRy_CONFIG.WARN_ACTION, and MCWDTx_CTRy_CONFIG.UPPER_ACTION are activated:

- Counter is serviced between MCWDTx_CTRy_LOWER_LIMIT and WARN_LIMIT: This is the regular behavior of the MCWDT. No WARN interrupt is issued and no RESET is done.
- Counter is serviced between WARN_LIMIT and MCWDTx_CTRy_UPPER_LIMIT: The service is done late; a WARN interrupt is issued but no RESET is done.
- Counter is not serviced at all: WARN interrupt is issued but the CTR0_SERVICE or CTR1_SERVICE bit is not set. When the counter reaches the MCWDTx_CTRy_UPPER_LIMIT, a FAULT is issued. If the firmware does not handle this FAULT to bring the system back into a safe state, a RESET is issued after a fixed number of LFCLK cycles.
- Counter is serviced before the MCWDTx_CTRy_LOWER_LIMIT is reached: The counter is serviced too early; a FAULT is issued followed by a RESET in case the FAULT is not handled in time by the firmware.

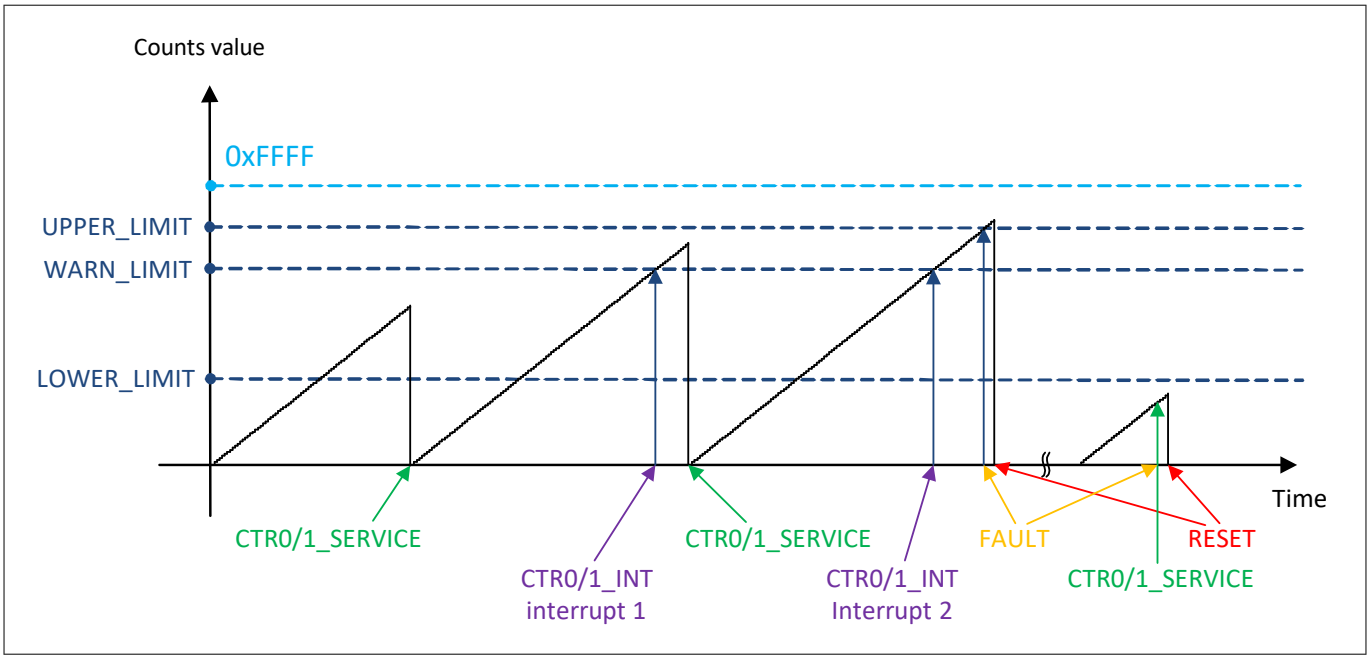


Figure 92 Subcounter 0/1 operation in window mode with FAULT and RESET action

***Note**: This figure illustrates the different scenarios with or without servicing the watchdog counter. It does not consider the WDT configuration, especially after a reset.*

Table 123 MCWDT subcounter 0 and subcounter 1 configuration options

Register [Bit_Pos]	Bit name	Description
MCWDTx_CTRy_CONFIG[1:0]	LOWER_ACTION	Action taken if this watchdog is serviced before MCWDTx_CTRy_LOWER_LIMIT is reached 0: Do nothing 1: FAULT 2: FAULT_THEN_RESET
MCWDTx_CTRy_CONFIG[5:4]	UPPER_ACTION	Action taken if this watchdog is not serviced before MCWDTx_CTRy_UPPER_LIMIT is reached 0: Do nothing 1: FAULT 2: FAULT_THEN_RESET
MCWDTx_CTRy_CONFIG[8]	WARN_ACTION	Action taken when the count value reaches WARN_LIMIT 0: Do nothing 1: Interrupt
MCWDTx_CTRy_CONFIG[12]	AUTO_SERVICE	Automatically service when the count value reaches WARN_LIMIT

(table continues...)

Table 123 (continued) MCWDT subcounter 0 and subcounter 1 configuration options

Register [Bit_Pos]	Bit name	Description
MCWDTx_CTRy_CONFIG[28]	DEBUG_TRIGGER_EN	Enables the trigger input for the MCWDT to pause the counter in debug mode. 0: Pauses the counter when a debug probe is connected. 1: Pauses the counter when a debug probe is connected and the trigger input is high.
MCWDTx_CTRy_CONFIG[30]	SLEEPDEEP_PAUSE	Pauses/runs this counter when the corresponding processor is in SLEEPDEEP 0: Counter runs normally regardless of processor mode. 1: Counter pauses when corresponding processor is in SLEEPDEEP.
MCWDTx_CTRy_CONFIG[31]	DEBUG_RUN	Pauses/runs this counter while a debugger is connected 0: Counter pauses according to DEBUG_TRIGGER_EN configuration. 1: Counter runs normally when debugger connected.
MCWDTx_CTRy_CTL[31]	ENABLE	Enable or disable the watchdog reset 0: Counter is disabled (not clocked) 1: Counter is enabled (counting up)
MCWDTx_CTRy_CTL[0]	ENABLED	Indicates actual state of watchdog
MCWDTx_CTRy_CNT[15:0]	CNT	Current value of subcounter for this MCWDT
MCWDTx_CTRy_LOWER_LIMIT[15:0]	LOWER_LIMIT	Lower limit for watchdog
MCWDTx_CTRy_UPPER_LIMIT[15:0]	UPPER_LIMIT	Upper limit for watchdog
MCWDTx_CTRy_WARN_LIMIT[15:0]	WARN_LIMIT	Warn limit for watchdog
MCWDTx_LOCK[1:0]	MCWDT_LOCK	Prohibits Writing Control and Configuration registers related to this MCWDT when not equal to '0'. 0: No effect 1: Clear bit 0 2: Clear bit 1 3: Set both bit 0 and 1 (lock enabled)
MCWDTx_INTR[0]	CTR0_INT	MCWDT interrupt request for subcounter 0
MCWDTx_INTR[1]	CTR1_INT	MCWDT interrupt request for subcounter 1

(table continues...)

Table 123 (continued) MCWDT subcounter 0 and subcounter 1 configuration options

Register [Bit_Pos]	Bit name	Description
MCWDTx_INTR_MASK[0]	CTR0_INT	Mask for subcounter 0 for warning interrupt 0: MCWDT interrupt is masked to CPU. 1: MCWDT interrupt is not masked to CPU.
MCWDTx_INTR_MASK[1]	CTR1_INT	Mask for subcounter 1 for warning interrupt 0: MCWDT interrupt is masked to CPU. 1: MCWDT interrupt is not masked to CPU.

20.4.2.2 32-bit counter operation

The subcounter 2 (MCWDTx_CNT2) is a 32-bit free-running counter that can be configured to generate an interrupt. The MCWDTx_CTR2_CNT register holds the current value of subcounter 2. Subcounter 2 does not support the window mode. However, it can be configured to generate an interrupt when one of the counter bits toggles. The BITS[20:16] bit field of the MCWDTx_CTR2_CONFIG register selects the bit on which the subcounter 2 interrupt is asserted. ACTION bit [0] of the MCWDTx_CTR2_CONFIG register decides whether to assert an interrupt on bit toggle or not. Figure 93 shows the subcounter 2 counter operation.

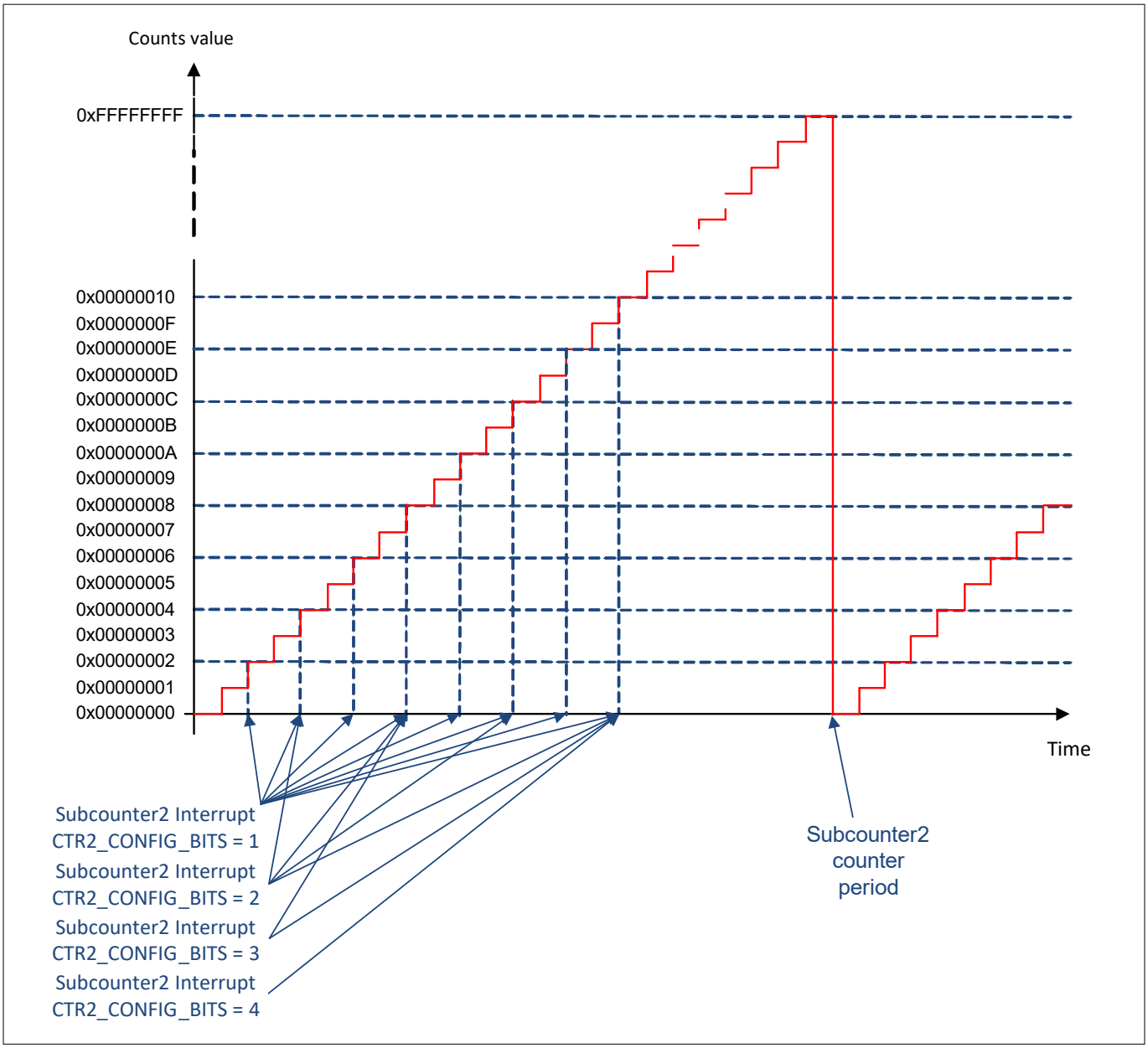


Figure 93 Subcounter 2 operation

Table 124 MCWDT subcounter 2 configuration options

Register [Bit_Pos]	Bit name	Description
MCWDTx_CTR2_CONFIG[0]	ACTION	Action taken when the specified BIT toggles 0: Nothing 1: Trigger an interrupt
MCWDTx_CTR2_CONFIG[20:16]	BITS	Bit to observe for a toggle: 0: Do ACTION after CTR2_CNT[0] toggles (every tick) ... 31: Do ACTION after CTR2_CNT[31] toggles (every 2^31 ticks)

(table continues...)

Table 124 (continued) MCWDT subcounter 2 configuration options

Register [Bit_Pos]	Bit name	Description
MCWDTx_CTR2_CONFIG[28]	DEBUG_TRIGGER_EN	Enables the trigger input for the MCWDT to pause the counter in debug mode. 0: Pauses the counter when a debug probe is connected. 1: Pauses the counter when a debug probe is connected and the trigger input is HIGH.
MCWDTx_CTR2_CONFIG[30]	SLEEPDEEP_PAUSE	Pauses/runs this counter when the corresponding processor is in SLEEPDEEP 0: Counter runs normally regardless of processor mode. 1: Counter pauses when corresponding processor is in SLEEPDEEP.
MCWDTx_CTR2_CONFIG[31]	DEBUG_RUN	Pauses/runs this counter while a debugger is connected 0: Counter pauses according to DEBUG_TRIGGER_EN configuration. 1: Counter runs normally when debugger connected.
MCWDTx_INTR[2]	CTR2_INT	MCWDT interrupt request for subcounter 2
MCWDTx_INTR_MASK[2]	CTR2_INT	MCWDT Interrupt Mask register for subcounter 2 0: MCWDT interrupt is masked to CPU 1: MCWDT interrupt is not masked to CPU
MCWDTx_ INTR_MASKED[2]	CTR2_INT	MCWDT interrupt masked register for subcounter 2. Logical AND of corresponding request and mask bits

20.4.3 Enabling and disabling MCWDT

The MCWDT counters are enabled by setting the ENABLE[31] bit in the MCWDTx_CTRy_CTL and MCWDTx_CTR2_CTL registers and are disabled by clearing it. Enabling or disabling a MCWDT counter requires two LFCLK cycles to come into effect. Therefore, the ENABLE bit value must not be changed more than once in that period and the ENABLED[0] bit of the MCWDTx_CTRy_CTL and MCWDTx_CTR2_CTL registers can be used to monitor the enabled/disabled state of the counter. The CTR0_SERVICE[0] and CTR1_SERVICE[1] bits of the MCWDTx_SERVICE register clears the corresponding subcounter when set in firmware. The hardware clears the bit after the MCWDT counter resets. This option is useful when subcounter 0 or subcounter 1 is configured to generate a device reset after a FAULT event. After the MCWDT counter is enabled, it is not recommended to write to the MCWDT configuration (MCWDTx_CTRy_CONFIG and MCWDTx_CONFIG) and control (MCWDTx_CTRy_CTL and MCWDTx_CTR2_CTL) registers. Accidental corruption of MCWDT registers can be prevented by setting the MCWDT_LOCK[1:0] bit of the MCWDTx_LOCK register. If the application requires updating any register while the WDT is running, the MCWDT_LOCK bits must be cleared. The MCWDT_LOCK bits require two different writes to clear both the bits. Writing a '1' to the bits clears bit 0. Writing a '2' clears bit 1. Writing a '3' sets both the bits and writing '0' does not have any effect. Note that the MCWDT_LOCK bits are only protecting following registers:

- MCWDTx_CTRy_CTL
- MCWDTx_CTRy_LOWER_LIMIT
- MCWDTx_CTRy_UPPER_LIMIT

- MCWDTx_CTRy_WARN_LIMIT
- MCWDTx_CTRy_CONFIG
- MCWDTx_CTRy_CNT
- MCWDTx_CTR2_CTL
- MCWDTx_CTR2_CONFIG
- MCWDTx_CTR2_CNT
- MCWDTx_SERVICE

Table 125 Watchdog configuration options

Register [Bit_Pos]	Bit name	Description
MCWDTx_CTRy_CTL[31] MCWDTx_CTL[31]	ENABLE ENABLE	Enable or disable the watchdog reset 0: Counter is disabled (not clocked) 1: Counter is enabled (counting up)
MCWDTx_CTRy_CTL[0] MCWDTx_CTL[0]	ENABLED ENABLED	Indicates actual state of watchdog
MCWDTx_LOCK[1:0]	MCWDT_LOCK	Locks or unlocks write access to the MCWDT registers. When the bits are set, the lock is enabled. 0: No effect 1: Clears bit 0 2: Clears bit 1 3: Sets both bit 0 and 1 (lock enabled)
MCWDTx_SERVICE[0] MCWDTx_SERVICE[1]	CTR0_SERVICE CTR1_SERVICE	Services subcounter 0. This resets the count value for subcounter 0 to zero Services subcounter 1. This resets the count value for subcounter 1 to zero

***Note**: When the watchdog counters are configured to generate an interrupt every LFCLK cycle, make sure you read the MCWDTx_INTR register after clearing the watchdog interrupt (setting the CTR0_INT, CTR1_INT, and CTR2_INT bits in the MCWDTx_INTR register). Failure to do this may result in missing the next interrupt. Hence, the interrupt cycle will become LFCLK/2.*

20.4.4 Watchdog reset

Subcounter 0 and subcounter 1 can be configured to generate a device reset similar to the basic WDT reset. Follow these steps to use subcounter 0 or subcounter 1 of an MCWDT block to generate a system reset. Note that a reset is asserted after an unhandled FAULT condition. The subcounters can be individually configured whether to generate only a FAULT, or a reset after a FAULT event.

1. Configure the MCWDT to generate a reset by setting MCWDTx_CTRy_CONFIG.LOWER_ACTION[1:0] or MCWDTx_CTRy_CONFIG.UPPER_ACTION[5:4] bits in the MCWDTx_CTRy_CONFIG register to '2'.
2. Calculate the watchdog reset period such that firmware is able to reset the watchdog at least once during the period, even along the longest firmware delay path, and write the value into the MCWDTx_CTRy_UPPER_LIMIT register. In window mode define an adequate MCWDTx_CTRy_LOWER_LIMIT, which cannot be violated by the shortest firmware delay path.
3. Enable MCWDT by setting the ENABLE[31] bit in the MCWDTx_CTRy_CTL register. Wait until the ENABLED[0] bit is set.

4. Lock the MCWDT configuration by setting the MCWDT_LOCK bits of the MCWDTx_LOCK register to '3'.
5. In the firmware, feed (reset) the watchdog by writing '1' into the CTR0_SERVICE[0] or CTR1_SERVICE[1] bit in the MCWDTx_SERVICE register.

It is not recommended to reset watchdog in the MCWDT ISR.

20.4.5 Watchdog interrupt

When configured to generate an interrupt, the CTR0_INT (subcounter 0), CTR1_INT (subcounter 1), and CTR2_INT (subcounter 2) bits of the MCWDTx_INTR register provide the status of any pending watchdog interrupts. The firmware must clear the interrupt by setting the same bit to '1'. The CTR0_INT, CTR1_INT, and CTR2_INT bits of the MCWDTx_INTR_MASK register unmask the corresponding MCWDT interrupt to the CPU.

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

1. Write the desired warning threshold value to the WARN_LIMIT register for subcounter 0 and subcounter 1 or the BITS[20:16] value to the MCWDTx_CTR2_CONFIG register for subcounter 2.
2. For subcounter 0 and subcounter 1 configure the MCWDT to generate an interrupt using the MCWDTx_CTRy_CONFIG.WARN_ACTION[8] bit in MCWDTx_CTRy_CONFIG register. For subcounter 2, set the ACTION[0] bit in the MCWDTx_CTR2_CONFIG register.
3. Set the CTR0_INT, CTR1_INT, and CTR2_INT bits in MCWDTx_INTR to clear any pending interrupt.
4. Set the MCWDTx_CTRy_CONFIG.AUTO_SERVICE[12] bit in MCWDTx_CTRy_CONFIG for subcounter 0 and subcounter 1 to reset the corresponding watchdog counter to '0' on a warning interrupt event.

***Note**: For subcounter 2, no automatic counter clearing is supported.*

5. Unmask the MCWDT interrupt to the CPU by setting the CTRx_INT bit in the MCWDTx_INTR_MASK register.
6. Enable MCWDT by setting the ENABLE[31] bit in the MCWDTx_CTRy_CTL register. Wait until the ENABLED[0] bit is set.
7. Enable the MCWDT interrupt to the CPU by configuring the appropriate ISER register. Refer to the Interrupts.
8. In the ISR, clear the MCWDT interrupt by setting the CTRx_INT bit in the MCWDTx_INTR register.

Note that interrupts from all three subcounters of the MCWDT block are mapped as a single interrupt to the CPU.

In the interrupt service routine, the CTRx_INT bits of the MCWDTx_INTR register can be read to identify the interrupt source. However, each MCWDT block has its own interrupt to the CPU. For details on interrupts, see the Interrupts. The MCWDT block can send interrupt requests to the CPU in active power mode and to the WIC in sleep and deep sleep power modes. It works similar as the basic WDT.

The hardware does not support changing the timeout for deep sleep mode. However, subcounters 0 and subcounter 1 can work together to get a similar behavior. Subcounter 0 can be configured with a timeout threshold suitable to protect running firmware and configured to stop during deep sleep. Subcounter 1 can be configured with a longer timeout that continues to operate in deep sleep. For this usage example to work, the early window thresholds should be the same (if window mode is enabled) and firmware should service both these subcounters at the same time.

20.5 Reset cause detection

The RESET_WDT bit [0] in the RES_CAUSE register indicates the reset generated by the basic WDT. The RESET_MCWDT0 [5], RESET_MCWDT1 [6], RESET_MCWDT2 [7], and RESET_MCWDT3 [8] bits in the RES_CAUSE register indicate the reset generated by the MCWDTx block. These bits remain set until cleared or until a power-on reset (POR), brownout reset (BOD), or external reset (XRES_L) occurs. All other resets leave the bits unaltered. For more details, see the Reset system.

20.6 Debug mode

For both types of WDTs, watchdog resets are automatically blocked by hardware during debugging, and window mode is automatically paused. By default, all the WDTs also stop counting. Two configuration bits (per WDT) configure the behavior of the counter when the debugger is connected. The recommended procedure to disconnect a debug probe is to service any active watchdog timers using the debug probe, then disconnect the probe. The firmware begins running again and the next service will realign the window and resume normal window operation.

In a multi-core environment, 'debug state' indicates that at least one of the CPUs is in debug state. If one CPU is debugged but another or multiple other CPUs are continuously running, then the user can configure the counter via the debugger to continue or pause depending on which CPU is using the counter.

The configuration is done with DEBUG_TRIGGER_ENABLE[28] and DEBUG_RUN[31] bits, which are both located in the related CONFIG register for basic WDT and MCWDT. Table 126 shows the configuration options.

Table 126 Debug modes

DEBUG_RUN	DEBUG_TRIGGER_ENABLE	Description
0	0	Counter is stopped when a debugger is connected.
0	1	Counter is stopped only when a debugger is connected and the CPU is halted during a breakpoint.
1	x	Counter is running when debugger is connected. No reset is issued when the CPU is halted during a breakpoint but the counter is not stopped.

Note that in each case, no reset and no FAULT is issued when the debugger is connected to the target system.

To pause at a breakpoint while debugging, configure the trigger matrix to connect the related CPU halted signal to the trigger input for the related watchdog timer. It takes up to two LFCLK cycles for the trigger signal to be processed. Triggers that are less than two LFCLK cycles may be missed. Synchronization errors can accumulate each time it is halted.

Note that it may take up to two ILO0 (or LFCLK for MCWDT) clock cycles for the counter to pause, due to internal synchronization. After the debugger is disconnected, the MCWDTx_CTRy_CONFIG.LOWER_ACTION is ignored until after the first service. This prevents an unintentional trigger of the MCWDTx_CTRy_CONFIG.LOWER_ACTION before the firmware realigns the servicing period. After the first service, MCWDTx_CTRy_CONFIG.LOWER_ACTION behaves as configured.

20.7 CPU select

In a multi-core system it is recommended to assign one MCWDT to a dedicated CPU to select the SLEEPDEEP signal to be used to control the counter in sleepdeep power mode. The counter pauses in sleepdeep power mode in case SLEEPDEEP_PAUSE[30] bit is set to '1' in CTR2_CONFIG register.

A single MCWDT is not intended to be used simultaneously by multiple CPUs because of the complexity involved in coordination.

CPU_SEL[1:0] bits in the CPU_SELECT register are defined in Table 127.

Table 127 MCWDT assignment to the cores

CPU_SEL[1:0]	CPU
0	CM0+
1	CM7_0
2	CM7_1

20.8 Registers

Table 128 WDT registers

Register	Name	Description
WDT_CTL	Watchdog Control register	Control register for the basic WDT
WDT_LOWER_LIMIT	WDT Lower Limit register	Lower limit for the basic WDT
WDT_UPPER_LIMIT	WDT Upper Limit register	Upper limit for the basic WDT
WDT_WARN_LIMIT	WDT Warn Limit register	Warn limit for the basic WDT
WDT_CONFIG	WDT Configuration register	Configuration for the basic WDT. Includes the ACTION configuration for upper, lower, and warn limits, auto-servicing, and pause settings in low-power and debug modes.
WDT_CNT	WDT Count register	Count value for the basic WDT
WDT_LOCK	WDT Lock register	Lock or unlock the basic WDT registers
WDT_SERVICE	WDT Service register	Clears the basic WDT counter
WDT_INTR	WDT Interrupt register	Interrupt signal from basic WDT
WDT_INTR_SET	WDT Interrupt Set register	Sets interrupts for firmware testing
WDT_INTR_MASK	WDT Interrupt Mask register	Controls whether interrupt is forwarded to CPU. All masks block the interrupt when '0' and forward the interrupt when '1'.
WDT_INTR_MASKED	WDT Interrupt Masked register	Bitwise AND between the Interrupt Request and Mask registers so firmware can read the status of all mask enabled interrupt causes with a single load operation.
MCWDTx_CTRy_CTL	MCWDT Subcounter 0/1 Control register	Control register for MCWDT Subcounter
MCWDTx_CTRy_LOWER_LIMIT	MCWDT Subcounter 0/1 Lower Limit register	Lower limit for this MCWDT Subcounter
MCWDTx_CTRy_UPPER_LIMIT	MCWDT Subcounter 0/1 Upper Limit register	Upper limit for this MCWDT Subcounter
MCWDTx_CTRy_WARN_LIMIT	MCWDT Subcounter 0/1 Warn Limit register	Warn limit for this MCWDT Subcounter
MCWDTx_CTRy_CONFIG	MCWDT Subcounter 0/1 Configuration register	Configuration for this MCWDT Subcounter. Includes the ACTION configuration for upper, lower, and warn limits.
MCWDTx_CTRy_CNTy	MCWDT Subcounter 0/1 Count register	Count value for this MCWDT Subcounter
MCWDTx_CTR2_CTL	MCWDT Subcounter 2 Control register	Control register for MCWDT Subcounter 2
MCWDTx_CTR2_CONFIG	MCWDT Subcounter 2 Configuration register	Configuration for MCWDT Subcounter 2

(table continues...)

Table 128 (continued) WDT registers

Register	Name	Description
MCWDTx_CTR2_CNT	MCWDT Subcounter 2 Count register	Count value for this MCWDT Subcounter 2
MCWDTx_LOCK	MCWDT Lock register	Lock or unlock the respective Configuration registers of Subcounters 0/1/2 of this MCWDT.
MCWDTx_SERVICE	MCWDT Service register	Includes service bits to clear subcounter 0/1 of this MCWDT.
MCWDTx_INTR	MCWDT Interrupt register	Interrupt Status register for subcounters 0/1/2 for this MCWDT.
MCWDTx_INTR_SET	MCWDT Interrupt Set register	Triggers an interrupt for firmware testing
MCWDTx_INTR_MASK	MCWDT Interrupt Mask register	Controls whether a subcounter interrupt is forwarded to the corresponding processor. All masks block the interrupt when ‘0’ and forward the interrupt when ‘1’.
MCWDTx_INTR_MASKED	MCWDT Interrupt Masked register	Bitwise AND between the Interrupt Request and Mask registers so firmware can read the status of all mask enabled interrupt causes with a single load operation.
CLK_SELECT	Clock Selection register	Clock Source Selection register
CLK_ILO0_CONFIG	ILO0 Configuration	ILO0 Configuration
RES_CAUSE	Reset Cause Observation register	Reset Cause Observation register

***Note**: In MCWDTx_CTRy, 'x' signifies the instance and 'y' signifies the subcounter (0/1). Refer to the device datasheet or the registers TRM for more information.*

21 Real-time clock

The real-time clock (RTC) system is an “always-on” function, which is a part of the backup domain. It contains a real-time clock with alarm feature, supported by a 32768-Hz watch crystal oscillator (WCO).

Backup is not a power mode; the Backup domain always runs on VDDD. For more details, see the Power supply and monitoring, the Device power modes, and the Clocking system for WCO.

21.1 Features

- Fully-featured RTC
  - Year/month/date, day-of-week, hour: minute: second fields (All fields Integer)
  - Supports both 12-hour and 24-hour formats
  - Automatic leap year correction
- Configurable alarm function
  - Alarm on month/date, day-of-week, hour: minute: second fields
  - Two independent alarms
- Calibration for a 32768-Hz WCO
- Calibration waveform output
  - Supports 512 Hz, 1 Hz, and 2 Hz
- Backup registers

21.2 Block diagram

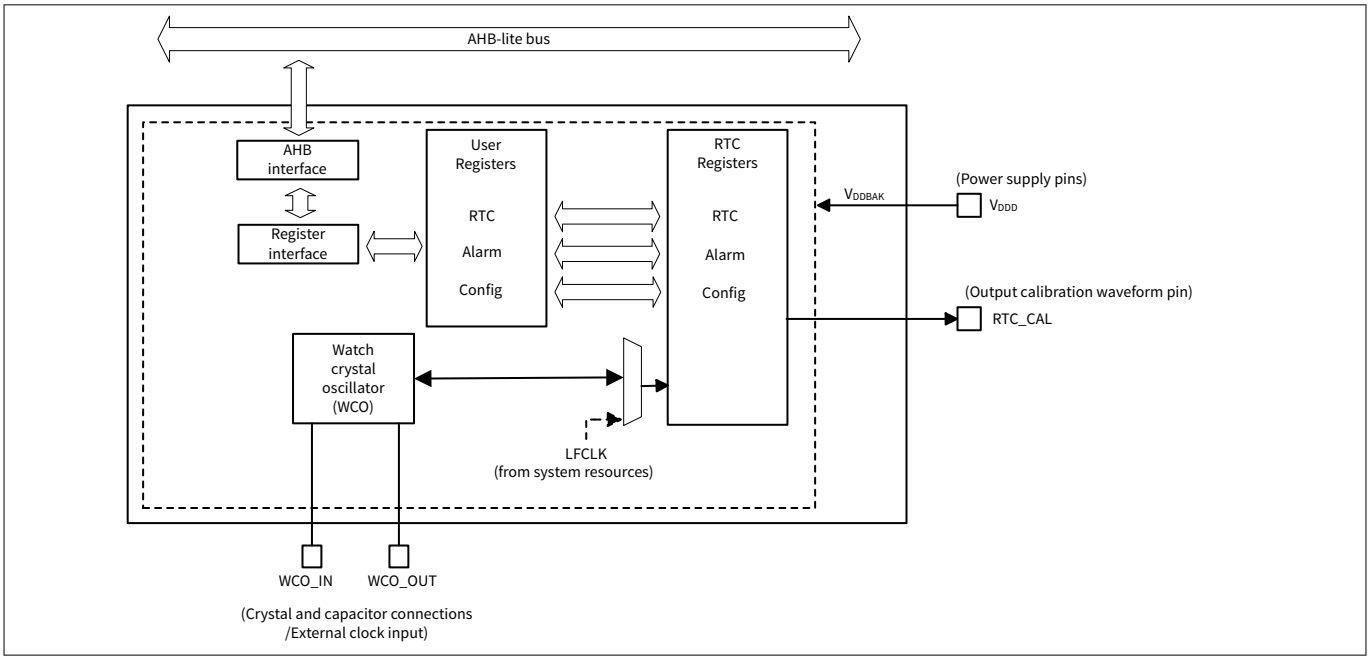


Figure 94 Block diagram

The RTC system includes an accurate WCO that can generate the required clock with the help of an external crystal or external clock inputs. The RTC has a programmable alarm feature, which can generate interrupts to the CPU. An AHB-Lite interface provides firmware access to registers interface in the backup domain.

The VDDBAK in the backup domain is always supplied from VDDD.

The domain also has a Backup registers block, which can retain its contents even when the device enters hibernate or XRES mode. The RTC system can also output a calibration waveform.

21.3 Power supply

Power to the RTC system is VDDD (unregulated main supply). See the Power supply and monitoring for more details.

It is possible to monitor the backup domain voltage (VDDD) using the low-voltage detect (LVD) feature of XMC7000. For more information on LVD, see the Power supply and monitoring.

21.4 Clocking

The RTC primarily runs from a 32768-Hz clock, after it is scaled down to one-second ticks. This clock signal can come from either of these internal sources:

- Watch-crystal oscillator (WCO). This is a high-accuracy clock generator that is suitable for RTC applications and requires a 32768-Hz external crystal populated on the application board. WCO can also operate without crystal, using external clock wave input. These additional operating modes are explained in the Clocking system. WCO is supplied by the Backup domain.
- Alternate backup clock (ALTBAK): This option allows the use of CLK_LF generated by the SRSS as the backup domain clock. Note that CLK_LF is not always available in all device power modes. See the Device power modes for more details. CLK_LF is described in the Clocking system. Clock glitches can propagate into the RTC system when CLK_LF is enabled or disabled by the SRSS. In addition, CLK_LF may not be as accurate as WCO depending on the actual source of CLK_LF. Because of these reasons, CLK_LF is not recommend for RTC applications. Also, if the WCO is intended as the clock source then choose it directly instead of routing through CLK_LF.
- Internal low-frequency oscillator (ILO): This option allows the use of ILO0. ILO0 is described in the Clocking system.

For more details on these clocks and calibration, see the Clocking system.

The RTC clock source can be selected using the BACKUP_CTL.CLK_SEL bit. The BACKUP_CTL.WCO_EN bit can be used to enable or disable the WCO. If the WCO operates with an external crystal, make sure the BACKUP_CTL.WCO_BYPASS bit is cleared before enabling the WCO. In addition, the BACKUP_CTL.PRESCALER bit must be configured for a prescaler value of 32768.

***Note**: External crystal and bypass capacitors of proper values must be connected to WCO_IN and WCO_OUT pins. See the device datasheet for details of component values and electrical connections. In addition, GPIOs must be configured for WCO_OUT and WCO_IN signals. See the I/O system to know how to configure the GPIOs.*

***Note**: If WCO is used as an RTC clock, then it is important to make sure that the WCO is running stable; that is, wait for BACKUP_STATUS.WCO_OK, before writing to the RTC registers.*

21.5 Reset

To keep the RTC operating through resets, the backup domain should not be reset under most circumstances.

The RTC initializes itself at power up; it cannot be reset by other internal and external resets such as BOD reset, OVD, OCD, WDT, and XRES_L. See the Reset system for more details.

The RTC system is reset only when all the power supplies are removed from the backup domain. Also, user firmware can reset the RTC system logic by using BACKUP_RESET.RESET.

If RES_CAUSE reports a BOD/OVD/OCD event, user firmware should initialize the RTC system by writing BACKUP_RESET.RESET=1 because faulty supplies may have corrupted the backup domain contents.

If RES_CAUSE reports a XRES_L or WDT event, it is an application-specific decision whether to trust the backup domain contents.

Although rare, XRES_L/WDT may mean that the backup domain contents were corrupted by faulty user firmware execution or by interrupting an AHB write to the backup logic.

21.6 Real-time clock

The RTC consists of seven integer fields and one control bit as shown in the following table:

Table 129 RTC fields

Bit field name	Description
RTC_SEC	Calendar seconds, value range = 0-59
RTC_MIN	Calendar minutes, value range = 0-59
RTC_HOUR	Calendar hours, value depends on 12-hour or 24-hour format set in the BACKUP_RTC_TIME.CTRL_12HR bit. In 12-hour mode, bit BACKUP_RTC_TIME.RTC_HOUR[4] = 0 for AM and 1 for PM, bits BACKUP_RTC_TIME.RTC_HOUR[3:0] = 1–12 In 24-hour mode, bits BACKUP_RTC_TIME.RTC_HOUR[4:0] = 0–23
CTRL_12HR	Select the 12-hour or 24-hour mode: 1=12HR, 0=24HR
RTC_DAY	Calendar day of the week, value range = 1-7 The user should define the meaning of the values
RTC_DATE	Calendar day of the month, value range = 1-31 Automatic leap year correction until 2400
RTC_MON	Calendar month, value range = 1-12
RTC_YEAR	Calendar year, value range = 0-99

RTC value fields indicate an integer format. Constant bits are omitted in the RTC implementation. For example, the maximum BACKUP_RTC_TIME.RTC_SEC is 59, which can be represented as one byte 0b00111011. However, the most significant bit is always zero and is therefore omitted, making the BACKUP_RTC_TIME.RTC_SEC a 6-bit field.

The RTC supports both 12-hour format with AM/PM flag, and 24-hour format for the “hours” field. The RTC also includes a “day of the week” field, which counts from 1 to 7. The user should define which weekday is represented by a value of ‘1’.

The RTC implements automatic leap year correction for the date field (day of the month). If the Year is divisible by four, the month of February (Month=2) will have 29 days instead of 28. When the Year field rolls over from 99 to 00, the firmware should update the otherwise static century value and therefore an interrupt is raised. This interrupt is called the century interrupt.

User registers containing these bit fields are BACKUP_RTC_TIME and BACKUP_RTC_DATE. See the corresponding register descriptions in the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual for details.

As the User registers are in the high-frequency bus-clock domain and the actual RTC registers run from the low-frequency 32768-Hz clock, reading and writing RTC registers require special care. These processes are explained in the following sections.

21.6.1 Reading RTC User registers

To start a read transaction, the firmware should set the BACKUP_RTC_RW.READ bit. When this bit is set, the RTC registers will be copied to User registers and frozen so that a coherent RTC value can safely be read by the firmware. The read transaction is completed by clearing the BACKUP_RTC_RW.READ bit.

BACKUP_RTC_RW.READ bit cannot be set if:

- RTC is still busy with a previous operation (that is, the BACKUP_STATUS.RTC_BUSY bit is set)
- The BACKUP_RTC_RW.WRITE bit is set

The firmware should verify that the above bits are not set before setting the BACKUP_RTC_RW.READ bit.

21.6.2 Writing to RTC User registers

When the BACKUP_RTC_RW.WRITE bit is set, data can be written into the RTC User registers; otherwise, writes to the RTC User registers are ignored. When all the RTC writes are done, the firmware needs to clear the BACKUP_RTC_RW.WRITE bit for the RTC update to take effect. After the BACKUP_RTC_RW.WRITE bit is cleared, the hardware will copy all the new data on one single WCO clock edge to ensure coherency to the actual RTC registers.

BACKUP_RTC_RW.WRITE bit cannot be set if:

- RTC is still busy with a previous operation (that is, the BACKUP_STATUS.RTC_BUSY bit is set)
- BACKUP_RTC_RW.READ bit is set

The firmware should make sure that the values written to the RTC fields form a coherent legal set. The hardware does not check the validity of the written values. Writing illegal values results in undefined behavior of the RTC.

When in the middle of an RTC update with the BACKUP_RTC_RW.WRITE bit set, and a brownout, reset, or entry to deep sleep or hibernate mode occurs, the write operation will not be complete. This is because the BACKUP_RTC_RW.WRITE bit will be cleared by a reset, and the RTC update is only triggered when this bit is cleared by an AHB WRITE transaction. If the write operation is in progress (BACKUP_STATUS.RTC_BUSY), data corruption can occur if the system is reset or enters deep sleep or hibernate mode.

To update only one or a few of the RTC fields, for example, when the RTC is adjusted for daylight saving time (DST), then only the Hour field needs an update, although the seconds and minutes fields should not be disturbed – they should continue running. For that reason, an ‘Update’ flag is maintained for each RTC field. Only those fields that have been updated will be copied to the actual RTC when the BACKUP_RTC_RW.WRITE bit is cleared.

21.7 WCO calibration

It is possible to improve the accuracy of the RTC by calibrating the WCO. The CLK_LF can also be calibrated. See the Clocking system for details.

The WCO accuracy is affected by an absolute crystal accuracy. This occurs because the crystal itself oscillates slightly faster or slower due to imperfect manufacturing. The user firmware can calibrate the RTC accuracy. The calibration bit fields are as follows:

Table 130 Calibration bit fields

Bit field name	Description
CALIB_VAL	Calibration value for absolute frequency. Each step causes 128 ticks to be added or removed each hour.
CALIB_SIGN	0: Negative sign: remove pulses (it takes more clock ticks to count one second) 1: Positive sign: add pulses (it takes less clock ticks to count one second)

(table continues...)

Table 130 (continued) Calibration bit fields

Bit field name	Description
CAL_SEL	Select calibration wave output signal 0: 512-Hz wave, not affected by calibration setting 1: Reserved 2: 2-Hz wave, includes the effect of the calibration setting 3: 1-Hz wave, includes the effect of the calibration setting
CAL_OUT	Output enable for wave signal for calibration, and allow BACKUP_CAL_CTL.CALIB_VAL to be written.

21.7.1 Absolute accuracy calibration

To measure the WCO error, the CAL_OUT bit must be set; this will cause a clock derived from the RTC system to be output on the RTC_CAL pin. The user should measure the deviation from 512 Hz, convert that to a ppm value, and derive the calibration settings to be used to correct the error.

The calibration correction is done by either adding or removing pulse counts from the oscillator divider each hour, which respectively speeds up or slows down the clock. After a calibration starts, it is performed hourly; it is applied as 64 ticks every 30 seconds until there are 2 × BACKUP_CAL_CTL.CALIB_VAL adjustments.

Because this is digital calibration, changing the calibration value does not affect the 512 Hz calibration output clock signal. XMC7000 supports two others calibration waveform frequencies; 1 Hz and 2 Hz. However, those calibration waveforms are affected by the current calibration.

The calibration register can only be written when the BACKUP_RTC_RW.WRITE bit is set. See Writing to RTC User registers.

21.8 Alarm feature

The alarm feature allows the RTC to generate an interrupt, which may be used to wake up the system from sleep, deep sleep, and hibernate power modes. The alarm feature consists of six fields corresponding to the fields of the RTC: month/date, day-of-week, and hour: minute: second. Each alarm field has an enable bit that needs to be set to enable matching; if the bit is cleared, then the field will be ignored for matching. Table 131 shows the alarm bit fields.

Table 131 Alarm bit fields

Bit field name	Description
ALM_SEC	Alarm seconds, value range = 0-59
ALM_SEC_EN	Alarm second enable: 0=disable, 1=enable
ALM_MIN	Alarm minutes, value range = 0-59
ALM_MIN_EN	Alarm minutes enable: 0=disable, 1=enable
ALM_HOUR	Alarm hours, value depending on the 12-hour or 24-hour mode. In 12-hour mode, bit BACKUP_ALMx_TIME.ALM _HOUR[4] = 0 for AM and 1 for PM, bits BACKUP_ALMx_TIME.ALM_HOUR[3:0] = 1–12 In 24-hour mode, bits BACKUP_ALMx_TIME.ALM_HOUR[4:0] = 0–23
ALM_HOUR_EN	Alarm hour enable: 0=disable, 1=enable

(table continues...)

Table 131 (continued) Alarm bit fields

Bit field name	Description
ALM_DAY	Calendar day of the week, value range = 1-7 The user should define the meaning of the values
ALM_DAY_EN	Alarm day of the week enable: 0=disable, 1=enable
ALM_DATE	Alarm day of the month, value range = 1-31
ALM_DATE_EN	Alarm day of the month enable: 0=disable, 1=enable
ALM_MON	Alarm month, value range = 1-12
ALM_MON_EN	Alarm month enable: 0=disable, 1=enable
ALM_EN	Master enable for alarm. 0: Alarm is disabled. Fields for date and time are ignored. 1: Alarm is enabled. If none of the date and time fields are enabled, then this alarm triggers once every second.

If the master enable (BACKUP_ALMx_DATE.ALM_EN) is set, but all alarm fields for date and time are disabled, an alarm interrupt will be generated once every second. Note that there is no alarm field for Year because the life expectancy of a chip is about 20 years. Thus, setting an alarm for a certain year indicates that the alarm matches either once or never in the lifetime of the chip.

XMC7000 has two independent alarms. See the BACKUP_ALM1_TIME, BACKUP_ALM1_DATE,BACKUP_ALM2_TIME, and BACKUP_ALM2_DATE registers in the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual for details.

Note that the Alarm User registers, similar to RTC User registers, require the same steps for read/write operations, as explained in Reading RTC User registers and Writing to RTC User registers.

Interrupts must be properly configured for the RTC to generate interrupts/wakeup events. Also, to enable RTC interrupts to wake up the device from hibernate mode, the PWR_HIBERNATE.MASK_HIBALARM bit must be set. See the Device power modes and the Interrupts for details.

BACKUP_INTR_MASK register can be used to disable certain interrupts from the RTC system.

Table 132 Interrupt mask bits

Bit name	Description
ALARM1	Mask bit for interrupt generated by ALARM1
ALARM2	Mask bit for interrupt generated by ALARM2
CENTURY	Mask bit for century interrupt, generated when the Year field rolls over from 99 to 00

21.9 Backup registers

The RTC system has several registers (BACKUP_BREGx), which can be used to store important information/ flags. This includes information that need to be retained when the device enters hibernate mode. For the number of BACKUP_BREGx registers, see the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual.

21.10 Real time Clock registers

***Note**: Refer to the device-specific datasheet to see whether this feature is supported.*

Table 133 Backup registers

Register	Name	Description
BACKUP_CTL	Control register	This register provides several settings of RTC operation. This register is hold in all device power modes including sleep, low-power sleep, deep sleep and hibernate.
BACKUP_RTC_RW	RTC Read Write register	This register provides read and write control function. This register is reset in deepsleep.
BACKUP_CAL_CTL	Oscillator Calibration Control register	This register provides oscillator calibration for absolute frequency.
BACKUP_STATUS	Status register	This register provides status of the RTC System. Firmware must monitor these bits to execute some operation. This register is hold in all device power modes including sleep, low-power sleep, deep sleep, and hibernate.
BACKUP_RTC_TIME	RTC Time register	This register provides calendar seconds, minutes, hours, and day of week.
BACKUP_RTC_DATE	RTC Date register	This register provides calendar day of month, month, and year.
BACKUP_ALM1_TIME	Alarm1 Time register	This register provides alarm 1 seconds, minute, hours, and day of week.
BACKUP_ALM1_DATE	Alarm1 Date register	This register provides alarm 1 day of month, and month.
BACKUP_ALM2_TIME	Alarm2 Time register	This register provides alarm 2 seconds, minute, hours, and day of week.
BACKUP_ALM2_DATE	Alarm2 Date register	This register provides alarm 2 day of month, and month.
BACKUP_INTR	Interrupt Request register	This register holds Interrupt signals. This register is sets by hardware if Interrupts condition occur. Firmware can clear these bits with writing '1'.
BACKUP_INTR_SET	Interrupt Set Request register	This register is for firmware testing. Interrupts occur if firmware set '1' to these bits. (For firmware testing purpose)
BACKUP_INTR_MASK	Interrupt Mask register	This register provides Interrupt mask. When Mask bit is set, the interrupt is enabled.
BACKUP_INTR_MASKED	Interrupt Masked Request register	This register allows the firmware 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.
BACKUP_BREGx	Backup register	These registers provide backup register regions. 'x' signifies the number of backup registers.
BACKUP_RESET	RTC System Reset register	This register is used to reset the RTC system from firmware.

***Note**: 'x' signifies the number of backup register. Refer to the registers TRM for more information.*

22 I/O system

Top-level architecture

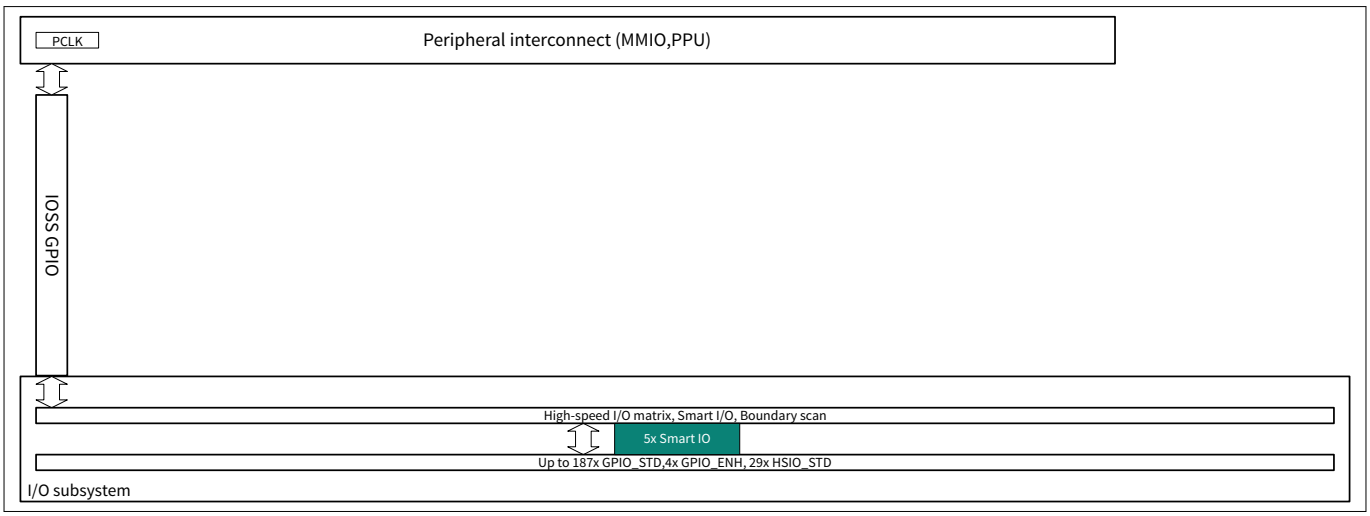


Figure 95 I/O system block diagram

This chapter explains the XMC7000 I/O system, its features, architecture, operating modes, and interrupts. The I/O system provides the interface between the CPU core and peripheral components. The flexibility of XMC7000 devices and the capability of its I/O to route most signals to most pins simplifies circuit design and board layout. The GPIO pins are grouped into ports; a port can have a maximum of eight GPIOs.

This chapter describes the following:

- Features and overview
- I/O cell architecture
- GPIO port configuration, interrupt support, and software I/O functionality
- I/O subsystem
- Smart I/O

22.1 Features

The XMC7000 family GPIOs have these features:

- Analog and digital input and output capabilities
- Eight drive strength modes
- Separate Port Read and Write registers
- Edge-triggered interrupts on rising edge, falling edge, or on both the edges, on all GPIO
- Slew rate control
- Hold mode for latching previous state (used to retain the I/O state in deepsleep mode)
- Selectable CMOS, TTL, and industrial input buffer mode
- Smart I/O provides the ability to perform Boolean functions in the I/O signal path

22.2 GPIO interface overview

Each of the GPIOs may fall into one of the following categories:

- GPIO cells provide a means for the CPU and peripherals to communicate off-chip. All GPIO cells are software-controllable and observable by the CPU. Some or all GPIO cells may be routed to one or more peripherals. A peripheral I/O signal may be routed to multiple GPIO cells; HSIOM Control registers specify the active route connection.
- System function cells such as reset or power supplies.
- Application-specific I/O pins

Analog peripheral connectivity:

- Some GPIO cells have dedicated analog connections to programmable analog peripherals, such as SARMUX.

XMC7000 is equipped with analog and digital peripherals. Figure 96 shows an overview of the routing between the peripherals and pins.

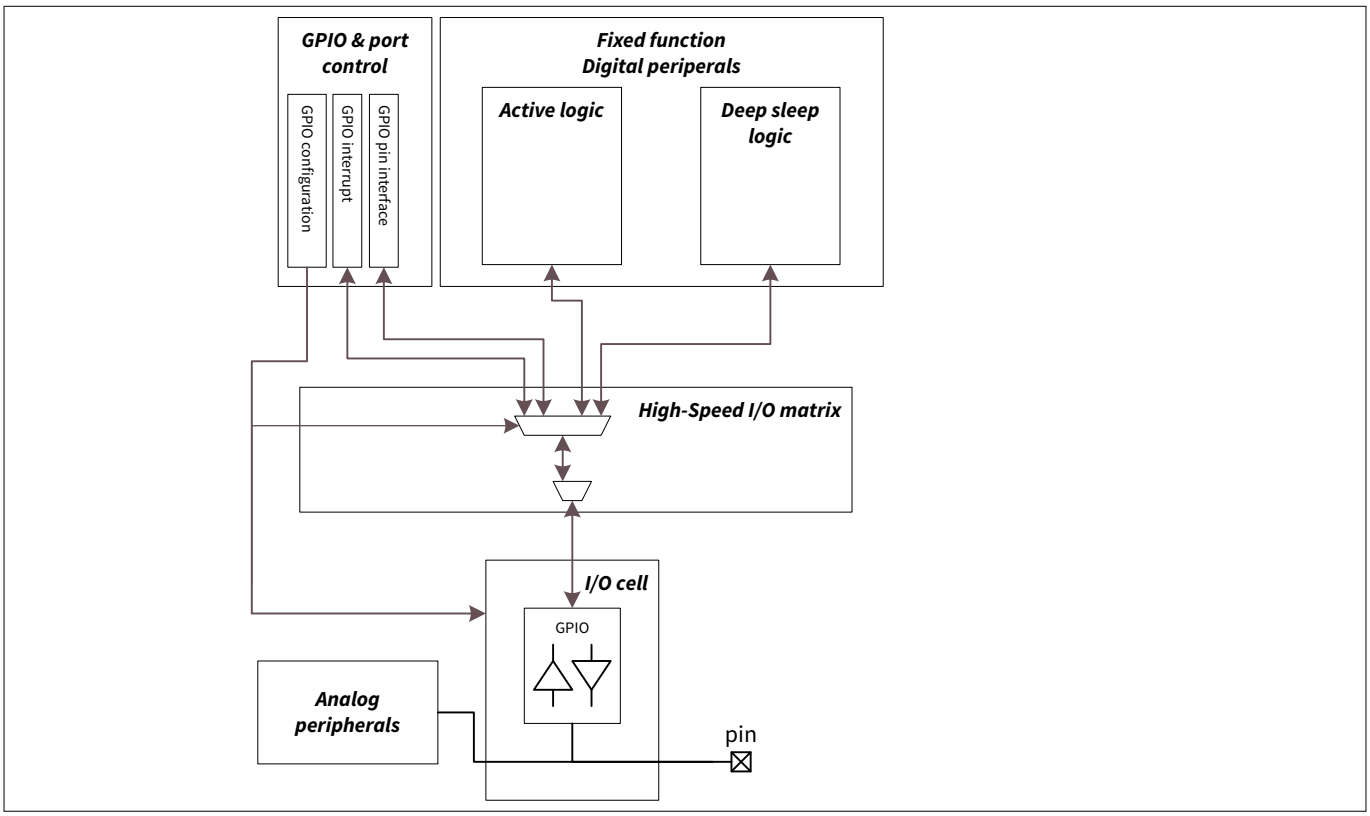


Figure 96 GPIO interface overview

The device has several options for interfacing to external signals and devices operating from a different supply voltage.

XMC7000 devices may optionally include a V_{DDIO} supply pin with a different voltage supply from the V_{DDD} pin. Where included, the V_{DDIO} pin may be used to power some or all of the GPIO cells, providing a built-in level-translator capability.

22.3 I/O cell architecture

Figure 97 shows the I/O cell architecture present in every GPIO cell. It comprises an input buffer and an output driver that connect to the HSIOM multiplexers for digital input and output signals. Analog peripherals connect directly to the pin for point-to-point connections or use of the AMUXBUS.

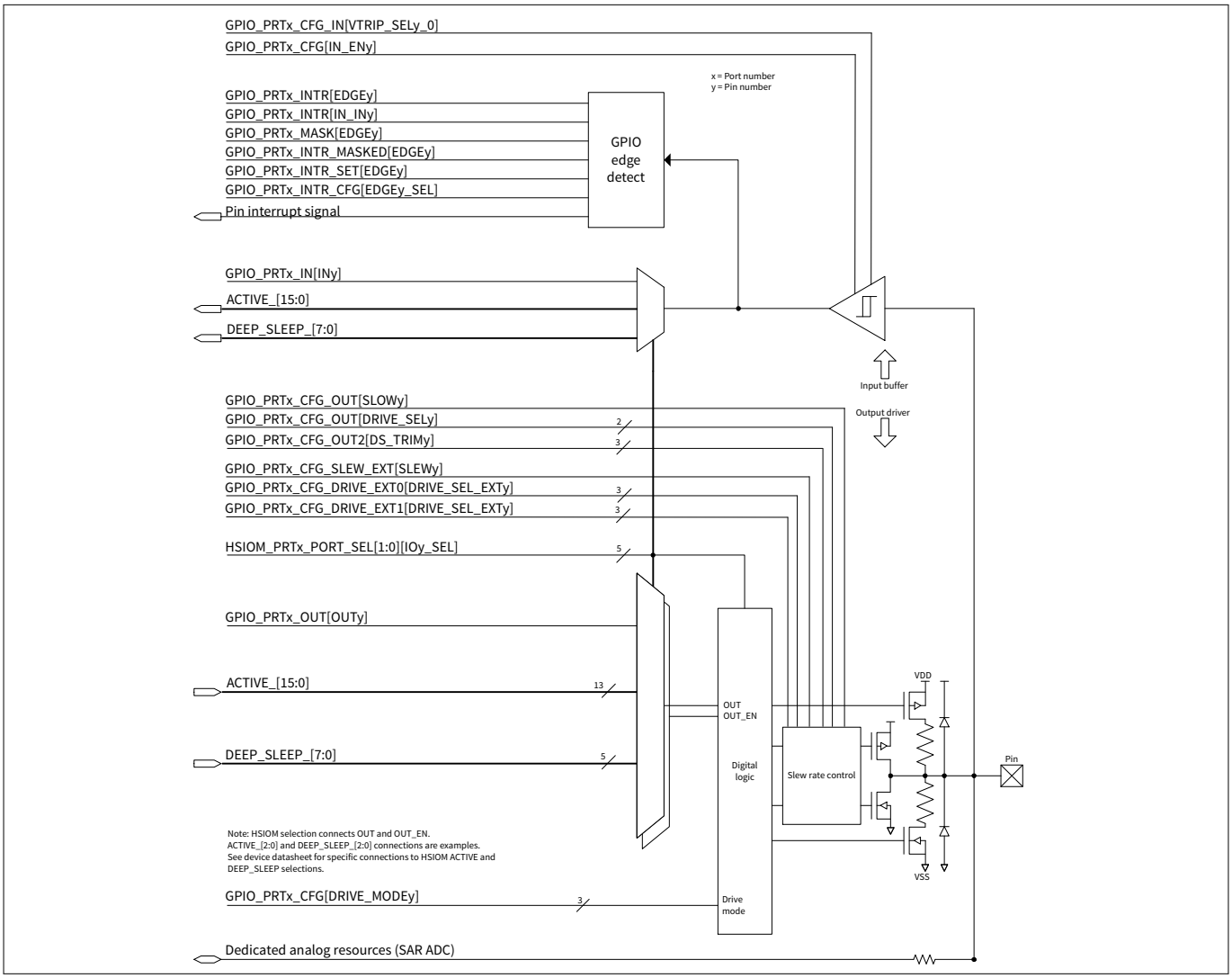


Figure 97 GPIO cell architecture

The GPIO component provides the I/O cell configuration information through registers. These registers are retained in deep sleep power mode, but are reset to their default value in hibernate power mode. To allow for hibernate Interrupt functionality, the I/O cells hold/freeze their configuration information when entering either deep sleep or hibernate power mode. As a result, the configuration signals can be routed in the active power domain.

If the HSIOM makes a functional connection to an I/O cell, the GPIO provides the configuration information. If the HSIOM makes a test connection (scan, PTM or JTAG) to an I/O cell, the GPIO configuration information is ignored, and the HSIOM provides the required configuration information.

I/O cell configuration includes information such as drive mode (pull-up/pull-down) and drive strength. Configuration information may be for a specific I/O pad: drive mode, drive strength, fast versus slow slew control transitioning, input buffer mode, and so on.

The I/Os in an I/O port are accessible by software to provide controllability of the I/O output signals and observability of the I/O input signals. Combined, controllability and observability provide software bit banging functionality.

Each I/O port has a GPIO_PRTx_OUT register field that specifies the data and data enable to be fed to the I/O cells output drivers. Each I/O cell has a dedicated 1-bit data and data enable field. Three additional registers are provided to ease/speedup software bit banging functionality. These registers allow software to manipulate individual I/O output signals without requiring a 'read modify-write' sequence. The GPIO_PRTx_OUT_SET register allows software to set specific data/data enable fields to '1', without affecting the signal level of the other data fields. The GPIO_PRTx_OUT_CLR register allows software to set specific data/data enable fields to '0', without affecting the signal level of the other data fields. The MMIO OUT_INV register allows software to invert the value of specific data/data enable fields, without affecting the signal level of the other data fields.

Note that the GPIO_PRTx_OUT_SET, GPIO_PRTx_OUT_CLR, and GPIO_PRTx_OUT_INV registers all operate on the OUT register data fields; no dedicated flip-flops are created for these registers.

Each I/O port has a GPIO_PRTx_IN (I/O cell input buffer state) register that reflects the I/O cells inputs. Note that the I/O cell inputs may be different from the data fed to the I/O cell output drives (GPIO_PRTx_OUT register).

The GPIO data input and data output/data output enable signals for I/O cells are on the HSIOM functional connections. The specific connection is under control of HSIOM register fields.

22.4 High speed I/O (HSIO)

These types of I/O ports are designed for high-speed operations supporting interfaces such as QSPI/OSPI, HYPERBUS™, SD standard, and Ethernet. Being optimized for high-speed operations, these ports do not offer slew rate control, deep sleep operation, and analog connections.

HSIOs can be used as the standard GPIO in active mode only. In low-power mode HSIO retains their state while the GPIO can toggle. Drive strength can be controlled using the GPIO_PRTx_CFG_OUT.DRIVE_SEL bits.

***Note**: Refer to the device datasheet for the availability of HSIO. Not all device support HSIO functionality.*

22.5 Digital input buffer

The digital input buffer provides a high-impedance buffer for the external digital input. The buffer is enabled or disabled by the GPIO_PRTx_CFG.IN_ENy bit (where 'x' is the port number and 'y' is the pin number).

The input buffer is connected to the HSIOM for routing to the CPU Port registers and selected peripherals. Writing to the HSIOM Port Select register (HSIOM_PRTx_PORT_SEL) selects the pin connection. See the device datasheet for the specific connections available for each pin. A port pin can be used as an input and output at the same time.

If a pin is only connected to an analog signal, the input buffer should be disabled to avoid crowbar currents.

Each pin's input buffer trip point and hysteresis are configurable for the following modes:

- CMOS + I^2C
- TTL
- Industrial

CMOS and TTL buffer modes are selected by the GPIO_PRTx_CFG_IN.VTRIP_SELy_0 bit. To set the mode to industrial use the GPIO_PRTxCFG_IN_AUTOLVL.VTRIP_SELy_1 bit to enable/disable the mode.

***Note**: Set the GPIO_PRTx_CFG_IN mode to CMOS if enabling the industrial mode. The trip levels of CMOS and industrial are shown in Figure 98.*

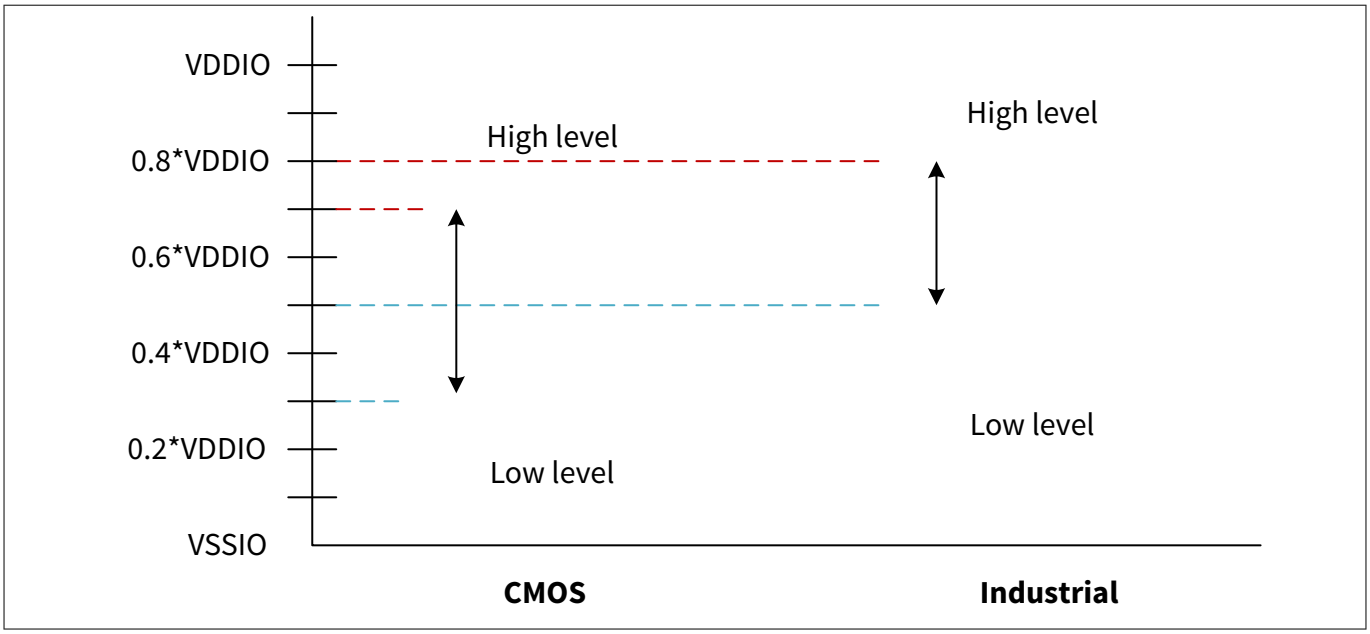


Figure 98 Input buffer mode's tripping levels

22.6 Digital output driver

Pins are driven by the digital output driver. It consists of circuitry to implement different drive modes and slew rate control for the digital output signals. The HSIOM selects the control source for the output driver. The two primary types of control sources are Port Configuration registers and Fixed-Function Digital peripherals. A particular HSIOM connection is selected by writing to the HSIOM Port Select register (HSIOM_PRTx_PORT_SEL).

Each GPIO pin has ESD diodes to clamp the pin voltage to the I/O supply source. Ensure that the voltage at the pin does not exceed the I/O supply voltage $V_{DDIO}/V_{DDD}/V_{DDA}$ or drop below $V_{SSIO}/V_{SSD}/V_{SSA}$. For the absolute maximum and minimum GPIO voltage, see the device datasheet.

The digital output driver can be enabled or disabled in hardware by the Output Data register (GPIO_PRTx_OUT) associated with the output pin. Peripherals other than GPIO port, directly control both the output and output-enable of the output buffer.

22.6.1 Drive modes

Each I/O is individually configurable to one of eight drive modes by the DRIVE_MODE[7:0] field of the Port Configuration register, GPIO_PRTx_CFG. Table 134 lists the drive modes. Drive mode '1' is reserved and should not be used in most designs. CPU register connections support seven discrete drive modes to maximize design flexibility. Fixed-function digital peripherals, such as SCB and TCPWM blocks, support modified functionality for the same seven drive modes compatible with fixed peripheral signaling. Figure 99 shows simplified output driver diagrams of the pin view for the CPU registers on each of the eight drive modes. Figure 100 is a simplified output driver diagram that shows the pin view for fixed-function-based peripherals for each of the eight drive modes.

Table 134 Drive mode settings

Drive mode	Value	GPIO port configuration register, AMUXBUS				Fixed-function digital peripheral			
		OUT_EN = 1		OUT_EN = 0		OUT_EN = 1		OUT_EN = 0	
		OUT = 1	OUT = 0	OUT = 1	OUT = 0	OUT = 1	OUT = 0	OUT = 1	OUT = 0
High impedance	0	High-Z	High-Z	High-Z	High-Z	High-Z	High-Z	High-Z	High-Z
Reserved	1	–	–	–	–	–	–	–	–
Resistive pull up	2	Weak 1	Strong 0	High-Z	High-Z	Strong 1	Strong 0	Weak 1	Weak 1
Resistive pull down	3	Strong 1	Weak 0	High-Z	High-Z	Strong 1	Strong 0	Weak 0	Weak 0
Open drain, drives low	4	High-Z	Strong 0	High-Z	High-Z	Strong 1[1)]	Strong 0	High-Z	High-Z
Open drain, drives high	5	Strong 1	High-Z	High-Z	High-Z	Strong 1	Strong 0	High-Z	High-Z
Strong	6	Strong 1	Strong 0	High-Z	High-Z	Strong 1	Strong 0	High-Z	High-Z
Resistive pull up or pull down	7	Weak 1	Weak 0	High-Z	High-Z	Strong 1	Strong 0	Weak 1	Weak 0

1) When there is I2C operation in the SCB block, output OUT=1 is 'High-Z' to keep open drain operation. Accordingly, the corresponding Drive Mode block diagram of the GPIO Port function is also valid.

***Note**: OUT_EN is not user configurable; its value is set according to the pin mode. For example, in GPIO mode OUT_EN = 1. See Table 141.*

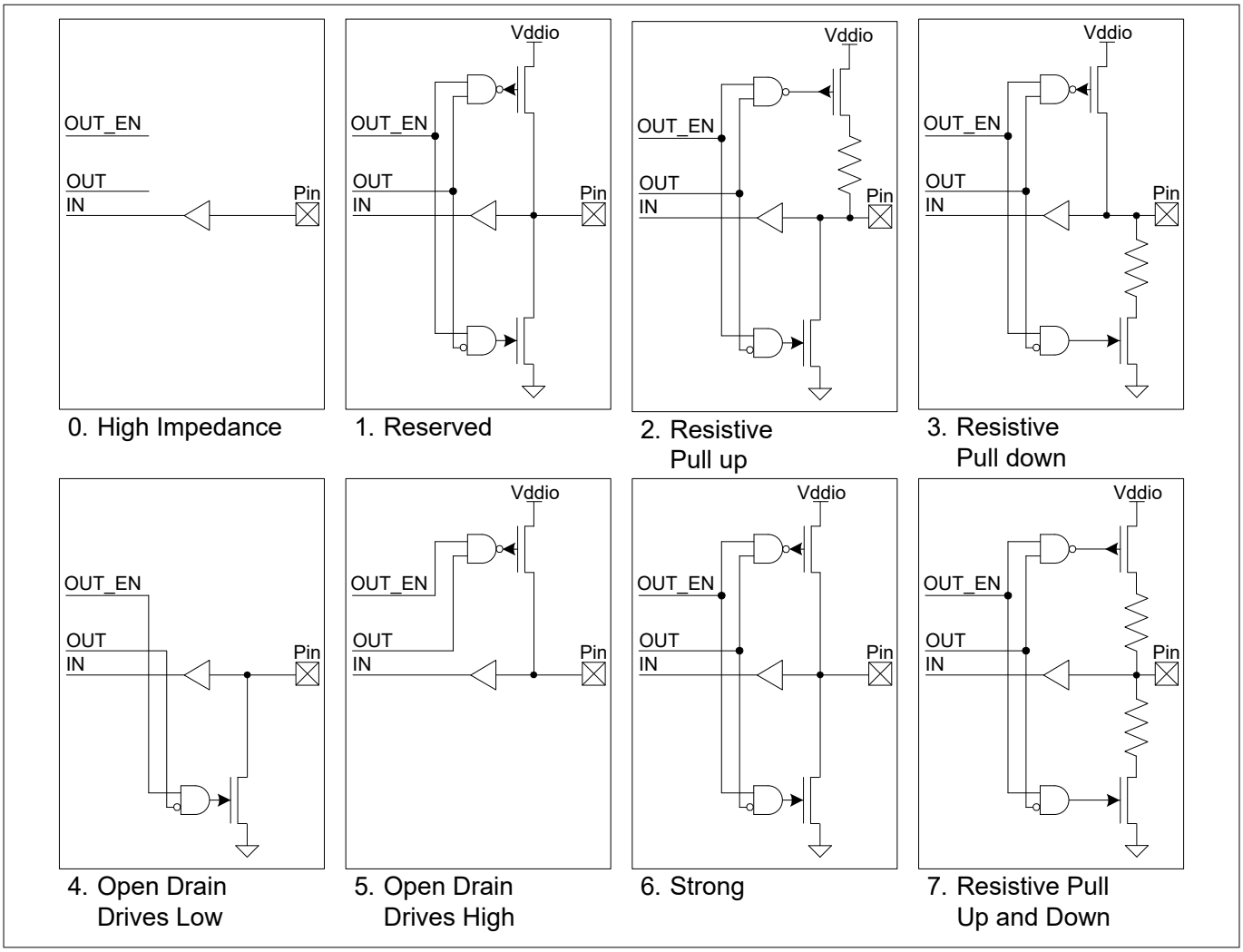


Figure 99 GPIO port, drive mode block diagram

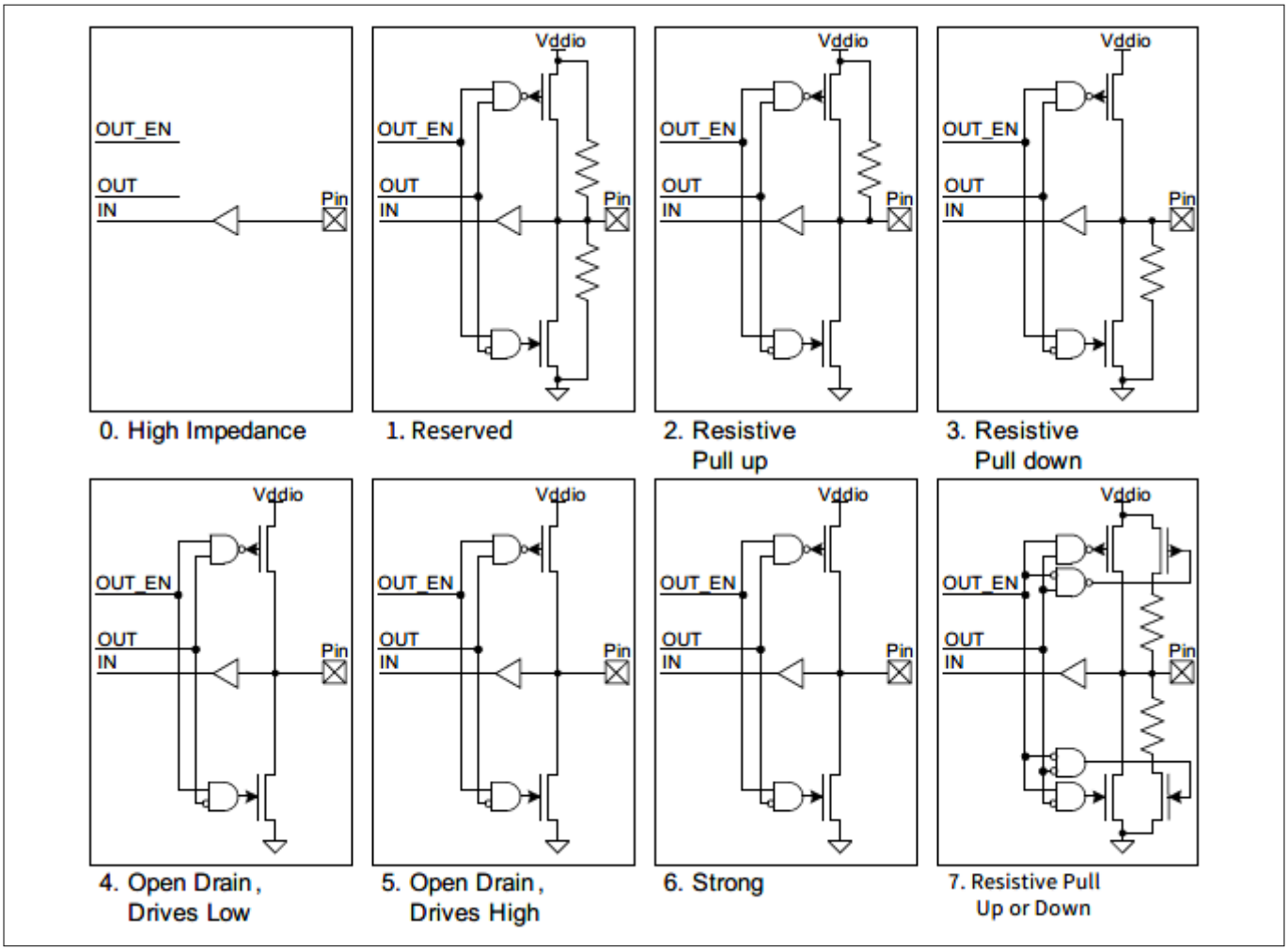


Figure 100 Fixed-function peripheral I/O drive mode block diagrams

- High-impedance

High-impedance mode is the standard high-impedance (High-Z) state recommended for analog and digital inputs. For digital signals, the input buffer is enabled; for analog signals, the input buffer is typically disabled to reduce crowbar current and leakage in low-power designs. To achieve the lowest device current, unused GPIOs must be configured to the high-impedance drive mode with input buffer disabled. High-impedance drive mode with input buffer disabled is also the default pin reset state.

- Resistive pull-up mode and resistive pull-down mode

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 XMC7000 with open drain drive lines. Resistive pull-up is used when the input is open drain low and resistive pull-down is used when the input is open drain HIGH.

- Open drain drives high and open drain drives low

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

- Strong drive

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

- Resistive pull-up or resistive pull-down

In the resistive pull-up or 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 useful when the pin is driven by other signals that may cause shorts.

22.6.2 Slew rate control

Some GPIO pins have fast and slow output slew rate options for the strong drivers configured using the SLOW bit of the Port Output Configuration register (GPIO_PRTx_CFG_OUT). By default, this bit is cleared and the port works in fast slew mode. This bit can be set if a slow slew rate is required. Slower slew rate results in reduced EMI and crosstalk and are recommended for low-frequency signals or signals without strict timing constraints.

When configured for fast slew rate, the drive strength can be set to one of four levels using the GPIO_PRTx_CFG_OUT.DRIVE_SELy. The drive strength field determines the active portion of the output drivers used and can affect the slew rate of output signals. Drive strength options are full drive strength (default), one-half strength, and one-quarter strength. Drive strength must be set to full drive strength when the slow slew rate bit (SLOW) is set.

Only an enhanced I/O port will support slew rate control; for standard ports slew rate can be controlled using drive strength. See the device datasheet for I/O ports with Enhanced functionality.

Table 135 Drive select for GPIO_STD

DRIVE_SEL[0:1]	Description
00	GPIO full drive, fast slew mode (default)
01	GPIO full drive strength
10	GPIO 1/2 drive strength
11	GPIO 1/4 drive strength

***Note**: See the device datasheet for the drive strength and AC specification.*

Table 136 Drive select for GPIO_ENH

DRIVE_SEL[0:1]	SLOW	Description
00	0	GPIO_ENH full drive, fast slew mode (default)
	1	GPIO_ENH full drive, slow slew mode
01	X (don’t care)	GPIO full drive strength
10	X (don’t care)	GPIO 1/2 drive strength
11	X (don’t care)	GPIO 1/4 drive strength

***Note**: See the device datasheet for the drive strength and AC specification.*

Table 137 Drive select for HSIO_STD

DRIVE_SEL[0:1]	Description
00	133 MHz at 20 pF, 50 MHz at 40 pF (default)
01	80 MHz at 15 pF, 32 MHz at 20 pF
10	15 MHz at 10 pF
11	10 MHz at 6 pF

***Note**: See the device datasheet for the drive strength and AC specification.*

Table 138 Drive select for HSIO_ENH

DS_TRIM[0:2]	Description
000	50 Ω, 3.3 V single memory (default)
001	15 Ω, 1.8 V memory (xSPI-400)
010	30 Ω, 3.3 V memory (xSPI-266 / xSPI-200, SPI-SDR / SPI-DDR)
011	30 Ω, 1.8 V memory
100	40 Ω, 1.8 V memory
101	20 Ω, 1.8 V memory (SPI-333, xSPI-266, xSPI.200, SPI-SDR, SPI-DDR)
110	25 Ω, 1.8 V memory
111	15 Ω, 1.8 V memory (xSPI-400)

***Note**: See the device datasheet for the drive strength and AC specification.*

Table 139 Drive select for HSIO_STD_LN

DRIVE_SEL_EXT[2:0]	SLEW	Description
000	0	133MHz at 15pF, xSPI-266 mode (default)
	1	125MHz at 15pF
001	0	100MHz at 15pF, xSPI-200 mode
	1	90MHz at 15pF
010	0	80MHz at 15pF, Graphics
	1	60MHz at 15pF
011	0	64MHz at 15pF
	1	50MHz at 15pF, Ethernet
100	Don't Care	12MHz at 20pF, 25MHz at 10pF, SPI
101 - 111	Don't Care	reserved, N/A

***Note**: See the device datasheet for the drive strength and AC specification.*

Table 140 Drive Select for GPIO_SMC

DRIVE_SEL[0:1]	SLOW	Description
00	0	GPIO full drive, fast slew mode (default)
	1	SMC full drive, high current slow slew mode
01	Don't care	GPIO full drive strength
10	Don't care	GPIO 1/2 drive strength
11	Don't care	GPIO 1/4 drive strength

***Note**: See the device datasheet for the drive strength and AC specification.*

22.7 High-speed I/O matrix

The high-speed I/O matrix (HSIOM) is a set of high-speed multiplexers that route internal CPU and peripheral signals to and from GPIOs. HSIOM allows GPIOs to be shared with multiple functions and multiplexes the pin connection to a particular peripheral selected by the user. The HSIOM_PRTx_PORT_SEL registers allow a single selection from up to 32 different connections to each pin as listed in Table 141.

Table 141 HSIOM connections

SELy_SEL	Name	Digital driver signal source		Digital input signal destination	Description
		OUT	OUT_EN		
0	GPIO	OUT register	1	IN register	GPIO_PRTx_OUT register controls "out"
1	Reserved	–	–	–	–
2	Reserved	–	–	–	–
3	Reserved	–	–	–	–
4	Reserved	–	–	–	–
5	Reserved	–	–	–	–
6	Reserved	–	–	–	–
7	Reserved	–	–	–	–
8	ACT_0	Active source OUT	Active source OUT_EN	Active source IN	Active functionality 0 - See device datasheet for specific pin connectivity
9	ACT_1	Active source OUT	Active source OUT_EN	Active source IN	Active functionality 1 - See device datasheet for specific pin connectivity
10	ACT_2	Active source OUT	Active source OUT_EN	Active source IN	Active functionality 2 - See device datasheet for specific pin connectivity
11	ACT_3	Active source OUT	Active source OUT_EN	Active source IN	Active functionality 3 - See device datasheet for specific pin connectivity

(table continues...)

Table 141 (continued) HSIOM connections

SELy_SEL	Name	Digital driver signal source		Digital input signal destination	Description
		OUT	OUT_EN		
12	DS_0	Deep sleep source OUT	Deep sleep source OUT_EN	Deep sleep IN	Deep sleep functionality 0 - See device datasheet for specific pin connectivity
13	DS_1	Deep sleep source OUT	Deep sleep source OUT_EN	Deep sleep IN	Deep sleep functionality 1 - See device datasheet for specific pin connectivity
14	DS_2	Deep sleep source OUT	Deep sleep source OUT_EN	Deep sleep IN	Deep sleep functionality 2 - See device datasheet for specific pin connectivity
15	DS_3	Deep sleep source OUT	Deep sleep Source OUT_EN	Deep sleep IN	Deep sleep functionality 3 - See device datasheet for specific pin connectivity
16	ACT_4	Active source OUT	Active source OUT_EN	Active source IN	Active functionality 4 - See device datasheet for specific pin connectivity
17	ACT_5	Active source OUT	Active source OUT_EN	Active source IN	Active functionality 5 - See device datasheet for specific pin connectivity
18	ACT_6	Active source OUT	Active source OUT_EN	Active source IN	Active functionality 6 - See device datasheet for specific pin connectivity
19	ACT_7	Active source OUT	Active source OUT_EN	Active source IN	Active functionality 7 - See device datasheet for specific pin connectivity
20	ACT_8	Active source OUT	Active source OUT_EN	Active source IN	Active functionality 8 - See device datasheet for specific pin connectivity
21	ACT_9	Active source OUT	Active source OUT_EN	Active source IN	Active functionality 9 - See device datasheet for specific pin connectivity
22	ACT_10	Active source OUT	Active source OUT_EN	Active source IN	Active functionality 10 - See device datasheet for specific pin connectivity
23	ACT_11	Active source OUT	Active source OUT_EN	Active source IN	Active functionality 11 - See device datasheet for specific pin connectivity
24	ACT_12	Active source OUT	Active source OUT_EN	Active source IN	Active functionality 12 - See device datasheet for specific pin connectivity

(table continues...)

Table 141 (continued) HSIOM connections

SELy_SEL	Name	Digital driver signal source		Digital input signal destination	Description
		OUT	OUT_EN		
25	ACT_13	Active source OUT	Active source OUT_EN	Active source IN	Active functionality 13 - See device datasheet for specific pin connectivity
26	ACT_14	Active source OUT	Active source OUT_EN	Active source IN	Active functionality 14 - See device datasheet for specific pin connectivity
27	ACT_15	Active source OUT	Active source OUT_EN	Active source IN	Active functionality 15 - See device datasheet for specific pin connectivity
28	DS_4	Deep sleep source OUT	Deep sleep source OUT_EN	Deep sleep IN	Deep sleep functionality 4 - See device datasheet for specific pin connectivity
29	DS_5	Deep sleep source OUT	Deep sleep source OUT_EN	Deep sleep IN	Deep sleep functionality 5 - See device datasheet for specific pin connectivity
30	DS_6	Deep sleep source OUT	Deep sleep source OUT_EN	Deep sleep IN	Deep sleep functionality 6 - See device datasheet for specific pin connectivity
31	DS_7	Deep sleep source OUT	Deep sleep source OUT_EN	Deep sleep IN	Deep sleep functionality 7 - See device datasheet for specific pin connectivity

***Notes**:*
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. If the JTAG input pin is configured to the SWJ_TRSTN mode upon reset (refer to the related device datasheet for the pin number), change the mode of the pin from SWJ_TRSTN to GPIO according to the following sequence:

1. *HSIOM_PRTx_PORT_SEL = 0 (GPIO)*
2. *GPIO_PRTx_CFG = 0*

22.8 I/O state on power up

During power up, all the GPIOs are in high-impedance analog state and the input buffers are disabled. During runtime, GPIOs can be configured by writing to the associated registers. Note that the pins supporting debug access port (DAP) connections (SWD lines) are always enabled as SWD lines during power up. The DAP connection does not provide pull-up or pull-down resistors; therefore, if left floating some crowbar current is possible. The DAP connection can be disabled or reconfigured for general-purpose use through the HSIOM only after the device boots and starts executing code.

22.9 Behavior in low-power modes

To allow for deep sleep interrupt and hibernate wake up functionality, the GPIOs hold/freeze their configuration information when entering either deep sleep or hibernate power mode. As a result, the configuration signals can be routed in the active power domain. Table 142 shows the status of GPIOs in low-power modes.

Table 142 GPIO in low-power modes

Low-power mode	Status
Sleep	• Standard GPIO pins are active and can be driven by most peripherals such as TCPWM and SCB, which can operate in sleep mode. • Inputs buffers are active; thus an interrupt on any I/O can be used to wake the CPU.
Deep sleep	• GPIO pins, connected to deep sleep domain peripherals, are functional. All other pins are hold/frozen and will maintain the last output driver state and configuration. • Pin interrupts are functional on all I/Os and can be used to wake the device.
Hibernate	• Pin output states and configuration are latched and remain in the hold/frozen state. • Pin interrupts are functional on only select I/Os and can be used to wake the device. See the device datasheet for specific hibernate pin connectivity.

22.10 Interrupt

All port pins have the capability to generate interrupts. There are two routing possibilities for pin signals to generate interrupts, as shown in Figure 101.

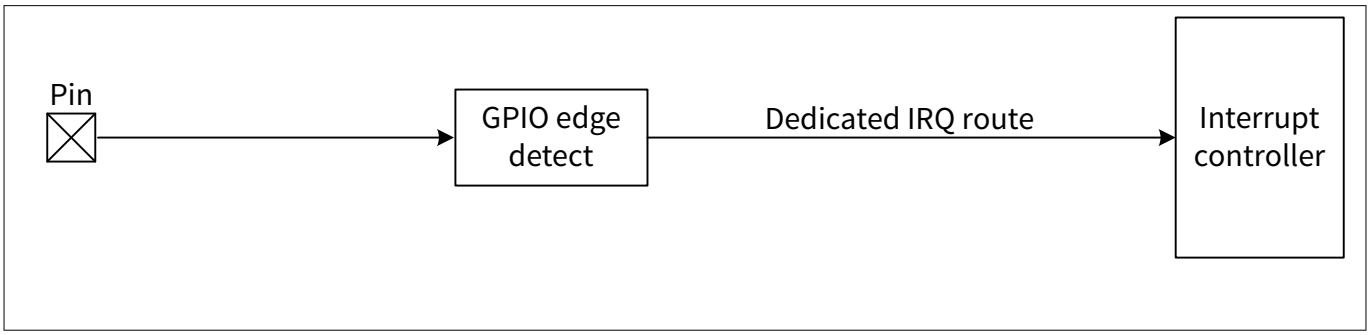


Figure 101 Interrupt signal routing

Pin signal through the ‘GPIO edge detect’ block with direct connection to the CPU interrupt controller. Interrupt generation is independent of HSIOM configuration, consider disabling it (GPIO_PRTx_INTR_CFG.EDGEy_SEL = 00) to avoid unwanted GPIO interrupt. Figure 102 shows the block diagram of the GPIO Edge Detect block.

Each GPIO pin interrupt can be activated either on GPIO pin rising or GPIO pin falling edge changes based on GPIO_PRTx_INTR_CFG register selection. Each of the up to eight GPIO pads in an I/O port has interrupt cause fields that are set to ‘1’ on a GPIO pin input signal rising and falling edge respectively (GPIO_PRTx_INTR register). The GPIO_PRTx_INTR_MASK register of the I/O port specifies which interrupt is propagated to its interrupt output. This propagated GPIO pin interrupt is a deep sleep functionality interrupt and this allows for a GPIO input signal change to wake up the CPU from deep sleep power mode after combined with other given GPIO port “i” pin interrupts as shown in Figure 102 to form a port interrupt interrupts_gpio[i].

In addition, an register field specifies one specific I/O input signal that is routed to a 50-ns glitch filter (GPIO_PRTx_INTR_CFG.FLT_SEL register). The glitch filter output has a dedicated detection circuitry and has dedicated detection control fields (like GPIO_PRTx_INTR.FLT_EDGE). Each I/O port has a dedicated interrupt associated to it (interrupts_gpio[i] for I/O port i). Figure 101 illustrates the interrupt functionality. Figure 102 shows the GPIO edge detect block architecture.

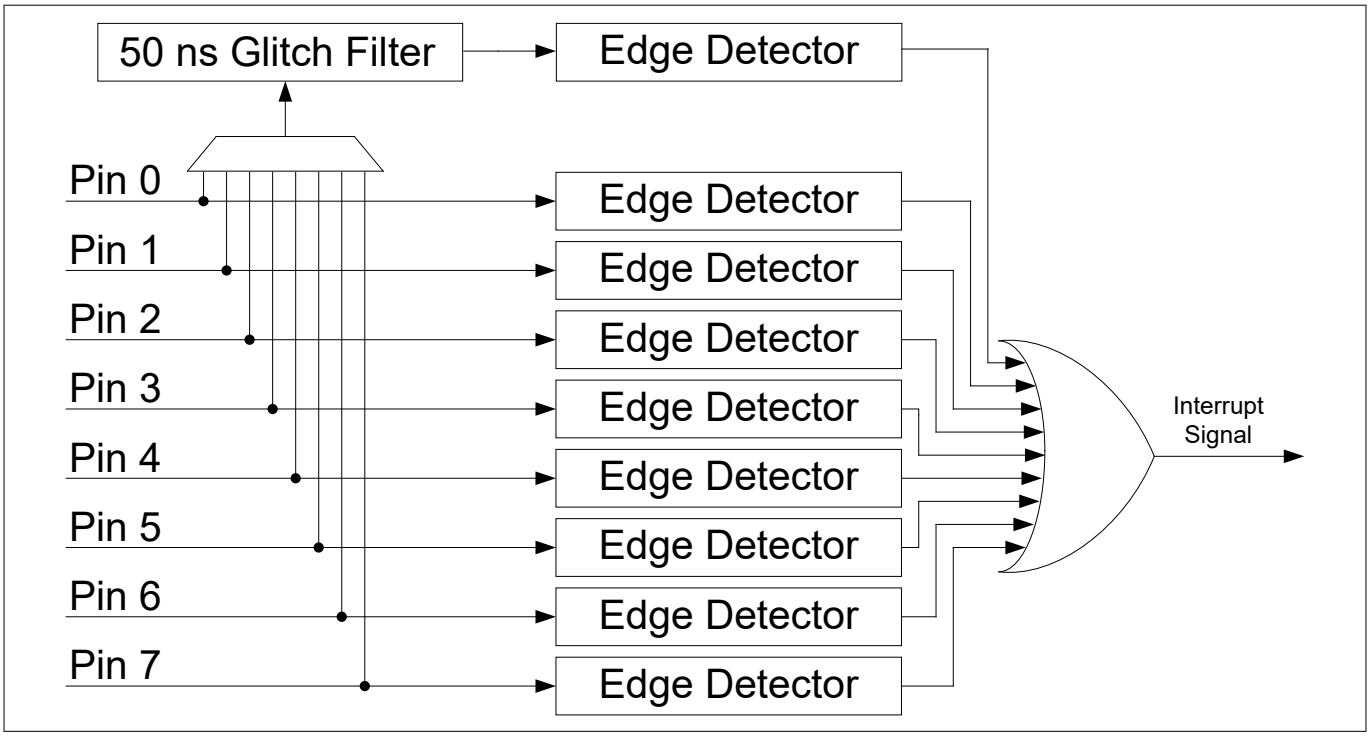


Figure 102 GPIO edge detect block architecture

The software ISR can read the 8+1 interrupt cause fields to determine the I/O or glitch filter signal(s) that caused the interrupt activation. The ISR needs to clear the interrupt cause fields to deactivate the interrupt.

An edge detector is present at each pin. It is capable of detecting rising edge, falling edge, and both edges without any reconfiguration. The edge detector is configured by writing into the GPIO_PRTx_INTR_CFG.EDGEy_SEL field, as shown in Table 143.

Table 143 Edge detector configuration

EDGE_SEL	Configuration
00	Interrupt is disabled
01	Interrupt on rising edge
10	Interrupt on falling edge
11	Interrupt on both edges

Writing '1' to the corresponding status bit clears the pin edge state. It is important to clear the edge state status bit; otherwise, an interrupt can occur repeatedly for a single trigger or respond only once for multiple triggers, which is explained later in this section. When the Port Interrupt Control Status register is read at the same time an edge is occurring on the corresponding port, it can result in the edge 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.

Firmware and the debug interface are able to trigger a hardware interrupt from any pin by setting the corresponding bit in the GPIO_PRTx_INTR_SET register.

In addition to the pins, each port provides a glitch filter connected to its own edge detector. This filter can be driven by one of the pins of a port. The selection of the driving pin is done by writing to the GPIO_PRTx_INTR_CFG.FLT_SEL field as shown in Table 144.

Table 144 Glitch filter input selection

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

When a port pin edge occurs, it is required to know which pin caused the edge. This is done by reading the port Interrupt Status register, GPIO_PRTx_INTR. This register includes both the latched information on which pin detected an edge and the current pin status. This allows the CPU to read both information in a single read operation. This register has an additional use - to clear the latched edge state.

The GPIO_PRTx_INTR_MASK register enables forwarding of the GPIO_PRTx_INTR edge detect signal to the interrupt controller when a '1' is written to a pin's corresponding bit field. The GPIO_PRTx_INTR_MASKED register can then be read to determine the specific pin that generated the interrupt signal forwarded to the interrupt controller. The masked edge detector outputs of a port are then ORed together and routed to the interrupt controller (NVIC in the CPU subsystem). Thus, there is only one interrupt vector per port.

The masked and ORed edge detector block output is routed to the interrupt source multiplexer (see the Interrupts for details), which gives an option of level and rising edge detection. If the Level option is selected, an interrupt is triggered repeatedly as long as the port interrupt status register bit is set. If the rising edge detect option is selected, an interrupt is triggered only once if the Port Interrupt Status register is not cleared. Thus, it is important to clear the interrupt status bit if the edge detect block is used.

There is a dedicated interrupt vector for each port when the interrupt signal is routed through the fixed-function route.

All the port interrupt vectors are also ORed together into a single interrupt vector for use on devices with more ports than there are interrupt vectors available. To determine the port that triggered the interrupt, the GPIO_INTR_CAUSEx registers can be read. A '1' present in a bit location indicates that the corresponding port has a pending interrupt. The indicated GPIO_PRTx_INTR register can then be read to determine the pin source.

The GPIO_VDD_ACTIVE register provides the capability to read the state of the external power supplies. It indicates the absence or presence of VDDIO supplies, VDDA and VDDD. In addition, power supply interrupts can be configured to generate interrupts on supply state change, The GPIO_VDD_INTR_MASK register is used to mask/enable the forwarding of interrupt to the CPUs. The status of interrupt can be checked with the GPIO_VDD_INTR register.

22.11 Peripheral connections

22.11.1 Firmware-controlled GPIO

For standard firmware-controlled GPIO using registers, the GPIO mode must be selected in the HSIOM_PRTx_PORT_SEL register.

The GPIO_PRTx_OUT register is used to read and write the output buffer state for GPIOs. A write operation to this register changes the GPIO's output driver state to the written value. A read operation reflects the output data written to this register and the resulting output driver state. It does not return the current logic level

present on GPIO pins, which may be different. Using the GPIO_PRTx_OUT register, read-modify-write sequences can be safely performed on a port that has both input and output GPIOs.

In addition to the Data register, three other registers –GPIO_PRTx_OUT_SET, GPIO_PRTx_OUT_CLR, and GPIO_PRTx_OUT_INV – are provided to set, clear, and invert the output data respectively on specific pins in a port without affecting other pins. This avoids the need for read-modify- write operations in most use cases. Writing '1' to these register bit fields will set, clear, or invert the respective pin; writing '0' will have no effect on the pin state.

GPIO_PRTx_IN is the Port I/O Pad register that provides the actual logic level present on the GPIO pin when read. Writes to this register have no effect.

22.11.2 Analog I/O

Analog resources, such as SAR ADC, which require low-impedance routing paths have dedicated pins. Dedicated analog pins provide direct connections to specific analog blocks. They help improve performance and should be given priority over other pins when using these analog resources. See the device datasheet for details on these dedicated pins of XMC7000.

To configure a GPIO as a dedicated analog I/O, it should be configured in high-impedance analog mode (see Table 134) with input buffer disabled and the respective connection should be enabled via registers in the specific analog resource.

While it is preferred that analog pins disable the input buffer, it is acceptable to enable the input buffer if simultaneous analog and digital input features are required.

22.11.3 Serial communication block (SCB)

SCB can be configured for UART, I^2C, and SPI communication protocols. The SCB has dedicated connections to pins through the HSIOM. See the device datasheet for details on the dedicated pin connections. When the SCB I^2C, UART, or SPI modes are used, the SCB controls the interface pins digital output state and output enable. In principle, all the peripherals with output connected to HSIOM also control the output enable for the pin. For details on the recommended interface pin drive modes and controlling signals, refer to the Serial communications block (SCB).

22.12 Smart I/O

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

- Integrate board-level Boolean logic functionality into a port
- Ability to pre-process HSIOM input signals from the GPIO port pins
- Ability to post-process HSIOM output signals to the GPIO port pins
- Support in all device power modes
- Integrate closely to the I/O pads, providing shortest signal paths with programmability

22.12.1 Overview

The smart I/O block is positioned in the signal path between the HSIOM and the I/O port. The HSIOM multiplexes the output signals from fixed-function peripherals and CPU to a specific port pin and vice-versa. The smart I/O block is placed on this signal path, acting as a bridge that can process signals between port pins and HSIOM, as shown in Figure 103.

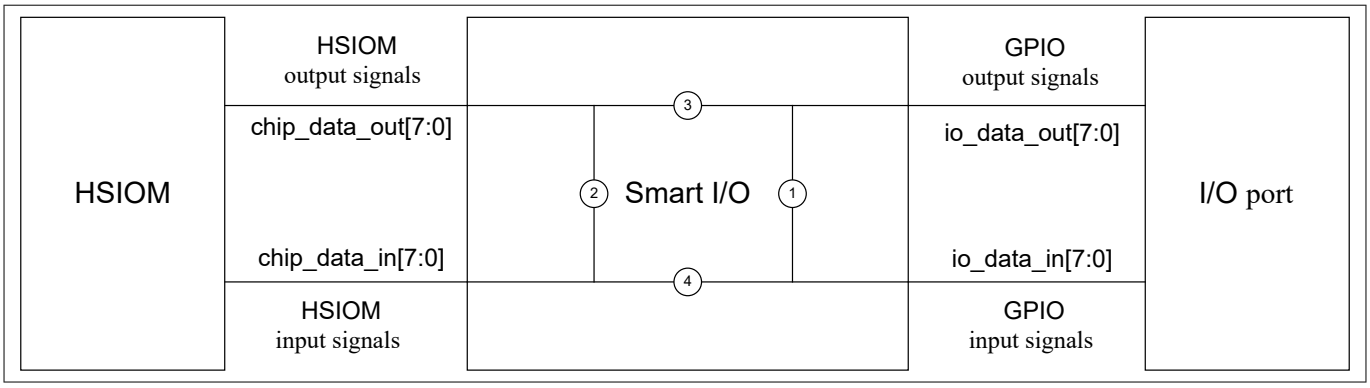


Figure 103 Smart I/O interface

The signal paths supported through the smart I/O block as shown in Figure 103 are as follows:

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

The following sections discuss the Smart I/O block components, routing, and configuration in detail. In these sections, the GPIO signal (io_data_in) refers to the input signal from the I/O port; device or chip (chip_data) signals refer to the output signal from HSIOM. smartio_data is the output of Smart I/O interface depending on the configuration of the blocks.

22.12.2 Block components

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

- Clock/reset
- Synchronizers
- Three-input lookup table (LUT)
- Data unit

22.12.2.1 Clock and reset

The clock and reset component selects the smart I/O block's clock (clk_block) and reset signal (rst_block_n). A single clock and reset signal is used for all components in the block. The clock and reset sources are determined by the SMARTIO_PRTx_CTL.CLOCK_SRC field. The selected clock is used for the synchronous logic in the block components, which includes the I/O input synchronizers, LUT, and data unit components. The selected reset is used to asynchronously reset the synchronous logic in the LUT and data unit components.

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

The following clock sources are available for selection:

- GPIO input signals io_data_in[7:0]. These clock sources have no associated reset.
- HSIOM output signals chip_data[7:0]. These clock sources have no associated reset.
- The smart I/O clock (PCLK_SMARTIOx_CLOCK). This is derived from the system clock (CLK_SYS/CLK_HF) using a peripheral clock divider. See the Clocking system for details on peripheral clock dividers. This clock is available only in active and sleep power modes. The clock can have one out of two associated resets: rst_sys_act_n and rst_sys_dpslp_n. These resets determine in which system power modes the block

synchronous state is reset; for example, rst_sys_act_n is intended for smart I/O synchronous functionality in the active power mode and reset is activated in the deep sleep power mode.

- The low-frequency (40 kHz) system clock (clk_lf). This clock is available in deep sleep power mode. This clock has an associated reset, rst_lf_dpslp_n.

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

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

Table 145 Clock and Reset register Control

Register[BIT_POS]	Bit name	Description
SMARTIO_PRTx_CTL[12:8]	CLK_SRC[4:0]	Clock (clk_block)/reset (rst_block_n) source selection: 0: io_data_in[0]/'1' ... 7: io_data_in[7]/'1' 8: chip_data[0]/'1' ... 15: chip_data[7]/'1' 16: clk_smartio/rst_sys_act_n; asserts reset in any power mode other than active; that is, smart I/O is active only in active power mode with clock from the peripheral divider. 17: clk_smartio/rst_sys_dpslp_n. Smart I/O is active in all power modes with clock from the peripheral divider. However, the clock will not be active in deep sleep power mode. 19: clk_lf/rst_lf_dpslp_n. Smart I/O is active in all power modes with clock from ILO. 20-30: Clock source is a constant '0'. 31: CLK_SYS/'1'. This selection is not intended for clk_sys operation.

22.12.2.2 Synchronizer

Each GPIO input signal and device input signal (HSIOM input) can be used either asynchronously or synchronously. To use the signals synchronously, a double flip-flop synchronizer, as shown in Figure 104, is placed on both these signal paths to synchronize the signal to the smart I/O clock (clk_block). The synchronization for each pin/input is enabled or disabled by setting or clearing the IO_SYNC_EN[i] bit field for GPIO input signal and CHIP_SYNC_EN[i] for HSIOM signal in the SMARTIO_PRT0_SYNC_CTL register, where 'i' is the pin number.

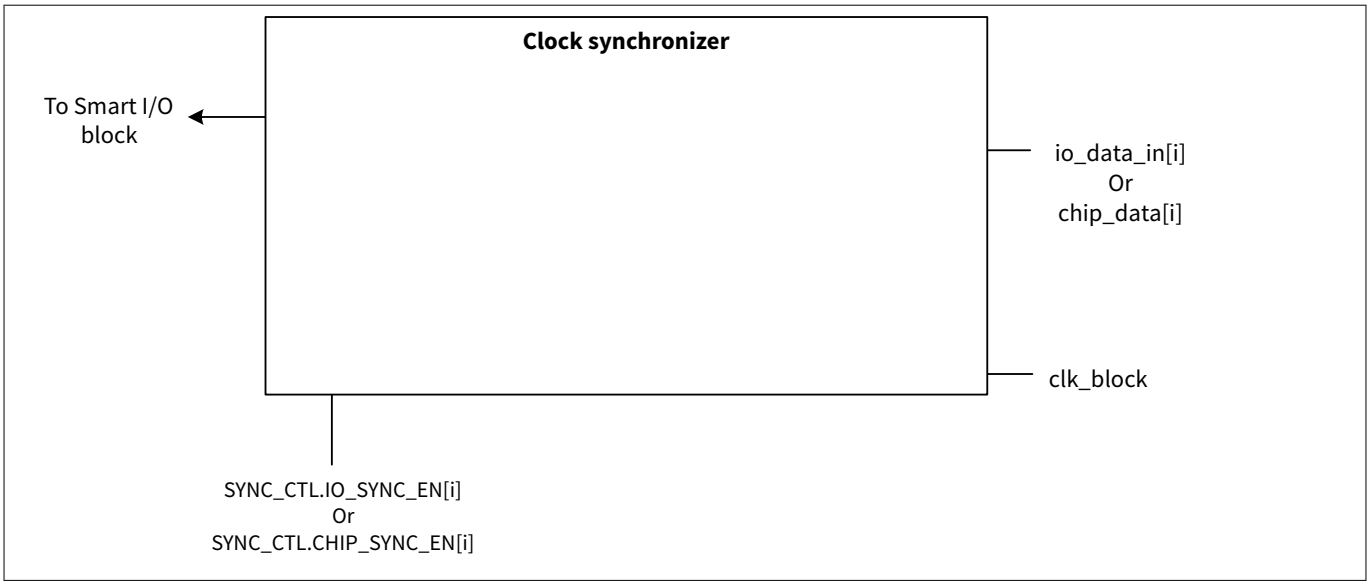


Figure 104 Smart I/O clock synchronizer

22.12.2.3 LUT

Each smart I/O block contains eight lookup table (LUT) components. The LUT component consists of a three-input LUT and a flip-flop. Each LUT block takes three input signals and generates an output based on the configuration set in the SMARTIO_PRTx_LUT_CTLy register ('y' denotes the LUT number). For each LUT, the configuration is determined by an 8-bit lookup vector LUT[7:0] and a 2-bit opcode OPC[1:0] in the SMARTIO_PRTx_LUT_CTLy register. The 8-bit vector is used as a lookup table for the three input signals. The 2-bit opcode determines the usage of the flip-flop. The LUT configuration for different opcode is shown in Figure 105.

SMARTIO_PRTx_LUT_SELy registers select the three input signals (tr0_in, tr1_in, and tr2_in) going into each LUT.

The input can come from the following sources:

- Data unit output
- Other LUT output signals (tr_out)
- HSIOM output signals (chip_data[7:0])
- GPIO input signals (io_data_in[7:0])

SMARTIO_PRTx_LUT_SELy.LUT_TR0_SEL register selects the tr0_in signal for the y^{th} LUT. Similarly, SMARTIO_PRTx_LUT_SELy.LUT_TR1_SEL bits and SMARTIO_PRTx_LUT_SELy.LUT_TR2_SEL bits select the tr1_in and tr2_in signals respectively. See Table 146 for details.

Table 146 LUT register control

Register[BIT_POS]	Bit name	Description
SMARTIO_PRTx_LUT_CTLy[7:0]	LUT[7:0]	LUT configuration. Depending on the LUT opcode (LUT_OPC), internal state, and LUT input signals tr0_in, tr1_in, and tr2_in, the LUT configuration is used to determine the LUT output signal and the next sequential state.
SMARTIO_PRTx_LUT_CTLy[9:8]	LUT_OPC[1:0]	LUT opcode specifies the LUT operation as illustrated in Figure 105.

(table continues...)

Table 146 (continued) LUT register control

Register[BIT_POS]	Bit name	Description
SMARTIO_PRTx_LUT_SELy[3:0]	LUT_TR0_SEL[3:0]	LUT input signal tr0_in source selection: 0: Data unit output 1: LUT 1 output 2: LUT 2 output 3: LUT 3 output 4: LUT 4 output 5: LUT 5 output 6: LUT 6 output 7: LUT 7 output 8: chip_data[0] (for LUTs 0, 1, 2, 3); chip_data[4] (for LUTs 4, 5, 6, 7) 9: chip_data[1] (for LUTs 0, 1, 2, 3); chip_data[5] (for LUTs 4, 5, 6, 7) 10: chip_data[2] (for LUTs 0, 1, 2, 3); chip_data[6] (for LUTs 4, 5, 6, 7) 11: chip_data[3] (for LUTs 0, 1, 2, 3); chip_data[7] (for LUTs 4, 5, 6, 7) 12: io_data_in[0] (for LUTs 0, 1, 2, 3); io_data_in[4] (for LUTs 4, 5, 6, 7) 13: io_data_in[1] (for LUTs 0, 1, 2, 3); io_data_in[5] (for LUTs 4, 5, 6, 7) 14: io_data_in[2] (for LUTs 0, 1, 2, 3); io_data_in[6] (for LUTs 4, 5, 6, 7) 15: io_data_in[3] (for LUTs 0, 1, 2, 3); io_data_in[7] (for LUTs 4, 5, 6, 7)

(table continues...)

Table 146 (continued) LUT register control

Register[BIT_POS]	Bit name	Description
SMARTIO_PRTx_LUT_SELy[11:8]	LUT_TR1_SEL[3:0]	LUT input signal tr1_in source selection: 0: LUT 0 output 1: LUT 1 output 2: LUT 2 output 3: LUT 3 output 4: LUT 4 output 5: LUT 5 output 6: LUT 6 output 7: LUT 7 output 8: chip_data[0] (for LUTs 0, 1, 2, 3); chip_data[4] (for LUTs 4, 5, 6, 7) 9: chip_data[1] (for LUTs 0, 1, 2, 3); chip_data[5] (for LUTs 4, 5, 6, 7) 10: chip_data[2] (for LUTs 0, 1, 2, 3); chip_data[6] (for LUTs 4, 5, 6, 7) 11: chip_data[3] (for LUTs 0, 1, 2, 3); chip_data[7] (for LUTs 4, 5, 6, 7) 12: io_data_in[0] (for LUTs 0, 1, 2, 3); io_data_in[4] (for LUTs 4, 5, 6, 7) 13: io_data_in[1] (for LUTs 0, 1, 2, 3); io_data_in[5] (for LUTs 4, 5, 6, 7) 14: io_data_in[2] (for LUTs 0, 1, 2, 3); io_data_in[6] (for LUTs 4, 5, 6, 7) 15: io_data_in[3] (for LUTs 0, 1, 2, 3); io_data_in[7] (for LUTs 4, 5, 6, 7)
SMARTIO_PRTx_LUT_SELy[19:16]	LUT_TR2_SEL[3:0]	LUT input signal tr2_in source selection. Encoding is the same as for LUT_TR1_SEL.

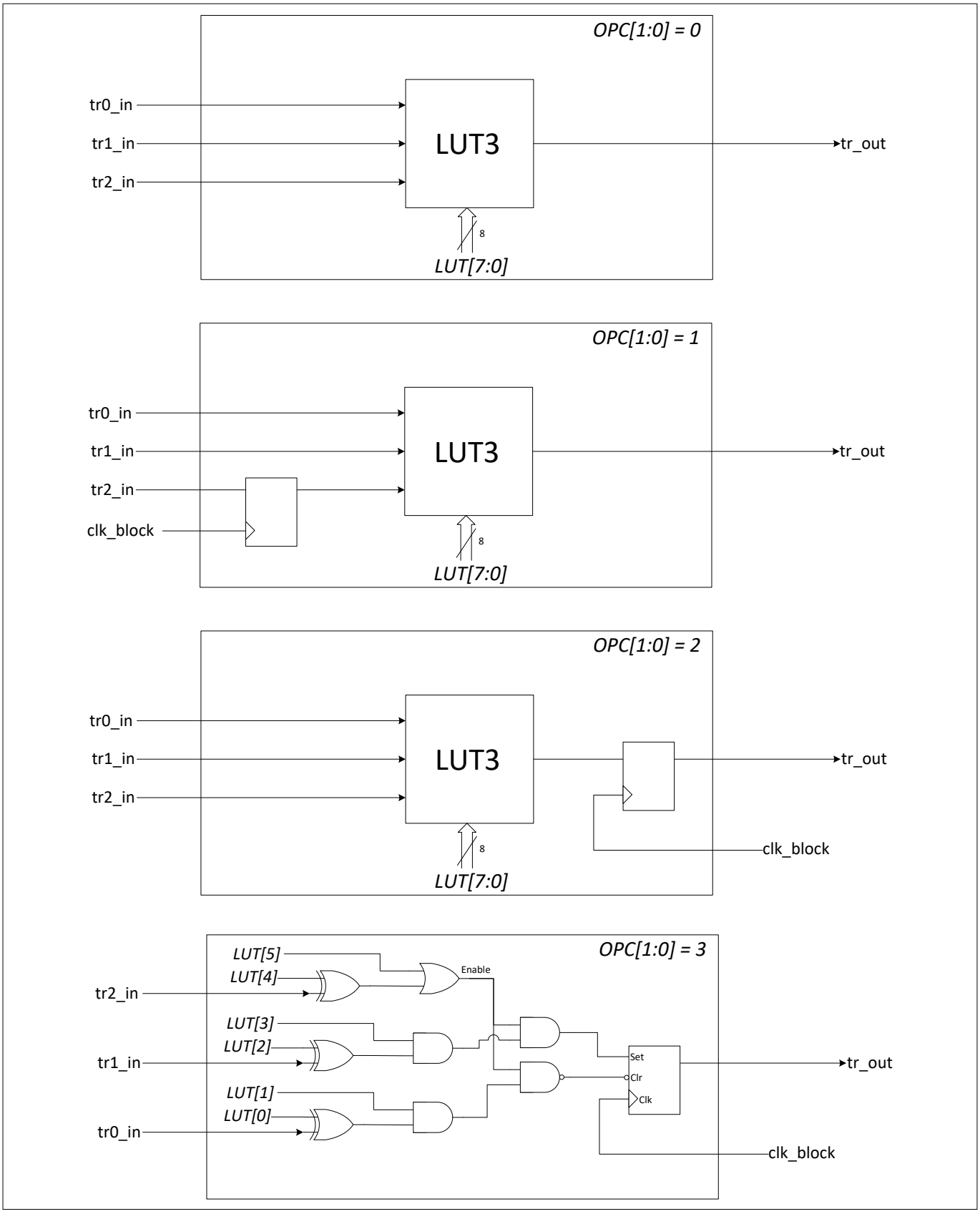


Figure 105 Smart I/O LUT configuration

22.12.2.4 Data unit

Each smart I/O block includes a data unit (DU) component. The data unit consists of an 8-bit datapath. It is capable of performing simple increment, decrement, increment/decrement, shift, and AND/OR operations. The operation performed by the DU is selected using a 4-bit opcode SMARTIO_PRTx_DU_CTL.DU_OPC field.

The data unit component supports up to three input trigger signals (tr0_in, tr1_in, tr2_in) similar to the LUT component. These signals are used to initiate an operation defined by the DU opcode. In addition, the data unit also includes two 8-bit data inputs (data0_in[7:0] and data1_in[7:0]) that are used to initialize the 8-bit internal state (data[7:0]) or to provide a reference. The 8-bit data input source is configured as:

- Constant '0x00'
- io_data_in[7:0]
- chip_data[7:0]
- SMARTIO_PRTx_DATA.DATA field

The trigger signals are selected using the SMARTIO_PRTx_DU_SEL.DU_TRy_SEL field. The SMARTIO_PRTx_DU_SEL.DU_DATAy_SEL field select the 8-bit input data source. The size of the DU (number of bits used by the datapath) is defined by the SMARTIO_PRTx_DU_CTL.DU_SIZE field. See Table 147 for register control details.

Table 147 Data Unit register control

Register[BIT_POS]	Bit name	Description
SMARTIO_PRTx_DU_CTL[2:0]	DU_SIZE[2:0]	Size/width of the data unit (in bits) is DU_SIZE+1. For example, if DU_SIZE is 7, the width is 8 bits.
SMARTIO_PRTx_DU_CTL[11:8]	DU_OPC[3:0]	Data unit opcode specifies the data unit operation: 1: INCR 2: DECR 3: INCR_WRAP 4: DECR_WRAP 5: INCR_DECR 6: INCR_DECR_WRAP 7: ROR 8: SHR 9: AND_OR 10: SHR_MAJ3 11: SHR_EQL Otherwise: Undefined.
SMARTIO_PRTx_DU_SEL[3:0]	DU_TR0_SEl[3:0]	Data unit input signal tr0_in source selection: 0: Constant '0' 1: Constant '1' 2: Data unit output 10-3: LUT 7 - 0 outputs Otherwise: Undefined
SMARTIO_PRTx_DU_SEL[11:8]	DU_TR1_SEl[3:0]	Data unit input signal tr1_in source selection. Encoding same as DU_TR0_SEL

(table continues...)

Table 147 (continued) Data Unit register control

Register[BIT_POS]	Bit name	Description
SMARTIO_PRTx_DU_SEL[19:16]	DU_TR2_SEl[3:0]	Data unit input signal tr2_in source selection. Encoding same as DU_TR0_SEL
SMARTIO_PRTx_DU_SEL[25:24]	DU_DATA0_SEL[1:0]	Data unit input data data0_in source selection: 0: 0x00 1: chip_data[7:0] 2: io_data_in[7:0] 3: SMARTIO_PRTx_DATA.DATA[7:0] register field
SMARTIO_PRTx_DU_SEL[29:28]	DU_DATA1_SEL[1:0]	Data unit input data data1_in source selection. Encoding same as DU_DATA0_SEL.
SMARTIO_PRTx_DATA[7:0]	DATA[7:0]	Data unit input data source.

The data unit generates a single output trigger signal (tr_out). The internal state (du_data[7:0]) is captured in flip-flops and requires clk_block.

The following pseudo code describes the various datapath operations supported by the DU opcode. Note that “Comb” describes the combinatorial functionality – that is, functionalities that operate independent of

previous output states. “Reg” describes the registered functionality – that is, functionalities that operate on inputs and previous output states (registered using flip-flops).

```
// The following is shared by all operations.

data_eql_data1_in = (data & mask) == (data1_in & mask));
data_eql_0        = (data & mask) == 0);
data_incr         = (data + 1) & mask;
data_decr         = (data - 1) & mask;
data0_masked      = data_in0 & mask;

// INCR operation:
Comb: tr_out = data_eql_data1_in;
Reg:  data ≤ data;
      if (tr0_in)      data ≤ data0_masked;
      else if (tr1_in) data ≤ data_eql_data1_in ? data : data_incr;

// INCR_WRAP operation:
Comb: tr_out = data_eql_data1_in;
Reg:  data ≤ data;
      if (tr0_in)      data ≤ data0_masked;
      else if (tr1_in) data ≤ data_eql_data1_in ? data0_masked : data_incr;

// DECR operation:
Comb: tr_out = data_eql_0;
Reg:  data ≤ data;
      if (tr0_in)      data ≤ data0_masked;
      else if (tr1_in) data ≤ data_eql_0        ? data : data_decr;

// DECR_WRAP operation:
Comb: tr_out = data_eql_0;
Reg:  data ≤ data;
      if (tr0_in)      data ≤ data0_masked;
      else if (tr1_in) data ≤ data_eql_0        ? data0_masked: data_decr;

// INCR_DECR operation:
Comb: tr_out = data_eql_data1_in | data_eql_0;
Reg:  data ≤ data;
      if (tr0_in)      data ≤ data0_masked;
      else if (tr1_in) data ≤ data_eql_data1_in ? data : data_incr;
      else if (tr2_in) data ≤ data_eql_0        ? data : data_decr;

// INCR_DECR_WRAP operation:
Comb: tr_out = data_eql_data1_in | data_eql_0;
Reg:  data ≤ data;
      if (tr0_in)      data ≤ data0_masked;
      else if (tr1_in) data ≤ data_eql_data1_in ? data0_masked : data_incr;
      else if (tr2_in) data ≤ data_eql_0        ? data0_masked : data_decr;

// ROR operation:
Comb: tr_out = data[0];
Reg:  data ≤ data;
      if (tr0_in)      data ≤ data0_masked;
```

```
      else if (tr1_in) {
                        data          ≤ {0, data[7:1]} & mask;
                        data[du_size] ≤ data[0];
      }

// SHR operation:
Comb: tr_out = data[0];
Reg:  data ≤ data;
      if (tr0_in)       data ≤ data0_masked;
      else if (tr1_in) {
                        data          ≤ {0, data[7:1]} & mask;
                        data[du_size] ≤ tr2_in;
      }

// SHR_MAJ3 operation:
Comb: tr_out =   (data == 0x03)
               | (data == 0x05)
               | (data == 0x06)
 | (data == 0x07);
Reg:  data ≤ data;
      if (tr0_in)       data ≤ data0_masked;
      else if (tr1_in) {
                        data          ≤ {0, data[7:1]} & mask;
                        data[du_size] ≤ tr2_in;
      }

// SHR_EQL operation:
Comb: tr_out = data_eql_data1_in;
Reg:  data ≤ data;
      if      (tr0_in) data ≤ data0_masked;
      else if (tr1_in) {
                        data          ≤ {0, data[7:1]} & mask;
                        data[du_size] ≤ tr2_in;
      }

// AND_OR operation:
Comb: tr_out = | (data & data1_in & mask);
Reg:  data ≤ data;
      if (tr0_in) data ≤ data0_masked;
```

The SHR_MAJ3 operation is useful to implement a digital filter. The filter selects the majority value of three signal values.

22.12.3 Routing

The smart I/O block includes many switches that are used to route the signals in and out of the block and also between various components present in the block. The routing switches are handled through the SMARTIO_PRTx_LUT_SELy and SMARTIO_PRTx_DU_SEL registers. Refer to the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual for details. The smart I/O internal routing is shown in Figure 106. In the figure, note that LUT7 to LUT4 operate on io_data/chip_data[7] to io_data/chip_data[4] whereas LUT to LUT0 operate on io_data/chip_data[3] to io_data/chip_data[0].

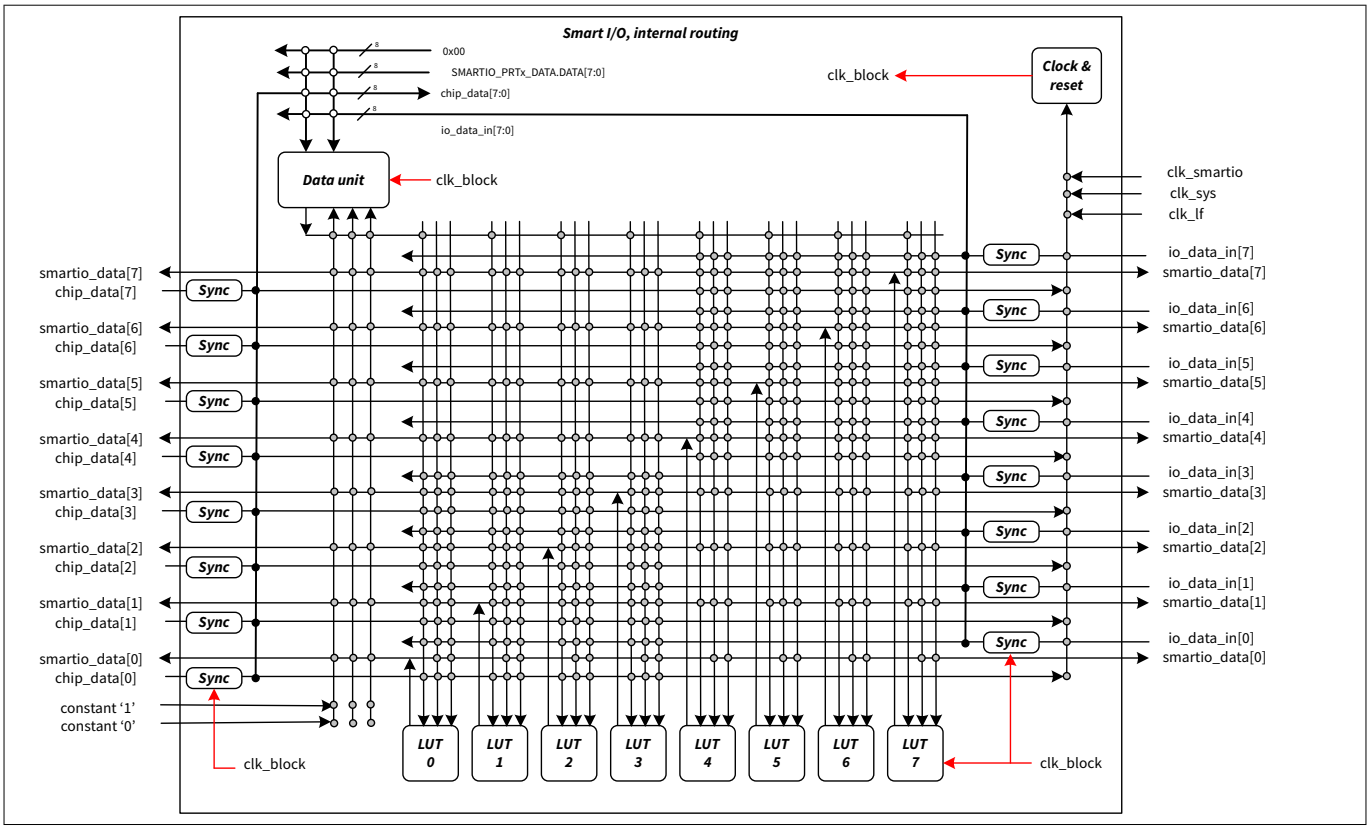


Figure 106 Smart I/O routing

22.12.4 Operation

The smart I/O block should be configured and operated as follows:

1. Before enabling the block, all the components and routing should be configured.
2. In addition to configuring the components and routing, some block level settings need to be configured correctly for desired operation.
   - **a.** Bypass control: The smart I/O path can be bypassed for a particular GPIO signal by setting the SMARTIO_PRTx_CTL.BYPASS bit register. When bit 'i' is set in the SMARTIO_PRTx_CTL.BYPASS bit field, the ith GPIO signal is bypassed to the HSIOM signal path directly - smart I/O logic will not be present in that signal path. This is useful when the smart I/O functionality is required only on select I/Os.
   - **b.** Pipelined trigger mode: The LUT input multiplexers and the LUT component itself do not include any combinatorial loops. Similarly, the data unit also does not include any combinatorial loops. However, when one LUT interacts with the other or to the data unit, inadvertent combinatorial loops are possible. To overcome this limitation, the SMARTIO_PRTx_CTL.PIPELINE_EN bit is used. When set, all the outputs (LUT and data unit) are registered before branching out to other components.
3. After the smart I/O block is configured for the desired functionality, the block can be enabled by setting the SMARTIO_PRTx_CTL.ENABLE bit. If disabled, the smart I/O block is put in bypass mode, where the GPIO signals are directly controlled by the HSIOM signals and vice-versa. The smart I/O block must be configured; that is, all register settings must be updated before enabling the block to prevent glitches during register updates.

Table 148 Smart I/O block controls

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

22.12.5 Example application

Smart I/O can be useful when the application involves simple logic operations and routing of the signal coming from or going to the I/O pin. No CPU is required for these operations. Some applications of smart I/O are:

- Change routing to/from pins (within the port)
- Invert the polarity of signal
- Clock or signal buffer
- Detect a pattern on pins

The following are detailed implementation of a few examples to better understand the smart I/O operation.

Example 1

Change routing to and from pins (within the port)

Consider an example where HSIOM is sending data to pin 1 at PORT 0 and you want to route it to a different pin (Pin 7, for example) at PORT0.

Figure 107 shows how the Smart I/O routing will look after configuration. Table 149 and Table 150 show LUT1 and LUT7 after configuration through the SMARTIO_PRT0_LUT_CTLx register.

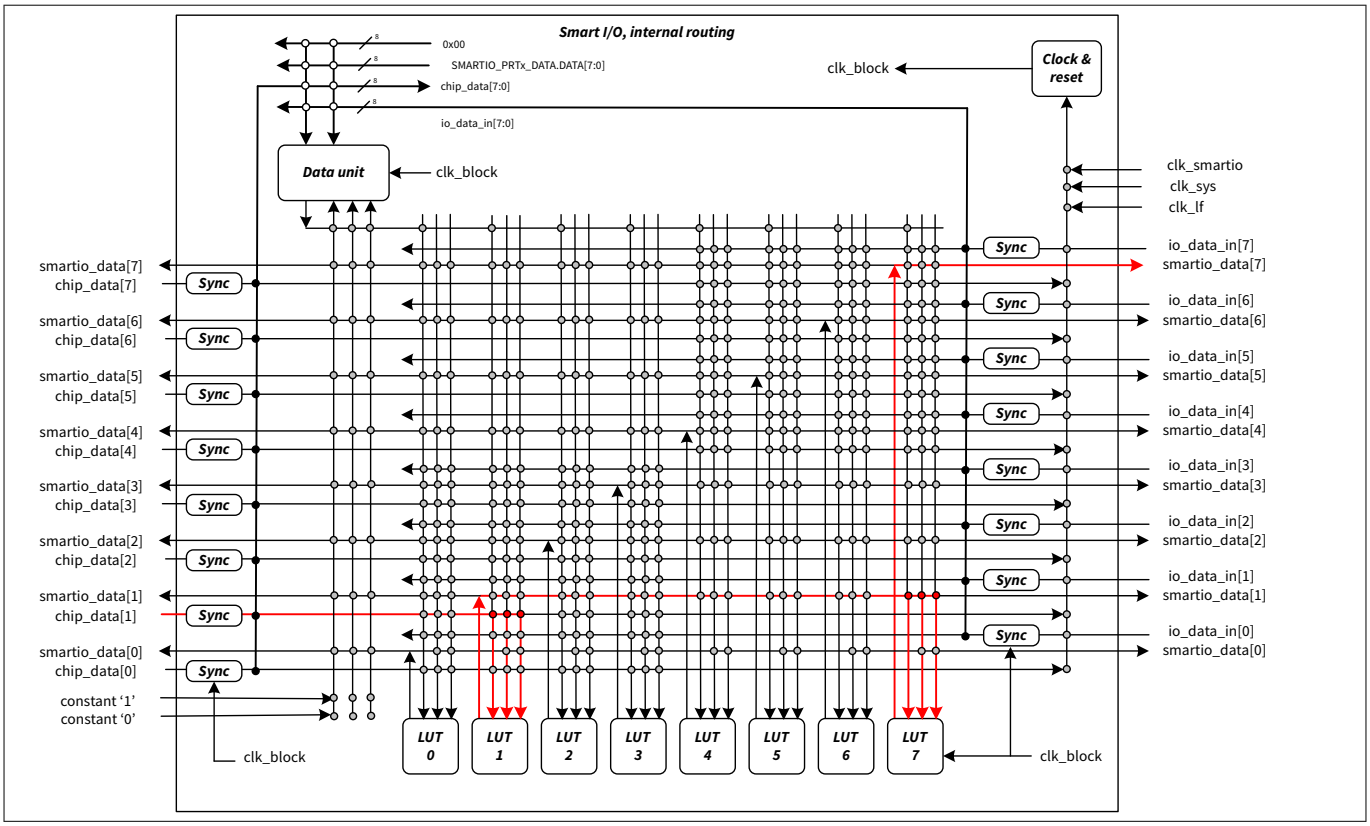


Figure 107 Smart I/O routing connections after configuration

Table 149 Lookup table LUT1

tr2_in	tr1_in	tr0_in	Out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Table 150 Lookup table LUT7

tr2_in	tr1_in	tr0_in	Out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0

(table continues...)

Table 150 (continued) Lookup table LUT7

tr2_in	tr1_in	tr0_in	Out
1	1	0	0
1	1	1	1

```
SMARTIO_PRT0_LUT_CTL1:
LUT = 0x80
LUT_OPC= 0x0 (Combinatorial)
LUT_TR0_SEL= 0x9
LUT_TR1_SEL= 0x9
LUT_TR2_SEL= 0x9

SMARTIO_PRT0_LUT_CTL7:
LUT = 0x80
LUT_OPC= 0x0 (Combinatorial)
LUT_TR0_SEL= 0x1
LUT_TR1_SEL= 0x1
LUT_TR2_SEL= 0x1

SMARTIO_PRT0_CTL:
CLK_SRC=0x10
BYPASS =0x7F
ENABLED=0x1 (Enable after all configuration is done)
```

Figure 108 Register settings for example 1

Example 2

Breathing LED using a constant PWM signal.

Consider a TCPWM that is sourced by a 1-MHZ clock and has a period of 65535 with a compare value of 32768. This generates a 50-percent duty cycle square wave with a period of approximately 65.5 ms. The CPU is used only to initialize the TCPWM.

Assume that the smart I/O is clocked at 30 Hz using the divided clock sourced from CLK_HF and implements several logic functions using the LUTs. Figure 109 shows the smart I/O LUT configuration. The smart I/O can implement a divide-by-two circuit from the 30-Hz clock. Therefore, LUT2 will produce a signal with a period of approximately 66.6 ms. The signal is then XORed with the PWM output (coming for port 1 pin 4) using LUT4 to generate a signal whose duty cycle gradually increases and decreases over time as shown in Figure 109 and inverted by LUT6. The example also shows how only one signal is routed to two output pins. Figure 110 shows the register settings required for the configuration given here.

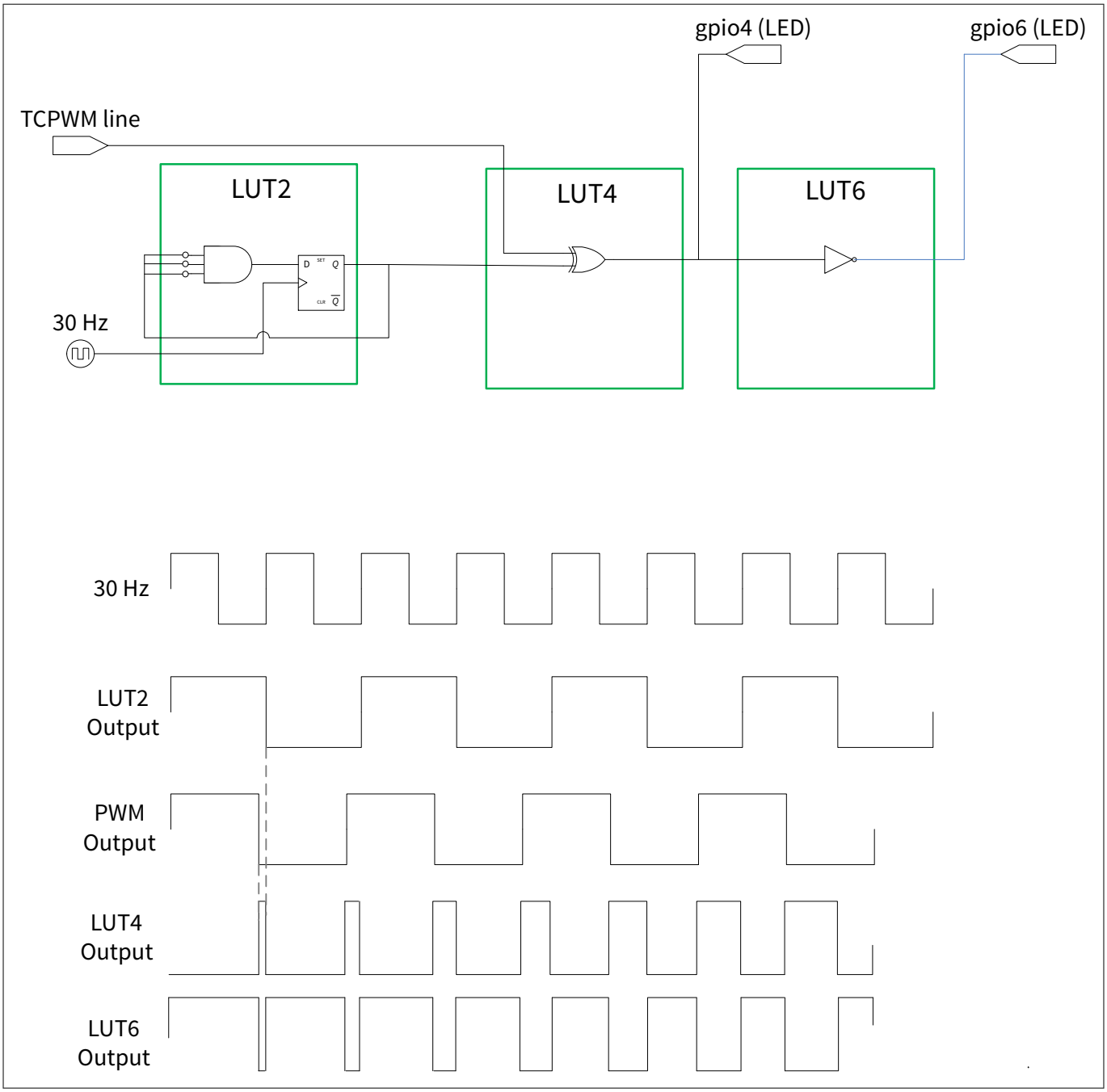


Figure 109 LUT configuration and timing diagram

Table 151 LUT2

tr2_in	tr1_in	tr0_in	Out
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0

(table continues...)

Table 151 (continued) LUT2

tr2_in	tr1_in	tr0_in	Out
1	1	0	0
1	1	1	0

Table 152 LUT4

tr2_in	tr1_in	tr0_in	Out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Table 153 LUT6

tr2_in	tr1_in	tr0_in	Out
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

```
SMARTIO_PRT1_LUT_CTL2:
LUT = 0x01
LUT_OPC= 0x2 (Gated Output)
LUT_TR0_SEL= 0x2
LUT_TR1_SEL= 0x2
LUT_TR2_SEL= 0x2

SMARTIO_PRT1_LUT_CTL4:
LUT = 0x42
LUT_OPC= 0x0 (Combinatorial)
LUT_TR0_SEL= 0x2
LUT_TR1_SEL= 0x2
LUT_TR2_SEL= 0x8

SMARTIO_PRT1_LUT_CTL6:
LUT = 0x01
LUT_OPC= 0x0 (Combinatorial)
LUT_TR0_SEL= 0x4
LUT_TR1_SEL= 0x4
LUT_TR2_SEL= 0x4

SMARTIO_PRT1_CTL:
CLK_SRC=0x10
BYPASS =0xAF
ENABLED=0x1 (Enable after all configuration is done)
```

Figure 110 Register settings for example 2

22.13 Registers

Table 154 I/O registers

Register	Name	Description
GPIO_PRTx_OUT	Port Output register	Port Output Data register reads and writes the output driver data for I/O pins in the port.
GPIO_PRTx_OUT_CLR	Port Output Clear register	Port Output Data Clear register clears output data of specific I/O pins in the port.
GPIO_PRTx_OUT_SET	Port Output Set register	Port Output Data Set register sets output data of specific I/O pins in the port.
GPIO_PRTx_OUT_INV	Port Output Invert register	Port Output Data Invert register inverts output data of specific I/O pins in the port.
GPIO_PRTx_IN	Port Input register	Port Input State register reads the current pin state present on I/O pin inputs.
GPIO_PRTx_INTR	Port Interrupt Status register	Port Interrupt Status register reads the current pin interrupt state.

(table continues...)

Table 154 (continued) I/O registers

Register	Name	Description
GPIO_PRTx_INTR_MASK	Port Interrupt Mask register	Port Interrupt Mask register configures the mask that forwards pin interrupts to the CPU's interrupt controller. This register only masks forwarding of interrupts to the CPU's interrupt controller; it does not enable/disable the logging of interrupts into the INTR register.
GPIO_PRTx_INTR_MASKED	Port Interrupt Masked Status register	This register contains the AND-ed values of INTR and INTR_MASK registers forwarded to the CPU interrupt controller.
GPIO_PRTx_INTR_SET	Port Interrupt Set register	Port Interrupt Set register allows firmware to set pin interrupts.
GPIO_PRTx_INTR_CFG	Port Interrupt Configuration register	Port Interrupt Configuration register selects the edge detection type for each pin interrupt.
GPIO_PRTx_CFG	Port Configuration register	Port Configuration register selects the drive mode and input buffer enable for each pin.
GPIO_PRTx_CFG_IN	Port Input Configuration register	Port Input Buffer Configuration register configures the input buffer mode (CMOS or TTL) for each pin. VTRIP_SEL[7:0]_0.
GPIO_PRTx_CFG_IN_AUTOLVL	Port Input Configuration register	Configures the input buffer upper bit i.e. VTRIP_SEL for each pin. Lower bit is still selected by CFG_IN.VTRIP_SEL[7:0]_1 field.
GPIO_PRTx_CFG_OUT	Port Output Configuration register	Port Output Buffer Configuration register selects the output driver slew rate for each pin.
GPIO_PRTx_CFG_OUT2	Port Output Configuration register	Port Output Buffer Configuration register 2 selects the output drive select trim for each I/O pin.
GPIO_PRTx_CFG_DRIVE_EXT	Port Output Buffer Drive Select Extension Configuration register	Port Output Buffer Drive Select Extension Configuration register configures the output driver for each pin.
GPIO_PRTx_CFG_SLEW_EXT	Port Output Buffer Slew Extension Configuration register	Port Output Buffer Slew Extension Configuration Register controls the slew rate for HSIO_STD_LN using the slew_ctl and HSIO_ENH by using slew_sel.
HSIOM_PRTx_PORT_SELy	HSIOM Port sSlect Register	High-speed I/O Mux (HSIOM) Port Selection Register selects the hardware peripheral connection to I/O pins.
GPIO_INTR_CAUSEz	Interrupt Port Cause Register	This register provides interrupt status corresponding to ports (0 + z × 32) to (31 + z × 32). "z" can be from 0 to 3.
GPIO_VDD_ACTIVE	External Power Supply Detection Register	This register provides external power supply status.

(table continues...)

Table 154 (continued) I/O registers

Register	Name	Description
GPIO_VDD_INTR	Supply Detection Interrupt Register	This register is set whenever a supply ramp up or ramp down is detected. Some bits may be set after system power-up, depending on power supply sequencing.
GPIO_VDD_INTR_MASK	Supply Detection Interrupt Mask Register	This register configures the supply detection interrupts for all supplies. It only masks the forwarding of interrupts to the CPUs and does not enable/disable the logging of interrupts into the VDD_INTR Register.
GPIO_VDD_INTR_MASKED	Supply Detection Interrupt Masked Register	This register contains the AND-ed values of VDD_INTR and VDD_INTR_MASK Registers.
GPIO_VDD_INTR_SET	Supply Detection Interrupt Set Register	This register allows firmware or debugger to set interrupt bits in the VDD_INTR Register by writing a '1' to the corresponding bit field. When read, it returns the same value as the VDD_INTR Register.
SMARTIO_PRTx_CTL	SMARTIO Control Register	This is the control register for SMARTIO on the specific port. It controls enable, clock source, bypass, and so on
SMARTIO_PRTx_SYNC_CTL	Synchronization Control Register	SMARTIO synchronization control
SMARTIO_PRTx_LUT_SELy	LUT Component Input Selection Register	LUT Input Selection Register. LUT_TR0_SEL, LUT_TR1_SEL, and LUT_TR2_SEL
SMARTIO_PRTx_LUT_CTLy	LUT Component Control Register	The LUT Control Register provides opcode for LUT
SMARTIO_PRTx_DU_SEL	Data Unit Component Input Selection Register	Data Unit Input Selection Register
SMARTIO_PRTx_DU_CTL	Data Unit Component Control Register	Data Unit Control Register
SMARTIO_PRTx_DATA	Data Register	Data unit input data source

***Note**: The 'x' in GPIO_PRTx/HSIOM_PRTx/SMARTIO_PRTx denotes the port number. For example, GPIO_PTR1_OUT is the port 1 output data register. 'y'/'z' can be '0' or '1'.*

23 Serial communications block (SCB)

Top-level architecture

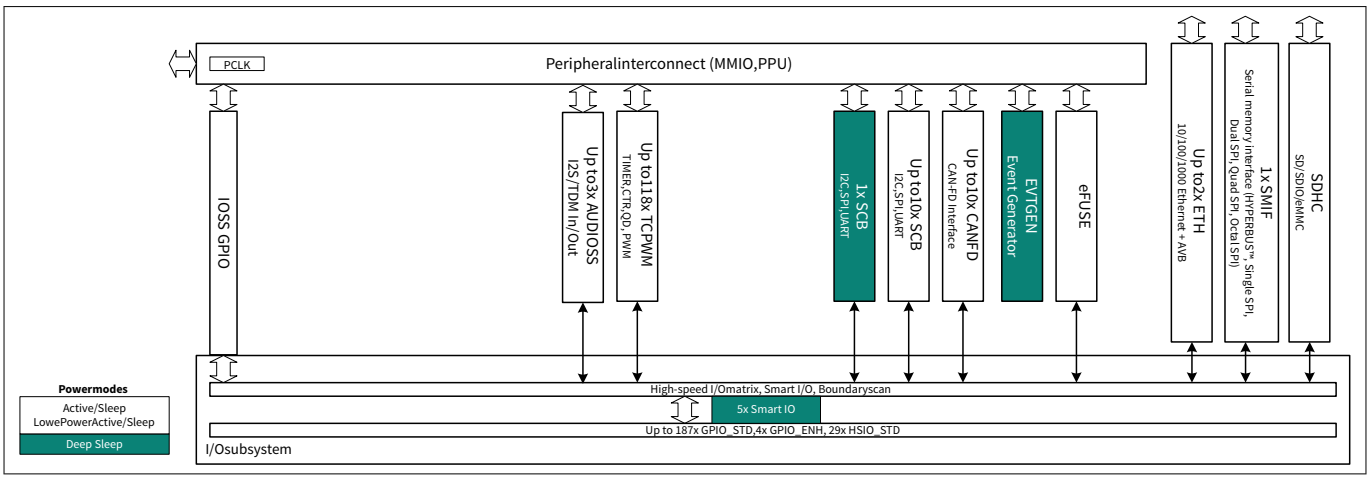


Figure 111 Digital system block diagram

The serial communications block (SCB) supports three serial communication protocols: serial peripheral interface (SPI), universal asynchronous receiver transmitter (UART), and inter-integrated circuit (I^2C or IIC). Only one of the protocols is supported by an SCB at any given time. XMC7000 MCUs have several SCBs. One of them supports only I^2C slave mode and SPI slave mode. This is the only SCB that is available in the deep sleep power mode.

23.1 Features

The SCB supports the following features:

- Standard SPI master and slave functionality with Motorola, Texas Instruments, and National Semiconductor protocols
- Standard UART functionality with SmartCard reader and IrDA protocols
- Standard I^2C master and slave functionality
- EZ mode for SPI and I^2C slaves; allows operation without CPU intervention
- CMD_RESP mode for SPI and I^2C slaves; allows operation without CPU intervention and is available only on deep sleep-capable SCB
- Low-power (deep sleep) mode of operation for SPI and I2C slaves (using external clocking), available only on deep sleep-capable SCB
- Deep sleep wakeup on I^2C slave address match or SPI slave selection; available only on deep sleep-capable SCB
- Trigger outputs for connection to DMA
- Multiple interrupt sources to indicate status of FIFOs and transfers
- Local loop-back control

23.2 Block diagram

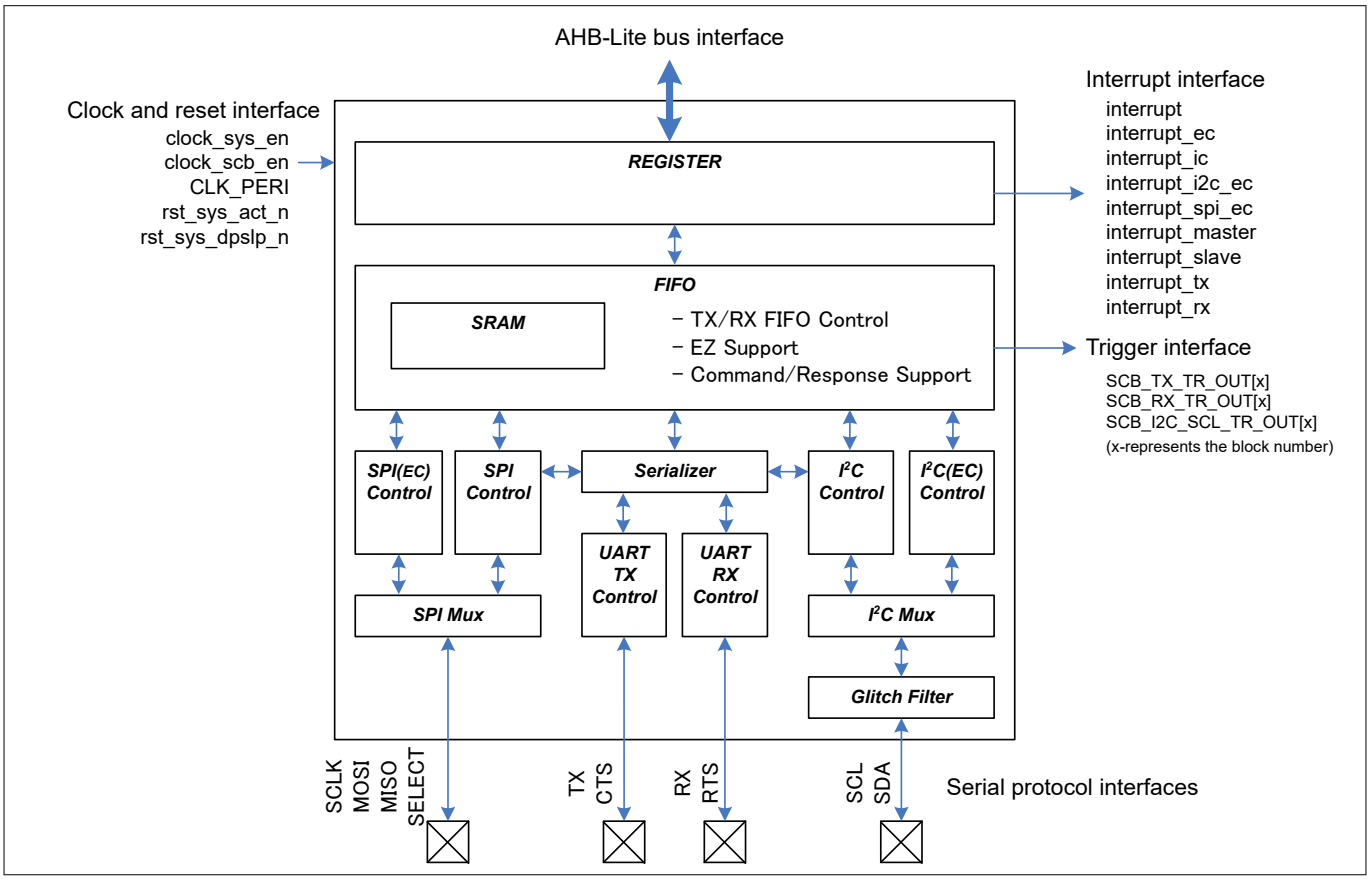


Figure 112 SCB block diagram

23.2.1 AHB-Lite bus interface

The SCB block is connected to the bus infrastructure through an AHB-Lite interface. This interface provides bus masters (such as the CPU) with access to the SCB block's registers. The registers control the block's operation and provide status information. The register map provides the details on the registers and register fields.

The AHB-Lite interface handles all accesses to the SCB block's 64-KB memory aperture.

- The AHB-Lite interface generates an AHB-Lite bus error for non 32-bit accesses.
- The AHB-Lite interface generates an AHB-Lite bus error for non-aligned 32-bit accesses.
- Read accesses to memory aperture locations that are not populated by a register, return '0'.
- Write accesses to memory aperture locations that are not populated by a register, are ignored (no AHB-Lite bus error is generated).

23.2.2 Trigger interface

23.2.2.1 DMA/DW trigger signals

The trigger interface provides status information on the TX FIFO (tr_tx_req) and RX FIFO (tr_rx_req):

- SCB_TX_TR_OUT[x] indicates that the TX FIFO can accept a data element (to be transmitted), controlled by SCBx_TX_FIFO_CTRL.TRIGGER_LEVEL.
- SCB_RX_TR_OUT[x] indicates that the RX FIFO can provide a (received) data element, controlled by SCBx_RX_FIFO_CTRL.TRIGGER_LEVEL.

These two signals are level-sensitive.

The trigger interface is typically connected (directly or indirectly) to a DW/DMA controller.

For a RX FIFO read case, it takes “2 CLK_AHB, 1 CLK_PERI” cycles, from AHB read RX FIFO to tr_rx_req being cleared. If CLK_AHB = CLK_PERI, it takes three CLK_PERI cycles.

For TX FIFO write case, it takes “3 CLK_AHB, 1 CLK_PERI” cycles, from AHB write TX FIFO to tr_rx_req being cleared. If CLK_AHB = CLK_PERI, it takes four CLK_PERI cycles.

23.2.2.2 SCB_I2C_SCL_TR_OUT[x] signal

The SCB_I2C_SCL_TR_OUT[x](where x represents the block number) signal is added for timeout detection on the I^2C SCL line. It is connected to SCL input analog filter output, so glitches are removed.

Along with the TCPWM block, it can be used for SMBus timeout, which is required as per the SMBus specification. It can also be used to detect SCL stretching.

23.2.2.3 Serial protocol interfaces

These are the SPI, UART, and I^2C signal interfaces.

The interface signals connect to the high-speed I/O matrix (HSIOM) in the I/O subsystem (IOSS). The HSIOM multiplexes between on-chip signals and I/O pads. The multiplexing flexibility is chip-specific and is controlled by the IOSS/HSIOM Registers. The HSIOM can expose a single serial interface at different pad locations to provide system-level flexibility.

If SMARTIO is available, it can be configured to short UART_TX and UART_RX to support single-line half-duplex UART, or short MOSI and MISO to support single-line half-duplex SPI.

23.2.3 Clock and reset interface

The SCB block receives the following clock and reset related signals:

- A high-frequency clock, CLK_PERI. This clock is used to derive an AHB-Lite interface clock (CLK_SYS) and an SCB functionality clock (CLK_SCB).
- A system clock enable (clock_sys_en) from SRSS, which is used to derive a system clock CLK_SYS from CLK_PERI.
- An SCB clock enable (clock_scb_en) from the PCLK component in PERI. This clock enable is used to derive CLK_SCB from CLK_PERI.
- CLK_SCB can be divided from integer or fractional divider in the PERI block.
- The fractional divider causes varying cycle times in generated CLK_SCB.
- The integer divider must be used for I^2C and SPI (synchronous interface),
- Both integer and fractional dividers can be used for UART (asynchronous interface).
- CLK_SCB is used for internally-clocked mode. Note that CLK_SCB is available only in active/sleep power modes. As a result, internally-clocked mode is not available in deep sleep power mode. The serial interface

protocols (UART TX/RX functionality and I^2C/SPI master functionality) are implemented using CLK_SCB as an “oversampled multiple” of the desired interface clock. For example, to implement a 100-kHz UART, CLK_SCB can be set to 1 MHz and the oversample factor set to ‘10’ (SCBx_CTRL.OVS = 10 – 1).

- A low/’0’ active system reset for active functionality rst_sys_act_n.
- A low/’0’ active system reset for deep sleep functionality rst_sys_dpslp_n.

In externally-clocked slave mode, serial interface input signals are used as clock (I2C: “SCL”, SPI: “SCLK”). These clocks are asynchronous to CLK_SYS and CLK_AHB, which are derived from CLK_PERI. In externally-clocked slave mode, reset signals are derived from rst_sys_dpslp_n that have a synchronous de-assertion with reference to the serial interface clock.

23.2.4 Block enable

More details about initializing is given in the description field of the SCBx_CTRL.ENABLED Register.

***Note**: All registers flagged with “NonRetention” will also be reset to the default state when the block is disabled. This includes the SCBx_INTR_XXX and SCBx_INTR_XXX_SET Registers.*

23.2.5 Interrupt interface

The SCB block has six types of interrupts. Each interrupt has dedicated registers. For details, see SCB interrupts and the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual.

Table 155 Interrupt Interface Signals and Registers

Interrupt	Functionality	Active/Deep sleep	Sync/async	Registers
interrupt_master	I2C master and SPI master functionality	Active	Sync	SCBx_INTR_M, SCBx_INTR_M_SET, SCBx_INTR_M_MASK, SCBx_INTR_M_MASKED
interrupt_slave	I2C slave and SPI slave functionality	Active	Sync	SCBx_INTR_S, SCBx_INTR_S_SET, SCBx_INTR_S_MASK, SCBx_INTR_S_MASKED
interrupt_tx	UART transmitter and TX FIFO functionality	Active	Sync	SCBx_INTR_TX, SCBx_INTR_TX_SET, SCBx_INTR_TX_MASK, SCBx_INTR_TX_MASKED
interrupt_rx	UART receiver and RX FIFO functionality	Active	Sync	SCBx_INTR_RX, SCBx_INTR_RX_SET, SCBx_INTR_RX_MASK, SCBx_INTR_RX_MASKED
interrupt_i2c_ec	Externally clocked I2C slave functionality	Deep sleep	Async	SCBx_INTR_I2C_EC, SCBx_INTR_I2C_EC_MASK, SCBx_INTR_I2C_EC_MASKED

(table continues...)

Table 155 (continued) Interrupt Interface Signals and Registers

Interrupt	Functionality	Active/Deep sleep	Sync/async	Registers
interrupt_spi_ec	Externally clocked SPI slave functionality	Deep sleep	Async	SCBx_INTR_SPI_EC, SCBx_INTR_SPI_EC_MASK, SCBx_INTR_SPI_EC_MASKED

The active functionality interrupts are generated synchronously to CLK_PERI. The deep sleep functionality interrupts are generated asynchronously to CLK_PERI and need synchronization in the CPUSS interrupt multiplexer.

For chips with a limited number of available interrupt lines, the SCB block also provides “combined functionality” interrupts as follows:

- interrupt_ec is the OR of interrupt_i2c_ec and interrupt_spi_ec.
- interrupt_ic is the OR of interrupt_master, interrupt_slave, interrupt_tx, and interrupt_rx.
- “interrupt” is the OR of all six individual interrupts.

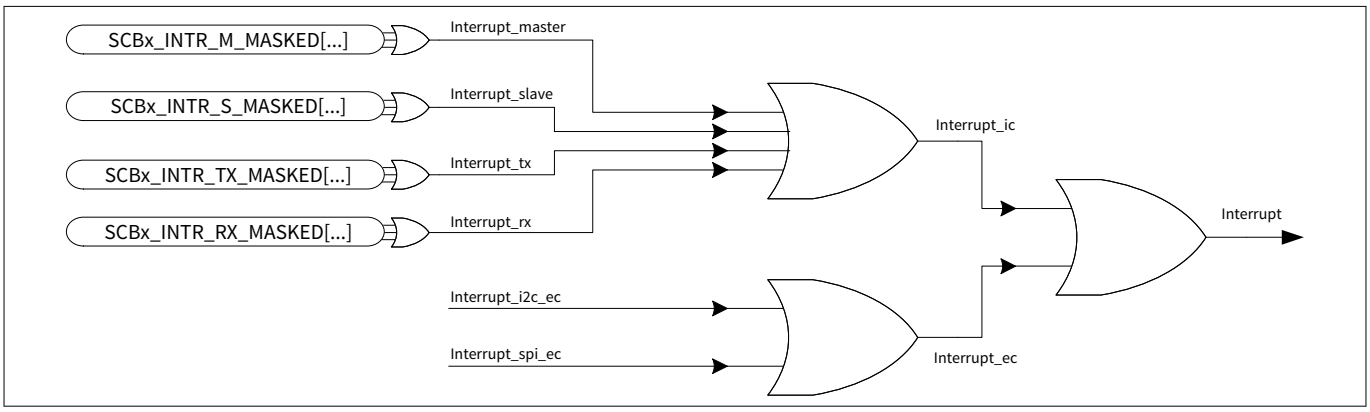


Figure 113 Interrupt lines

SCBx_INTR_M, SCBx_INTR_S, SCBx_INTR_TX, and SCBx_INTR_RX are interrupts from internal-clocked logic; SCBx_INTR_I2C_EC and SCBx_INTR_SPI_EC are interrupts from external-clocked logic.

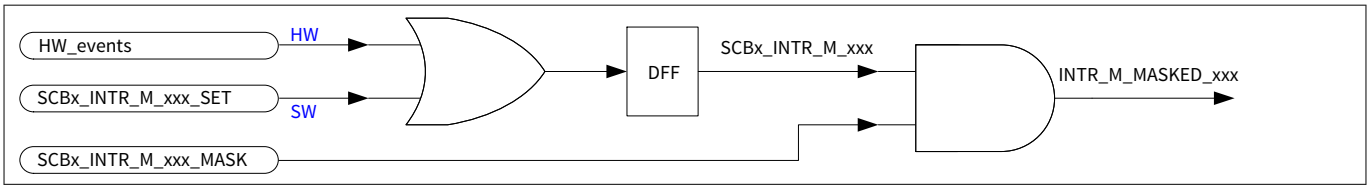


Figure 114 SCBx_INTR_M generation

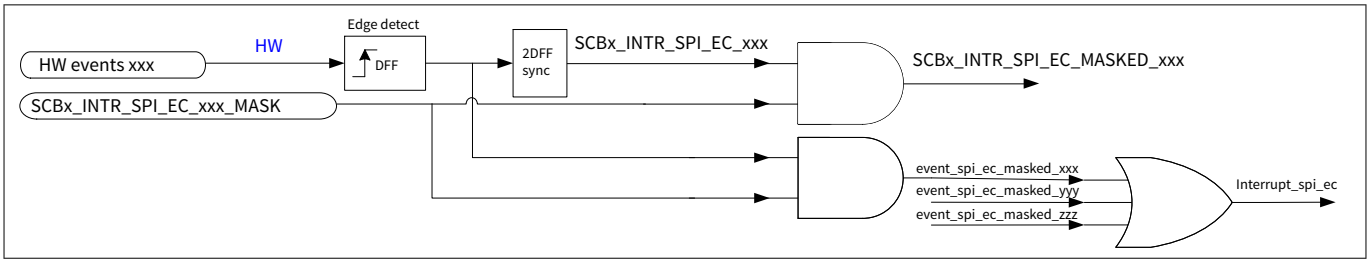


Figure 115 SCBx_INTR_SPI_EC and interrupt_spi_ec generation

***Note**: Interrupt Request Registers such as SCBx_INTR_M can be set only by hardware (HW RW1S) and cleared only by software (SW RW1C).*

To avoid being triggered by events from previous transactions, whenever the firmware enables an interrupt mask register bit (for example, SCBx_INTR_M_MASK.I2C_STOP), it should clear the Interrupt Request Register (for example, SCBx_INTR_M.I2C_STOP) in advance.

23.3 Operation modes

23.3.1 Buffer modes

Each SCB has 256 bytes of dedicated RAM for transmit and receive operation. This RAM can be configured in three different modes (FIFO, EZ, or CMD_RESP). The following sections give a high-level overview of each mode. The sections on each protocol will provide more details.

- Masters can only use FIFO mode
- Slaves can use all three modes.

***Note**: CMD Response mode is available only on deep sleep-capable SCB*

- UART only uses FIFO mode

Figure 116 shows the buffer modes using a dedicated SRAM.

- Two 32 deep FIFOs for up to 32-bit data elements (SCBx_CTRL.MEM_WIDTH Register is ‘2’)
- Two 64 deep FIFOs for up to 16-bit data elements (SCBx_CTRL.MEM_WIDTH Register is ‘1’)
- Two 128 deep FIFOs for up to 8-bit data elements (SCBx_CTRL.MEM_WIDTH Register is ‘0’)
- One 256 Byte EZ memory buffer
- One 256 Byte CMD_RESP memory buffer

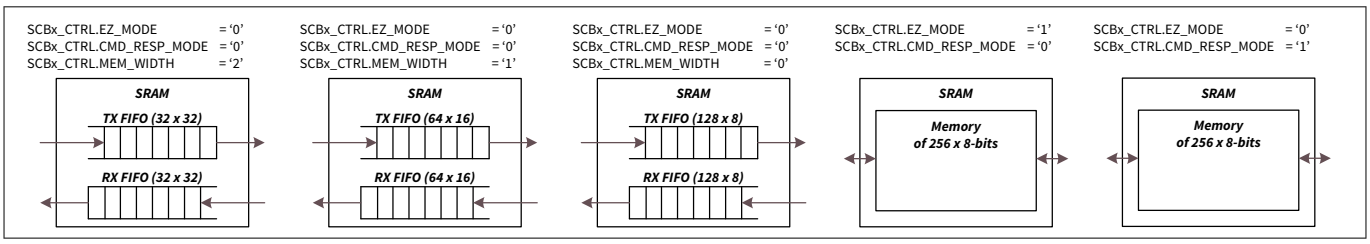


Figure 116 Buffer modes using a dedicated SRAM

23.3.1.1 FIFO mode

In this mode the RAM is split into two 128-byte FIFOs, one for transmit (TX) and one for receive (RX). The FIFOs can be configured to be 8 bits x 128 elements or 16 bits x 64 elements.

FIFO mode of operation is available only in active and sleep power modes. However, the I^2C address or SPI slave select can be used to wake the device from deep sleep on the deep sleep-capable SCB.

A write access to the TX FIFO uses the SCBx_TX_FIFO_WR.DATA Register. A read access from the RX FIFO uses the SCBx_RX_FIFO_RD.DATA Register.

Furthermore, it is possible that reading a data frame will not remove the data frame from the FIFO using the SCBx_RX_FIFO_RD_SILENT.DATA Register.

Status is provided for both the RX and TX buffers. Multiple interrupt sources that indicate the status of the FIFOs are available, such as full or empty; see SCB interrupts.

23.3.1.2 EZ mode

In easy (EZ) mode the RAM is used as a single 256-byte buffer. The external master sets a base address and reads and writes start from that base address.

EZ mode is available only for SPI slave and I²C slave. It is available only on the deep sleep-capable SCB.

EZ mode is available in active, sleep, and deep sleep power modes.

23.3.1.3 CMD_RESP mode

Command response (CMD_RESP) mode is similar to EZ mode except that the base address is provided by the CPU not the external master.

CMD_RESP mode is available only for SPI slave and I²C slave. It is available only on the deep sleep-capable SCB.

CMD_RESP mode operation is available in active, sleep, and deep sleep power modes.

23.3.2 Clocking modes

The SCB can be clocked either by an internal clock provided by the peripheral clock dividers or by the external master.

- UART, SPI master, and I²C master modes must use the internal clock, called CLK_SCB in the rest of this chapter.
- Only SPI slave and I²C slave can use the clock from an external master, and only the deep sleep-capable SCB supports this.

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

- SCBx_CTRL.EC_AM_MODE indicates whether SPI slave selection or I²C address matching is clocked internally ('0') or externally ('1').
- SCBx_CTRL.EC_OP_MODE indicates whether the rest of the protocol operation (besides SPI slave selection and I²C address matching) is clocked internally ('0') or externally ('1').

When using externally-clocked slave functionality, it is important to realize that:

- FIFO mode is not supported with externally-clocked operation (SCBx_CTRL.EC_OP_MODE is '1')
- EZ and CMD_RESP modes are supported with externally-clocked operation (SCBx_CTRL.EC_OP_MODE is '1')
- Before going to deep sleep mode, the SCBx_CTRL.EC_ACCESS Register should be set to ‘1’. When waking up from deep sleep mode and PLL is locked (CLK_SCB is at the expected frequency), the SCBx_CTRL.EC_ACCESS should be set to ‘0’.

The following table provides an overview of which clocking modes and which buffer modes are supported for each communication mode.

Table 156 Clock mode compatibility

	Internally-clocked (“IC”)			Externally-clocked (“EC”)		
	FIFO	**EZ**	**CMD_RESP**	**FIFO**	**EZ**	**CMD_RESP**
I²C master	Yes	No	No	No	No	No
I²C slave	Yes	Yes	No	Yes[1)]	Yes	Yes
I²C master-slave	Yes	No	No	No	No	No
SPI master	Yes	No	No	No	No	No
SPI slave	Yes	Yes	No	Yes[2)]	Yes	Yes

(table continues...)

Table 156 (continued) Clock mode compatibility

	Internally-clocked (“IC”)			Externally-clocked (“EC”)		
	FIFO	EZ	CMD_RESP	FIFO	EZ	CMD_RESP
UART transmitter	Yes	No	No	No	No	No
UART receiver	Yes	No	No	No	No	No

1) In deep sleep mode the externally-clocked logic can handle slave address matching; it then triggers an interrupt to wake up the CPU. The slave can be programmed to stretch the clock, or NACK until internal logic takes over. This only applies to the deep sleep-capable SCB.

2) In deep sleep mode the externally-clocked logic can handle slave selection detection; it then triggers an interrupt to wake up the CPU. Writes will be ignored and reads will return 0xFF until internal logic takes over. This only applies to the deep sleep-capable SCB.

23.4 Serial peripheral interface (SPI)

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

In the XMC7000 MCU, all SCB blocks support full SPI and only one SCB (SCB[0]) is available in deep sleep power mode; this allows externally-clocked operations.

23.4.1 Features

- Supports master and slave functionality
- Supports three types of SPI protocols:
  - Motorola SPI - modes 0, 1, 2, and 3
  - Texas Instruments SPI, with coinciding and preceding data frame indicator - mode 1 only
  - National Semiconductor (Microwire) SPI - mode 0 only
- Master supports up to four slave select lines
  - Each slave select has configurable active polarity (high or low)
  - Slave select can be programmed to stay active for a whole transfer, or just for each byte
- Master supports late sampling for better timing margin
- Master supports continuous SPI clock
- Data frame size programmable from 4 bits to 32 bits
- Variable SELECT output signal timing (SPI master):
  - SELECT setup time (select active to SPI clock)
  - SELECT hold time (SPI clock to select inactive)
  - Inter-data frame deselect time (select inactive to select active)
- Parity support (odd and even parity)
- Interrupts or polling CPU interface
- Programmable oversampling
- MSB or LSB first
- Median filter available for inputs
- Supports FIFO mode, EZ mode (slave only), and CMD_RESP mode (slave only).
- Wake-up interrupt cause activated on slave selection (SCB[0] only)
- Local loop-back control

23.4.2 General description

Figure 117 illustrates an example of SPI master with four slaves.

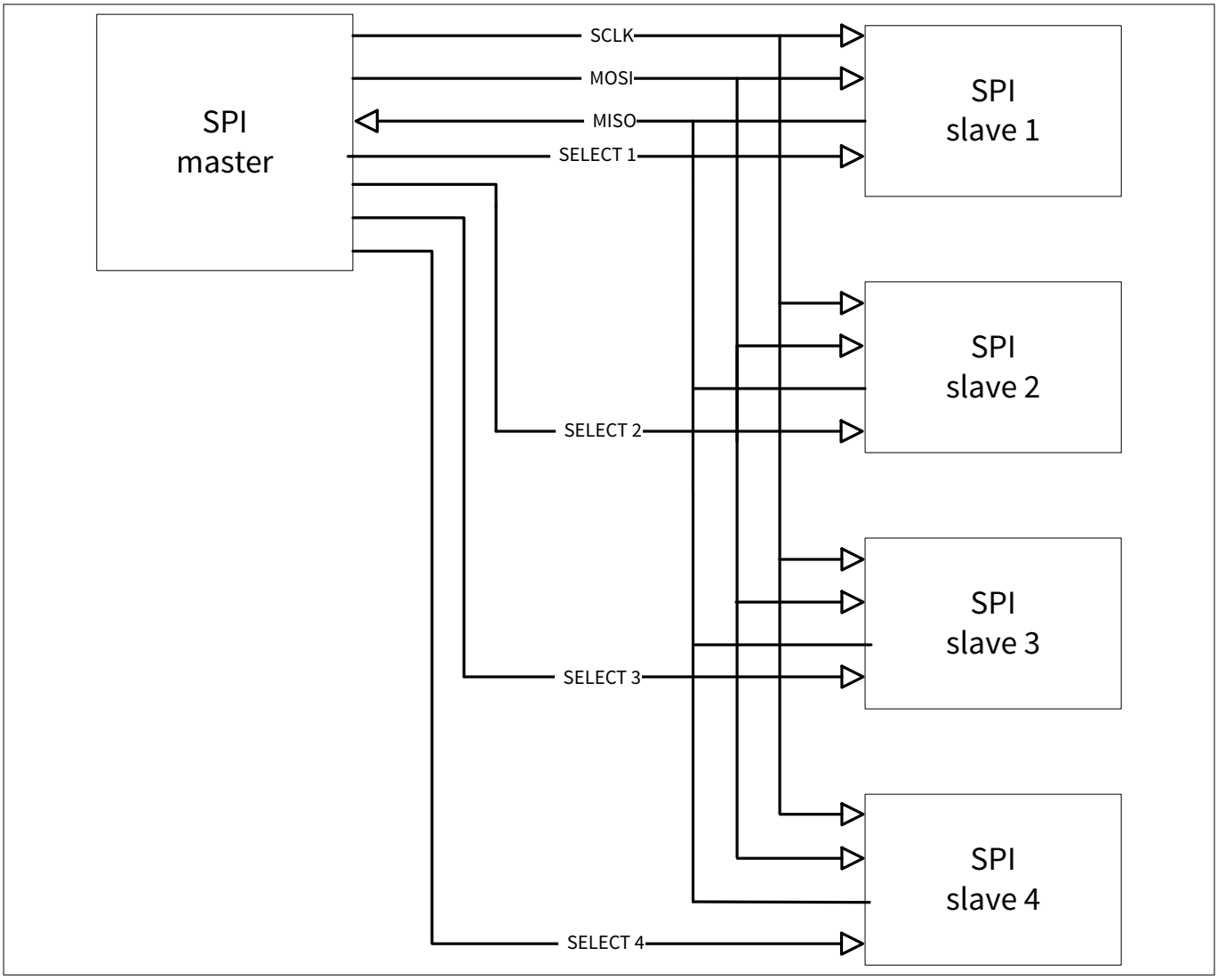


Figure 117 SPI example

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).
- SELECT: 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 SELECT line, then it drives data on the MOSI line and a clock on the SCLK line. The slave uses either of the edges of SCLK depending on the configuration to capture the data on the MOSI line; it also drives data on the MISO line, which is captured by the master.

By default, the SPI interface supports a data frame size of eight bits (1 byte). The data frame size can be configured to any value in the range 4 to 32 bits. The serial data can be transmitted either most significant bit (MSb) first or least significant bit (LSb) first.

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

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

Notes about duplex control:

- Motorola and Texas Instruments modes are full-duplex; National Semiconductors mode is half-duplex.
- Full-duplex modes also work similar to half-duplex, controlled by SCBx_TX_FIFO_CTRL.FREEZE or SCBx_RX_FIFO_CTRL.FREEZE, to transmit dummy data words or ignore received data words.
- The MOSI can be set to Hi-Z state using IOSS/GPIO configuration.

23.4.3 SPI modes of operation

23.4.3.1 Motorola SPI

The original SPI protocol was defined by Motorola. It is a full duplex protocol. Multiple data transfers may happen with the SELECT 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 SELECT line is held at '1' and SCLK is typically pulled low.

Clock modes of Motorola SPI

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

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

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

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

Figure 118 illustrates driving and capturing of MOSI/MISO data as a function of SCBx_SPI_CTRL.CPOL and SCBx_SPI_CTRL.CPHA.

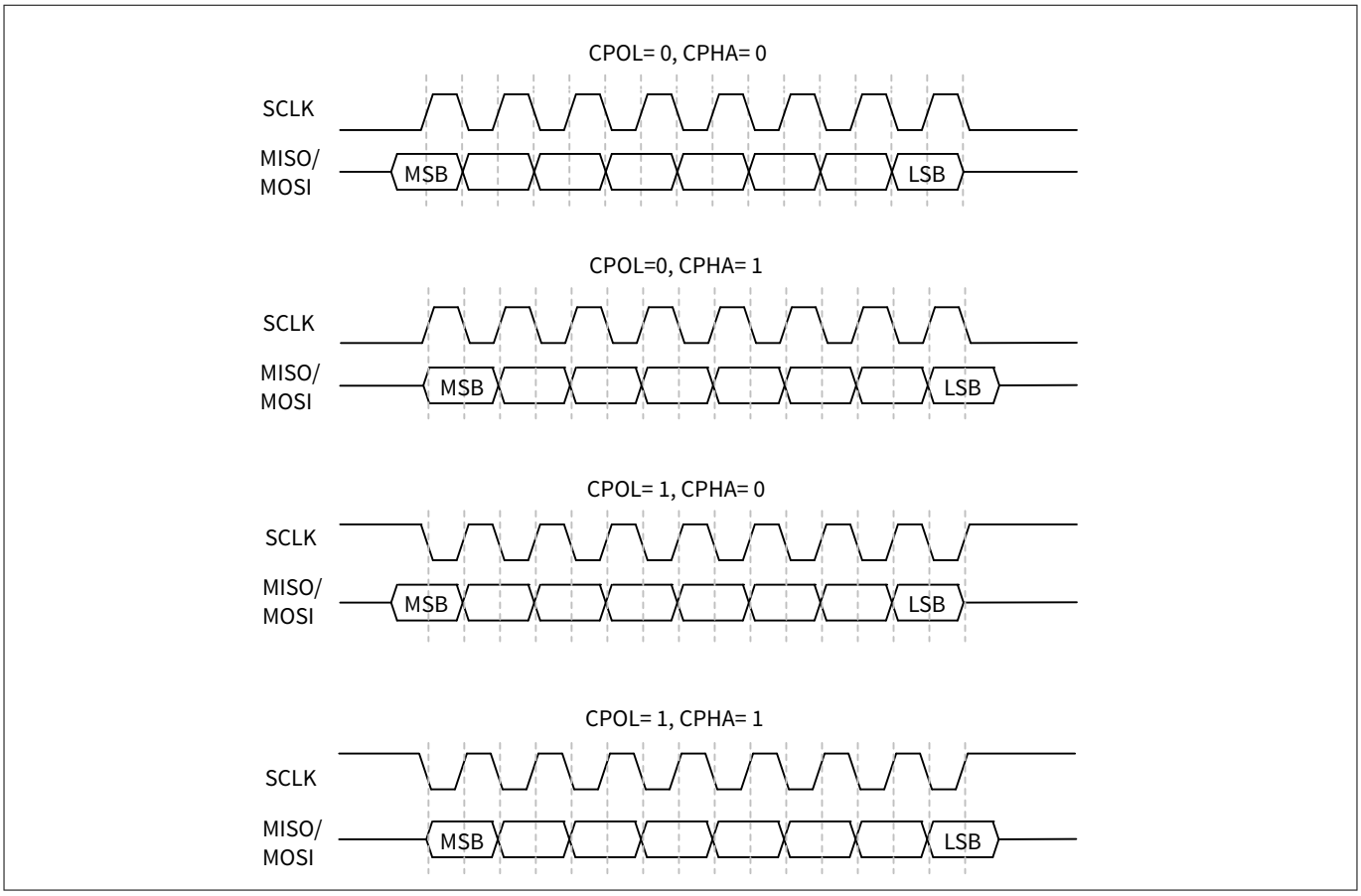


Figure 118 SPI Motorola, 4 modes

Figure 119 shows a single 8-bit and two successive 8-bit data transfers in mode 0 (SCBx_SPI_CTRL.CPOL is '0', SCBx_SPI_CTRL.CPHA is '0').

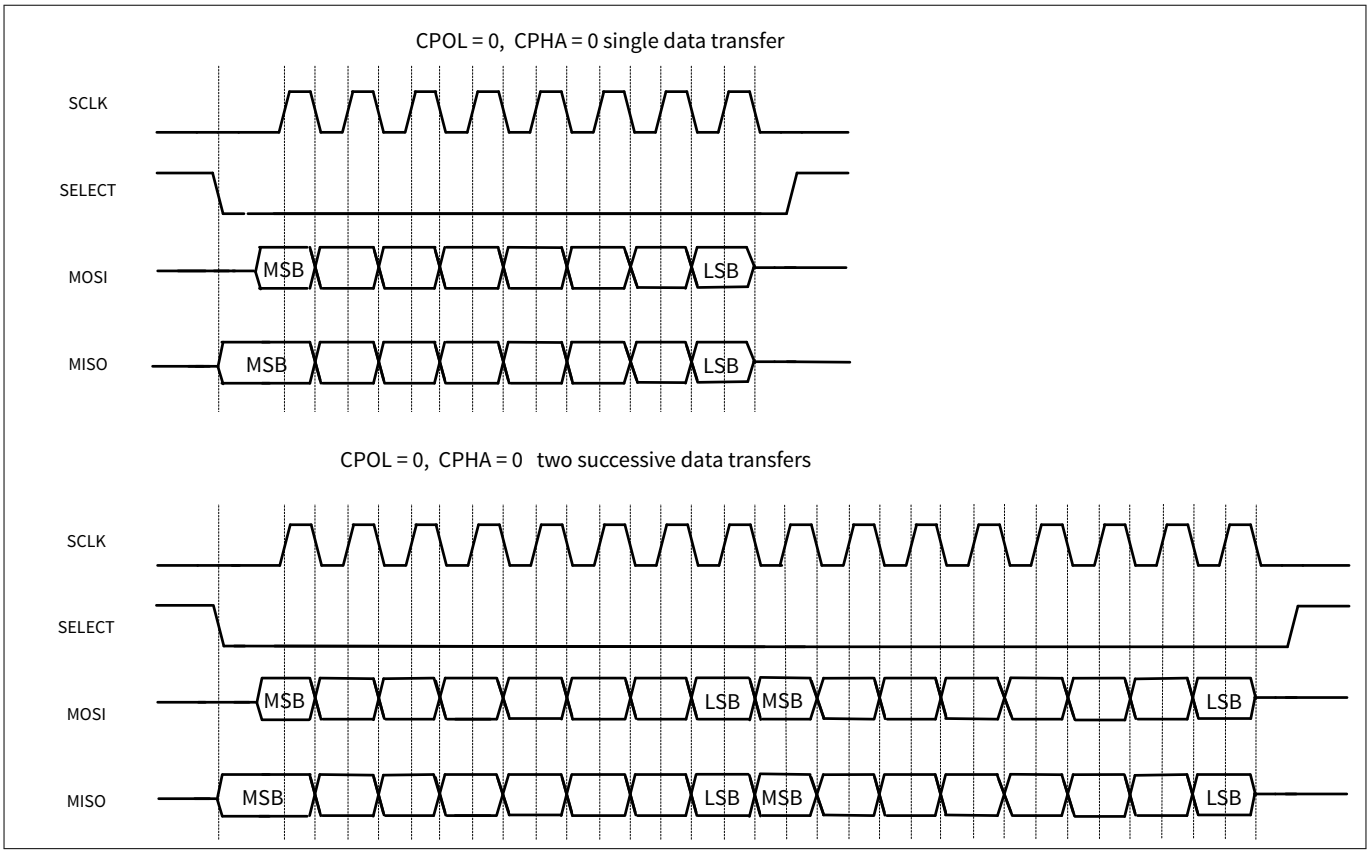


Figure 119 SPI Motorola data transfer example

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 SCBx_CTRL.MODE Register.
2. Select SPI Motorola mode by writing '00' to the SCBx_CTRL.MODE Register.
3. Select the mode of operation in Motorola by writing to the SCBx_SPI_CTRL.CPHA and SCBx_SPI_CTRL.CPOL Register.
4. Follow steps 2 to 4 mentioned in Enabling and initializing SPI.

For more information on these registers, see the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual.

23.4.3.2 Texas Instruments SPI

The Texas Instruments' SPI protocol redefines the use of the SELECT signal. It uses the signal to indicate the start of a data transfer, rather than a low active slave select signal, as in the 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 of a single-bit transfer period. This pulse may occur one cycle before the transmission of the first data bit, or may coincide with the transmission of the first data bit. The TI SPI protocol supports only mode 1 (SCBx_SPI_CTRL.CPOL is '0' and SCBx_SPI_CTRL.CPHA is '1'): data is driven on a rising edge of SCLK and captured on a falling edge of SCLK.

Figure 120 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.

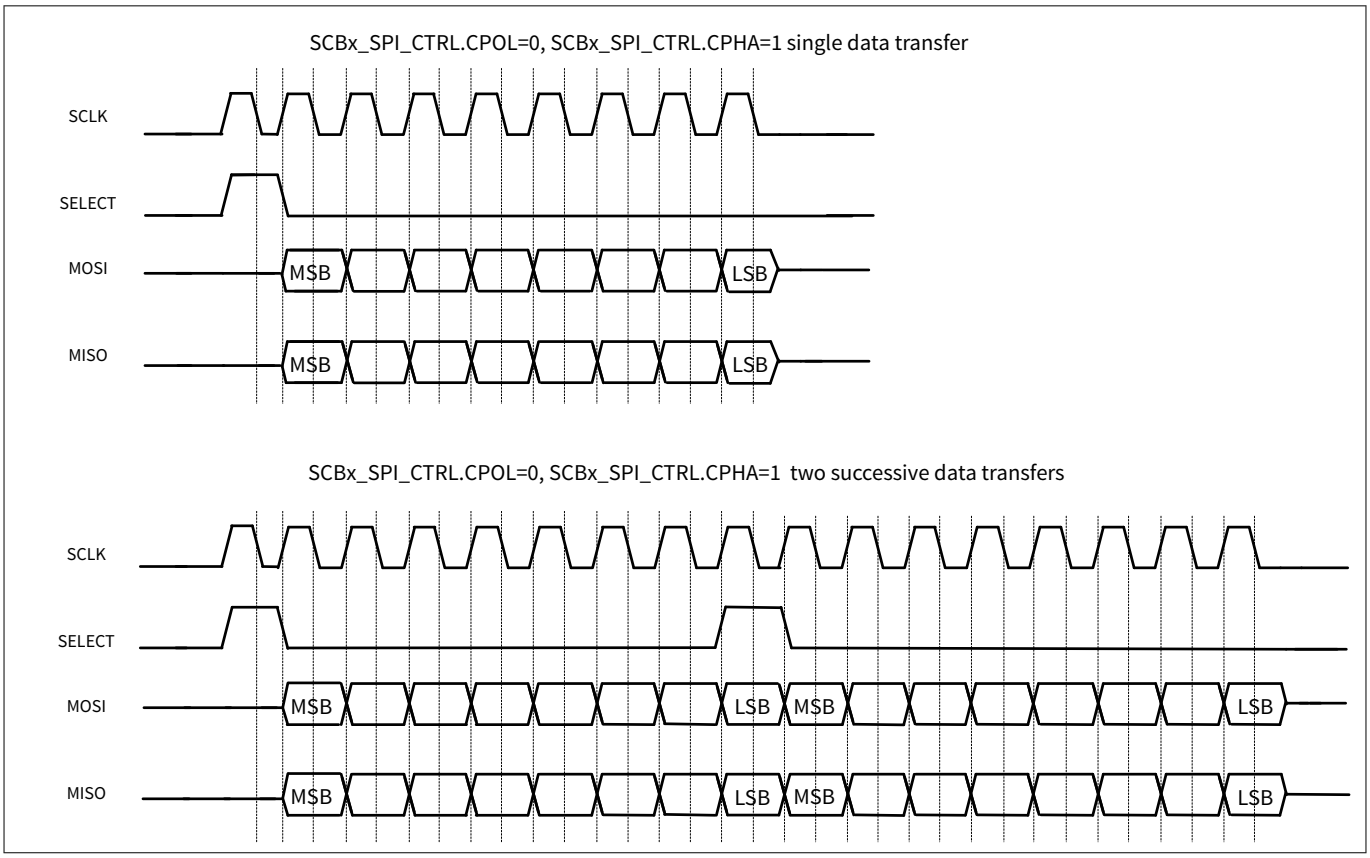


Figure 120 SPI TI data transfer example

Figure 121 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.

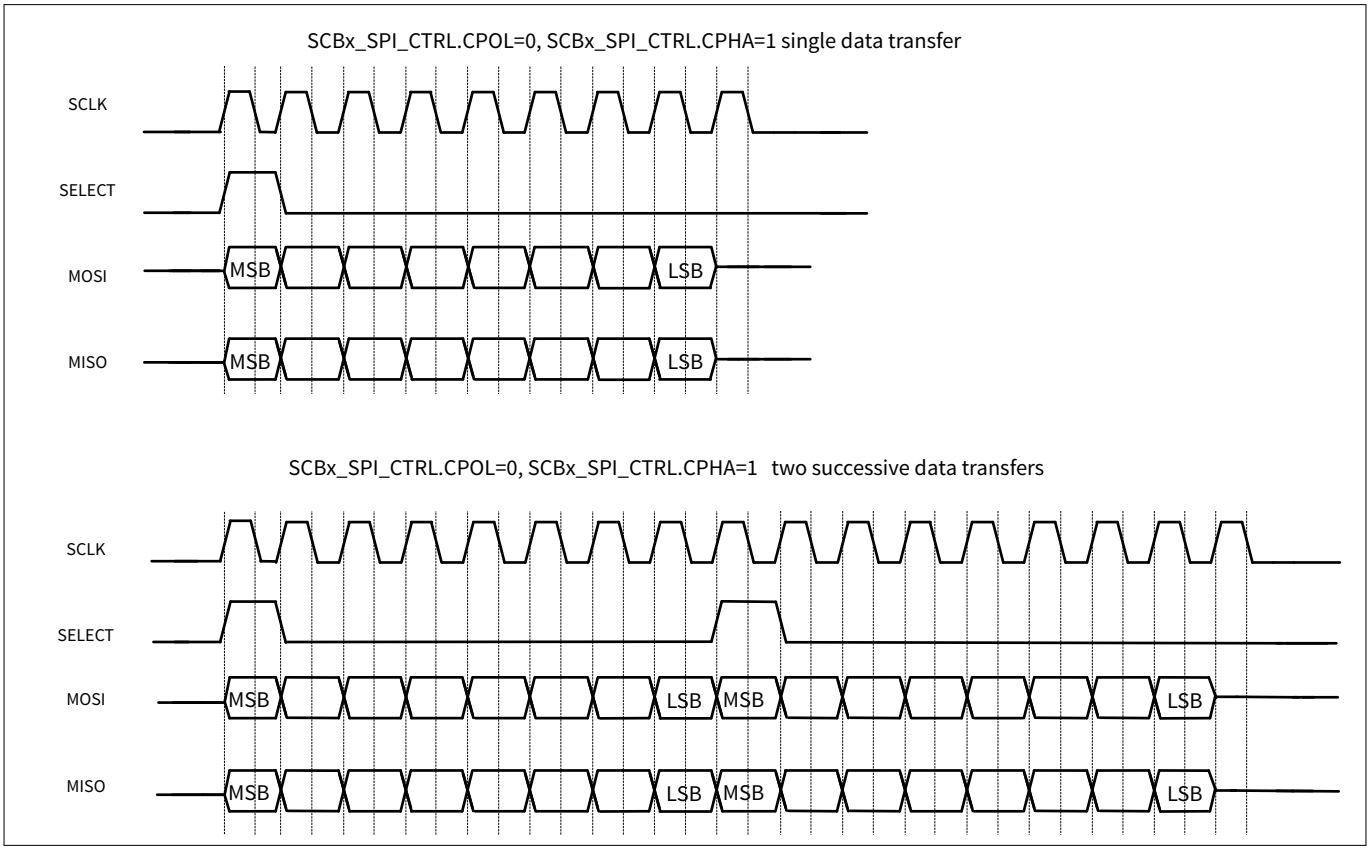


Figure 121 SPI TI data transfer example

Configuring SCB for SPI TI mode

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

1. Select SPI by writing '01' to the SCBx_CTRL.MODE Register.
2. Select SPI TI mode by writing '01' to the SCBx_CTRL.MODE Register.
3. Select the mode of operation in TI by writing to the SCBx_SPI_CTRL.SELECT_PRECEDE Register ('1' configures the SELECT pulse to precede the first bit of next frame and '0' otherwise).
4. Follow steps 2 to 4 mentioned in Enabling and initializing SPI.

For more information on these registers, see the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual.

23.4.3.3 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 (= '0') 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.

Figure 122 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.

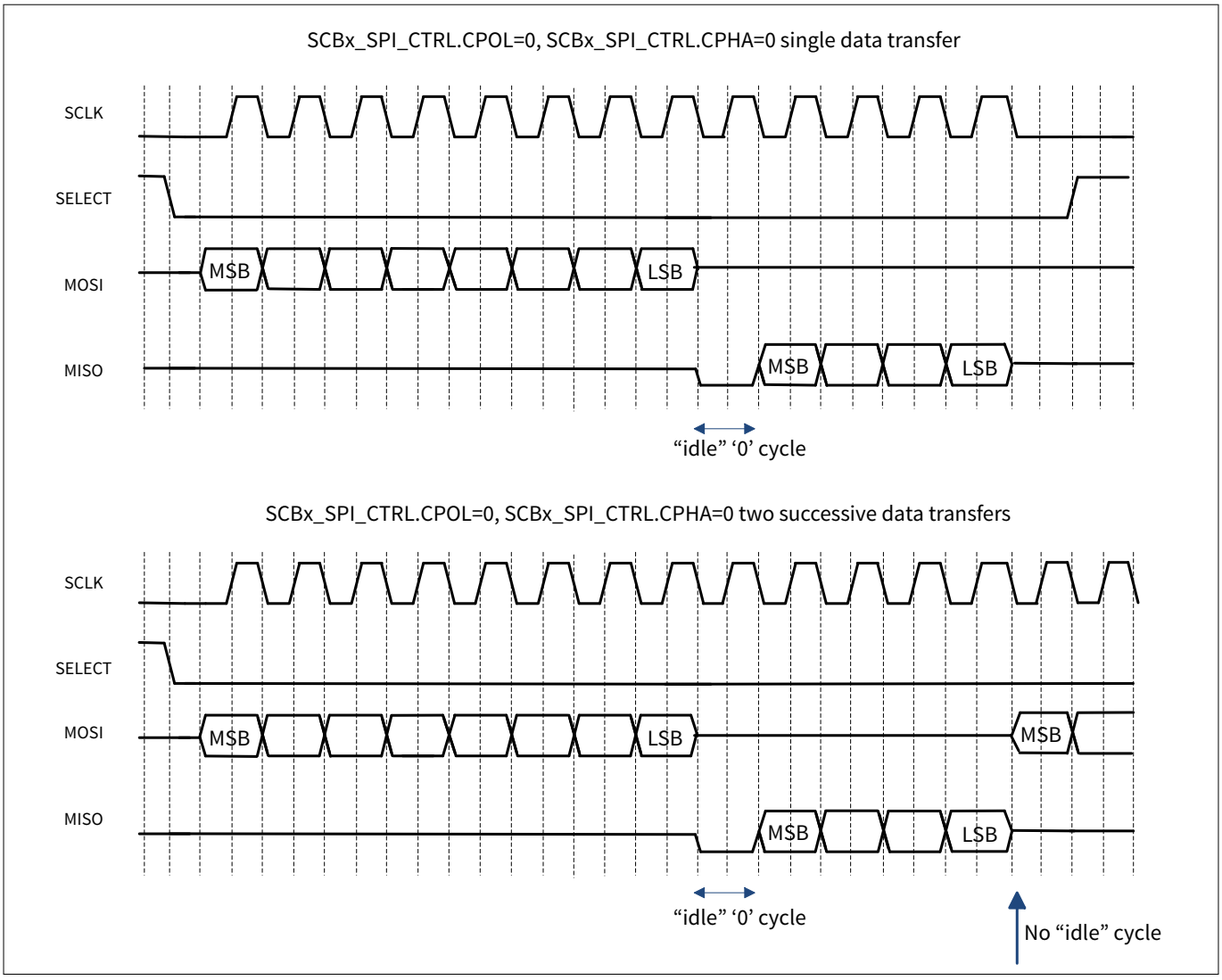


Figure 122 SPI NS data transfer example

Configuring SCB for SPI NS mode

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

1. Select SPI by writing '01' to the SCBx_CTRL.MODE Register.
2. Select SPI NS mode by writing '10' to the SCBx_CTRL.MODE Register.
3. Follow steps 2 to 4 mentioned in Enabling and initializing SPI.

For more information on these registers, see the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual.

23.4.4 SPI buffer modes

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

23.4.4.1 FIFO mode

The FIFO mode has a TX FIFO for the data being transmitted and an RX FIFO for the data received. Each FIFO is constructed out of the SRAM buffer. The FIFOs are either 32 elements deep with 32-bit data elements or 64 elements deep with 16-bit data elements or 128 elements deep with 8-bit data elements. The width of a FIFO is configured using the SCBx_CTRL.MEM_WIDTH Register.

FIFO mode is available only in active and sleep power modes, and not in the deep sleep mode.

Transmit and receive FIFOs allow write and read accesses. A write access to the transmit FIFO uses the SCBx_TX_FIFO_WR Register. A read access from the receive FIFO uses the SCBx_RX_FIFO_RD Register.

Transmit and receive FIFO status information is available through Status Registers, SCBx_TX_FIFO_STATUS and SCBx_RX_FIFO_STATUS. It is possible to define a programmable threshold that indicates a number of FIFO entries, a trigger/event is generated when the following conditions are met:

- The transmit FIFO has a SCBx_TX_FIFO_CTRL.TRIGGER_LEVEL. A trigger/event is generated when the number of entries in the transmit FIFO is less than SCBx_TX_FIFO_CTRL.TRIGGER_LEVEL.
- The receive FIFO has an SCBx_RX_FIFO_CTRL.TRIGGER_LEVEL. A trigger/event is generated when the number of receive FIFO entries is greater than the SCBx_RX_FIFO_CTRL.TRIGGER_LEVEL.

These triggers can be connected to a DMA channel.

Furthermore, numerous interrupt status bits are provided for both the RX and TX FIFOs. These can be found looking at SCBx_INTR_TX and SCBx_INTR_RX.

Deep sleep to active transition

SCBx_CTRL.EC_AM_MODE = 1, SCBx_CTRL.EC_OP_MODE = 0, FIFO mode.

MISO transmits 0xFF until internally-clocked logic takes over and CPU writes to TX FIFO. Data on MOSI is ignored until internally-clocked logic takes over. When the internally-clocked logic takes over, there is no guarantee that the internal clock will be at the correct frequency due to PLL/FLL locking times. This may lead to corrupted data in the RX FIFO. Therefore, it is recommended to clear the RX FIFO before writing new data into the TX FIFO after the transition from deep sleep to active. Another option is to disable CLK_SCB before going to deep sleep, and then wait to enable it until the PLL and FLL have stabilized. The external master needs to be aware that when it reads 0xFF on MISO the device is not ready yet.

23.4.4.2 EZSPI mode

The easy SPI (EZSPI) protocol is based on the Motorola SPI operating in any mode (0, 1, 2, or 3). It allows communication between master and slave without the need for CPU intervention. In XMC7000 MCU, only one SCB block supports EZSPI mode; the deep sleep-capable SCB.

The EZSPI protocol defines a single memory buffer with an 8-bit EZ address that indexes the buffer (256-entry array of eight bit per entry) located on the slave device. The EZ address is used to address these 256 locations. All EZSPI data transfers have 8-bit data frames.

The CPU writes and reads to the memory buffer through the SCBx_EZ_DATA Registers. These accesses are word accesses, but only the least significant byte of the word is used.

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

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

EZ address write

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

Memory array write

A write to a memory array index starts with a command byte (0x01) on the MOSI line indicating the master's intent to write to the memory array. The slave then drives a reply byte on the MISO line to indicate that the command was registered (0xFE) or not (0xFF). Any additional 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 (256), it remains there and does not wrap around to 0. The EZ base address is reset to the address written in the EZ Address Write phase on each slave selection.

Memory array read

A read from a memory array index starts with a command byte (0x02) on the MOSI line indicating the master's intent to read from the memory array. The slave then drives a reply byte on the MISO line to indicate that the command was registered (0xFE) or not (0xFF). Any additional read data bytes on the MISO line are read from the memory array at locations indicated by the communicated EZ address. The EZ address is automatically incremented by the slave as bytes are read from the memory array. When the EZ address exceeds the maximum number of memory entries (256), it remains there and does not wrap around to 0. The EZ base address is reset to the address written in the EZ Address Write phase on each slave selection.

Figure 123 illustrates the write of EZ address, write to a memory array and read from a memory array operations in the EZSPI protocol.

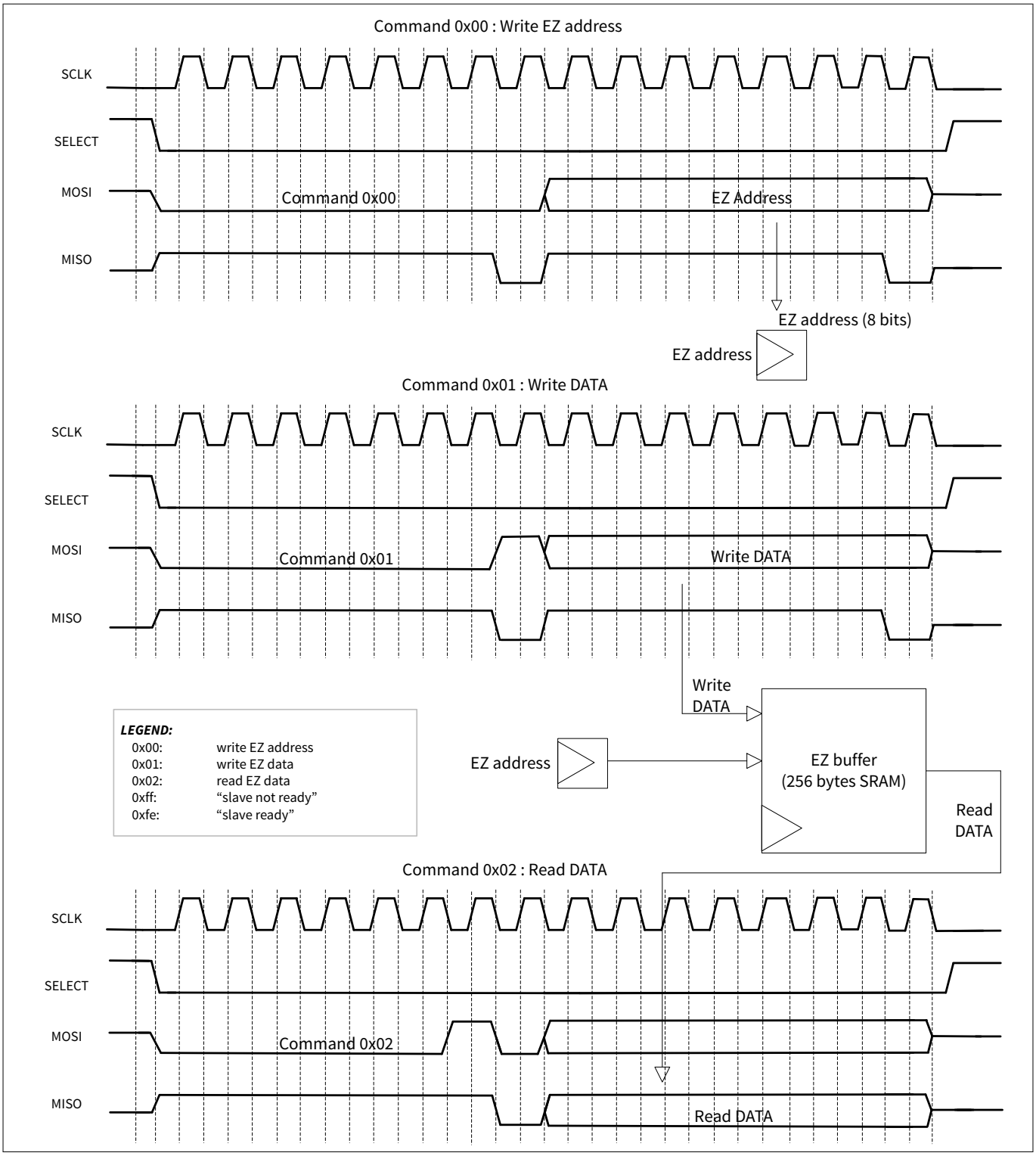


Figure 123 EZSPI example

Configuring SCB for EZSPI mode

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

1. Select EZ mode by writing '1' to the SCBx_CTRL.EZ_MODE Register.
2. Follow the steps in Configuring SCB for SPI Motorola mode.
3. Follow steps 2 to 4 mentioned in Enabling and initializing SPI.

For more information on these registers, see the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual.

Deep sleep to active transition

- **SCBx_CTRL.EC_AM_MODE = 1, SCBx_CTRL.EC_OP_MODE = 0, EZ mode:**

MISO transmits 0xFF until the internally-clocked logic takes over. Data on MOSI is ignored until the internally-clocked logic takes over. When this happens, there is no guarantee that the internal clock will be at the correct frequency due to PLL/FLL locking times. This may lead to corrupted data on MISO and in the EZ memory. Therefore, it is recommended to disable CLK_SCB before going to deep sleep, and then wait to enable it until the PLL/FLL have stabilized. The external master needs to be aware that when it reads 0xFF on MISO the device is not ready yet.

- **SCBx_CTRL.EC_AM_MODE = 1, SCBx_CTRL.EC_OP_MODE = 1, EZ mode:**

When transitioning from deep sleep to active mode, there is no guarantee that the internal clock will be at the correct frequency due to PLL/FLL locking times. This situation limits the SPI SCLK frequency to 2 MHz. After the FLL/PLL outputs have stabilized the clock can run faster.

23.4.4.3 Command-response mode

The command-response mode is defined only for an SPI slave. In the XMC7000 MCU, only one SCB (SCB[0]) supports the command-response mode. This mode has a single memory buffer, a base read address, a current read address, a base write address, and a current write address that are used to index the memory buffer. The base addresses are provided by the CPU. The current addresses are used by the slave to index the memory buffer for sequential accesses of the memory buffer. The memory buffer holds 256 8-bit data elements. The base and current addresses are in the range [0, 255].

The CPU writes and reads to the memory buffer through the SCBx_EZ_DATA Registers. These accesses are word accesses, but only the least significant byte of the word is used.

The slave interface accesses the memory buffer using the current addresses. At the start of a write transfer (SPI slave selection), the base write address is copied to the current write address. A data element write is to the current write address location. After the write access, the current address is incremented by '1'. At the start of a read transfer, the base read address is copied to the current read address. A data element read is to the current read address location. After the read data element is transmitted, the current read address is incremented by '1'.

If the current addresses equal the last memory buffer address (255), the current addresses are not incremented. Subsequent write accesses will overwrite any previously written value at the last buffer address. Subsequent read accesses will continue to provide the (same) read value at the last buffer address. The bus master should be aware of the memory buffer capacity in command-response mode.

The base addresses are provided through SCBx_CMD_RESP_CTRL.BASE_RD_ADDR and SCBx_CMD_RESP_CTRL.BASE_WR_ADDR. The current addresses are provided through SCBx_CMD_RESP_STATUS.CURR_RD_ADDR and SCBx_CMD_RESP_STATUS.CURR_WR_ADDR. At the end of a transfer (SPI slave de-selection), the difference between a base and current address indicates how many read/write accesses were performed. The block provides interrupt cause fields to identify the end of a transfer. Command-response mode operation is available in active, sleep, and deep sleep power modes.

The command-response mode has two phases of operation:

- Write phase - The write phase begins with a selection byte, which has its last bit set to '0' indicating a write. The master writes 8-bit data elements to the slave's memory buffer following the selection byte. The slave's current write address is set to the slave's base write address. Received data elements are written to the current write address memory location. After each memory write, the current write address is incremented.
- Read phase - The read phase begins with a selection byte, which has its last bit set to '1' indicating a read. The master reads 8-bit data elements from the slave's memory buffer. The slave's current read address is set to the slave's base read address. Transmitted data elements are read from the current address memory location. After each read data element is transferred, the current read address is incremented.

During the reception of the first byte, the slave (MISO) transmits either 0x62 (ready) or a value different from 0x62 (busy). When disabled or reset, the slave transmits 0xFF (busy). The byte value can be used by the master to determine whether the slave is ready to accept the SPI request.

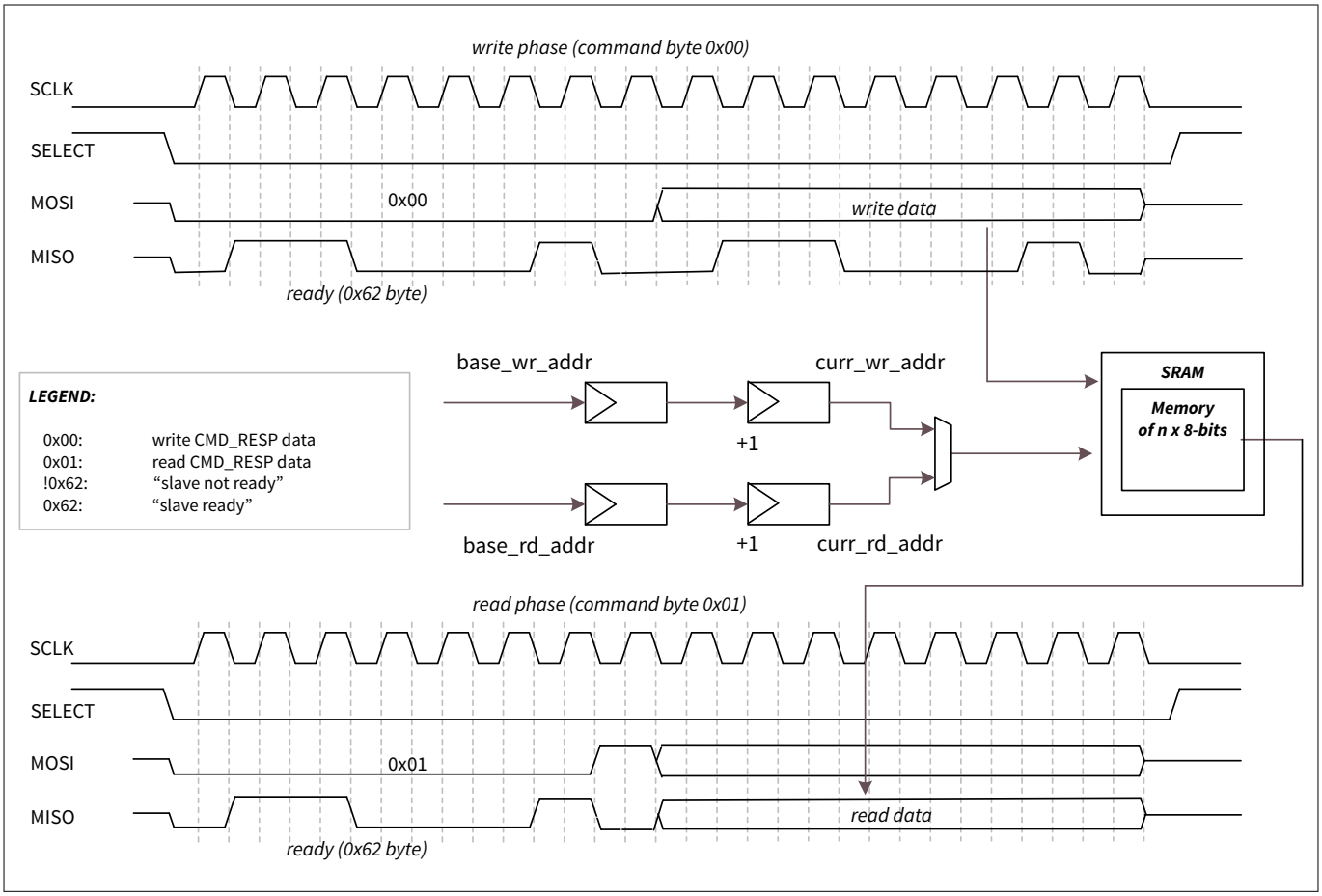


Figure 124 Command-response mode example

Note that a slave's base addresses are updated by the CPU and not by the master.

- **Deep sleep to active transition:**

SCBx_CTRL.EC_AM_MODE = 1, SCBx_CTRL.EC_OP_MODE = 1, CMD_RESP mode.

When transitioning from deep sleep to active mode there is no guarantee that the internal clock will be at the correct frequency due to PLL/FLL locking times. This situation limits the SPI SCLK frequency to 2 MHz. After the FLL/PLL outputs have stabilized the clock can run faster.

- **Configuring SCB for CMD_RESP mode:**

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

1. Select the CMD_RESP mode by writing '1' to the SCBx_CTRL.CMD_RESP_MODE Register.
2. Follow the steps in Configuring SCB for SPI Motorola mode.
3. Follow steps 2 to 4 mentioned in Enabling and initializing SPI.

For more information on these registers, see the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual.

23.4.5 Clocking and oversampling

23.4.5.1 Clock modes

The SCB SPI supports both internally- and externally-clocked operation modes. SCBx_CTRL.EC_AM_MODE and SCBx_CTRL.EC_OP_MODE Register determine the SCB clock mode. SCBx_CTRL.EC_AM_MODE indicates whether SPI slave selection is clocked internally (0) or externally (1). SCBx_CTRL.EC_OP_MODE indicates whether the rest of the protocol operation (besides SPI slave selection) is clocked internally (0) or externally (1).

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

***Note**: In the XMC7000 only the deep sleep-capable SCB supports externally-clocked mode of operation and only for SPI slave mode.*

An internally-clocked operation uses the programmable clock dividers. For more information on system clocking, see the Clocking system.

The SCBx_CTRL.EC_AM_MODE and SCBx_CTRL.EC_OP_MODE can be configured in the following ways.

- SCBx_CTRL.EC_AM_MODE is '0' and SCBx_CTRL.EC_OP_MODE is '0': Use this configuration when only active mode functionality is required.
  - FIFO mode: Supported
  - EZ mode: Supported
  - Command-response mode: Not supported. The slave (MISO) transmits a value different from a ready (0x62) byte during reception of the first byte, if the command-response mode is attempted in this configuration.
- SCBx_CTRL.EC_AM_MODE is '1' and SCBx_CTRL.EC_OP_MODE is '0': Use this configuration when both active and deep sleep functionality are required. This configuration relies on the externally-clocked functionality to detect the slave selection and relies on the internally-clocked functionality to access the memory buffer. The "handover" from external to internal functionality relies on a busy/ready byte scheme. This scheme relies on the master to retry the current transfer when it receives a busy byte and requires the master to support busy/ready byte interpretation. When the slave is selected, SCBx_INTR_SPI_EC.WAKE_UP is set to '1'. The associated deep sleep functionality interrupt brings the system into active power mode.
  - FIFO mode: Supported. The slave (MISO) transmits 0xFF until the CPU is awoken and the TX FIFO is populated. Any data on the MOSI line will be dropped until CLK_SCB is enabled see Deep sleep to active transition for more details
  - EZ mode: Supported. In deep sleep power mode, the slave (MISO) transmits a busy (0xFF) byte during the reception of the command byte. In active power mode, the slave (MISO) transmits a ready (0xFE) byte during the reception of the command byte.
  - CMD_RESP mode: Not supported. The slave transmits (MISO) a value different from a ready (0x62) byte during the reception of the first byte.
- SCBx_CTRL.EC_AM_MODE is '1' and SCBx_CTRL.EC_OP_MODE is '1'. Use this mode when both active and deep sleep functionality are required. When the slave is selected, SCBx_INTR_SPI_EC.WAKE_UP is set to

'1'. The associated deep sleep functionality interrupt brings the system into active power mode. When the slave is deselected, SCBx_INTR_SPI_EC.EZ_STOP and/or SCBx_INTR_SPI_EC.EZ_WRITE_STOP are set to '1'.

- FIFO mode: Not supported
- EZ mode: Supported
- CMD_RESP mode: Supported.

Table 157 SPI modes compatibility

	Internally-clocked (IC)			Externally-clocked (EC)		
	FIFO	**EZ**	**CMD_RESP**	**FIFO**	**EZ**	**CMD_RESP**
SPI master	Yes	No	No	No	No	No
SPI slave	Yes	Yes	No	Yes[1)]	Yes	Yes

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

If SCBx_CTRL.EC_OP_MODE is '1', the external interface logic accesses the memory buffer on the external interface clock (SPI SCLK). This allows for EZ and CMD_RESP mode functionality in active and deep sleep power modes.

In active system power mode, the memory buffer requires arbitration between external interface logic (on SPI SCLK) and the CPU interface logic (on system peripheral clock). This arbitration always gives the highest priority to the external interface logic (host accesses). The external interface logic takes two serial interface clock/bit periods for SPI. During this period, the internal logic is denied service to the memory buffer. The XMC7000 MCU provides two programmable options to address this “denial of service”:

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

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

If a series of read or write accesses is performed and SCBx_CTRL.BLOCK is '0', a failure is detected by comparing the “logical OR” of all read values to 0xFFFF:FFFF and checking the SCBx_INTR_TX.BLOCKED and SCBx_INTR_RX.BLOCKED fields to determine whether a failure occurred for a series of write or read operations.

23.4.5.2 Using SPI master to clock slave

In a normal SPI master mode transmission, the SCLK is generated only when the SCB is enabled and data is being transmitted. This can be changed to always generate a clock on the SCLK line while the SCB is enabled. This is used when the slave uses the SCLK for functional operations other than the SPI functionality. To enable this, write '1' to the SCBx_SPI_CTRL.SCLK_CONTINUOUS Register.

23.4.5.3 Oversampling and bit rate

SPI master mode:

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

The SCBx_CTRL.OVS Register specify the oversampling. The oversampling rate is calculated as the value in SCBx_CTRL.OVS Register + 1. In SPI master mode, the valid range for oversampling is 4 to 16, when MISO is used; if MISO is not used then the valid range is 2 to 16. The bit rate is calculated as follows.

Bit Rate = Input Clock/SCBx_CTRL.OVS

Hence, with an input clock of 100 MHz, the maximum bit rate is 25 Mbps with MISO, or 50 Mbps without MISO.

The numbers above indicate how fast the SCB hardware can run SCLK. It does not indicate that the master will be able to correctly receive data from a slave at those speeds. To determine that, the path delay of MISO must be calculated. It can be calculated using the following equation:

$$\frac{1}{2} * t_{SCLK} \geq t_{SCLK_PBC_D} + t_{DSO} + t_{MISO_PCB_D} + t_{DSI} \quad (7)$$

Where:

t_{SCLK} is the period of the SPI clock

$t_{SCLK_PCB_D}$ is the SCLK PCB delay from master to slave

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

$t_{MISO_PCB_D}$ is the MISO PCB delay from slave to master

t_{DSI} is the master setup time

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

After doing these calculations, if the desired speed cannot be achieved then consider using the MISO late sample feature of the SCB. MISO late sample tells the SCB to sample the incoming MISO signal on the next edge of SCLK, thus allowing for a one-half SCLK cycle more timing margin, see Figure 125.

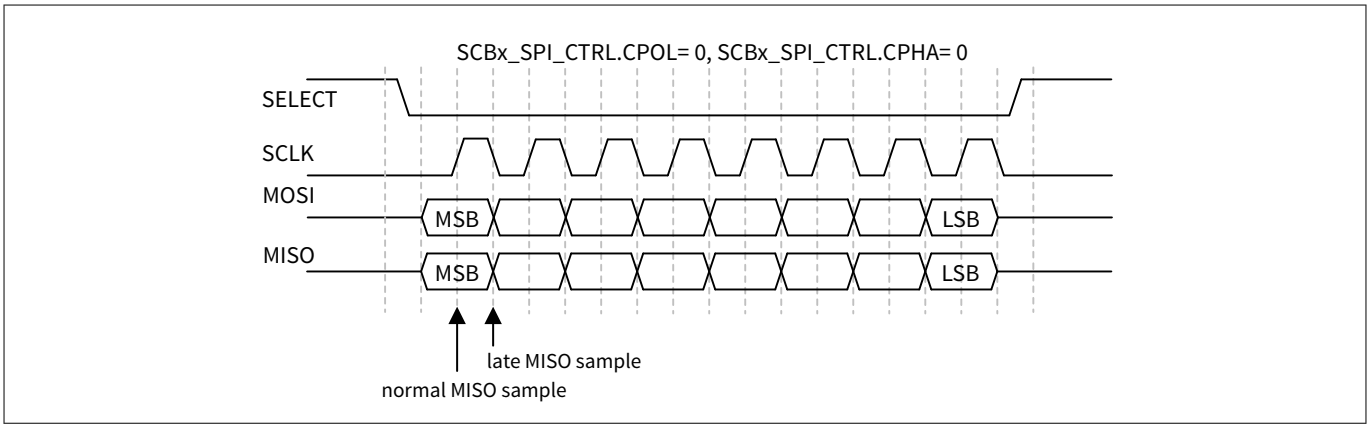


Figure 125 MISO sampling timing

This changes the equation to:

$$t_{SCLK} \geq t_{SCLK_PCB_D} + t_{DSO} + t_{MISO_PCB_D} + t_{DSI} \quad (8)$$

Because late sample allows for better timing, it is recommended to leave it enabled all the time.
The tDSI specification in the device datasheet assumes that the late sample is enabled.

***Note**: The SCBx_SPI_CTRL.LATE_MISO_SAMPLE is set to '0' by default.*

SPI slave mode:

In SPI slave mode, the SCBx_CTRL.OVS Register is not used. The data rate is determined by Equation 23-1 and Equation 23-2. Late MISO sample is determined by the external master and not by SCBx_SPI_CTRL.LATE_MISO_SAMPLE.

For XMC7000 MCUs, tDSO is given in the device datasheet. For internally-clocked mode, it is proportional to the frequency of the internal clock. For example, it may be 20 nsec + 3 × tCLK_SCB. Assuming 0 nsec PCB delays, and a 0 nsec external master tDSI Equation 28-1 can be rearranged to tCLK_SCB ≤ ((tSCLK) – 40 nsec)/6.

23.4.6 SPI master SELECT output timing control

The SPI master SELECT output signal “spi_select” timing is made variable. This applies to:

- The SELECT setup time (select active to SPI clock)
- The SELECT hold time (SPI clock to select inactive)
- The inter-data frame deselect time (select inactive to select active)

The following options can be selected for these delays:

- SELECT setup time (SCBx_SPI_CTRL.SSEL_SETUP_DEL Register):
  - When SCBx_SPI_CTRL.CPHA = 0: 0.75 or 1.75 SPI clock cycles
  - When SCBx_SPI_CTRL.CPHA = 1: 0.25 or 1.25 SPI clock cycles
- SELECT hold time (SCBx_SPI_CTRL.SSEL_HOLD_DEL register):
  - When SCBx_SPI_CTRL.CPHA = 0: 0.25 or 1.25 SPI clock cycles
  - When SCBx_SPI_CTRL.CPHA = 1: 0.75 or 1.75 SPI clock cycles

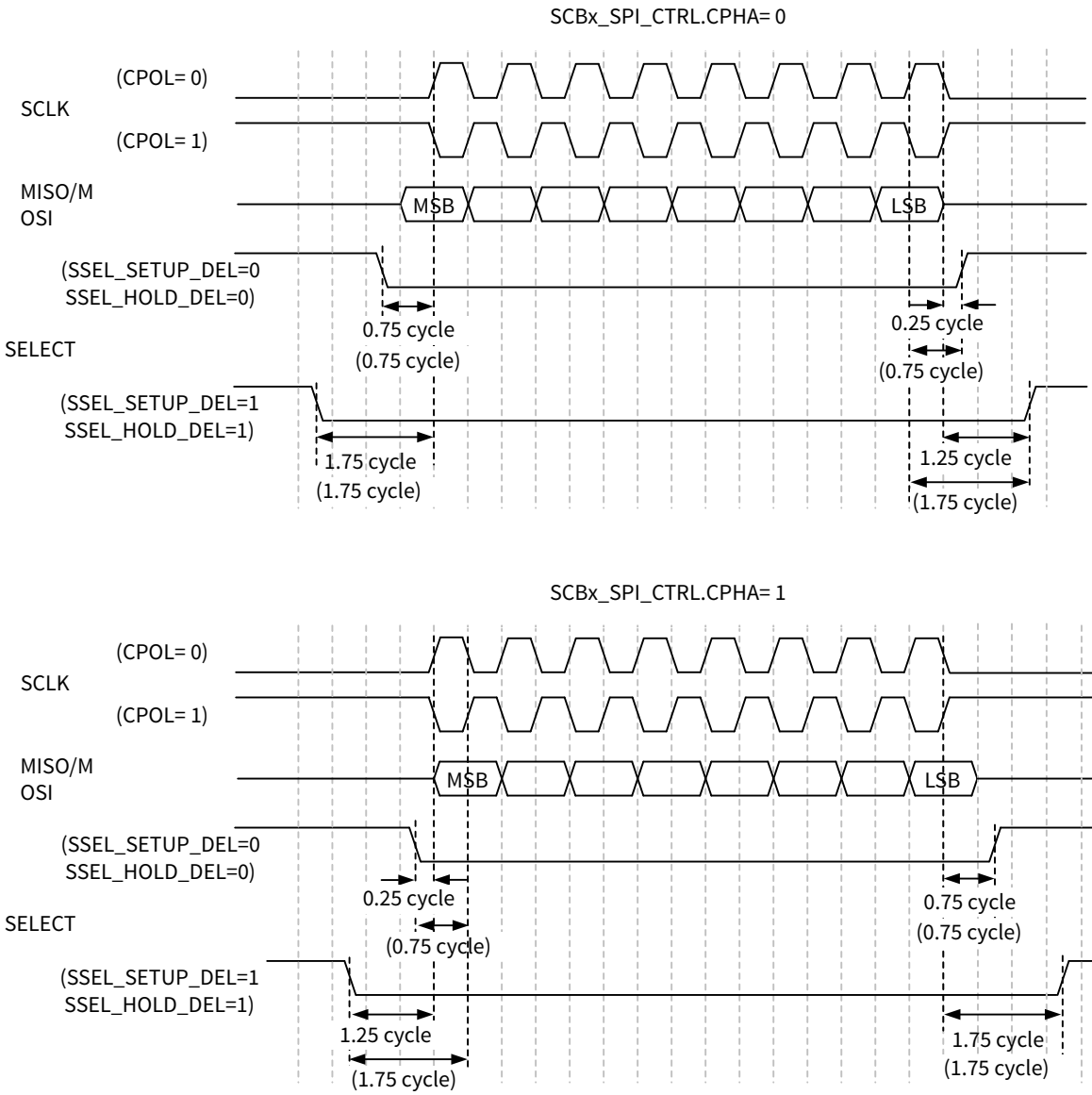


Figure 126 SELECT setup/hold delay

- INTER-FRAME deselect time (SCBx_SPI_CTRL.SSEL_INTER_FRAME_DEL Register):
  - 1.5 SPI clock cycles or 2.5 SPI clock cycles

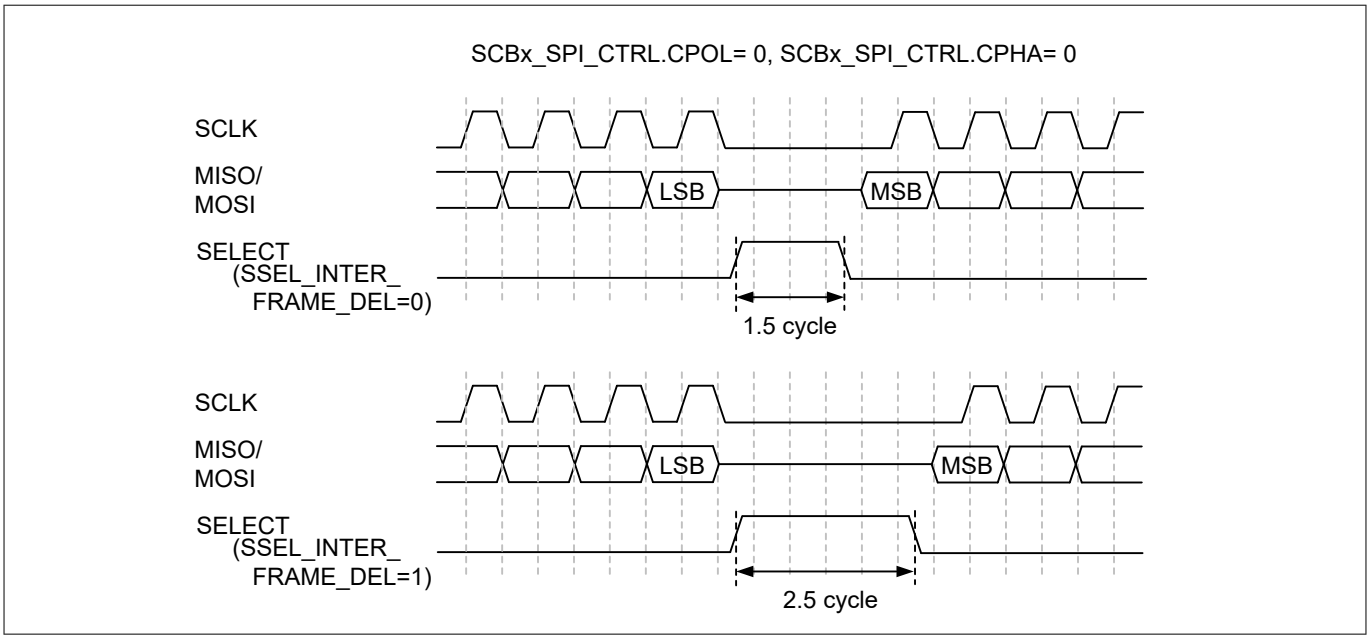


Figure 127 SELECT INTER-FRAME deselect time

23.4.7 SPI parity functionality

Parity functionality is added to SPI mode.

- This applies to the SPI master and SPI slave with internally-clocked operation.
- Parity functionality adds a parity bit to the data frame and is used to identify single-bit data frame errors. The parity bit directly follows the data frame bits.
- Parity functionality can be configured to be enabled or disabled using SCBx_SPI_TX_CTRL.PARITY_ENABLED and SCBx_SPI_RX_CTRL.PARITY_ENABLED individually.
- When transmitting, a parity bit can be inserted. When receiving, the parity bit can be checked. If parity fails, it is possible to select whether the received data is sent to the RX FIFO or is dropped and lost, using the SCBx_SPI_RX_CTRL.DROP_ON_PARITY_ERROR Register.
- Even and odd parity is supported (SCBx_SPI_TX_CTRL.PARITY, SCBx_SPI_RX_CTRL.PARITY).

23.4.8 Loop-back

In SPI Master mode, SCB supports internal loop-back from an output signal for MOSI to an input signal for MISO without affecting the information on the pins. It is configured using the SCBx_SPI_CTRL.LOOPBACK Register.

This loop-back is not supported in National Semiconductors mode.

23.4.9 Enabling and initializing SPI

The SPI must be programmed in the following order:

1. Program protocol specific information using the SCBx_SPI_CTRL Register. This includes selecting the submodes of the protocol (MODE), master-slave functionality (MASTER_MODE), one of four SELECT (SSEL), whether SELECT stays active for a whole transfer or just for each dataframe width (SSEL_CONTINUOUS), and SELECT polarity (SSEL_POLARITY0-3). EZSPI and CMD_RESP can be used with slave mode only.
2. Program the generic transmitter and receiver information using the SCBx_TX_CTRL and SCBx_RX_CTRL Registers:
   a. Specify the data frame width. This should always be 8 for EZSPI and CMD_RESP.
   b. Specify whether MSb or LSb is the first bit to be transmitted/received. This should always be MSb first for EZSPI and CMD_RESP.
3. Program the transmitter and receiver FIFOs using the SCBx_TX_FIFO_CTRL and SCBx_RX_FIFO_CTRL Registers respectively, as shown in SCBx_TX_FIFO_CTRL/SCBx_RX_FIFO_CTRL Registers. Only for FIFO mode:
   a. Set the trigger level (TRIGGER_LEVEL).
   b. Clear the transmitter and receiver FIFO and Shift Registers (CLEAR).
4. Enable the block (write a '1' to the SCBx_CTRL.ENABLED Register). After the block is enabled, control bits should not be changed. Changes should be made after disabling the block; for example, to modify the operation mode (from Motorola mode to TI mode) or to go from externally-clocked to internally-clocked operation. The change takes effect only after the block is re-enabled. Note that re-enabling the block causes reinitialization and the associated state is lost (for example, FIFO content).

23.4.10 I/O pad connection

23.4.10.1 SPI master

In SPI master mode, the SCB provides data transmit and data receive functionality. Figure 128 and Table 158 list the use of the I/O pads for the SPI master.

Typically, the strong drive mode (GPIO_PRTx_CFG.DRIVE_MODEy = 6) is used for output signals. When SCB is disabled, the respective out_en signals will be 0, so the output will be High-Z; to avoid High-Z state, do one of the following:

- Use GPIO to drive the output to idle level, or
- Use pull-up or pull-down drive modes with an internal pull-up/pull-down resistor (fixed resistance number), or
- Use the strong drive mode, using an external pull-up/pull-down resistor (flexible resistance number)

The internal and external pull-up/pull-down resistors have a negative impact on the maximum data rate.

For SPI MISO input in normal full-duplex mode, when the SPI slave device is not selected, its MISO output will be High-Z. If all SPI slave devices connected to SPI master are not selected, the MISO line will be High-Z. A pull-up resistor is needed on the MISO line to avoid High-Z state.

- High-impedance (High-Z) drive mode, using external pull-up/pull-down resistor
- Pull-up/pull-down drive mode, using internal pull-up/pull-down resistor

The internal and external pull-up/pull-down resistors have a negative impact on the maximum data rate.

Half-duplex mode is not supported, because it drives “strong pull-up, strong pull-down” in normal functional mode (when spi_mosi_out_en = 1).

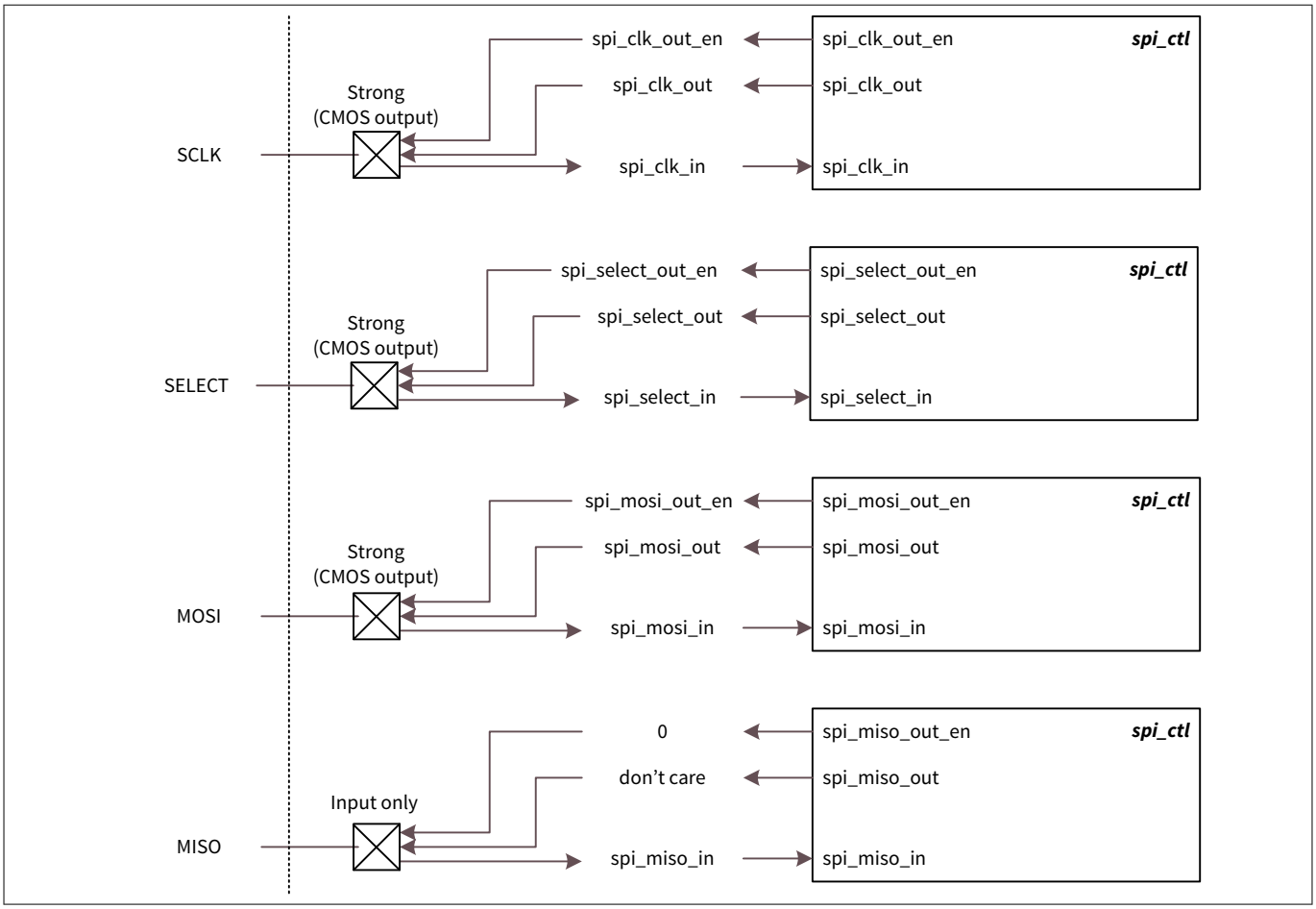


Figure 128 SPI master I/O pad connections

Table 158 SPI master I/O pad connection usage

I/O pads	Drive mode	On-chip I/O signals	Usage
SCLK	Strong (CMOS output)	spi_clk_out_en spi_clk_out	Transmit a clock signal
SELECT	Strong (CMOS output)	spi_select_out_en spi_select_out	Transmit a select signal
MOSI	Strong (CMOS output)	spi_mosi_out_en spi_mosi_out	Transmit a data element
MISO	Input only	spi_miso_in	Receive a data element

23.4.10.2 SPI slave

In SPI slave mode, the SCB provides data transmit and data receive functionality. Figure 129 and Table 159 list the use of I/O pads for SPI slave.

Typically, the strong drive mode (GPIO_PRTx_CFG.DRIVE_MODEy = 6) is used for output signals. When SCB is disabled, the respective out_en signals will be 0, so the output will be High-Z; to avoid High-Z state, do one of the following:

- Use GPIO to drive the output to idle level, or

- Use pull-up or pull-down drive modes with an internal pull-up/pull-down resistor (fixed resistance number), or
- Use the Strong drive mode, using an external pull-up/pull-down resistor (flexible resistance number)

The internal and external pull-up/pull-down resistors have a negative impact on the maximum data rate.

When SCBx_TX_CTRL.OPEN_DRAIN = 1, MOSI and MISO can be shorted together to work in half-duplex mode.

- The drive mode of MISO output can be open drain drives low only
- Users should add an external pull-up resistor on MISO line.

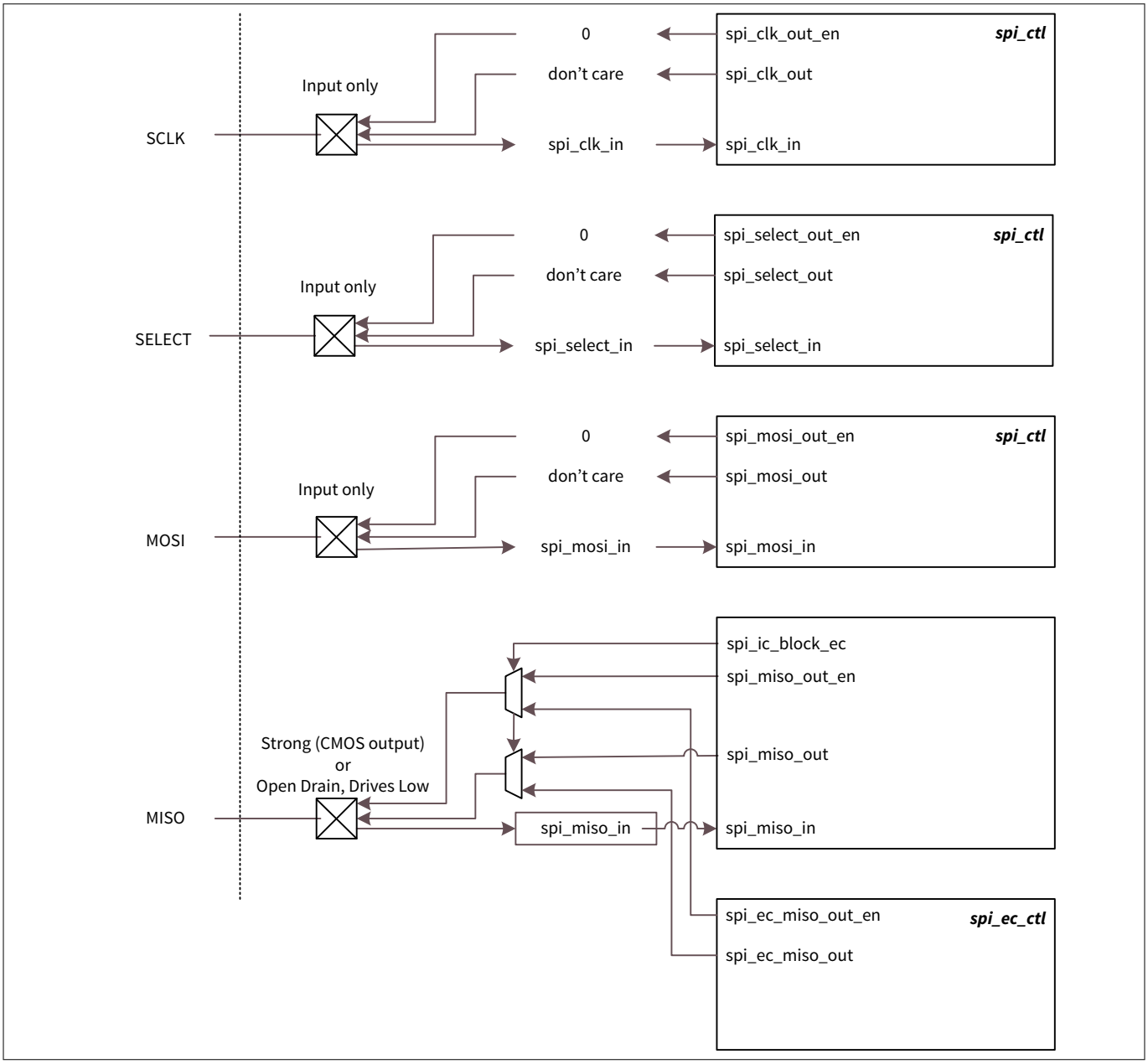


Figure 129 SPI slave I/O pad connections

Table 159 SPI slave I/O signal description

I/O pads	Drive mode	On-chip I/O signals	Usage
SCLK	Input only	spi_clk_in	Receive a clock signal
SELECT	Input only	spi_select_in	Receive a select signal
MOSI	Input only	spi_mosi_in	Receive a data element
MISO	Strong (CMOS output), or open drain drives low	spi_miso_out_en spi_miso_out	Transmit a data element

23.4.11 SPI Registers

The SPI interface is controlled using a set of 32-bit Control and Status Registers listed in Table 172. For more information on these registers, see the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual.

23.5 UART

The universal asynchronous receiver/transmitter (UART) protocol is an asynchronous serial interface protocol. UART communication is typically point-to-point. The UART interface consists of two signals:

- TX: Transmitter output
- RX: Receiver input

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

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

23.5.1 Features

- Supports UART protocol
  - Standard UART
  - Multi-processor mode
- SmartCard (ISO 7816) reader
- IrDA
- Data frame size programmable from 4 to 16 bits
- Programmable number of STOP bits, which can be set in terms of half bit periods between 1 and 4
- Parity support (odd and even parity)
- Median filter on RX input
- Programmable oversampling
- Start skipping
- FIFO mode operation only
- Local loop-back control

23.5.2 General description

Figure 130 illustrates a standard UART TX and RX.

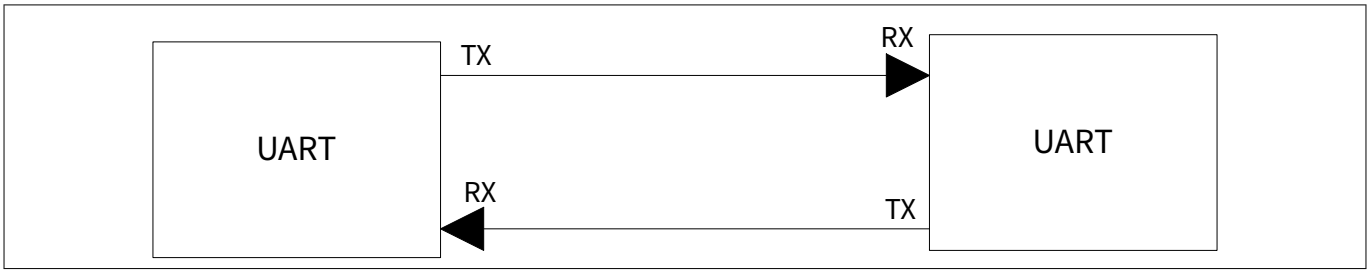


Figure 130 UART example

A typical UART transfer consists of a start bit followed by multiple data bits, optionally followed by a parity bit and finally completed by one or more stop bits. The start and stop bits indicate the start and end of data transmission. The parity bit is sent by the transmitter and is used by the receiver to detect single-bit errors. Because the interface does not have a clock (asynchronous), the transmitter and receiver use their own clocks; thus, the transmitter and receiver need to agree on the baud rate.

By default, UART supports a data frame width of eight bits. However, this can be configured to any value in the range of 4 to 9. This does not include start, stop, and parity bits. The number of stop bits can be in the range of 1 to 7 (SCBx_UART_TX_CTRL.STOP_BITS, SCBx_UART_RX_CTRL.STOP_BITS). 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 136 depicts the default configuration of the UART interface of the SCB.

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

***Note**: The behavior of UART when an error is detected in a start or stop period is determined by the SCBx_UART_RX_CTRL.DROP_ON_FRAME_ERROR Register.*

23.5.3 UART modes of operation

23.5.3.1 Standard protocol

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

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

The transition of a stop bit to a start bit is represented by a change from '1' to '0' on the TX line. This transition can be used by the receiver to synchronize with the transmitter clock. Synchronization at the start of each data transfer allows error-free transmission even in the presence of frequency drift between transmitter and receiver clocks. The required clock accuracy is dependent on the data transfer size. The stop period or the amount of stop bits between successive data transfers is typically agreed upon between transmitter and receiver, and is typically in the range of 1 to 3-bit transfer periods.

Figure 131 illustrates the UART protocol.

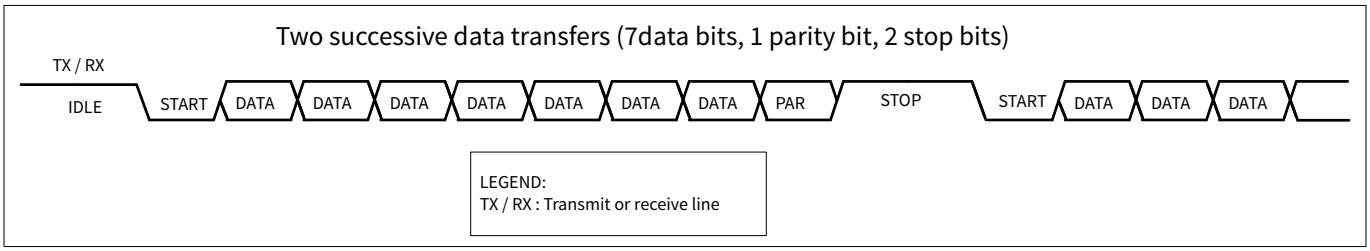


Figure 131 UART, standard protocol example

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 132 illustrates this.

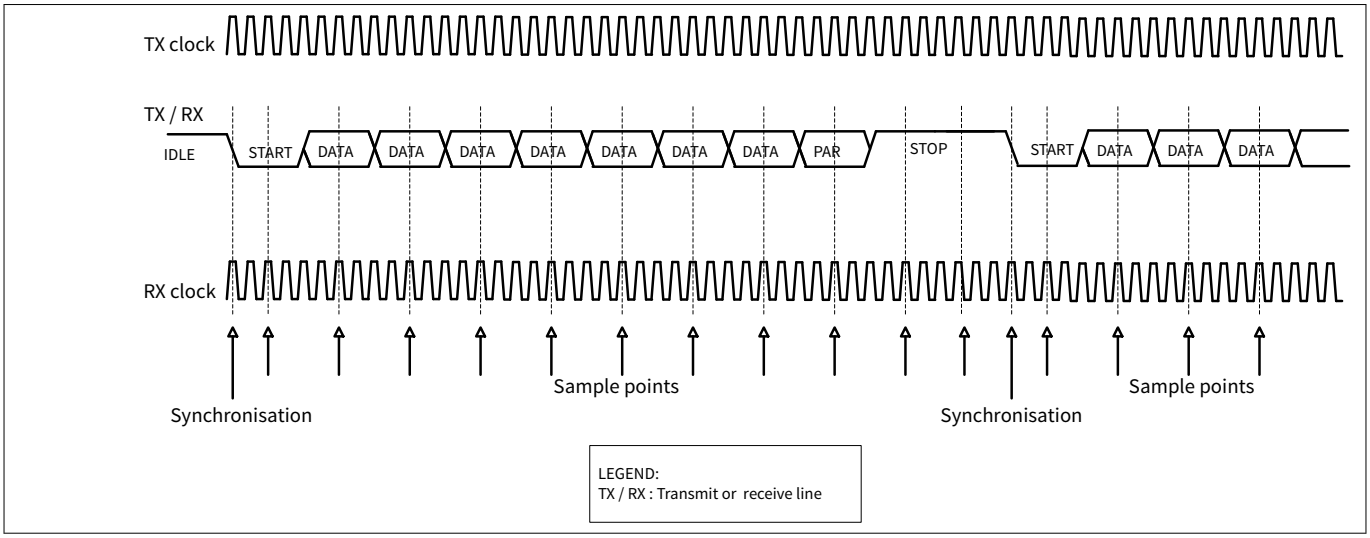


Figure 132 UART, standard protocol example (single sample)

Alternatively, three samples around the middle of the bit transfer period (on the receiver's clock) are used for a majority vote to increase accuracy; this is enabled by enabling the RX_CTRL.MEDIAN Register. Figure 133 illustrates this.

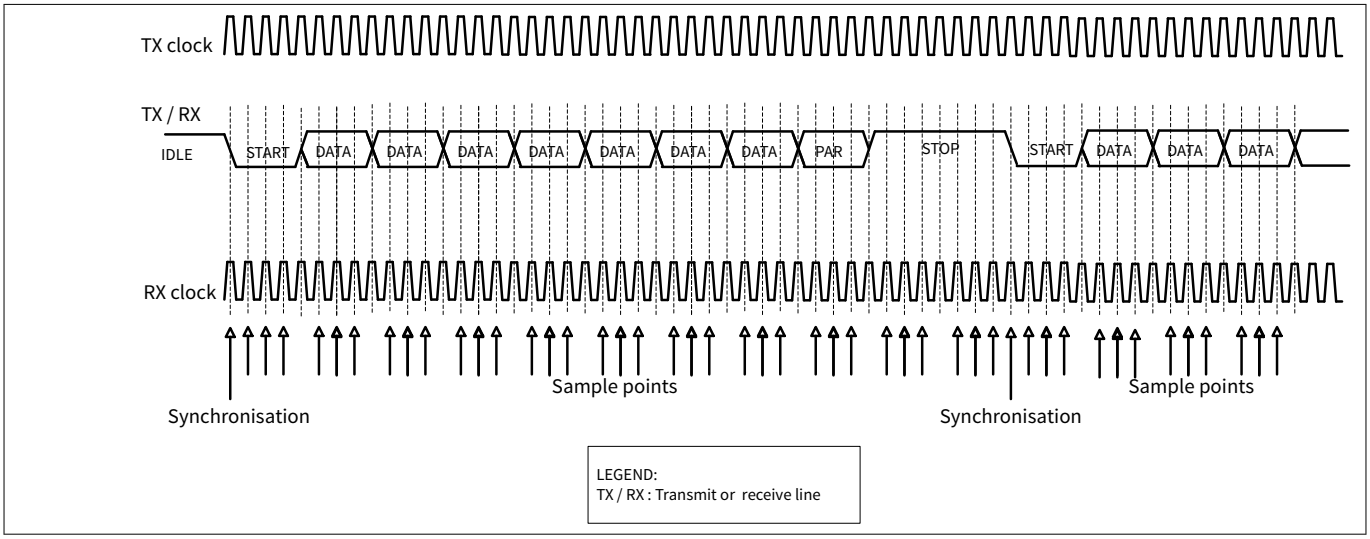


Figure 133 UART, standard protocol (multiple samples)

Parity

This functionality adds a parity bit to the data frame and is used to identify single-bit data frame errors. The parity bit directly follows the data frame bits. The transmitter calculates the parity bit (when SCBx_UART_TX_CTRL.PARITY_ENABLED is 1) from the data frame bits, such that data frame bits and parity bit have an even (SCBx_UART_TX_CTRL.PARITY is 0) or odd (SCBx_UART_TX_CTRL.PARITY is 1) parity. The receiver checks the parity bit (when SCBx_UART_RX_CTRL.PARITY_ENABLED is 1) from the received data frame bits, such that data frame bits and parity bit have an even (SCBx_UART_RX_CTRL.PARITY is '0') or odd (SCBx_UART_RX_CTRL.PARITY is '1') parity.

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

- Transmit functionality: SCBx_UART_TX_CTRL.PARITY and SCBx_UART_TX_CTRL.PARITY_ENABLED
- Receive functionality: SCBx_UART_RX_CTRL.PARITY and SCBx_UART_RX_CTRL.PARITY_ENABLED

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

The following figure illustrates the parity functionality (8-bit data frame).

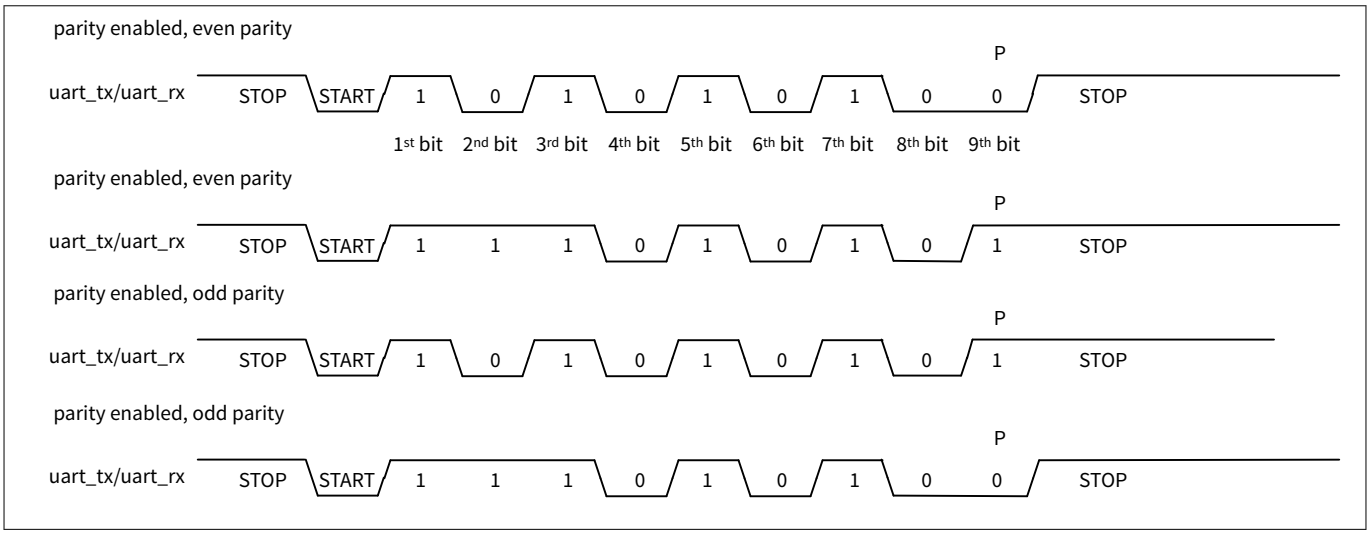


Figure 134 UART parity examples

Start skipping

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

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

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

Figure 135 illustrates the process.

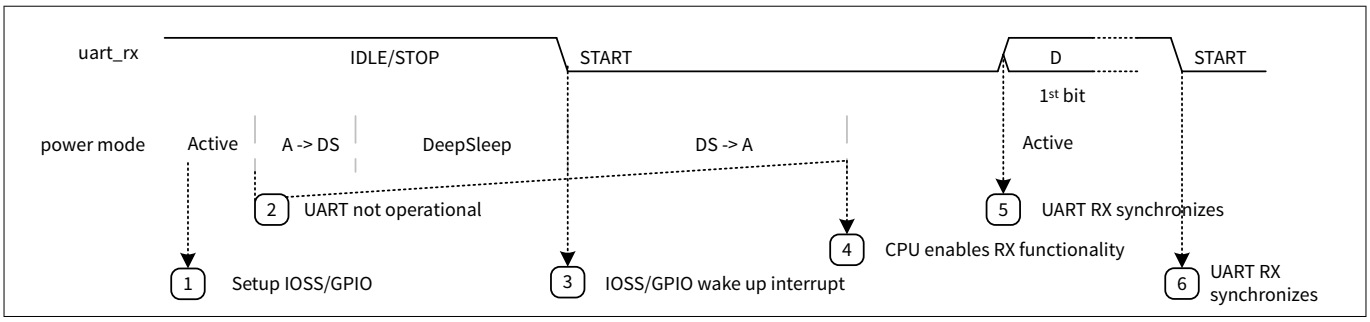


Figure 135 UART start skip and wakeup from deep sleep

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

In step 4 of the above process, it takes some time for the firmware to finish the wakeup interrupt routine and enable the UART receive functionality, before the block can detect the input rising edge on the UART RX line. If the above steps cannot be completed in less than 1 bit period, then it is recommended to first send a “dummy” byte to the device to wake it up before sending real UART data. In this case, the SCBx_UART_RX_CTRL.SKIP_START bit can be left as ’0’.

Break detection

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

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

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

When a break is detected, the SCBx_INTR_RX.BREAK_DETECT interrupt cause is set to '1'. Figure 136 illustrates a regular data frame and break frame (8-bit data frame, parity support, and a break width of 12-bit periods). When SCBx_UART_RX_CTRL.BREAK_LEVEL is set to ‘1’, idle line detection is possible. For example, after successive transfer of several UART data frames, an idle (high) level longer than normal data frame length (start+8data+1parity +1stop) indicates the end of this successive transfer.

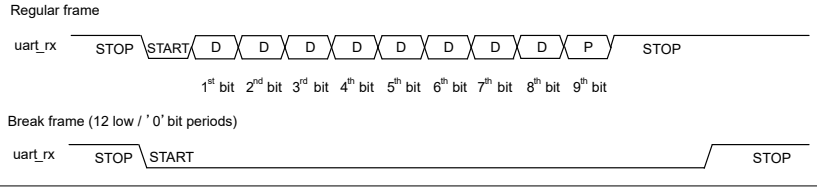


Figure 136 UART - regular frame and data frame

Flow control

The standard UART mode supports flow control. This modem flow control controls the pace at which the transmitter transfers data to the receiver. Modem flow control is enabled through the SCBx_UART_FLOW_CTRL.CTS_ENABLED register field. When this field is '0', the transmitter transfers data when

its TX FIFO is not empty. When '1', the transmitter transfers data when UART CTS line is active and its TX FIFO is not empty.

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

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

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

Typically, the UART side-band signals are active low. However, sometimes active high signaling is used. Therefore, the polarity of the side-band signals can be controlled using SCBx_UART_FLOW_CTRL.RTS_POLARITY and SCBx_UART_FLOW_CTRL.CTS_POLARITY bitfields. Figure 137 gives an overview of the flow control functionality.

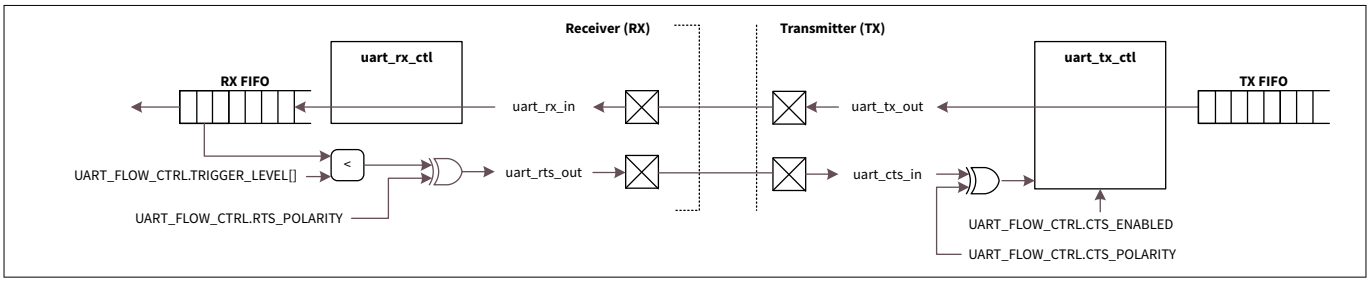


Figure 137 UART flow control connection

23.5.3.2 UART multi-processor mode

The UART_MP (multi-processor) mode is defined with single-master-multi-slave topology, as Figure 138 shows. This mode is also known as UART 9-bit protocol because the data field is nine bits wide. UART_MP is part of standard UART mode.

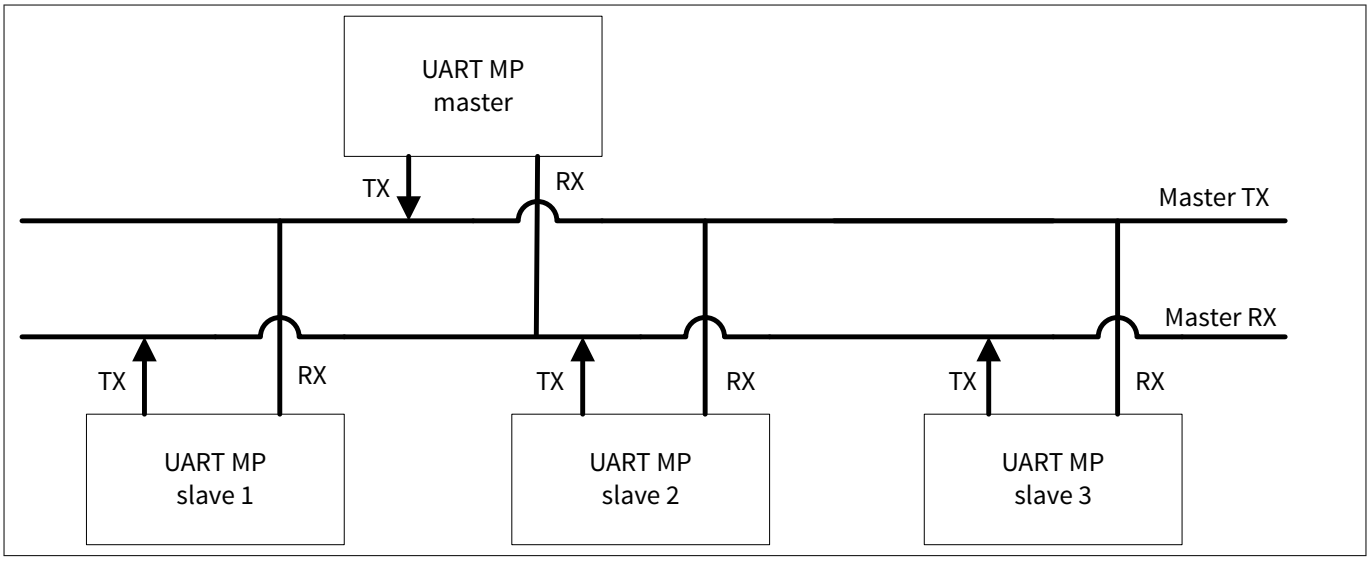


Figure 138 UART MP mode bus connections

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 139.
- Parity bit is disabled.

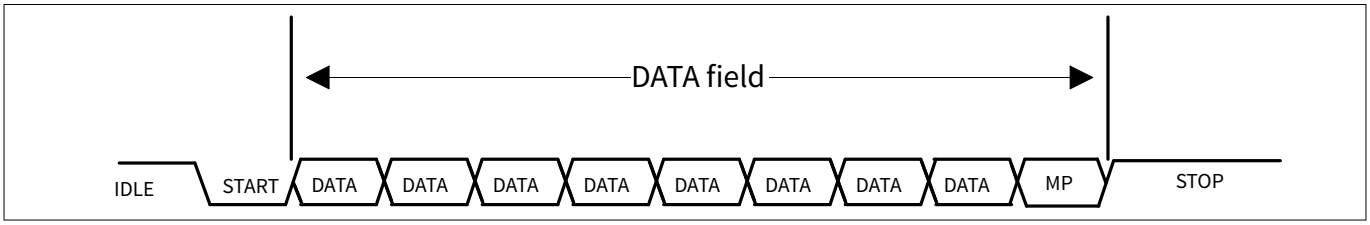


Figure 139 UART MP address and data frame

The SCB can be used either as a 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 SCBx_UART_RX_CTRL.MP_MODE 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 SCBx_CTRL.ADDR_ACCEPT Register is set to '1'. If received address does not match its own address, then the interface ignores the following data, until the next address is received for compare.

23.5.3.3 SmartCard (ISO 7816)

ISO 7816 is an 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 acknowledgment (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 (when SCBx_UART_TX_CTRL.RETRY_ON_NACK = 1). 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 140 illustrates the SmartCard protocol.

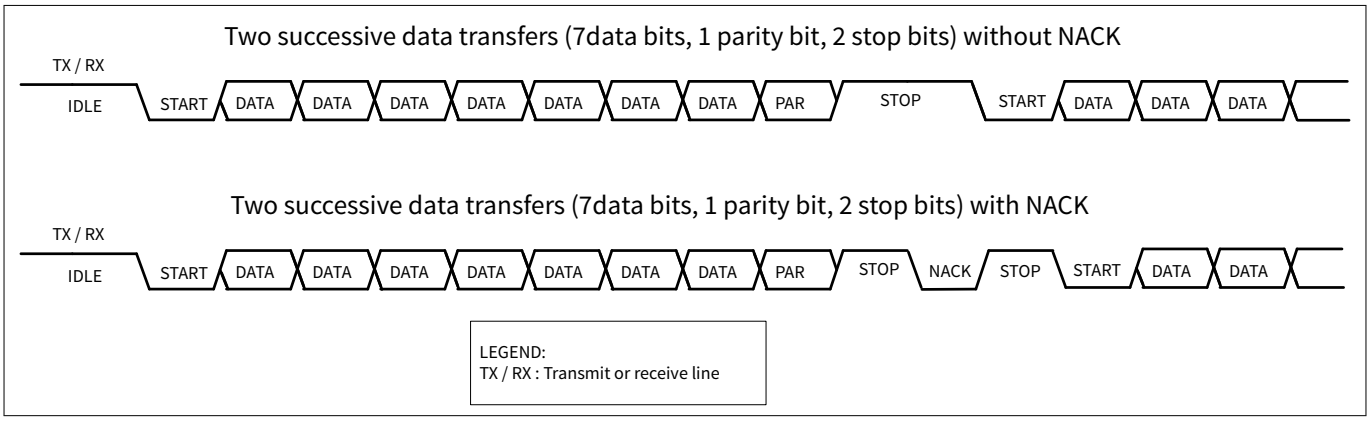


Figure 140 SmartCard example

The communication baud rate for ISO 7816 is given as:

Baud rate = f_{7816} × (D/F)

Where f_{7816} is the clock frequency, F is the clock rate conversion integer, and D is the baud rate adjustment integer.

By default, F = 372, D = 1, and maximum clock frequency is 5 MHz. Thus, maximum baud rate is 13.4 kbps. Typically, a 3.57-MHz clock is selected; the baud rate will then be 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 XMC7100 registers technical reference manual and XMC7200 registers technical reference manual.

1. Configure the SCB as UART interface by writing '10' to the SCBx_CTRL.MODE Register.
2. Configure the UART interface to operate as a Smart-Card protocol by writing '01' to the SCBx_UART_CTRL.MODE Register.
3. Follow steps 2 to 4 described in Enabling and initializing UART.

23.5.3.4 IrDA

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

The IrDA protocol adds a modulation scheme to the UART signaling. At the transmitter, bits are modulated. At the receiver, bits are demodulated. The modulation scheme uses a return-to-zero-Inverted (RZI) format. A bit value of '0' is signaled by a short '1' pulse on the line and a bit value of '1' is signaled by holding the line to '0'. For these data rates (≤ 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 SCBx_CTRL.OVS Register. The SCBx_UART_RX_CTRL.POLARITY Register can invert the incoming UART_RX line signal. In addition, the XMC7000 MCU SCB supports a low-power IrDA receiver mode, which allows it to detect pulses with a minimum width of 1.41 µs.

Different communication speeds under 115.2 kbps can be achieved by configuring the corresponding block clock frequency. Additional allowable rates are 2.4 kbps, 9.6 kbps, 19.2 kbps, 38.4 kbps, and 57.6 kbps. Figure 141 shows how a UART transfer is IrDA modulated.

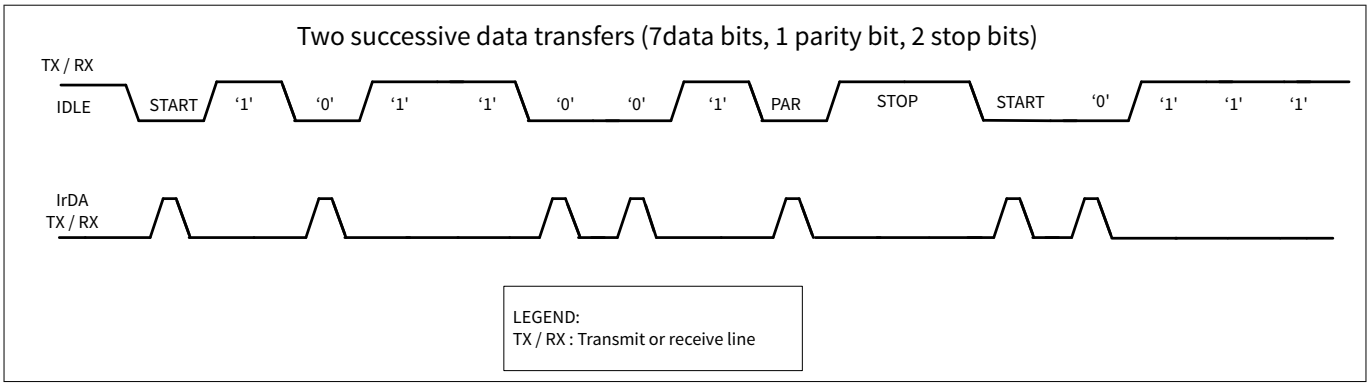


Figure 141 IrDA example

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 XMC7100 registers technical reference manual and XMC7200 registers technical reference manual.

1. Configure the SCB as UART interface by writing '10' to the SCBx_CTRL.MODE Register.
2. Configure the UART interface to operate as IrDA protocol by writing '10' to the SCBx_UART_CTRL.MODE Register.
3. Enable the median filter on the input interface line by writing '1' to SCBx_RX_CTRL.MEDIAN Register.
4. Configure the SCB as described in Enabling and initializing UART.

23.5.4 Clocking and oversampling

The UART protocol is implemented using the SCB input clock as an oversampled multiple of the baud rate. For example, to implement a 100-kHz UART, SCB input clock should be set to 1 MHz and the oversample factor set to '10'. The oversampling is set using the SCBx_CTRL.OVS Register field. The oversampling value is SCBx_CTRL.OVS + 1. In the UART standard sub-mode and the SmartCard sub-mode, the valid range for the SCBx_CTRL.OVS field is [7, 15].

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

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

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

The SCB input clock and the oversampling together determine the IrDA bit rate. Refer to the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual for more details on the SCBx_CTRL.OVS values for different baud rates.

23.5.5 Loop-back

SCB supports internal loop-back from an output signal for UART_TX and UART_RTS to an input signal for UART_RX and UART_CTS without affecting the information on the pins. It is configured using the SCBx_UART_CTRL.LOOPBACK Register.

23.5.6 Enabling and initializing UART

The UART must be programmed in the following order:

1. Program protocol specific information using the SCBx_UART_TX_CTRL, SCBx_UART_RX_CTRL, and SCBx_UART_FLOW_CTRL Registers. This includes selecting the submodes of the protocol, transmitter-receiver functionality, and so on.
2. Program the generic transmitter and receiver information using the SCBx_TX_CTRL and SCBx_RX_CTRL Registers.
   a. Specify the data frame width.
   b. Specify whether MSb or LSb is the first bit to be transmitted or received.
3. Program the transmitter and receiver FIFOs using the SCBx_TX_FIFO_CTRL and SCBx_RX_FIFO_CTRL Registers respectively.
   a. Set the trigger level (TRIGGER_LEVEL).
   b. Clear the transmitter and receiver FIFO and Shift Registers (CLEAR).
4. Enable the block (write a '1' to the SCBx_CTRL.ENABLED Register). After the block is enabled, control bits should not be changed. Changes should be made after disabling the block; for example, to modify the operation mode (from SmartCard to IrDA). The change takes effect only after the block is re-enabled. Note that re-enabling the block causes re-initialization and the associated state is lost (such as FIFO content).

23.5.7 I/O pad connection

23.5.7.1 Standard UART mode

Figure 142, Figure 143, Figure 144, and Table 160 list the use of the I/O pads for the standard UART mode.

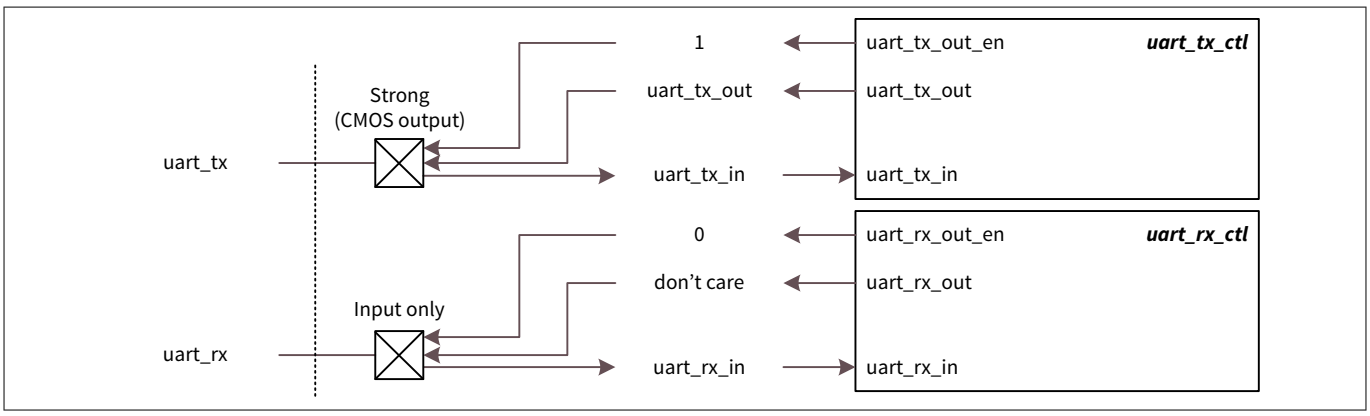


Figure 142 Standard UART mode, I/O pad connections

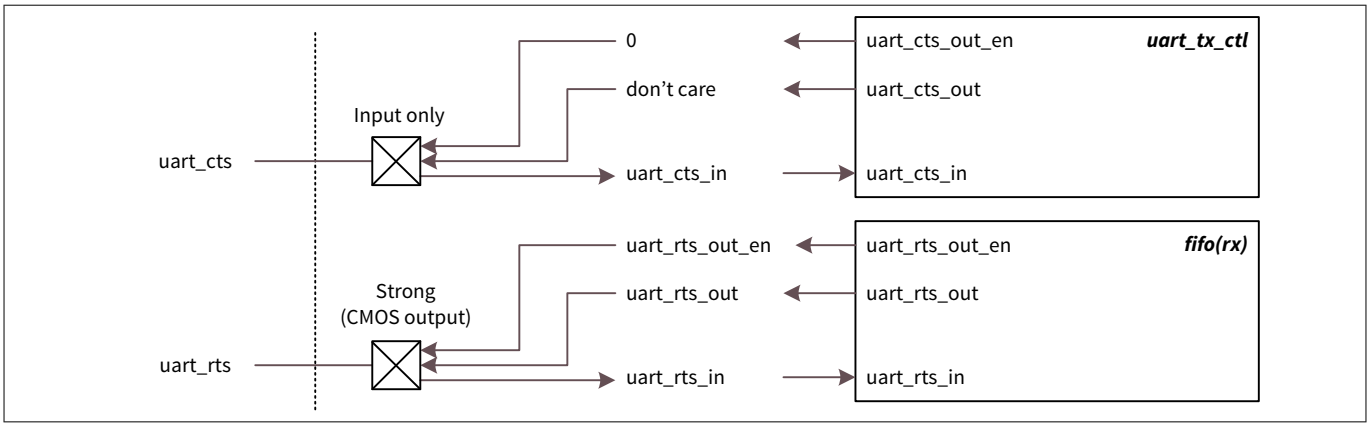


Figure 143 Standard UART mode, flow control I/O pad connection

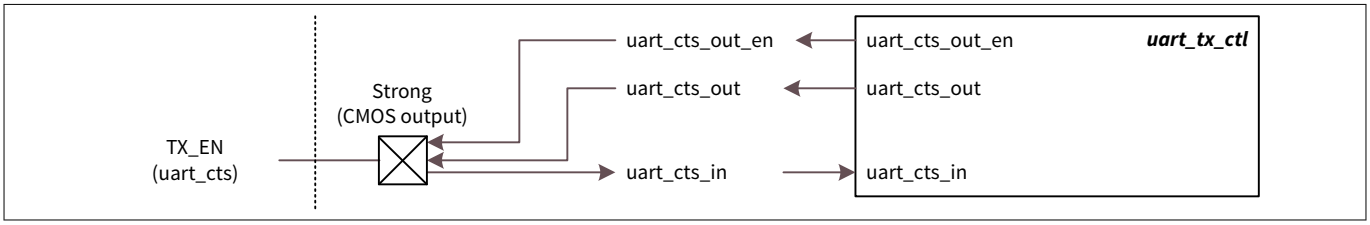


Figure 144 Standard UART mode, CTS Reused as TX_EN for RS485

Table 160 UART I/O pad connection usage

I/O pads	Drive mode	On-chip I/O signals	Usage
uart_tx	Strong (CMOS output)	uart_tx_out_en uart_tx_out	Transmit a data element
uart_rx	Input only	uart_rx_in	Receive a data element
uart_cts	Input only	Uart_cts_in	Indicate peer part readiness to receive data
uart_rts	Strong (CMOS output)	Uart_rts_out_en Uart_rts_out	Indicate DUT readiness to receive data
TX_EN (uart_cts)	Strong (CMOS output)	Uart_cts_out_en Uart_cts_out	Indicate DUT is transmitting data

23.5.7.2 SmartCard mode

Figure 145 and Table 161 list the use of the I/O pads for the SmartCard mode.

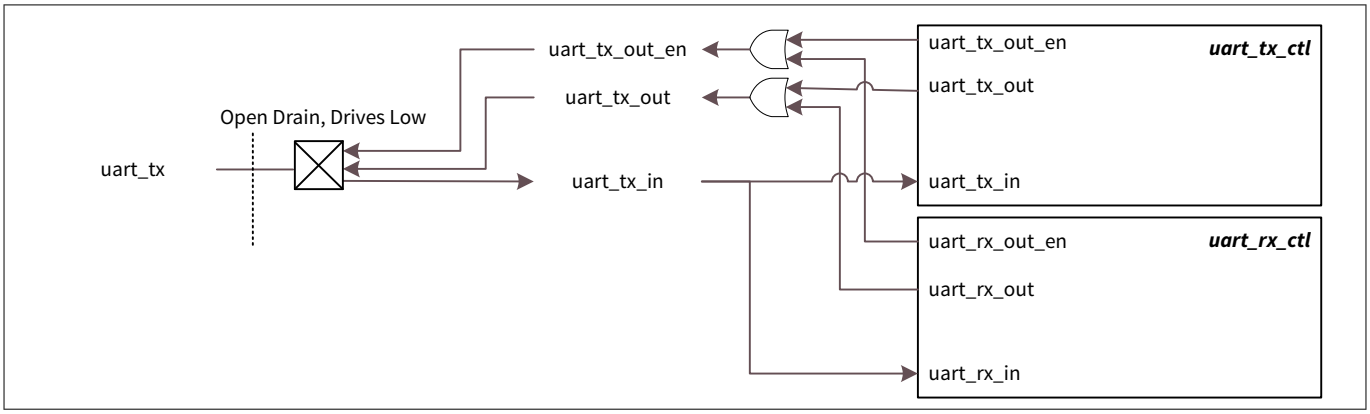


Figure 145 SmartCard mode I/O pad connections

Table 161 SmartCard mode I/O pad connections

I/O pads	Drive mode	On-chip I/O signals	Usage
uart_tx	Open drain drives low	uart_tx_in	Used to receive a data element. Receive a negative acknowledgment of a transmitted data element
		uart_tx_out_en uart_tx_out	Transmit a data element. Transmit a negative acknowledgment to a received data element.

23.5.7.3 IrDA mode

Figure 146 and Table 162 list the use of the I/O pads for IrDA mode.

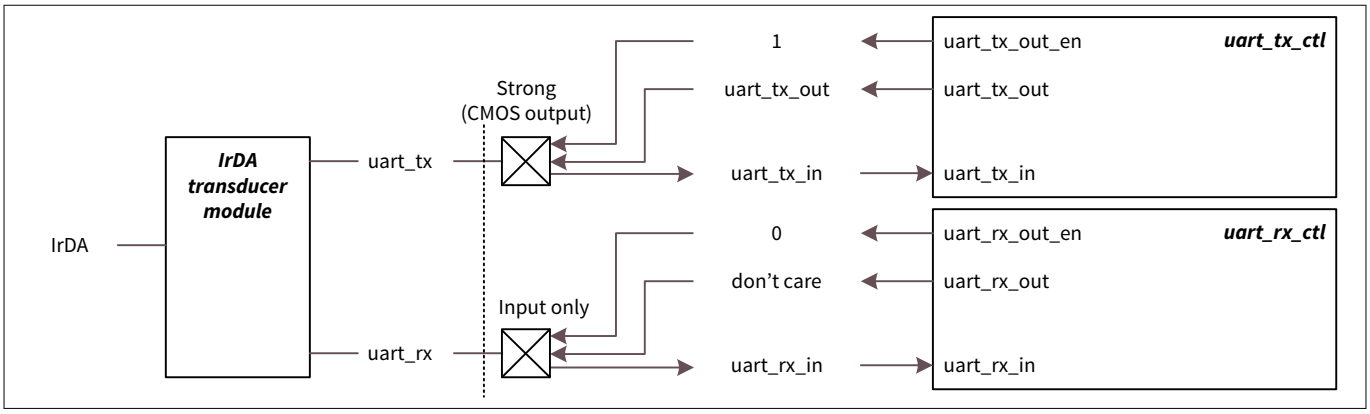


Figure 146 IrDA mode I/O pad connections

Table 162 IrDA mode I/O pad connections

I/O pads	Drive mode	On-chip I/O signals	Usage
uart_tx	Strong (CMOS output)	uart_tx_out_en uart_tx_out	Transmit a data element
uart_rx	Input only	uart_rx_in	Receive a data element

23.5.8 UART Registers

The UART interface is controlled using a set of 32-bit registers listed in Table 173. For more information on these registers, see the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual.

23.6 Inter integrated circuit (I^2C)

This section explains the I^2C implementation in the XMC7000 MCUs. For more information on the I^2C protocol specification, refer to the I^2C-bus specification available on the NXP website. In the XMC7000 MCU, all SCB blocks support both I^2C master and slave mode; only one SCB (SCB[0]) is available in deep sleep power mode and allows externally-clocked operations.

23.6.1 Features

This block supports the following features:

- Master, slave, and master/slave mode
- Standard-mode (100 kbps), fast-mode (400 kbps), and fast-mode plus (1000 kbps) data-rates
- 7-bit slave addressing
- Clock stretching
- Collision detection
- Programmable oversampling of I^2C clock signal (SCL)
- Auto ACK when RX FIFO not full, including address
- General address detection
- FIFO mode
- EZ and CMD_RESP modes
- Interrupts or polling CPU interface
- Analog glitch filter
- Local loop-back control

23.6.2 General description

Figure 147 illustrates an example of an I²C communication network.

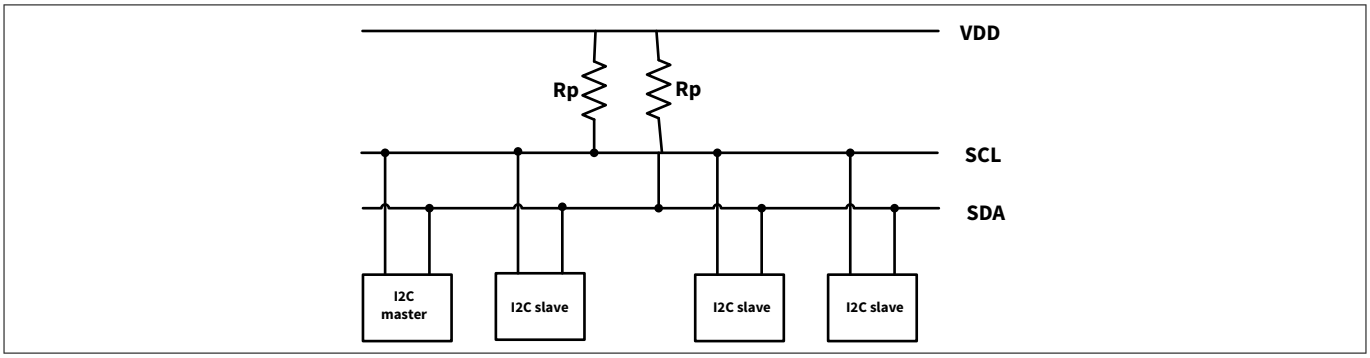


Figure 147 I²C interface block diagram

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.

23.6.3 Terms and definitions

Table 163 explains the commonly used terms in an I²C communication network.

Table 163 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.
Slave	The device addressed by a master.
Multi-master	More than one master can attempt to control the bus at the same time.
Arbitration	Procedure to ensure that, if more than one master simultaneously tries to control the bus, only one is allowed to do so and the winning message is not corrupted.
Synchronization	Procedure to synchronize the clock signals of two or more devices.

23.6.3.1 Clock stretching

When a slave device is not yet ready to process data, it may drive a '0' on the SCL line to hold it down. Due to the implementation of the I/O signal interface, the SCL line value will be '0', independent of the values that any other master or slave may be driving on the SCL line. This is known as clock stretching and is the only situation in which a slave drives the SCL line. The master device monitors the SCL line and detects it when it cannot generate a positive clock pulse ('1') on the SCL line. It then reacts by delaying the generation of a positive edge on the SCL line, effectively synchronizing with the slave device that is stretching the clock. The SCB on the XMC7000 MCU can and will stretch the clock.

23.6.3.2 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.

23.6.4 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. Table 164 illustrates the I²C modes of operation.

Table 164 I²C modes

Mode	Description
Slave	Slave only operation (default)
Master	Master only operation
Multi-master	Supports more than one master on the bus

Data transfer through the I²C bus follows a specific format. Table 165 lists some common bus events that are part of an I²C data transfer. The Write transfer and Read transfer sections explain the I²C bus bit format during data transfer.

Table 165 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. This indicates to the transmitter that the receiver received the byte properly.
NACK	The receiver does not pull the SDA line LOW and it remains HIGH during the HIGH period of clock pulse after the transmitter transmits each byte. This indicates to the transmitter that the receiver received the byte unsuccessfully.
Repeated START	START condition generated by master at the end of a transfer instead of a STOP condition
DATA	SDA status change while SCL is low (data changing), and no change while SCL is high (data valid).

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

Above START, STOP, ACK, NACK, and repeated START is controlled by the following registers. For more information, see the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual.

- SCBx_I2C_M_CMD.M_START
- SCBx_I2C_M_CMD.M_START_ON_IDLE
- SCBx_I2C_M_CMD.M_ACK

- SCBx_I2C_M_CMD.M_NACK
- SCBx_I2C_M_CMD.M_STOP
- SCBx_I2C_S_CMD.S_ACK
- SCBx_I2C_S_CMD.S_NACK

The behavior when received ACK or NACK can be configured by the following registers. For more information, see the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual.

- SCBx_I2C_CTRL.M_READY_DATA_ACK
- SCBx_I2C_CTRL.M_NOT_READY_DATA_NACK
- SCBx_I2C_CTRL.S_GENERAL_IGNORE
- SCBx_I2C_CTRL.S_READY_ADDR_ACK
- SCBx_I2C_CTRL.S_READY_DATA_ACK
- SCBx_I2C_CTRL.S_NOT_READY_ADDR_NACK
- SCBx_I2C_CTRL.S_NOT_READY_DATA_NACK

23.6.4.1 Write transfer

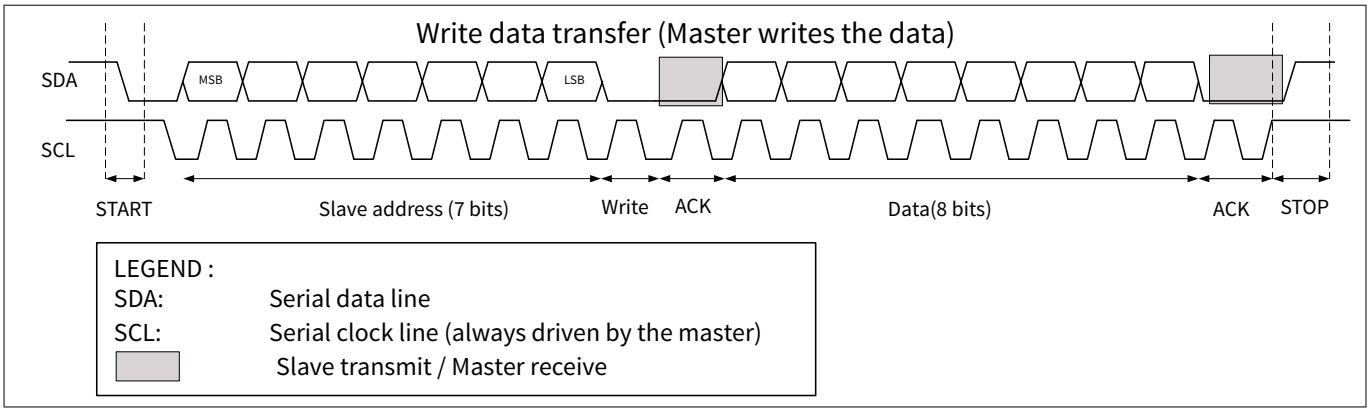


Figure 148 Master write data transfer

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

23.6.4.2 Read transfer

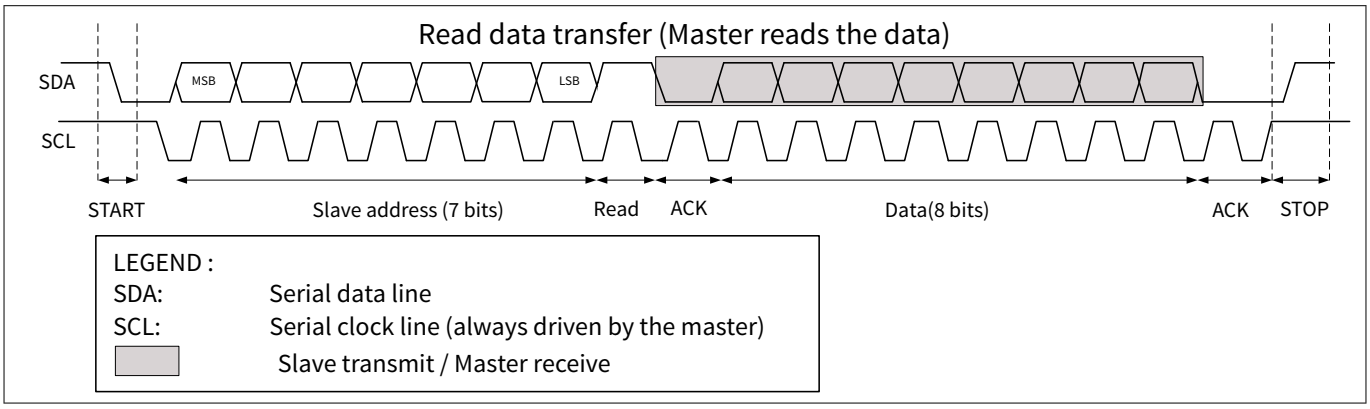


Figure 149 Master read data transfer

- 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 acknowledgment by pulling the data line low during the ninth bit time.
- If the slave address does not match with that of the connected slave device or if the addressed device does not want to acknowledge the request, a no acknowledgment (NACK) is transmitted by not pulling the SDA line low. The absence of an acknowledgment, results in an SDA line value of '1' due to the pull-up resistor implementation.
- If no acknowledgment is transmitted by the slave, the master may end the read transfer with a STOP event. The master can also generate a repeated START condition for a retry attempt.
- If the slave acknowledges the address, it starts transmitting data after the acknowledgment signal. The master transmits an acknowledgment to confirm the receipt of each data byte sent by the slave. Upon receipt of this acknowledgment, the addressed slave may transmit another data byte.
- The master can send a NACK signal to the slave to stop the slave from sending data bytes. This completes the read transfer.
- When the transfer is complete, the master generates a STOP condition.
- Individual data transfers (of one or more data elements) start with a START event and end with a STOP event. Combined data transfers consist of multiple individual transfers that are not separated by STOP events, but by repeated START events only.

23.6.5 I²C buffer modes

I²C can operate in three different buffered modes - FIFO, EZ, and CMD_RESP modes.The buffer is used in different ways in each of the modes. The following subsections explain each of these buffered modes in detail.

23.6.5.1 FIFO mode

The FIFO mode has a TX FIFO for the data being transmitted and an RX FIFO for the data being received. Each FIFO is constructed out of the SRAM buffer. The FIFOs are either 32 elements deep with 32-bit data elements or 64 elements deep with 16-bit data elements or 128 elements deep with 8-bit data elements. The width of the data elements are configured using the SCBx_CTRL.MEM_WIDTH. For I²C it is recommended to put the FIFO in BYTE mode because all transactions are a byte wide.

The FIFO mode operation is available only in active and sleep power modes, not in the deep sleep power mode. However, on the deep sleep-capable SCB the slave address can be used to wake the device from sleep.

Transmit and receive FIFOs allows write and read accesses. A write access to the transmit FIFO uses Register SCBx_TX_FIFO_WR. A read access from the receive FIFO uses Register SCBx_RX_FIFO_RD.

Transmit and receive FIFO status information is available through the SCBx_TX_FIFO_STATUS and SCBx_RX_FIFO_STATUS registers. It is possible to define a programmable threshold that indicates a number of FIFO entries, a trigger/event is generated when the following conditions are met:

- The transmit FIFO has a SCBx_TX_FIFO_CTRL.TRIGGER_LEVEL. A trigger/event is generated when number of entries in the transmit FIFO is less than SCBx_TX_FIFO_CTRL.TRIGGER_LEVEL.
- The receive FIFO has an SCBx_RX_FIFO_CTRL.TRIGGER_LEVEL. A trigger/event is generated when number of receive FIFO entries is greater than the SCBx_RX_FIFO_CTRL.TRIGGER_LEVEL.

Furthermore, several interrupt status bits are provided as well, which indicate if the FIFOs are full, empty, and so on.

Deep sleep to active transition

SCBx_CTRL.EC_AM_MODE = 1, SCBx_CTRL.EC_OP_MODE = 0, FIFO mode.

Master write:

- SCBx_I2C_CTRL.S_NOT_READY_ADDR_NACK = 0, SCBx_I2C_CTRL.S_READY_ADDR_ACK = 1. The clock is stretched until the internally-clocked logic takes over, at which point the address is ACK'd and the master can start writing data. Before going to deep sleep, CLK_SCB needs to be disabled. Upon wake up from deep sleep CLK_SCB must be re-enabled; this is when the clock stretch will be released.
- SCBx_I2C_CTRL.S_NOT_READY_ADDR_NACK = 0, SCBx_I2C_CTRL.S_READY_ADDR_ACK = 0. The clock is stretched until the internally-clocked logic takes over and the CPU writes either SCBx_I2C_S_CMD.S_ACK, or SCBx_I2C_S_CMD.S_NACK. Before going to deep sleep CLK_SCB needs to be disabled. Upon wake up from deep sleep CLK_SCB must be re-enabled, do this before setting SCBx_I2C_S_CMD.S_ACK or SCBx_I2C_S_CMD.S_NACK.
- SCBx_I2C_CTRL.S_NOT_READY_ADDR_NACK = 1, SCBx_I2C_CTRL.S_READY_ADDR_ACK = x. The incoming address is NACK'd until the internally-clocked logic takes over. When the internally-clocked logic takes over, there is no guarantee that the internal clock will be at the correct frequency due to PLL/FLL locking times. This may lead to incorrect timing on the I^2C bus for the ACK/NACK. To avoid this disable CLK_SCB before going to deep sleep, and then re-enable after the PLL/FLL have stabilized.

Master read:

- SCBx_I2C_CTRL.S_NOT_READY_ADDR_NACK = 0, SCBx_I2C_CTRL.S_READY_ADDR_ACK = x. The incoming address is stretched until the internally-clocked logic takes over and the CPU writes data into the TX FIFO. Before going to deep sleep CLK_SCB needs to be disabled. Upon wake up from deep sleep CLK_SCB must be re-enabled before writing data into the TX FIFO.
- SCBx_I2C_CTRL.S_NOT_READY_ADDR_NACK = 1, SCBx_I2C_CTRL.S_READY_ADDR_ACK = x. The incoming address is NACK'd until the internally-clocked logic takes over. When this happens, there is no guarantee that the internal clock will be at the correct frequency due to PLL/FLL locking times. This may lead to incorrect timing on the I^2C bus for the ACK/NACK. To avoid this, disable CLK_SCB before going to deep sleep, and then re-enable after the PLL/FLL have stabilized.

23.6.5.2 EZI2C mode

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

The EZI2C protocol defines a single memory buffer with an 8-bit address that indexes the buffer (256-entry array of 8-bit per entry is supported) located on the slave device. The EZ address is used to address these 256 locations. The CPU writes and reads to the memory buffer through the EZ_DATA Registers. These accesses are word accesses, but only the least significant byte of the word is used.

The slave interface accesses the memory buffer using the current address. At the start of a transfer (I2C START/ RESTART), the base address is copied to the current address. A data element write or read operation is to the current address location. After the access, the current address is incremented by '1'.

If the current address equals the last memory buffer address (255), the current address is not incremented. Subsequent write accesses will overwrite any previously written value at the last buffer address. Subsequent read accesses will continue to provide the (same) read value at the last buffer address. The bus master should be aware of the memory buffer capacity in EZ mode.

The I^2C base and current addresses are provided through I2C_STATUS. At the end of a transfer (I^2C), the difference between the base and current addresses indicates how many read or write accesses were performed. The block provides interrupt cause fields to identify the end of a transfer. EZ mode operation is available in active, sleep, and deep sleep power modes. In XMC7000 MCUs, only the deep sleep-capable SCB block operate in EZI2C mode.

EZI2C distinguishes three operation phases:

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

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

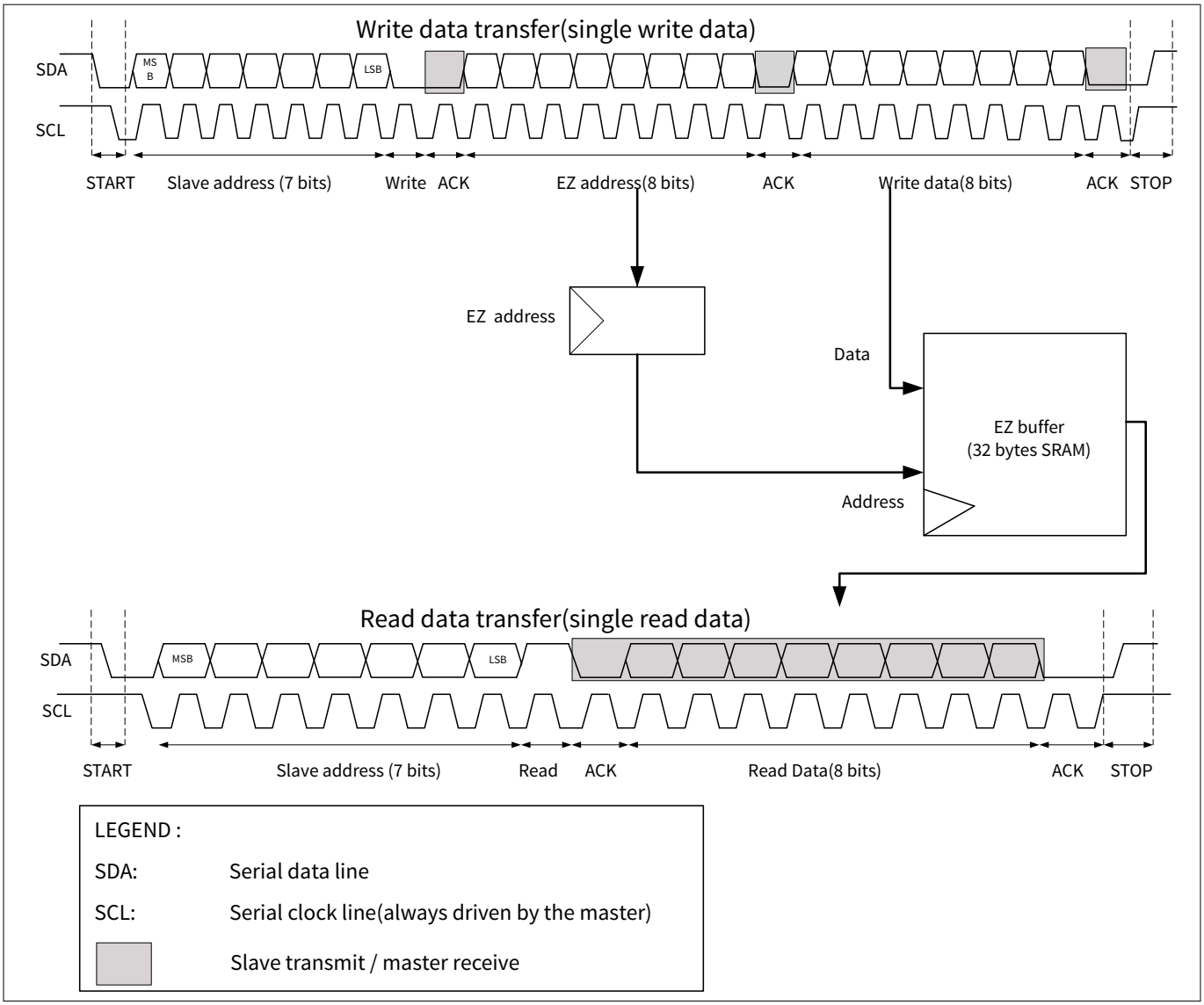


Figure 150 EZI2C write and read data transfer

Deep sleep to active transition

SCBx_CTRL.EC_AM_MODE = 1, SCBx_CTRL.EC_OP_MODE = 0, EZ mode.

- SCBx_I2C_CTRL.S_NOT_READY_ADDR_NACK = 0, SCBx_I2C_CTRL.S_READY_ADDR_ACK = 1. The clock is stretched until the internally-clocked logic takes over at which point the address is ACK'd and master can start writing data. Before going to deep sleep CLK_SCB needs to be disabled. Upon wake up from deep sleep CLK_SCB must be re-enabled this is when the clock stretch will be released.
- SCBx_I2C_CTRL.S_NOT_READY_ADDR_NACK = 1, SCBx_I2C_CTRL.S_READY_ADDR_ACK = x. The incoming address is NACK'd until the internally-clocked logic takes over. When this happens, there is no guarantee that the internal clock will be at the correct frequency due to PLL/FLL locking times. To avoid this, disable CLK_SCB before going to deep sleep, and then re-enable after the PLL/FLL have stabilized.

23.6.5.3 Command-response mode

In the XMC7000 MCU, only the deep sleep-capable SCB supports the command-response mode. This mode has a single memory buffer, a base read address, a current read address, a base write address, and a current write address that are used to index the memory buffer. The base addresses are provided by the CPU. The current

addresses are used by the slave to index the memory buffer for sequential accesses of the memory buffer. The memory buffer holds 256 8-bit data elements. The base and current addresses are in the range [0 to 255].

The CPU writes and reads to the memory buffer through the SCBx_EZ_DATA Registers. These are word accesses, but only the least significant byte of the word is used.

The slave interface accesses the memory buffer using the current addresses. At the start of a write transfer (I2C START/RESTART), the base write address is copied to the current write address. A data element write is to the current write address location. After the write access, the current address is incremented by '1'. At the start of a read transfer, the base read address is copied to the current read address. A data element read is to the current read address location. After the read data element is transmitted, the current read address is incremented by '1'.

If the current addresses equal the last memory buffer address (255), the current addresses are not incremented. Subsequent write accesses will overwrite any previously written value at the last buffer address. Subsequent read accesses will continue to provide the (same) read value at the last buffer address. The bus master should be aware of the memory buffer capacity in command-response mode.

The base addresses are provided through SCBx_CMD_RESP_CTRL.BASE_RD_ADDR and SCBx_CMD_RESP_CTRL.BASE_WR_ADDR. The current addresses are provided through SCBx_CMD_RESP_STATUS.CURR_RD_ADDR and SCBx_CMD_RESP_STATUS.CURR_WR_ADDR. At the end of a transfer (I^2CSTOP), the difference between a base and current address indicates how many read/write accesses were performed. The block provides interrupt cause fields to identify the end of a transfer. Command-response mode operation is available in active, sleep, and deep sleep power modes. The command-response mode has two phases of operation:

- Write phase - The write phase begins with a START/RESTART followed by the slave address with read/write bit set to '0' indicating a write. The slave's current write address is set to the slave's base write address. Received data elements are written to the current write address memory location. After each memory write, the current write address is incremented.
- Read phase - The read phase begins with a START/RESTART followed by the slave address with read/write bit set to '1' indicating a read. The slave's current read address is set to the slave's base read address. Transmitted data elements are read from the current address memory location. After each read data element is transferred, the current read address is incremented.

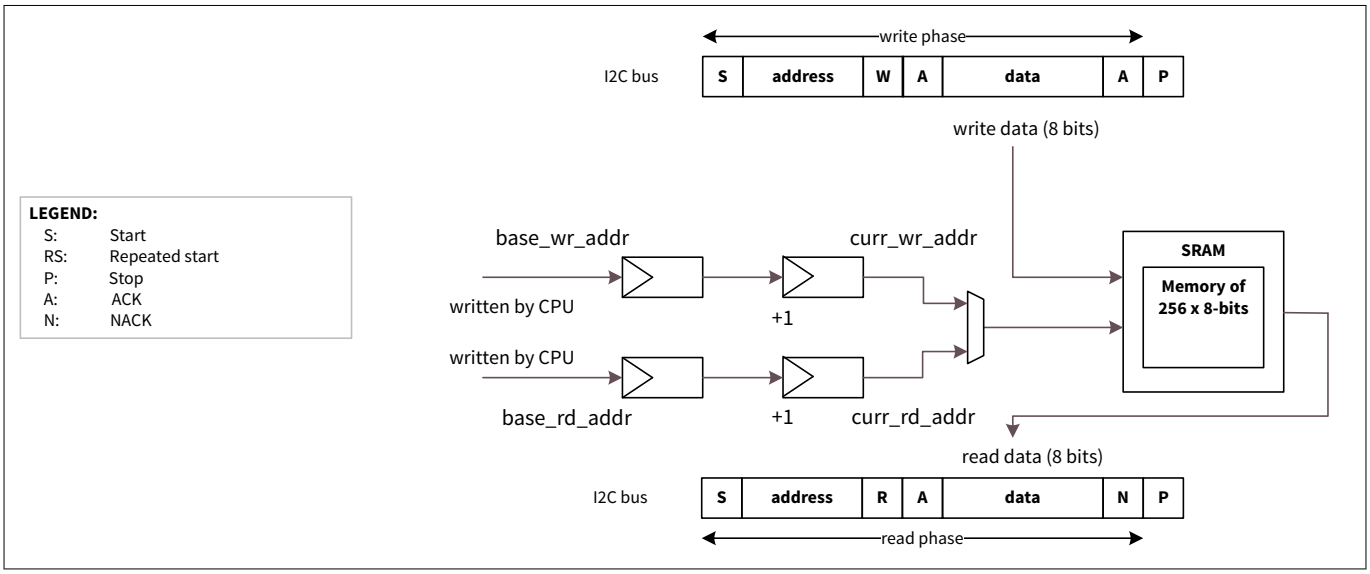


Figure 151 I^2C command-response mode

Note that a slave's base addresses are updated by the CPU and not by the master.

23.6.6 Clocking and oversampling

The SCB I²C supports both internally and externally-clocked operation modes. SCBx_CTRL.EC_AM_MODE and SCBx_CTRL_EC_OP_MODE Register determine the SCB clock mode. SCBx_CTRL.EC_AM_MODE indicates whether I²C address matching is clocked internally (0) or externally (1). I²C address matching comprises the first part of the I²C protocol. SCBx_CTRL.EC_OP_MODE indicates whether the rest of the protocol operation (besides I²C address matching) is clocked internally (0) or externally (1). The externally-clocked mode of operation is supported only in the I²C slave mode.

An internally-clocked operation uses the programmable clock dividers. For more information on system clocking, see the Clocking system. The internally-clocked mode does not support the command-response mode.

***Note**: In the XMC7000 MCUs, only one SCB supports externally-clocked mode of operation.*

The SCBx_CTRL.EC_AM_MODE and SCBx_CTRL.EC_OP_MODE can be configured in the following ways.

- SCBx_CTRL.EC_AM_MODE is '0' and SCBx_CTRL.EC_OP_MODE is '0': Use this configuration when only active mode functionality is required.
  - FIFO mode: Supported
  - EZ mode: Supported
  - Command-response mode: Not supported. The slave NACKs every slave address.
- SCBx_CTRL.EC_AM_MODE is '1' and SCBx_CTRL.EC_OP_MODE is '0': Use this configuration when both active and deep sleep functionality are required. This configuration relies on the externally-clocked functionality for the I²C address matching and relies on the internally-clocked functionality to access the memory buffer. The “handover” from external to internal functionality relies either on an ACK/NACK or clock stretching scheme. The former may result in termination of the current transfer and relies on a master retry. The latter stretches the current transfer after a matching address is received. This mode requires the master to support either NACK generation (and retry) or clock stretching. When the I²C address is matched, SCBx_INTR_I2C_EC.WAKE_UP is set to '1'. The associated deep sleep functionality interrupt brings the system into active power mode.
  - FIFO mode: See Deep sleep to active transition
  - EZ mode: See Deep sleep to active transition
  - CMD_RESP mode: Not supported. The slave NACKs every slave address.
- SCBx_CTRL.EC_AM_MODE is '1' and SCBx_CTRL.EC_OP_MODE is '1'. Use this mode when both active and deep sleep functionality are required. This mode may cause a “denial of service” for memory buffer accesses made by the CPU. When the slave is selected, SCBx_INTR_I2C_EC.WAKE_UP is set to '1'. The associated deep sleep functionality interrupt brings the system into active power mode. When the slave is deselected, SCBx_INTR_I2C_EC.EZ_STOP and SCBx_INTR_I2C_EC.EZ_WRITE_STOP are set to '1'.
  - FIFO mode: Not supported
  - EZ mode: Supported
  - CMD_RESP mode: Supported

Table 166 Clock configuration and mode support

Mode	SCBx_CTRL.EC_AM_MODE is '0'; SCBx_CTRL.EC_OP_MODE is '0'	SCBx_CTRL.EC_AM_MODE is '1'; SCBx_CTRL.EC_OP_MODE is '0'	SCBx_CTRL.EC_AM_MODE is '1'; SCBx_CTRL.EC_OP_MODE is '1'
FIFO mode	Yes	Yes	No
EZ mode	Yes	Yes	Yes

(table continues...)

Table 166 (continued) Clock configuration and mode support

Mode	SCBx_CTRL.EC_AM_MODE is '0'; SCBx_CTRL.EC_OP_MODE is '0'	SCBx_CTRL.EC_AM_MODE is '1'; SCBx_CTRL.EC_OP_MODE is '0'	SCBx_CTRL.EC_AM_MODE is '1'; SCBx_CTRL.EC_OP_MODE is '1'
CMD_RESP mode	No	No	Yes

An externally-clocked operation uses a clock provided by the serial interface. The externally-clocked mode does not support FIFO mode. If SCBx_CTRL.EC_OP_MODE is '1', the external interface logic accesses the memory buffer on the external interface clock (I^2C SCL). This allows for EZ and CMD_RESP mode functionality in active and deep sleep power modes.

In active system power mode, the memory buffer requires arbitration between external interface logic (on I^2C SCL) and the CPU interface logic (on system peripheral clock). This arbitration always gives the highest priority to the external interface logic (host accesses). The external interface logic takes one serial interface clock/bit periods for the I^2C. During this period, the internal logic is denied service to the memory buffer. The XMC7000 MCU provides two programmable options to address this “denial of service”:

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

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

If a series of read or write accesses is performed and SCBx_CTRL.BLOCK is '0', a failure is detected by comparing the “logical-or” of all read values to 0xFFFF:FFFF and checking the SCBx_INTR_TX.BLOCKED and SCBx_INTR_RX.BLOCKED fields to determine whether a failure occurred for a (series of) write or read operation(s).

***Note**: In XMC7000 MCUs, only one SCB supports externally-clocked mode of operation.*

23.6.6.1 Glitch filtering

The XMC7000 MCU SCB I^2C has analog and digital glitch filters. Analog glitch filters are applied on the i2c_scl_in and i2c_sda_in input signals (AF_in) to filter glitches of up to 50 ns. An analog glitch filter is also applied on the i2c_sda_out output signal (AF_out). Analog glitch filters are enabled and disabled in the SCBx_I2C_CFG Register. Do not change the _TRIM bitfields, only change the _SEL bitfields in this Register.

Digital glitch filters are applied on the i2c_scl_in and i2c_sda_in input signals (DF_in). The digital glitch filter is enabled in the SCBx_RX_CTRL.MEDIAN.

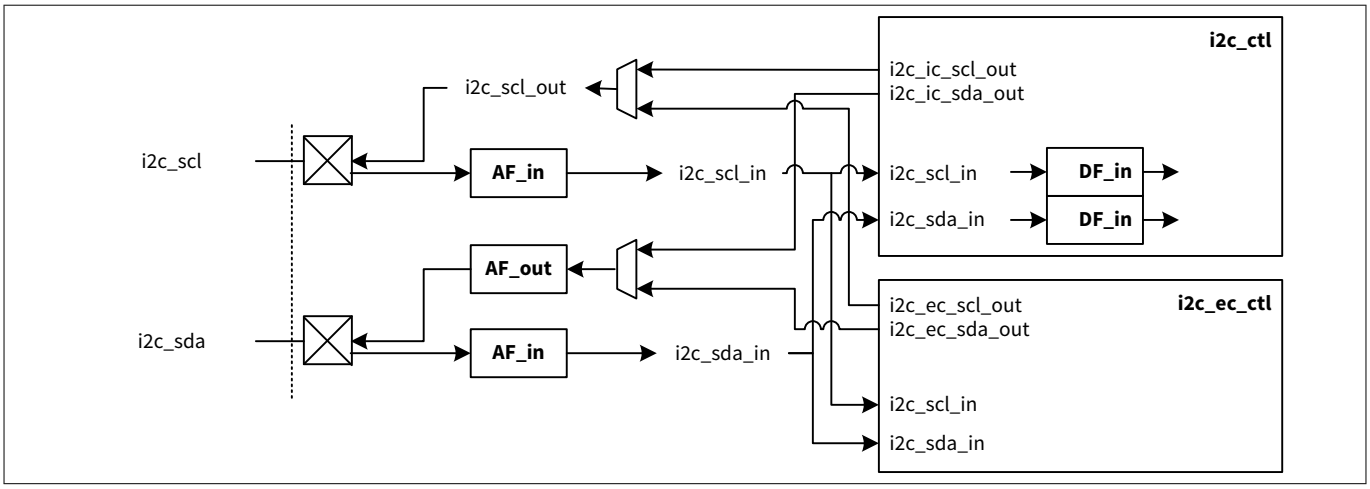


Figure 152 I²C glitch filtering connection

The following table lists the useful combinations of glitch filters.

Table 167 Glitch filter combinations

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

When operating in EC_OP_MODE = 1, the 100-kHz, 400-kHz, and 1000-kHz modes require the following settings for AF_out:

AF_in	AF_out	DF_in	
1	1	0	100-kHz mode: SCBx_I2C_CFG.SDA_OUT_FILT_SEL = 3 400-kHz mode: SCBx_I2C_CFG.SDA_OUT_FILT_SEL = 3 1000-kHz mode: SCBx_I2C_CFG.SDA_OUT_FILT_SEL = 1

23.6.6.2 Oversampling and bit rate

Internally-clocked master

The XMC7000 MCU implements the I²C clock as an oversampled multiple of the SCB input clock. In master mode, the block determines the I²C frequency. Routing delays on the PCB, on the device, and the block (including analog and digital glitch filters) all contribute to the signal interface timing. In master mode, the block operates off CLK_SCB and uses programmable oversampling factors for the SCL high SCBx_I2C_CTRL.HIGH_PHASE_OVS and low SCBx_I2C_CTRL.LOW_PHASE_OVS times.

Table 168 I²C frequency and oversampling requirements in I²C master mode

AF_in	AF_out	DF_in	Mode	Supported Frequency	SCBx_I2C_CTRL. LOW_PHASE_OVS	SCBx_I2C_CTRL. HIGH_PHASE_OVS	Input Clock Frequency
0	0	1	100 kHz	[62, 100] kHz	[9, 15]	[9, 15]	[1.98-3.2] MHz

(table continues...)

Table 168 (continued) I²C frequency and oversampling requirements in I²C master mode

AF_in	AF_out	DF_in	Mode	Supported Frequency	SCBx_I2C_CTRL. LOW_PHASE_OVS	SCBx_I2C_CTRL. HIGH_PHASE_OVS	Input Clock Frequency
			400 kHz	[264, 400] kHz	[13, 15]	[7, 15]	[8.45-10] MHz
			1000 kHz	[447, 1000] kHz	[8, 15]	[5, 15]	[14.32-25.8] MHz
1	0	0	100 kHz	[48, 100] kHz	[7, 5]	[7, 15]	[1.55-3.2] MHz
			400 kHz	[244, 400] kHz	[12, 15]	[7, 15]	[7.82-10] MHz
			1000 kHz	Not supported			

Table 168 assumes worst-case conditions on the I²C bus. The following equations can be used to determine the settings for your own system. This will involve measuring the rise and fall times on SCL and SDA lines in your system.

tCLK_SCB(Min) = (tLOW + tF)/SCBx_I2C_CTRL.LOW_PHASE_OVS

If CLK_SCB is any faster than this, the tLOW of the I²C specification will be violated. tF needs to be measured in your system.

tCLK_SCB(Max) = (tVD – tRF – 100 nsec)/3 (When analog filter is enabled and digital filter disabled)

tCLK_SCB(Max) = (tVD – tRF)/4 (When analog filter is disabled and digital filter is enabled)

tRF is the maximum of either the rise or fall time. If CLK_SCB is slower than this frequency, tVD will be violated.

Internal-clocked slave

In slave mode, the I²C frequency is determined by the incoming I²C SCL signal. To ensure proper operation, CLK_SCB must be significantly higher than the I²C bus frequency. Unlike master mode, this mode does not use programmable oversampling factors. Table 169 assumes worst-case conditions on the I²C bus including the chip internal delay.

Table 169 SCB input clock requirements in I²C slave mode

AF_in	AF_out	DF_in	Mode	CLK_SCB frequency range
0	0	1	100 kHz	[1.98-12.8] MHz
			400 kHz	[8.45-17.14] MHz
			1000 kHz	[14.32-44.77] MHz
1	0	0	100 kHz	[1.55-12.8] MHz
			400 kHz	[7.82-15.38] MHz
			1000 kHz	[15.84-89.0] MHz

tCLK_SCB(Max) = (tVD – tRF – 100 nsec) / 3 (When analog filter is enabled and digital filter disabled)

tCLK_SCB(Max) = (tVD – tRF) / 4 (When analog filter is disabled and digital filter is enabled)

tRF is the maximum of either the rise or fall time. If CLK_SCB is slower than this frequency, tVD will be violated.

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

tCLK_SCB(MIN) = (tSU;DAT(min) + tRF) /16

or

tCLK_SCB(min) = (0.6 × tF – 50 nsec) / 2 (When analog filter is enabled and digital filter disabled)

tCLK_SCB(min) = (0.6 × tF) / 3 (When analog filter is disabled and digital filter enabled)

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

Master-slave

In this mode, when the SCB is acting as a master device, the block determines the I²C frequency. When the SCB is acting as a slave device, the block does not determine the I²C frequency. Instead, the incoming I²C SCL signal does.

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

23.6.7 Loop-back

In master-slave mode, SCB supports internal SCL and SDA lines are routed internally in the peripheral. As a result, it is unaffected by other I²C devices.

It is configured using the SCBx_I2C_CTRL.LOOPBACK Register.

23.6.8 Enabling and initializing the I²C

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

23.6.8.1 Configuring for I²C FIFO mode

The I²C interface must be programmed in the following order.

1. Program protocol specific information using the SCBx_I2C_CTRL Register. This includes selecting master-slave functionality (MASTER_MODE, SLAVE_MODE).
2. Program the generic transmitter and receiver information using the SCBx_TX_CTRL and SCBx_RX_CTRL Registers.
   a. Specify the data frame width (DATA_WIDTH = 7).
   b. Specify that MSb is the first bit to be transmitted/received (MSB_FIRST = 1).
3. Set the SCBx_CTRL.MEM_WIDTH to ‘1’ to enable the byte mode.
4. Program the transmitter and receiver FIFOs using the SCBx_TX_FIFO_CTRL and SCBx_RX_FIFO_CTRL Registers respectively.
   a. Set the trigger level (TRIGGER_LEVEL).
   b. Clear the transmitter and receiver FIFO and Shift Registers (CLEAR).
5. Program the SCBx_CTRL Register to enable the I²C block and select the I²C mode. For a complete description of the I²C Registers, see the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual.

23.6.8.2 Configuring for EZ and CMD_RESP modes

To configure the I²C block for EZ and CMD_RESP modes, set the following I²C Register bits

- Select the EZI2C mode by writing '1' to the SCBx_CTRL.EZ_MODE Register.
- Select CMD_RESP mode by writing a 1 to the SCBx_CTRL.CMD_RESP Register.
- Set the S_READY_ADDR_ACK (bit 12) and SCBx_I2C_CTRL.S_READY_DATA_ACK Register.

23.6.9 I/O pad connections

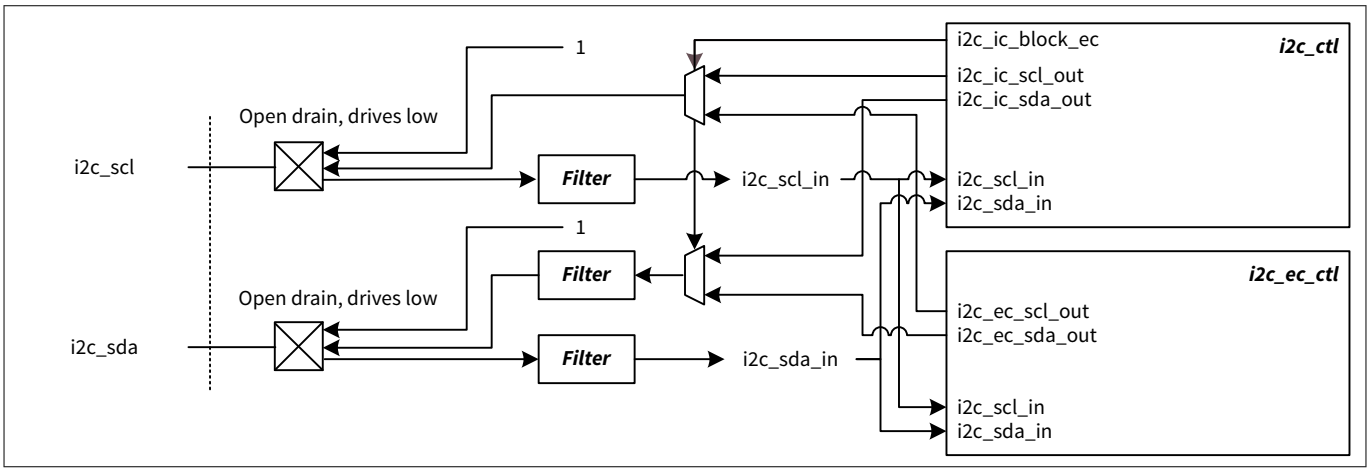


Figure 153 I²C I/O pad connections

Table 170 I²C I/O pad descriptions

I/O pads	Drive mode	On-chip I/O signals	Usage
i2c_scl	Open drain drives low	i2c_scl_in	Receive a clock
		i2c_scl_out	Transmit a clock
i2c_sda	Open drain drives low	i2c_sda_in	Receive data
		i2c_sda_out	Transmit data

23.6.10 I²C Registers

The I²C interface is controlled by reading and writing a set of configuration, control, and Status Registers, as listed in Table 174.

***Note**: Detailed descriptions of the I²C Register bits are available in the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual.*

23.7 SCB interrupts

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

Table 171 SCB interrupts

Interrupt	Functionality	Active/deep sleep	Registers
interrupt_master	I^2C master and SPI master functionality	Active	SCBx_INTR_M, SCBx_INTR_M_SET, SCBx_INTR_M_MASK, SCBx_INTR_M_MASKED
interrupt_slave	I^2C slave and SPI slave functionality	Active	SCBx_INTR_S, SCBx_INTR_S_SET, SCBx_INTR_S_MASK, SCBx_INTR_S_MASKED
interrupt_tx	UART transmitter and TX FIFO functionality	Active	SCBx_INTR_TX, SCBx_INTR_TX_SET, SCBx_INTR_TX_MASK, SCBx_INTR_TX_MASKED
interrupt_rx	UART receiver and RX FIFO functionality	Active	SCBx_INTR_RX, SCBx_INTR_RX_SET, SCBx_INTR_RX_MASK, SCBx_INTR_RX_MASKED
interrupt_i2c_ec	Externally-clocked I^2C slave functionality	Deep sleep	SCBx_INTR_I2C_EC, SCBx_INTR_I2C_EC_MASK, SCBx_INTR_I2C_EC_MASKED
interrupt_spi_ec	Externally-clocked SPI slave functionality	Deep sleep	SCBx_INTR_SPI_EC, SCBx_INTR_SPI_EC_MASK, SCBx_INTR_SPI_EC_MASKED

The following sections explain the different interrupt sources for each mode of SCB operation.

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

23.7.1 SPI interrupts

SPI interrupts can be classified as master interrupts, slave interrupts, TX interrupts, RX interrupts, and externally-clocked (EC) mode interrupts. Each interrupt output is the logical OR of the group of all possible interrupt sources classified under the section. For example, the TX interrupt output is the logical OR of the group of all possible TX interrupt sources. This signal goes high when any of the enabled TX interrupt sources are true. The SCB also provides an Interrupt Cause Register (SCBx_ INTR_CAUSE) that can be used to determine interrupt source. The Interrupt Registers are cleared by writing '1' to the corresponding bit field. Note that certain interrupt sources are triggered again as long as the condition is met even if the interrupt source was cleared. For example, the TX_FIFO_EMPTY is set as long as the transmit FIFO is empty even if the interrupt source is cleared. For more information on Interrupt Registers, see the XMC7100 registers technical reference

manual and XMC7200 registers technical reference manual. The SPI supports interrupts on the following events:

- SPI master interrupts (SCBx_INTR_M)
  - SPI master transfer done (SPI_DONE)
- SPI slave interrupts (SCBx_INTR_S)
  - SPI slave deselected after a write EZSPI transfer occurred (SPI_EZ_WRITE_STOP)
  - SPI slave deselected after any EZSPI transfer occurred (SPI_EZ_STOP)
  - SPI Bus Error – Slave deselected unexpectedly in the SPI transfer. The firmware may decide to clear the TX and RX FIFOs for this error. (SPI_BUS_ERROR)
- SPI TX (SCBx_INTR_TX)
  - TX FIFO has less entries than the value specified by SCBx_TX_FIFO_CTRL.TRIGGER_LEVEL (TRIGGER)
  - TX FIFO is not full (NOT_FULL)
  - TX FIFO is empty (EMPTY)
  - TX FIFO overflow (OVERFLOW)
  - TX FIFO underflow (UNDERFLOW)
- SPI RX (SCBx_INTR_RX)
  - RX FIFO has more entries than the value specified by SCBx_RX_FIFO_CTRL.TRIGGER_LEVEL (TRIGGER)
  - RX FIFO is not empty (NOT_EMPTY)
  - RX FIFO is full (FULL)
  - RX FIFO overflow (OVERFLOW)
  - RX FIFO underflow (UNDERFLOW)
- SPI externally-clocked (SCBx_INTR_SPI_EC)
  - Wake up request on slave select (WAKE_UP)
  - SPI STOP detection at the end of each transfer (EZ_STOP)
  - SPI STOP detection at the end of a write transfer (EZ_WRITE_STOP)
  - SPI STOP detection at the end of a read transfer (EZ_READ_STOP)

23.7.2 UART interrupts

UART interrupts can be classified as TX interrupts and RX interrupts. Each interrupt output is the logical OR of the group of all possible interrupt sources classified under the section. For example, the TX interrupt output is the logical OR of the group of all possible TX interrupt sources. This signal goes high when any of the enabled TX interrupt sources are true. The SCB also provides an Interrupt Cause Register (SCBx_INTR_CAUSE) that can be used to determine interrupt source. The Interrupt Registers are cleared by writing '1' to the corresponding bitfield. Note that certain interrupt sources are triggered again as long as the condition is met even if the interrupt source was cleared. For example, the TX_FIFO_EMPTY is set as long as the transmit FIFO is empty even if the interrupt source is cleared. For more information on Interrupt Registers, see the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual. The UART blocks generates interrupts on the following events:

- UART TX (SCBx_INTR_TX)
  - TX FIFO has less entries than the value specified by SCBx_TX_FIFO_CTRL.TRIGGER_LEVEL (TRIGGER)
  - TX FIFO is not full (NOT_FULL)
  - TX FIFO is empty (EMPTY)
  - TX FIFO overflow (OVERFLOW)
  - TX FIFO underflow (UNDERFLOW)
  - TX received a NACK in SmartCard mode (UART_NACK)

  - TX done. This happens when the UART completes transferring all data in the TX FIFO and the last stop field is transmitted (both TX FIFO and Transmit Shifter Register are empty). (UART_DONE)
  - Arbitration lost (in SmartCard modes) (UART_ARB_LOST)
- UART RX (INTR_RX)
  - RX FIFO has more entries than the value specified by SCBx_RX_FIFO_CTRL.TRIGGER_LEVEL (TRIGGER)
  - RX FIFO is not empty (NOT_EMPTY)
  - RX FIFO is full (FULL)
  - RX FIFO overflow (OVERFLOW)
  - RX FIFO underflow (UNDERFLOW)
  - Frame error in received data frame (FRAME_ERROR)
  - Parity error in received data frame (PARITY_ERROR)

23.7.3 I²C interrupts

I²C interrupts can be classified as master interrupts, slave Interrupts, TX interrupts, RX interrupts, and Externally-clocked (EC) mode interrupts. Each interrupt output is the logical OR of the group of all possible interrupt sources classified under the section. For example, the TX interrupt output is the logical OR of the group of all possible TX interrupt sources. This signal goes high when any of the enabled TX interrupt sources are true. The SCB also provides an Interrupt Cause Register (SCBx_INTR_CAUSE) that can be used to determine interrupt source. The Interrupt Registers are cleared by writing '1' to the corresponding bit field. Note that certain interrupt sources are triggered again as long as the condition is met even if the interrupt source was cleared. For example, the TX_FIFO_EMPTY is set as long as the transmit FIFO is empty even if the interrupt source is cleared. For more information on Interrupt Registers, see the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual. The I²C block generates interrupts for the following conditions.

- I²C master (SCBx_INTR_M)
  - I²C master lost arbitration (I2C_ARB_LOST)
  - I²C master received NACK (I2C_NACK)
  - I²C master received ACK (I2C_ACK)
  - I²C master sent STOP (I2C_STOP)
  - I²C bus error (unexpected stop/start condition detected) (I2C_BUS_ERROR)
- I²C slave (SCBx_INTR_S)
  - I²C slave lost arbitration (I2C_ARB_LOST)
  - I²C slave received NACK (I2C_NACK)
  - I²C slave received ACK (I2C_ACK)
  - I²C slave received Write STOP (I2C_WRITE_STOP)
  - I²C slave received STOP (I2C_STOP)
  - I²C slave received START (I2C_START)
  - I²C slave address matched (I2C_ADDR_MATCH)
  - I²C slave general call address received (I2C_GENERAL)
  - I²C bus error – unexpected stop/start condition detected (I2C_BUS_ERROR)
- I²C TX (SCBx_INTR_TX)
  - TX FIFO has less entries than the value specified by SCBx_TX_FIFO_CTRL.TRIGGER_LEVEL (TRIGGER)
  - TX FIFO is not full (NOT_FULL)
  - TX FIFO is empty (EMPTY)

  - TX FIFO overflow (OVERFLOW)
  - TX FIFO underflow (UNDERFLOW)
- I^2C RX (SCBx_INTR_RX)
  - RX FIFO has more entries than the value specified by SCBx_RX_FIFO_CTRL.TRIGGER_LEVEL (TRIGGER)
  - RX FIFO is not empty (NOT_EMPTY)
  - RX FIFO is full (FULL)
  - RX FIFO overflow (OVERFLOW)
  - RX FIFO underflow (UNDERFLOW)
- I^2C externally-clocked (SCBx_INTR_I2C_EC)
  - Wake up request on address match (WAKE_UP)
  - I^2C STOP detection at the end of each transfer (EZ_STOP)
  - I^2C STOP detection at the end of a write transfer (EZ_WRITE_STOP)
  - I^2C STOP detection at the end of a read transfer (EZ_READ_STOP)

23.8 Registers

23.8.1 SPI Registers

Table 172 SPI Registers

Register	Name	Description
SCBx_CTRL	SCB Control Register	Enables the SCB, selects the type of serial interface (SPI, UART, I^2C), and selects internally and externally-clocked operation, EZ and non-EZ modes of operation.
SCBx_STATUS	SCB Status Register	In EZ mode, this register indicates whether the externally-clocked logic is potentially using the EZ memory.
SCBx_SPI_CTRL	SCB SPI Control Register	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	SCB SPI Status Register	Indicates whether the SPI bus is busy and sets the SPI slave EZ address in the internally-clocked mode.
SCBx_TX_CTRL	SCB TX Control Register	Specifies the data frame width and specifies whether MSb or LSb is the first bit in transmission.
SCBx_RX_CTRL	SCB RX Control Register	Performs the same function as that of the SCBx_TX_CTRL Register, but for the receiver. Also decides whether a median filter is to be used on the input interface lines.
SCBx_TX_FIFO_CTRL	SCB TX FIFO Control Register	Specifies the trigger level, clears the Transmitter FIFO and Shift Registers, and performs the FREEZE operation of the transmitter FIFO.
SCBx_RX_FIFO_CTRL	SCB RX FIFO Control Register	Performs the same function as that of the SCBx_TX_FIFO_CTRL Register, but for the receiver.

(table continues...)

Table 172 (continued) SPI Registers

Register	Name	Description
SCBx_TX_FIFO_WR	SCB TX FIFO Write Register	Holds the data frame written into the transmitter FIFO. Behavior is similar to that of a PUSH operation.
SCBx_RX_FIFO_RD	SCB RX FIFO Read Register	Holds the data frame read from the receiver FIFO. Reading a data frame removes the data frame from the FIFO - behavior is similar to that of a POP operation. This register has a side effect when read by software: a data frame is removed from the FIFO.
SCBx_RX_FIFO_RD_SILENT	SCB RX FIFO Read Silent Register	Holds the data frame read from the receiver FIFO. Reading a data frame does not remove the data frame from the FIFO; behavior is similar to that of a PEEK operation.
SCBx_TX_FIFO_STATUS	SCB TX FIFO Status Register	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	SCB RX FIFO Status Register	Performs the same function as that of the SCBx_TX_FIFO_STATUS Register, but for the receiver.
SCBx_EZ_DATA	SCB EZ Data Register	Holds the data in EZ memory location

23.8.2 UART Registers

Table 173 UART Registers

Register	Name	Description
SCBx_CTRL	SCB Control Register	Enables the SCB; selects the type of serial interface (SPI, UART, I^2C)
SCBx_UART_CTRL	SCB UART Control register	Used to select the sub-modes of UART (standard UART, SmartCard, IrDA), also used for local loop back control.
SCBx_UART_RX_STATUS	SCB UART RX Status Register	Used to specify the BR_COUNTER value that determines the bit period. This is used to set the accuracy of the SCB clock. This value provides more granularity than the OVS bit in SCBx_CTRL Register.
SCBx_UART_TX_CTRL	SCB UART TX control Register	Used to specify the number of stop bits, enable parity, select the type of parity, and enable retransmission on NACK.
SCBx_UART_RX_CTRL	SCB UART RX Control Register	Performs same function as SCBx_UART_TX_CTRL but is also used for enabling multi processor mode.
SCBx_TX_CTRL	SCB TX Control Register	Used to specify the data frame width and to specify whether MSb or LSb is the first bit in transmission.

(table continues...)

Table 173 (continued) UART Registers

Register	Name	Description
SCBx_RX_CTRL	SCB RX Control Register	Performs the same function as that of the SCBx_TX_CTRL Register, but for the receiver. Also decides whether a median filter is to be used on the input interface lines.
SCBx_UART_FLOW_CONTROL	SCB UART Flow Control Register	Configures flow control for UART transmitter.

23.8.3 I²C Registers

Table 174 I²C Registers

Register	Name	Description
SCBx_CTRL	SCB Control Register	Enables the SCB 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	SCB I2C Control Register	Selects the mode (master, slave) and sends an ACK or NACK signal based on receiver FIFO status.
SCBx_I2C_STATUS	SCB I2C Status Register	Indicates bus busy status detection, read/write transfer status of the slave/master, and stores the EZ slave address.
SCBx_I2C_M_CMD	SCB I2C Master Command Register	Enables the master to generate START, STOP, and ACK/NACK signals.
SCBx_I2C_S_CMD	SCB I2C Slave Command Register	Enables the slave to generate ACK/NACK signals.
SCBx_STATUS	SCB Status Register	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.
SCBx_I2C_CFG	SCB I2C Configuration Register	Configures filters, which remove glitches from the SDA and SCL lines.
SCBx_TX_CTRL	SCB TX Control Register	Specifies the data frame width; also used to specify whether MSb or LSb is the first bit in transmission.
SCBx_TX_FIFO_CTRL	SCB TX FIFO Control Register	Specifies the trigger level, clearing of the transmitter FIFO and Shift Registers, and FREEZE operation of the transmitter FIFO.
SCBx_TX_FIFO_STATUS	SCB TX FIFO Status Register	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.

(table continues...)

Table 174 (continued) I²C Registers

Register	Name	Description
SCBx_TX_FIFO_WR	SCB TX FIFO Write Register	Holds the data frame written into the transmitter FIFO. Behavior is similar to that of a PUSH operation.
SCBx_RX_CTRL	SCB RX Control Register	Performs the same function as that of the SCBx_TX_CTRL Register, but for the receiver. Also decides whether a median filter is to be used on the input interface lines.
SCBx_RX_FIFO_CTRL	SCB RX FIFO Control Register	Performs the same function as that of the SCBx_TX_FIFO_CTRL Register, but for the receiver.
SCBx_RX_FIFO_STATUS	SCB RX FIFO Status Register	Performs the same function as that of the SCBx_TX_FIFO_STATUS Register, but for the receiver.
SCBx_RX_FIFO_RD	SCB RX FIFO Read Register	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	SCB RX FIFO Read Silent Register	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	SCB RX Match Register	Stores slave device address and is also used as slave device address MASK.
SCBx_EZ_DATA	SCB EZ Data Register	Holds the data in an EZ memory location.

24 CAN FD controller

24.1 Overview

The CAN FD controller complies with the ISO 11898-1 (CAN specification Rev. 2.0 parts A and B). In addition, it supports the time-triggered CAN (TTCAN) protocol defined in ISO 11898-4.

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

Two separate clocks are provided to the CAN FD controller: CAN clock (PCLK_CANFD[x]_CLOCK_CAN[y]) for CAN operation and system clock (CLK_SYS/CLK_GR5) for internal block operation. Acceptance filtering is implemented by a combination of up to 192 filter elements, where each can be configured as a range, as a bit mask, or as a dedicated ID filter.

The CAN FD controller functions only in active and sleep power modes. In deep sleep mode, it is not functional but is fully retained except the shared time stamp (TS) counter. In hibernate power mode, the controller is neither functional nor retained.

24.1.1 Features

The CAN FD controller has the following features:

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

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

24.1.2 Features not supported

- Asynchronous serial communication (ASC)
- Interrupt of bit error corrected (CANFDx_CHy_IR.BEC) in M_TTCAN
  - This bit is fixed at '0'. If this occurs, the fault structures report a correctable ECC error.

24.2 Configuration

24.2.1 Block diagram

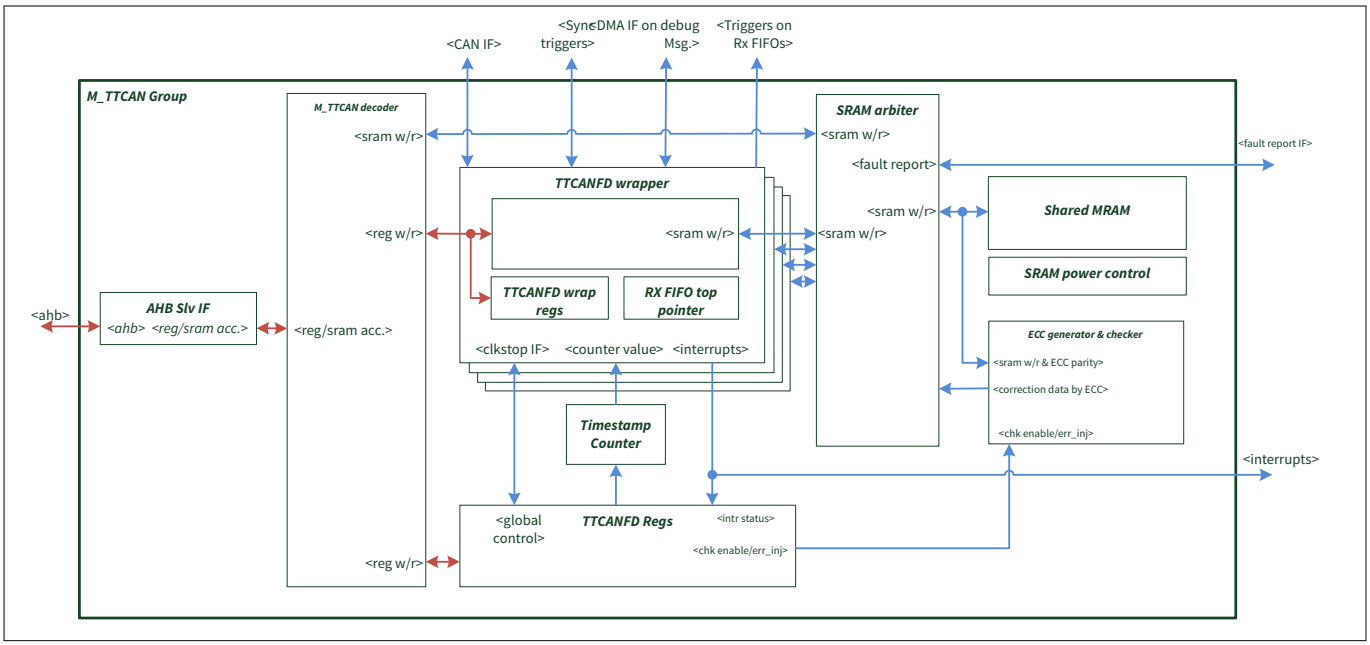


Figure 154 M_TTCAN block diagram

24.2.2 Dual clock sources

Each M_TTCAN channel has two clock inputs: PCLK_CANFD[x]_CLOCK_CAN[y] and CLK_GR5. The PCLK_CANFD[x]_CLOCK_CAN[y] (clk_cclk) is used for the CAN (or CAN FD) operation. The CLK_GR5 (CLK_SYS) is used for everything except CAN operation, such as register accesses and SRAM accesses. For the CAN FD operation, it is recommended to use 20 MHz, 40 MHz, or 80 MHz for PCLK_CANFD[x]_CLOCK_CAN[y] (clk_cclk). CLK_SYS must run equal or faster than clk_cclk.

Each M_TTCAN channel has its own PCLK_CANFD[x]_CLOCK_CAN[y]. This allows channels to have the flexibility to communicate at independent speeds. However, the host clock (CLK_SYS) will be the same for all M_TTCAN channels.

See the Clocking system for more details about clock configuration.

24.2.3 Interrupt lines

The two kind of interrupts from the M_TTCAN group are as follows:

- Interrupt0 and interrupt1 from each M_TTCAN channel within the M_TTCAN group
- Consolidated interrupt0 and consolidated interrupt1 for one M_TTCAN group

Each M_TTCAN channel provides two interrupt lines: interrupt0 and interrupt1. Interrupts from any source within the M_TTCAN channel can be routed either to interrupt0 or unterrupt1 by using CANFDx_CHy_ILS and CANFDx_CHy_TTILS Registers. By default, all interrupts are routed to interrupt0. By programming enable

interrupt line 0 (CANFDx_CHy_ILE.EINT0) and enable interrupt line 1 (CANFDx_CHy_ILE.EINT1), the interrupt lines can be enabled or disabled separately for each interrupt source.

In XMC7000, one device may contain multiple M_TTCAN channels in one M_TTCAN instance. Therefore, Interrupt line 0 and Interrupt line 1 from each M_TTCAN channel are routed to a common interrupt0 and interrupt1. Common interrupt0 and interrupt1 are ORed of all interrupt0 and interrupt1 coming from all present channels within one M_TTCAN group. Interrupt cause Registers CANFDx_INTR0_CAUSE and CANFDx_INTR1_CAUSE provide information about the active interrupt causing channel from a particular group.

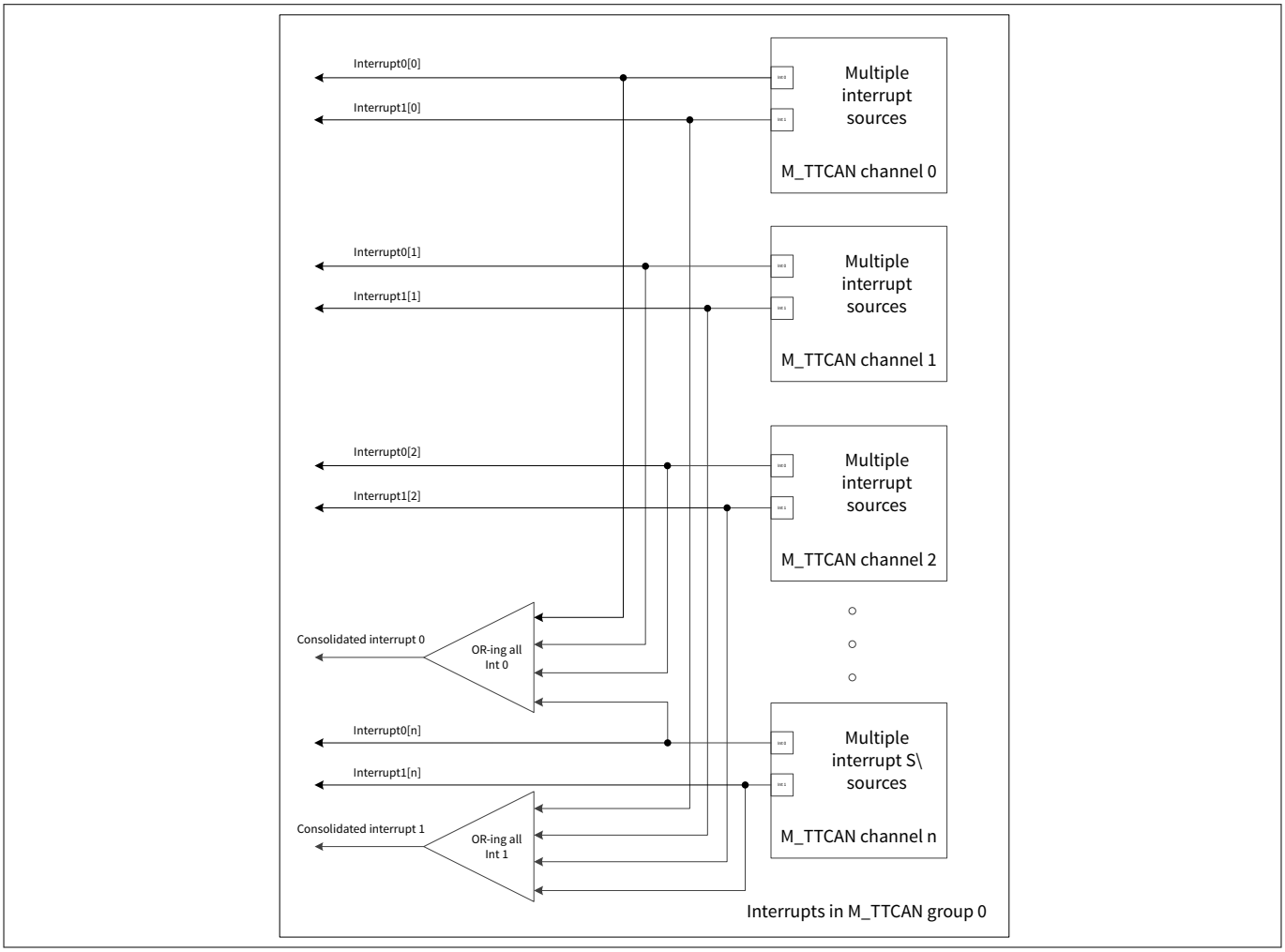


Figure 155 Interrupts in M_TTCAN group

24.3 Functional description

24.3.1 Operation modes

24.3.1.1 Software initialization

This refers to setting or resetting the initialization bit (CANFDx_CHy_CCCR.INIT). The CANFDx_CHy_CCCR.INIT bit is set

- Either by software or hardware reset
- When an uncorrected bit error is detected in message RAM
- By going bus OFF

While the CANFDx_CHy_CCCR.INIT is set:

- Message transfer from and to the CAN bus is stopped
- The status of the CAN bus output CANx_y_TX is recessive (HIGH)
- The protocol error counters are unchanged

Setting CANFDx_CHy_CCCR.INIT does not change any Configuration Register.

Resetting CANFDx_CHy_CCCR.INIT finishes the software initialization. The CAN FD controller then synchronizes itself to the data transfer on the CAN bus by waiting for the occurrence of a sequence of 11 consecutive recessive bits (Bus Idle) before it can take part in bus activities and start the message transfer.

Access/set/reset properties of registers affected by configuration change enable (CANFDx_CHy_CCCR.CCE)

Access to the Configuration Registers is only enabled when both bits CANFDx_CHy_CCCR.INIT and CANFDx_CHy_CCCR.CCE are set (write-protected). CANFDx_CHy_CCCR.CCE can only be set/reset while CANFDx_CHy_CCCR.INIT is 1. CANFDx_CHy_CCCR.CCE is automatically reset when CANFDx_CHy_CCCR.INIT is reset.

The following registers are reset when CANFDx_CHy_CCCR.CCE is set

- CANFDx_CHy_HPMS - High priority message status
- CANFDx_CHy_RXF0S - RX FIFO 0 status
- CANFDx_CHy_RXF1S - RX FIFO 1 status
- CANFDx_CHy_TXFQS - TX FIFO/queue status
- CANFDx_CHy_TXBRP - TX buffer request pending
- CANFDx_CHy_TXBTO - TX buffer transmission occurred
- CANFDx_CHy_TXBCF - TX buffer cancellation finished
- CANFDx_CHy_TXEFS - TX event FIFO status
- CANFDx_CHy_TTOST - TT operation status
- CANFDx_CHy_TTLGT - TT local and global time, only global time CANFDx_CHy_TTLGT.GT is reset
- CANFDx_CHy_TTCTC - TT cycle time and count
- CANFDx_CHy_TTCSM - TT cycle sync mark

In addition

- Timeout counter value (CANFDx_CHy_TOCV.TOC[15:0]) is preset to the value configured by the timeout period (CANFDx_CHy_TOCC.TOP[15:0]) when CANFDx_CHy_CCCR.CCE is set.
- State machines of TX and RX handlers are held in idle state while CANFDx_CHy_CCCR.CCE is 1.

The following registers are only writable while CANFDx_CHy_CCCR.CCE is 0.

- CANFDx_CHy_TXBAR - TX buffer add request
- CANFDx_CHy_TXBCR - TX buffer cancellation request

Test mode enable (CANFDx_CHy_CCCR.TEST) and bus monitoring mode (CANFDx_CHy_CCCR.MON) can only be set by the CPU while CANFDx_CHy_CCCR.INIT is 1 and CANFDx_CHy_CCCR.CCE is 1. Both bits may be reset at any time. Disable automatic retransmission (CANFDx_CHy_CCCR.DAR) can only be set/reset while CANFDx_CHy_CCCR.INIT is '1' and CANFDx_CHy_CCCR.CCE is '1'.

Message RAM initialization

Each message RAM word should be reset by writing 0x00000000 before configuration of the CAN FD controller. This prevents message RAM bit errors when reading uninitialized words, and also avoids unexpected filter element configurations in message RAM.

24.3.1.2 Normal operation

The M_TTCAN's default operating mode after hardware reset is event-driven CAN communication without time triggers (CANFDx_CHy_TTOCF.OM = 00). Both CANFDx_CHy_CCCR.INIT and CANFDx_CHy_CCCR.CCE must be set before the TT operation mode is changed.

When M_TTCAN is initialized and CANFDx_CHy_CCCR.INIT is reset to zero, M_TTCAN synchronizes itself to the CAN bus and is ready for communication.

After passing the acceptance filtering, received messages including Message ID and DLC are stored into a dedicated RX buffer or into RX FIFO 0 or RX FIFO 1.

For messages to be transmitted, dedicated TX buffers and a TX FIFO/TX queue can be initialized or updated. Automated transmission on reception of remote frames is not implemented.

24.3.1.3 CAN FD operation

The two variants in CAN FD frame transmission are:

- CAN FD frame without bit rate switching
- CAN FD frame where the control, data, and CRC fields are transmitted with a higher bit rate than the beginning and end of the frame

The previously reserved bit in CAN frames with 11-bit identifiers and 29-bit identifiers will now be decoded as FDF bit.

- FDF = recessive signifies a CAN FD frame
- FDF = dominant signifies a classic CAN frame

In a CAN FD frame, the two bits following FDF, reserved bits, and bit rate switch (BRS) decide whether the bit rate inside the CAN FD frame is switched. A CAN FD bit rate switch signified by res is dominant and BRS is recessive. The coding of res as recessive is reserved for future expansion of the protocol. If the M_TTCAN receives a frame with FDF and res as recessive, it will signal a protocol exception event by setting the CANFDx_CHy_PSR.PXE bit. When protocol exception handling is enabled (CANFDx_CHy_CCCR.PXHD = 0), it causes the operation state to change from receiver (CANFDx_CHy_PSR.ACT = 10) to Integrating (CANFDx_CHy_PSR.ACT = 00) at the next sample point. If protocol exception handling is disabled (CANFDx_CHy_CCCR.PXHD = 1), the M_TTCAN will treat a recessive res bit as a form error and respond with an error frame.

CAN FD operation is enabled by programming CANFDx_CHy_CCCR.FDOE. If CANFDx_CHy_CCCR.FDOE is '1', transmission and reception of CAN FD frames is enabled. Transmission and reception of classic CAN frames is always possible. Whether a CAN FD frame or a classic CAN frame is transmitted can be configured via the FDF bit in the respective TX buffer element. With CANFDx_CHy_CCCR.FDOE as '0', received frames are interpreted as classic CAN frames, which leads to the transmission of an error frame when receiving a CAN FD frame. When CAN FD operation is disabled, no CAN FD frames are transmitted even if the FDF bit of a TX buffer element is set. CANFDx_CHy_CCCR.FDOE and CANFDx_CHy_CCCR.BRSE can only be changed while CANFDx_CHy_CCCR.INIT and CANFDx_CHy_CCCR.CCE are both set.

With CANFDx_CHy_CCCR.FDOE as '0', the setting of FDF and BRS is ignored and frames are transmitted in classic CAN format. When CANFDx_CHy_CCCR.FDOE = 1 and CANFDx_CHy_CCCR.BRSE = 0, only FDF of a TX buffer element is evaluated. When CANFDx_CHy_CCCR.FDOE = 1 and CANFDx_CHy_CCCR.BRSE = 1, transmission of CAN FD frames with bit rate switching is enabled. All TX buffer elements with FDF and BRS bits set are transmitted in CAN FD format with bit rate switching.

A mode change during CAN operation is only recommended under the following conditions:

- The failure rate in the CAN FD data phase is significantly higher than in the CAN FD arbitration phase. In this case disable the CAN FD bit rate switching option for transmissions.
- During system startup, all nodes transmit classic CAN messages until it is verified that they can communicate in CAN FD format. If this is true, all nodes switch to CAN FD operation.
- Wake-up messages in CAN partial networking must be transmitted in classic CAN format.
- End-of-line programming occurs in case all nodes are not CAN FD capable. Non-CAN FD nodes are held in Silent mode until programming is completed. Then all nodes switch back to classic CAN communication.

In the CAN FD format, the coding of the DLC differs from the standard CAN format. The DLC codes 0 to 8 have the same coding as in standard CAN, codes 9 to 15, which in standard CAN have a data field of 8 bytes, are coded according to Table 175.

Table 175 Coding of DLC in CAN FD

DLC	9	10	11	12	13	14	15
Number of data bytes	12	16	20	24	32	48	64

In CAN FD frames, the bit timing will be switched inside the frame after the (BRS) bit, if this bit is recessive. Before the BRS bit, in the CAN FD arbitration phase, the nominal CAN bit timing is used as defined by the Nominal Bit Timing and Prescaler Register (CANFDx_CHy_NBTP). In the following CAN FD data phase, the data phase bit timing is used as defined by the Data Bit Timing and Prescaler Register (CANFDx_CHy_DBTP). The bit timing is switched back from the data phase timing at the CRC delimiter or when an error is detected, whichever occurs first.

The maximum configurable bit rate in the CAN FD data phase depends on the CAN clock frequency (clk_can). For example, with a CAN clock frequency of 20 MHz and the shortest configurable bit time of 4 tq, the bit rate in the data phase is 5 Mbit/s.

In both data frame formats, CAN FD and CAN FD with bit rate switching, the value of the bit ESI (error status indicator) is determined by the transmitter's error state at the start of the transmission. If the transmitter is error passive, ESI is transmitted recessive; otherwise, it is transmitted dominant.

24.3.1.4 Transmitter delay compensation

During the data phase of a CAN FD transmission only one node is a transmitter; all others are receivers. The length of the bus line has no impact. When transmitting via pin CANx_y_TX, the M_TTCAN receives the transmitted data from its local CAN transceiver via pin CANx_y_RX. The received data is delayed by the transmitter delay. In case this delay is greater than TSEG1 (time segment before sample point), a bit error is detected. To enable a data phase bit time that is even shorter than the transmitter delay, the delay compensation is introduced. Without transmitter delay compensation, the bit rate in the data phase of a CAN FD frame is limited by the transmitter delay.

Description

The M_TTCAN's protocol unit has implemented a delay compensation mechanism to compensate the transmitter delay. This enables transmission with higher bit rates during the CAN FD data phase, independent of the delay of a specific CAN transceiver.

To check for bit errors during the data phase of transmitting nodes, the delayed transmit data is compared against the received data at the secondary sample point (SSP). If a bit error is detected, the transmitter will

react on this bit error at the next following regular sample point. During the arbitration phase the delay compensation is always disabled.

The transmitter delay compensation enables configurations where the data bit time is shorter than the transmitter delay, it is described in detail in the new ISO 11898-1:2015. It is enabled by setting bit CANFDx_CHy_DBTP.TDC.

The received bit is compared against the transmitted bit at the SSP. The SSP position is defined as the sum of the measured delay from the M_TTCAN's transmit output CANx_y_TX through the transceiver to the receive input RX plus the transmitter delay compensation offset as configured by CANFDx_CHy_TDCR.TDCO. The transmitter delay compensation offset is used to adjust the position of the SSP inside the received bit (for example, half of the bit time in the data phase). The position of the secondary sample point is rounded down to the next integer number of mtq (PCLK_CANFD[x]_CLOCK_CAN[y] period).

CANFDx_CHy_PSR.TDCV shows the actual transmitter delay compensation value. CANFDx_CHy_PSR.TDCV is cleared when CANFDx_CHy_CCCR.INIT is set and is updated at each transmission of an FD frame while CANFDx_CHy_DBTP.TDC is set.

The following boundary conditions must be considered for the transmitter delay compensation implemented in the M_TTCAN:

- The sum of the measured delay from CANx_y_TX to CANx_y_RX and the configured transmitter delay compensation offset CANFDx_CHy_TDCR.TDCO must be less than 6 bit times in the data phase.
- The sum of the measured delay from CANx_y_TX to CANx_y_RX and the configured transmitter delay compensation offset CANFDx_CHy_TDCR.TDCO should be less than or equal 127 mtq. In case this sum exceeds 127 mtq, the maximum value of 127 mtq is used for transmitter delay compensation
- The data phase ends at the sample point of the CRC delimiter that stops checking of receive bits at the SSPs.

Transmitter delay compensation measurement

If transmitter delay compensation is enabled by programming CANFDx_CHy_DBTP.TDC = 1, the measurement is started within each transmitted CAN FD frame at the falling edge of bit FDF to bit res. The measurement is stopped when this edge is seen at the receive input CANx_y_RX of the transmitter. The resolution of this measurement is one mtq (minimum time quanta).

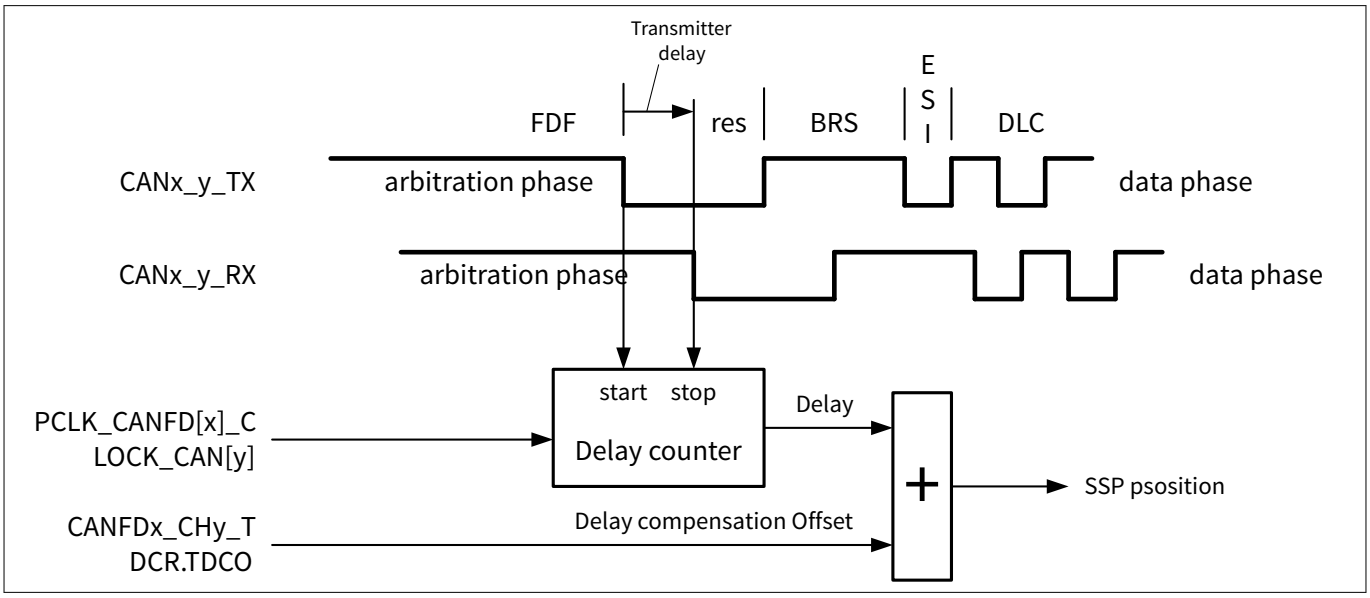


Figure 156 Transmitter delay measurement

To avoid this, a dominant glitch inside the received FDF bit ends the delay compensation measurement before the falling edge of the received res bit, resulting in an early SSP position. The use of a transmitter delay compensation filter window can be enabled by programming CANFDx_CHy_TDCR.TDCF. This defines a minimum value for the SSP position. Dominant edges on CANx_y_RX, that results in an earlier SSP position are

ignored for transmitter delay measurement. The measurement is stopped when the SSP position is at least CANFDx_CHy_TDCR.TDCF and CANx_y_RX is low.

24.3.1.5 Restricted operation mode

In restricted operation mode, the node is able to receive data and remote frames and acknowledge valid frames, but it does not send data frames, remote frames, active error frames, or overload frames. In case of an error or overload condition, it does not send dominant bits; instead it waits for the occurrence of a bus idle condition to resynchronize itself to the CAN communication. The error counters (CANFDx_CHy_ECR.REC and CANFDx_CHy_ECR.TEC) are frozen while error logging (CANFDx_CHy_ECR.CEL) is active.

The CPU can set the CAN FD controller into restricted operation mode by setting the restricted operation mode bit (CANFDx_CHy_CCCR.ASM). CANFDx_CHy_CCCR.ASM can only be set by the CPU when both CANFDx_CHy_CCCR.CCE and CANFDx_CHy_CCCR.INIT are set to ‘1‘. CANFDx_CHy_CCCR.ASM can be reset by the CPU at any time.

The CAN FD controller enters Restricted Operation mode automatically when the TX handler is not able to read data from the message RAM in time. To leave this mode, the CPU should reset CANFDx_CHy_CCCR.ASM.

The restricted operation mode can be used in applications that adapt themselves to different CAN bit rates. In this case, the application tests different bit rates and leaves the mode after it has received a valid frame.

***Note**: The restricted operation mode must not be combined with the loop back mode (internal or external).*

24.3.1.6 Bus monitoring mode

The M_TTCAN is set in bus monitoring mode by programming CANFDx_CHy_CCCR.MON to ‘1’ or when error level S3 (CANFDx_CHy_TTOST.EL = 11) is entered. In bus monitoring mode, the M_TTCAN is able to receive valid data frames and valid remote frames, but cannot start a transmission. In this mode, it sends only recessive bits on the CAN bus, if the M_TTCAN is required to send a dominant bit (ACK bit, overload flag, or active error flag), the bit is rerouted internally so that the M_TTCAN monitors this dominant bit, although the CAN bus may remain in recessive state. In bus monitoring mode, the CANFDx_CHy_TXBRP Register is held in reset state.

The bus monitoring mode can be used to analyze the traffic on a CAN bus without affecting it by the transmission of dominant bits. Figure 157 shows the connection of signals CANx_y_TX and CANx_y_RX to the M_TTCAN in bus monitoring mode.

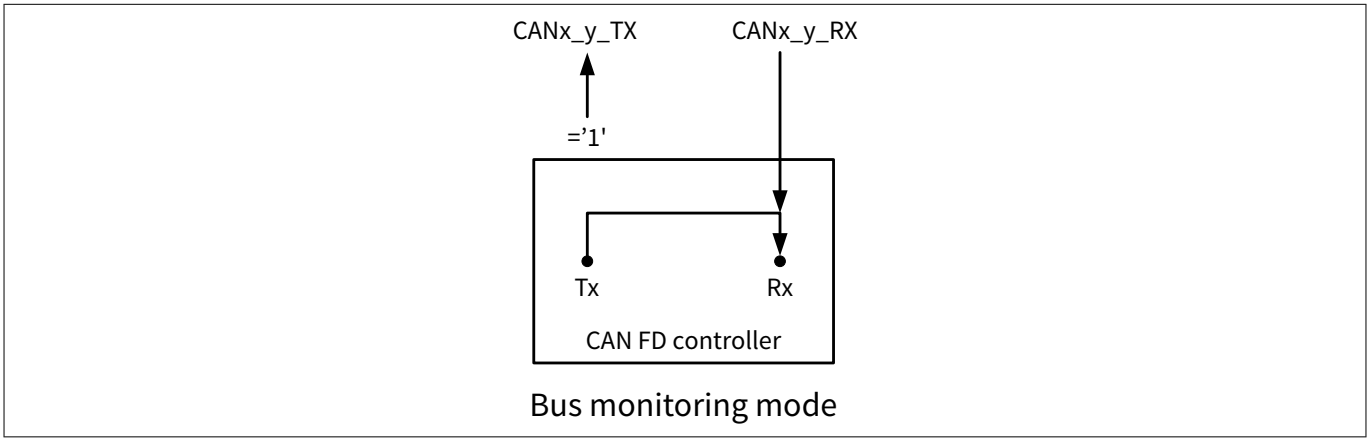


Figure 157 Pin control in bus monitoring mode

24.3.1.7 Disable automatic retransmission

M_TTCAN supports automatic retransmission of frames that have lost arbitration or that have been disturbed by errors during transmission. By default, automatic retransmission is enabled. To support time-triggered

communication (as described in ISO 11898-1:2015, chapter 9.2), the automatic retransmission may be disabled via CANFDx_CHy_CCCR.DAR.

In DAR mode, all transmissions are automatically canceled after they are started on the CAN bus. The TX request Pending bit CANFDx_CHy_TXBRP.TRPx is reset after successful transmission, when a transmission has not yet started at the point of cancellation, is aborted due to lost arbitration, or when an error occurred during frame transmission.

- Successful transmission:
  Corresponding TX buffer transmission occurred bit CANFDx_CHy_TXBTO.TOx set
  Corresponding TX buffer cancellation finished bit CANFDx_CHy_TXBCF.CFx not set
- Successful transmission in spite of cancellation:
  Corresponding TX buffer transmission occurred bit CANFDx_CHy_TXBTO.TOx set
  Corresponding TX buffer cancellation finished bit CANFDx_CHy_TXBCF.CFx set
- Arbitration lost or frame transmission disturbed:
  Corresponding TX buffer transmission occurred bit CANFDx_CHy_TXBTO.TOx not set
  Corresponding TX buffer cancellation finished bit CANFDx_CHy_TXBCF.CFx set

In successful frame transmissions, and if storage of TX events is enabled, a TX Event FIFO element is written with event type ET = 10 (transmission despite cancellation).

24.3.1.8 Power down (sleep mode)

The M_TTCAN channel can be set into power down mode via clock stop request (CANFDx_CTL.STOP_REQ). As long as clock stop request is active, STOP_REQ bit is read as one.

When all pending transmission requests have completed, the M_TTCAN waits until bus idle state is detected. Then the M_TTCAN sets CANFDx_CHy_CCCR.INIT to one to prevent any further CAN transfers. Now the M_TTCAN acknowledges that it is ready for power down by setting clock stop acknowledge (CANFDx_STATUS.STOP_ACK). Upon receiving acknowledgment from channel, hardware automatically switches off the clock to the respective channel.

To leave power down mode, the application must reset CANFDx_CTL.STOP_REQ. The M_TTCAN will acknowledge this by resetting CANFDx_STATUS.STOP_ACK. Afterwards, the application can restart CAN communication by resetting bit CANFDx_CHy_CCCR.INIT.

When the clock stop request is triggered through CANFDx_CTL.STOP_REQ, it must not be cleared before CANFDx_STATUS.STOP_ACK bit is set.

***Note**: Do not use the TTCAN CANFDx_CHy_CCCR.CSR Register for the power down control, instead use CANFDx_CTL.STOP_REQ. Similarly, use of CANFDx_CHy_CCCR.CSA should be avoided, instead use CANFDx_STAUTS.STOP_ACK.*

24.3.1.9 Test mode

To enable write access to CANFDx_CHy_TEST Register, Test Mode Enable bit (CANFDx_CHy_CCCR.TEST) must be set to one. This allows the configuration of the test modes and test functions.

Four output functions are available for the CAN transmit pin CANx_y_TX by programming CANFDx_CHy_TEST.TX. Apart from its default function of serial data output, it can drive the CAN sample point signal to monitor the M_TTCAN's bit timing; it can also drive constant dominant or recessive values. The actual value at the CANx_y_RX pin can be read from CANFDx_CHy_TEST.RX. Both functions can be used to check the CAN bus physical layer.

Due to the synchronization mechanism between CAN clock and host clock domain, there may be a delay of several host clock periods between writing to CANFDx_CHy_TEST.TX until the new configuration is visible at output pin CANx_y_TX. This applies also when reading input pin CANx_y_RX via CANFDx_CHy_TEST.RX.

***Note**: Test modes should be used for production tests or self-test only. The software control for pin CANx_y_TX interferes with all CAN protocol functions. It is not recommended to use test modes for application.*

External loop back mode

The M_TTCAN can be set in external loop back mode by programming CANFDx_CHy_TEST.LBCK to one. In loop back mode, the M_TTCAN treats its own transmitted messages as received messages and stores them (if they pass acceptance filtering) into an RX buffer or an RX FIFO. Figure 158 shows the connection of signals CANx_y_TX and CANx_y_RX to the M_TTCAN in external loop back mode.

This mode is provided for hardware self-test. To be independent from external stimulation, the M_TTCAN ignores acknowledge errors (recessive bit sampled in the acknowledge slot of a data/remote frame) in loop back mode. In this mode the M_TTCAN performs an internal feedback from its TX output to its RX input. The actual value of the CANx_y_RX input pin is disregarded by the M_TTCAN. The transmitted messages can be monitored at the CANx_y_TX pin.

Internal loop back mode

Internal loop back mode is entered by programming the CANFDx_CHy_TEST.LBCK and CANFDx_CHy_CCCR.MON bits to one. This mode can be used for a “Hot Selftest”, meaning the M_TTCAN can be tested without affecting a running CAN system connected to the CANx_y_TX and CANx_y_RX pins. In this mode, CANx_y_RX pin is disconnected from the M_TTCAN and CANx_y_TX pin is held recessive. Figure 158 shows the connection of CANx_y_TX and CANx_y_RX to the M_TTCAN in internal loop back mode.

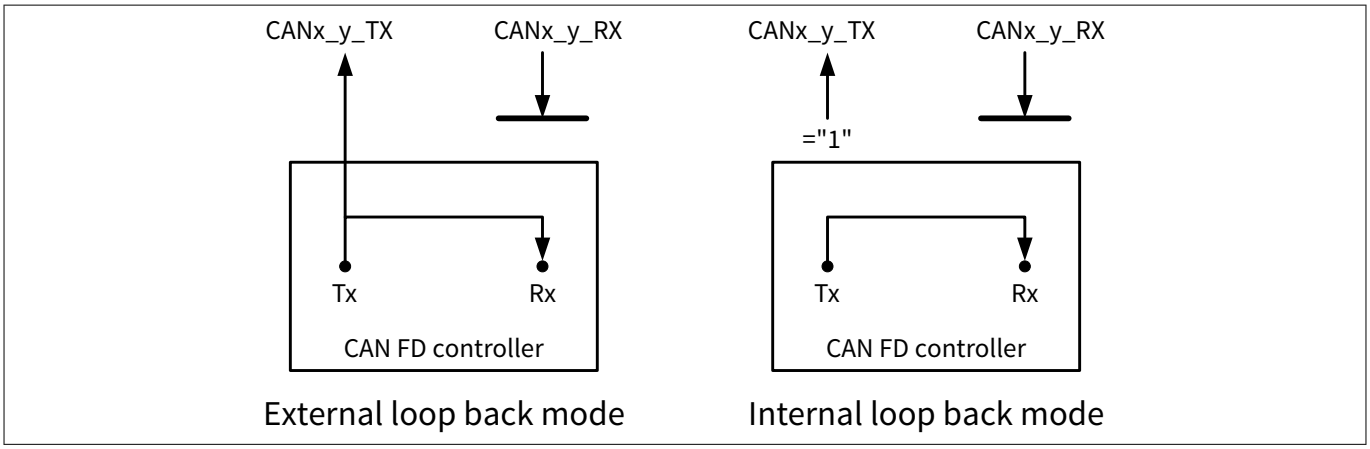


Figure 158 Pin control in loop back modes

24.3.1.10 Application watchdog

The application watchdog is served by reading the CANFDx_CHy_TTOST Register. When the application watchdog is not served in time, CANFDx_CHy_TTOST.AWE bit is set, all TTCAN communication is stopped, and the M_TTCAN is set into Bus Monitoring mode.

The TT application watchdog can be disabled by programming the application watchdog limit CANFDx_CHy_TTOCF.AWL to 0x00. The TT application watchdog should not be disabled in a TTCAN application program.

24.3.2 Timestamp generation

The M_TTCAN channel uses a 16-bit counter to record when messages are sent or received. This allows the application software to know the order in which events occurred.

To keep event ordering across multiple M_TTCAN channels, a global timestamp counter is implemented, which must be selected by setting '10' to CANFDx_CHy_TSCC.TSS[1:0]. This global timestamp counter is shared among all M_TTCAN groups present in the device. For instance, if the device contains two M_TTCAN groups, timestamp counter is shared among all the channels present in both the groups.

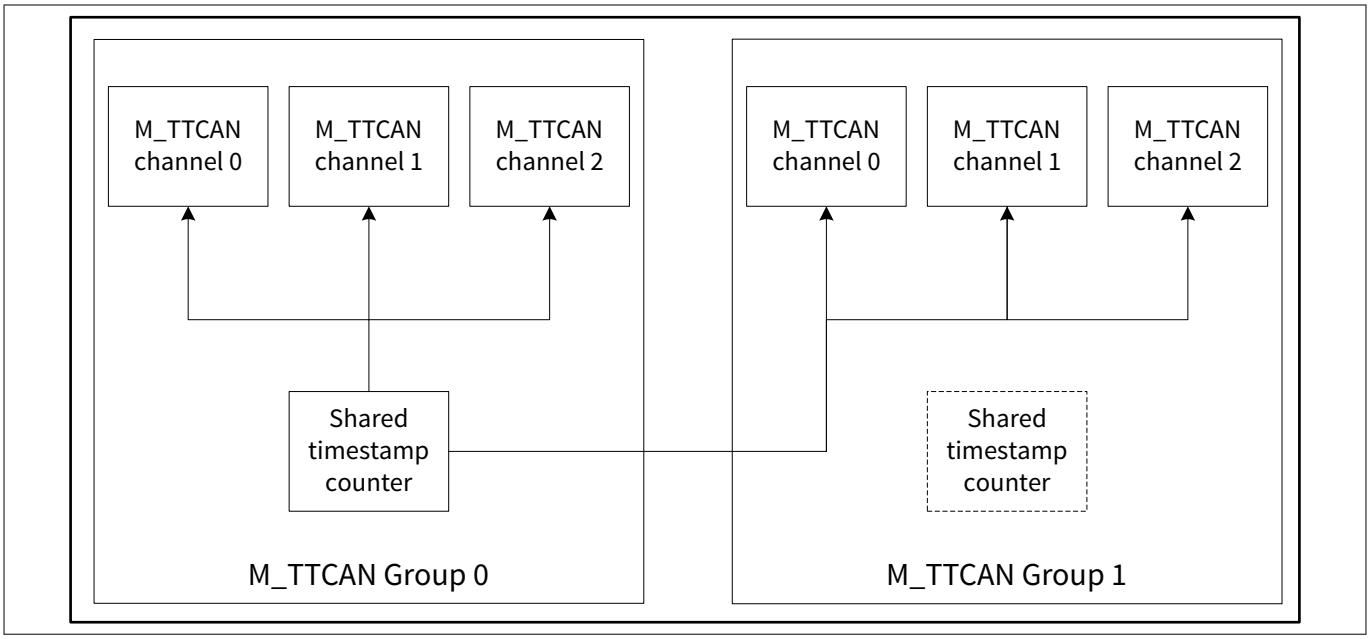


Figure 159 Timestamp connection between two M_TTCAN group

The timestamp counter is configured through the CANFDx_TS_CTL Register. The CANFDx_TS_CTL.ENABLED bit will enable the counter. Upon enabling, it will start incrementing according to the CANFDx_TS_CTL.PRESCALE [15:0]. The application can read the counter value through the CANFDx_TS_CNT Register. Write access to the CANFDx_TS_CNT Register will clear the CANFDx_TS_CNT.

When the timestamp counter is enabled, internal counter for prescaler counts with every cycle of CLK_SYS; when the counter value reaches the prescaler value, the timestamp counter increments by one and internal prescaler counter is cleared. When CANFDx_TS_CTL.PRESCALE changes, CANFDx_TS_CNT should be written to reset them. This can make the internal prescaler counter follow a new value of CANFDx_TS_CTL.PRESCALE immediately.

The shared timestamp counter is a wrap-around counter. When the counter wraps around, CANFDx_CHy_IR.TSW for all M_TTCAN channels will be raised.

On start of frame reception/transmission, the timestamp counter value is captured and stored into the timestamp section of an RX buffer/RX FIFO (RXTS [15:0]) or TX Event FIFO (TXTS [15:0]) element.

***Note**: The counter value CANFDx_TS_CNT is not retained in deep sleep mode whereas the CANFDx_TS_CTL is retained.*

24.3.3 Timeout counter

To signal timeout conditions for RX FIFO 0, RX FIFO 1, and the TX Event FIFO, the M_TTCAN supplies a 16-bit timeout counter. It operates as down-counter and uses the same prescaler controlled by CANFDx_CHy_TSCC.TCP as the timestamp counter. A prescaler CANFDx_CHy_TSCC.TCP should be configured to

clock the timeout counter in multiples of CAN bit times (1…16). The timeout counter is configured via Register CANFDx_CHy_TOCC. The actual counter value can be read from CANFDx_CHy_TOCV.TOC.

The timeout counter can only be started while CANFDx_CHy_CCCR.INIT = 0. It is stopped when CANFDx_CHy_CCCR.INIT = 1; for example, when the M_TTCAN enters Bus_Off state.

The operation mode is selected by CANFDx_CHy_TOCC.TOS. When operating in Continuous mode, the counter starts when CANFDx_CHy_CCCR.INIT is reset. A write to CANFDx_CHy_TOCV presets the counter to the value configured by CANFDx_CHy_TOCC.TOP and continues down-counting.

When the timeout counter is controlled by one of the FIFOs, an empty FIFO presets the counter to the value configured by CANFDx_CHy_TOCC.TOP. Down-counting is started when the first FIFO element is stored. Writing to CANFDx_CHy_TOCV has no effect.

When the counter reaches zero, interrupt flag CANFDx_CHy_IR.TOO is set. In continuous mode, the counter is immediately restarted at CANFDx_CHy_TOCC.TOP.

***Note**: The clock signal for the timeout counter is derived from the CAN Core's sample point signal. Therefore, the time the timeout counter is decremented may vary due to the synchronization/resynchronization mechanism of the CAN core. If the bit rate switch feature in CAN FD is used, the timeout counter is clocked differently in arbitration and data field.*

24.3.4 RX handling

The RX handler controls acceptance filtering, transfer of received messages to the RX buffers or to one of the two RX FIFOs, as well as RX FIFO's put and get indices.

24.3.4.1 Acceptance filtering

The M_TTCAN offers the possibility to configure two sets of acceptance filters, one for standard identifiers and one for extended identifiers. These filters can be assigned to an RX buffer or to RX FIFO 0,1. For acceptance filtering each list of filters is executed from element #0 until the first matching element. Acceptance filtering stops at the first matching element. The following filter elements are not evaluated for this message.

The main features are:

- Each filter element can be configured as
  - Range filter (from - to)
  - Filter for one or two dedicated IDs
  - Classic bit mask filter
- Each filter element is configurable for acceptance or rejection filtering
- Each filter element can be enabled/disabled individually
- Filters are checked sequentially; execution stops with the first matching filter element

Related configuration registers are:

- Global filter configuration (CANFDx_CHy_GFC)
- Standard ID filter configuration (CANFDx_CHy_SIDFC)
- Extended ID filter configuration (CANFDx_CHy_XIDFC)
- Extended ID AND mask (CANFDx_CHy_XIDAM)

Depending on the configuration of the filter element (SFEC/EFEC) a match triggers one of the following actions:

- Store received frame in FIFO 0 or FIFO 1
- Store received frame in RX buffer
- Store received frame in RX buffer and generate pulse at filter event pin
- Reject received frame

- Set high-priority message interrupt flag CANFDx_CHy_IR.HPM
- Set high-priority message interrupt flag CANFDx_CHy_IR.HPM and store received frame in FIFO 0 or FIFO 1

Acceptance filtering is started after the complete identifier is received. After acceptance filtering has completed, if a matching RX buffer or RX FIFO is found, the message handler starts writing the received message data in portions of 32 bits to the matching RX buffer or RX FIFO. If the CAN protocol controller has detected an error condition (such as CRC error), this message is discarded with the following impact on the affected RX buffer or RX FIFO:

- RX buffer

New data flag of the matching RX buffer is not set, but RX buffer is (partly) overwritten with received data. For error type, see CANFDx_CHy_PSR.LEC and CANFDx_CHy_PSR.DLEC, respectively.

- RX FIFO

Put index of matching RX FIFO is not updated, but related RX FIFO element is (partly) overwritten with received data. For error type, see CANFDx_CHy_PSR.LEC and CANFDx_CHy_PSR.DLEC, respectively. If the matching RX FIFO is operated in overwrite mode, the boundary conditions described in RX FIFO overwrite mode should be considered.

***Note**: When an accepted message is written to one of the two RX FIFOs, or into an RX buffer, the unmodified received identifier is stored independent of the filter(s) used. The result of the acceptance filter process depends on the sequence of configured filter elements.*

Range filter

The filter matches for all received frames with message IDs in the range defined by SF1ID/SF2ID resp. EF1ID/EF2ID.

The two possibilities when range filtering is used with extended frames are:

- EFT = 00: The message ID of received frames is ANDed with the CANFDx_CHy_XIDAM before the range filter is applied.
- EFT = 11: The CANFDx_CHy_XIDAM is not used for range filtering.

Filter for specific IDs

A filter element can be configured to filter one or two specific message IDs. To filter a specific message ID, the filter element should be configured with SF1ID = SF2ID and EF1ID = EF2ID, respectively.

Classic bit mask filter

Classic bit mask filtering is intended to filter groups of message IDs by masking single bits of a received message ID. With classic bit mask filtering, SF1ID/EF1ID is used as message ID filter, while SF2ID/EF2ID is used as filter mask.

A zero bit at the filter mask will mask the corresponding bit position of the configured ID filter; for example, the value of the received message ID at that bit position is not relevant for acceptance filtering. Only those bits of the received message ID where the corresponding mask bits are one are relevant for acceptance filtering.

In case all mask bits are one, a match occurs only when the received message ID and the message ID filter are identical. If all mask bits are zero, all message IDs match.

Standard message ID filtering

Figure 160 shows the flow for standard message ID (11-bit Identifier) filtering. The standard message ID filter element is described in Standard message ID filter element.

Controlled by the CANFDx_CHy_GFC and CANFDx_CHy_SIDFC message IDs, the remote transmission request bit (RTR) and the identifier extension bit (IDE) of received frames are compared against the list of configured filter elements.

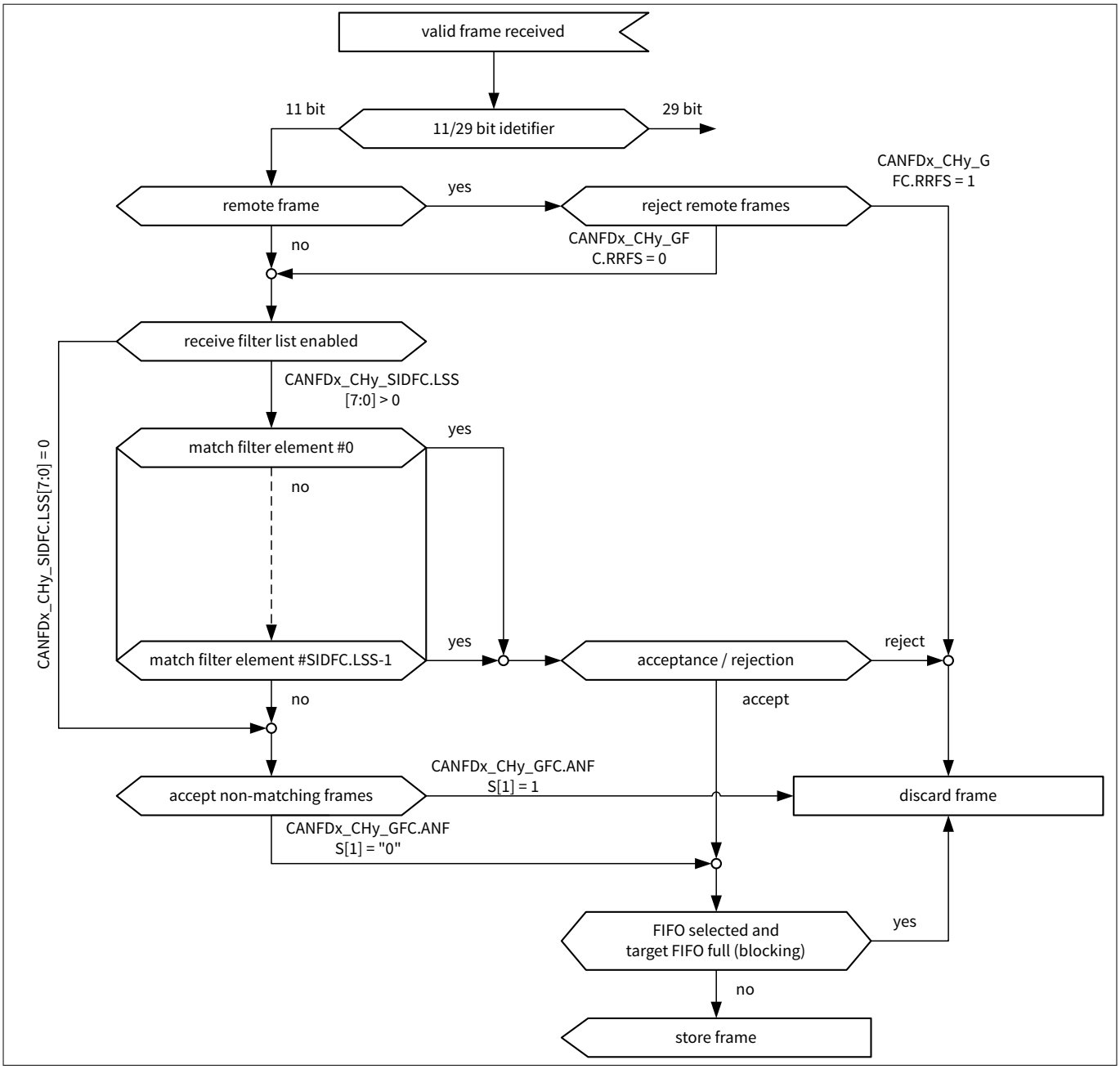


Figure 160 Standard message ID filter

Extended message ID filtering

Figure 161 shows the flow for extended Message ID (29-bit Identifier) filtering. The extended message ID filter element is described in Extended message ID filter element.

Controlled by the CANFDx_CHy_GFC and CANFDx_CHy_XIDFC Message IDs, the RTR bit, and IDE bit of received frames are compared against the list of configured filter elements.

The extended ID AND mask XIDAM[28:0] is ANDed with the received identifier before the filter list is executed.

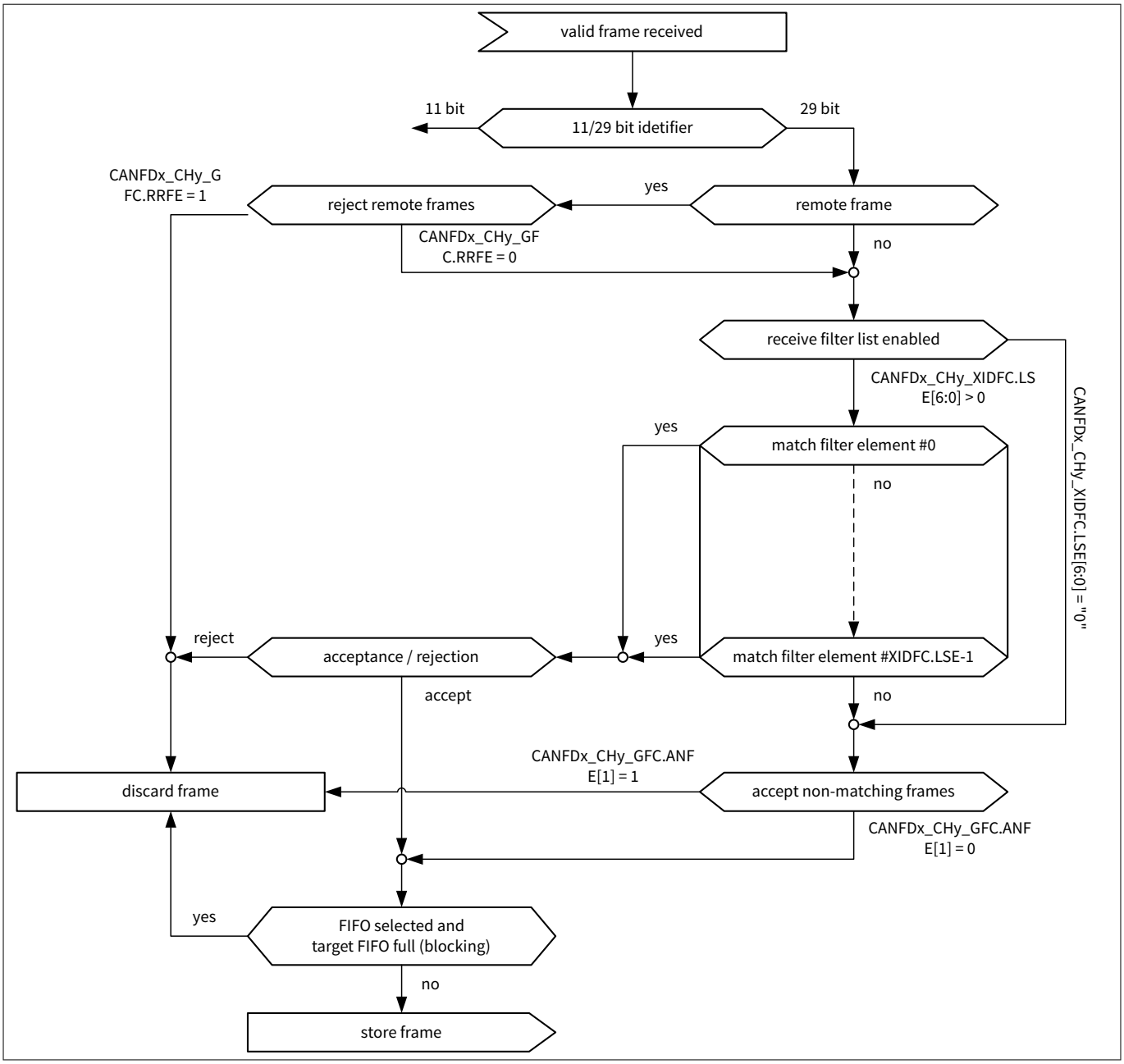


Figure 161 Extended message ID filter path

24.3.4.2 RX FIFOs

RX FIFO 0 and RX FIFO 1 can be configured to hold up to 64 elements each. The two RX FIFOs are configured via the CANFDx_CHy_RXF0C and CANFDx_CHy_RXF1C Registers.

Received messages that pass acceptance filtering are transferred to the RX FIFO as configured by the matching filter element. For a description of the filter mechanisms available for RX FIFO 0 and RX FIFO 1, see Acceptance filtering. The RX FIFO element is described in RX buffer and FIFO element.

To avoid an RX FIFO overflow, the RX FIFO watermark can be used. When the RX FIFO fill level reaches the RX FIFO watermark configured by CANFDx_CHy_RXFnC.FnWM, interrupt flag CANFDx_CHy_IR.RFnW is set. When the RX FIFO Put Index reaches the RX FIFO Get Index, an RX FIFO Full condition is signaled by CANFDx_CHy_RXFnS.FnF. In addition, interrupt flag CANFDx_CHy_IR.RFnF is set. The FIFO watermark interrupt flags can be used to trigger the DMA. DMA request for FIFO will remain set until the respective trigger is cleared by software. Software can clear the trigger by clearing the watermark flag.

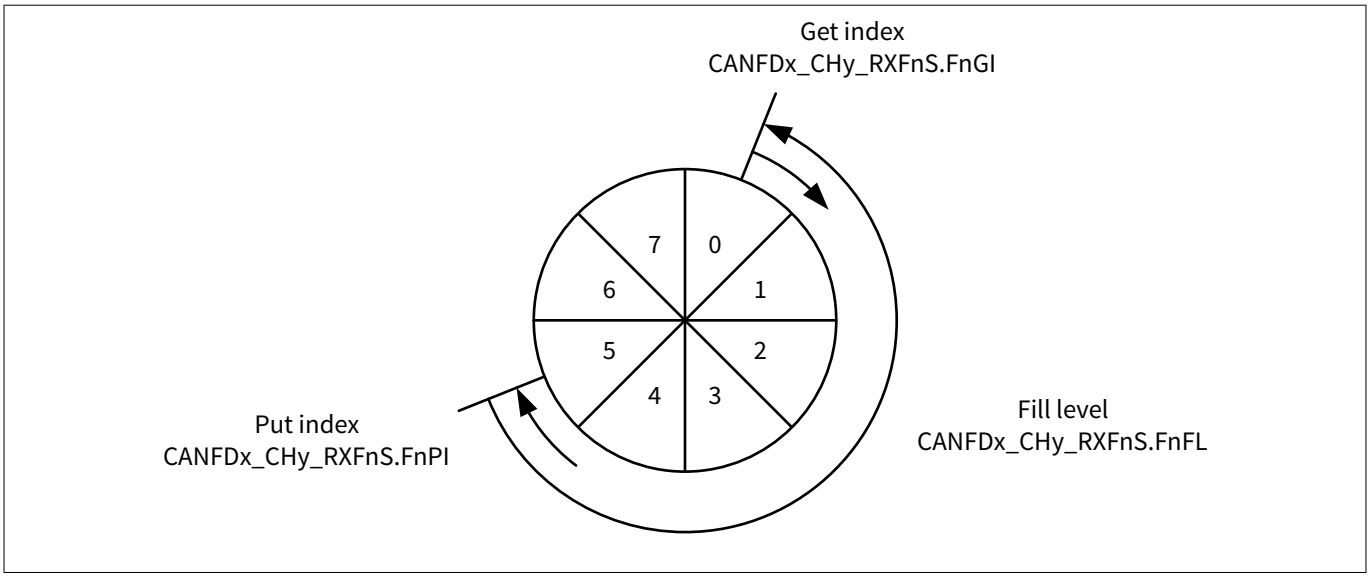


Figure 162 RX FIFO status

When reading from an RX FIFO, RX FIFO Get Index CANFDx_CHy_RXFnS.FnGI x FIFO element size has to be added to the corresponding RX FIFO start address CANFDx_CHy_RXFnC.FnSA. RX FIFO Top pointer logic is added to the CAN FD controller to make reading faster. See RX FIFO top pointer.

Table 176 RX buffer/FIFO element size

CANFDx_CHy_RXESC.RBDS[2:0] CANFDx_CHy_RXESC.FnDS[2:0]	Data field [bytes]	FIFO element size [RAM words]
000	8	4
001	12	5
010	16	6
011	20	7
100	24	8
101	32	10
110	48	14
111	64	18

RX FIFO blocking mode

The RX FIFO blocking mode is configured by CANFDx_CHy_RXFnC.FnOM = 0. This is the default operation mode for RX FIFOs.

When an RX FIFO full condition is reached (CANFDx_CHy_RXFnS.FnPI = CANFDx_CHy_RXFnS.FnGI), no further messages are written to the corresponding RX FIFO until at least one message is read and the RX FIFO Get Index is incremented. An RX FIFO full condition is signaled by CANFDx_CHy_RXFnS.FnF = 1. In addition, the interrupt flag CANFDx_CHy_IR.RFnF is set.

If a message is received while the corresponding RX FIFO is full, this message is discarded and the message lost condition is signaled by CANFDx_CHy_RXFnS.RFnL = 1. In addition, the interrupt flag CANFDx_CHy_IR.RFnL is set.

RX FIFO overwrite mode

The RX FIFO overwrite mode is configured by CANFDx_CHy_RXFnC.FnOM = 1.

When an RX FIFO full condition (CANFDx_CHy_RXFnS.FnPI = CANFDx_CHy_RXFnS.FnGI) is signaled by CANFDx_CHy_RXFnS.FnF = 1, the next message accepted for the FIFO will overwrite the oldest FIFO message. Put and Get indices are both incremented by one.

When an RX FIFO is operated in overwrite mode and an RX FIFO full condition is signaled, reading of the RX FIFO elements should start at least at get index + 1. This is because a received message may be written to the message RAM (put index) while the CPU is reading from the message RAM (get index). In this case, inconsistent data may be read from the respective RX FIFO element. Adding an offset to the get index when reading from the RX FIFO avoids this problem. The offset depends on how fast the CPU accesses the RX FIFO. Figure 163 shows an offset of two with respect to the get index when reading the RX FIFO. In this case, the two messages stored in element 1 and 2 are lost.

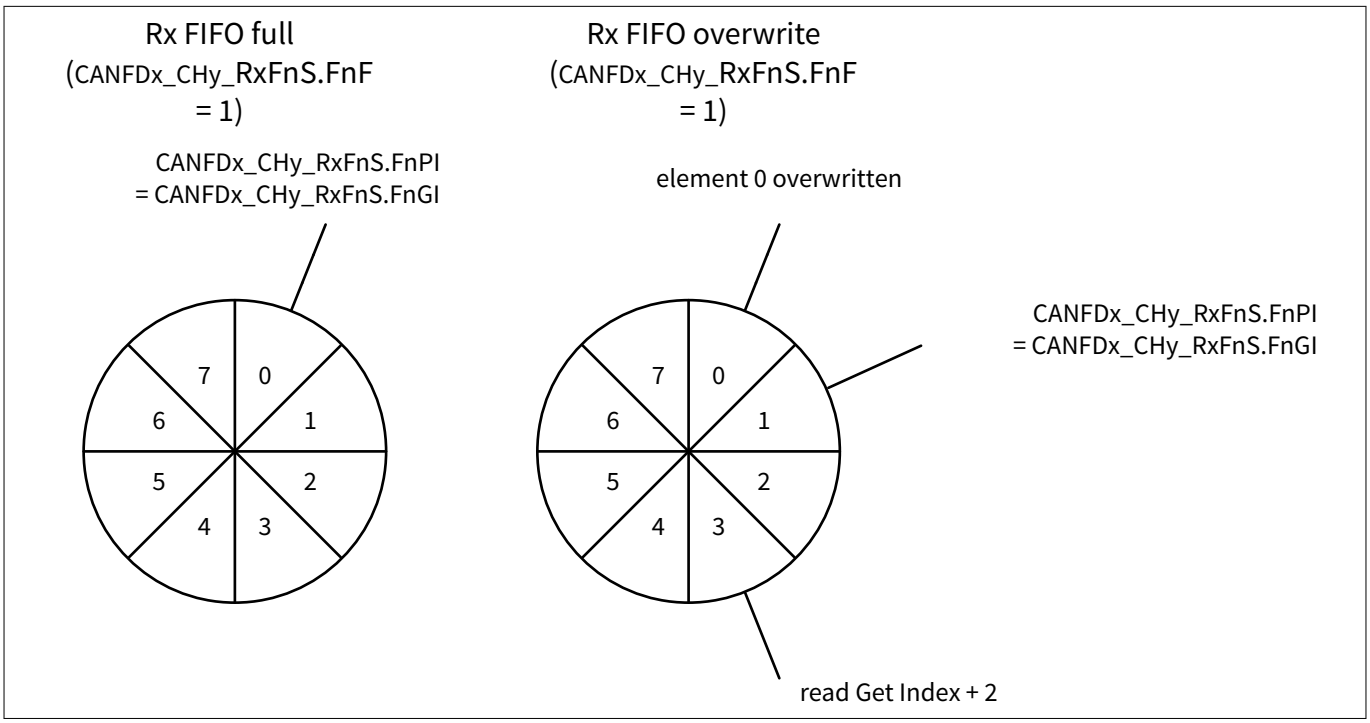


Figure 163 RX FIFO overflow handling

After reading from the RX FIFO, the number of the last element read must be written to the RX FIFO acknowledge Index CANFDx_CHy_RXFnA.FnA. This increments the get index to that element number. If the put index is not incremented to this RX FIFO element, the RX FIFO full condition is reset (CANFDx_CHy_RXFnS.FnF = 0).

RX FIFO top pointer

M_TTCAN supports two receive FIFOs. Reading from these FIFOs requires application to go through following steps:

- Retrieve read pointer
- Calculate correct message RAM address
- Read the data from message RAM
- Update the read pointer

To avoid all these steps, RX FIFO Top Pointer logic has been integrated in the CAN FD controller. It provides a single MMIO location (CANFDx_CHy_RXFTOPn_DATA; n = 0,1) to read the data from. Using such hardware logic has the following benefits:

- Higher performance data access
- Less bus traffic
- Reduced CPU load

- Reduced power
- Enables DMA access to FIFO

This logic is enabled when CANFDx_CHy_RXFTOP_CTL.FnTPE is set. Setting this bit enables the logic to set the FIFO top address (FnTA) and internal message word counter. Receive FIFO in the top Status Register (CANFDx_CHy_RXFTOPn_STAT) shows the respective FIFO top address and CANFDx_CHy_RXFTOPn_DATA provides the data located at the top address. Refer to register definitions for more details on both registers.

If CANFDx_CHy_RXFTOPn_DATA is read, the top pointer logic also updates the RX FIFO acknowledge index (CANFDx_CHy_RXFnA.FnA) in TTCAN channel.

***Note**: Top pointer logic is disabled when the channel is being configured (CANFDx_CHy_CCCR.CCE = 1). Reading CANFDx_CHy_RXFTOPn_DATA while the logic is disabled will return the invalid data.*

24.3.4.3 Dedicated RX buffers

The M_TTCAN supports up to 64 dedicated RX buffers. The start address of the dedicated RX buffer section is configured via CANFDx_CHy_RXBC.RBSA.

For each RX buffer, a standard or extended message ID filter element with SFEC/EFEC = 111 and SFID2/ EFID2[10:9] = 00 must be configured (see Standard message ID filter element and Extended message ID filter element).

After a received message is accepted by a filter element, the message is stored into the RX buffer in the message RAM referenced by the filter element. The format is the same as for an RX FIFO element. In addition, the flag CANFDx_CHy_IR.DRX (message stored in a dedicated RX buffer) in the Interrupt Register is set.

Table 177 Example filter configuration for RX buffers

Filter element	SFID1[10:0] EFID1[28:0]	SFID2[10:9] EFID2[10:9]	SFID2[5:0] EFID2[5:0]
0	ID message 1	00	00 0000
1	ID message 2	00	00 0001
2	ID message 3	00	00 0010

After the last word of a matching received message is written to the message RAM, the respective new data flag in CANFDx_CHy_NDAT1 and CANFDx_CHy_NDAT2 Registers is set. As long as the new data flag is set, the respective RX buffer is locked against updates from received matching frames. The new data flags should be reset by the host by writing a '1' to the respective bit position.

While an RX buffer's new data flag is set, a message ID filter element referencing the specific RX buffer will not match, causing the acceptance filtering to continue. The following message ID filter elements may cause the received message to be stored into another RX buffer, or into an RX FIFO, or the message may be rejected, depending on filter configuration.

RX buffer handling:

- Reset interrupt flag CANFDx_CHy_IR.DRX
- Read new data Registers
- Read messages from message RAM
- Reset new data flags of processed messages

24.3.4.4 Debug on CAN support

Debug messages are stored into RX buffers; three consecutive RX buffers (for example, #61, #62, and #63) should be used to store debug messages A, B, and C. The format is the same for RX buffer and RX FIFO elements.

To filter debug messages standard/extended filter elements with SFEC/EFEC = “111” should be set up. Messages that match these filter elements are stored into the RX buffers addressed by SFID2/EFID2[5:0].

After message C is stored, the DMA request is activated and the three messages can be read from the message RAM under DMA control. The RAM words holding the debug messages will not be changed by the M_TTCAN while DMA request is activated. The behavior is similar to that of an RX buffer with its new data flag set.

After the DMA transfer is completed, an acknowledge from DMA resets the DMA request. Now the M_TTCAN is prepared to receive the next set of debug messages.

Filtering debug messages

Debug messages are filtered by configuring one standard/extended message ID filter element for each of the three debug messages. To enable a filter element to filter debug messages, SFEC/EFEC should be programmed to “111”. In this case the SFID1/SFID2 and EFID1/EFID2 fields have a different meaning (see Standard message ID filter element and Extended message ID filter element). While SFID2/EFID2[10:9] controls the debug message handling state machine, SFID2/EFID2[5:0] controls the storage location of a received debug message.

When a debug message is stored, neither the respective new data flag nor CANFDx_CHy_IR.DRX are set. The reception of debug messages can be monitored via CANFDx_CHy_RXF1S.DMS.

Table 178 Example filter configuration for debug message

Filter element	SFID1[10:0] EFID1[28:0]	SFID2[10:9] EFID2[10:9]	SFID2[5:0] EFID2[5:0]
0	ID debug message A	01	11 1101
1	ID debug message B	10	11 1110
2	ID debug message C	11	11 1111

Debug message handling

The debug message handling state machine assures that debug messages are stored to three consecutive RX buffers in the correct order. If there are missing messages, the process is restarted. The DMA request is activated only when all three debug messages A, B, and C are received in correct order.

The status of the debug message handling state machine is signaled via CANFDx_CHy_RXF1S.DMS.

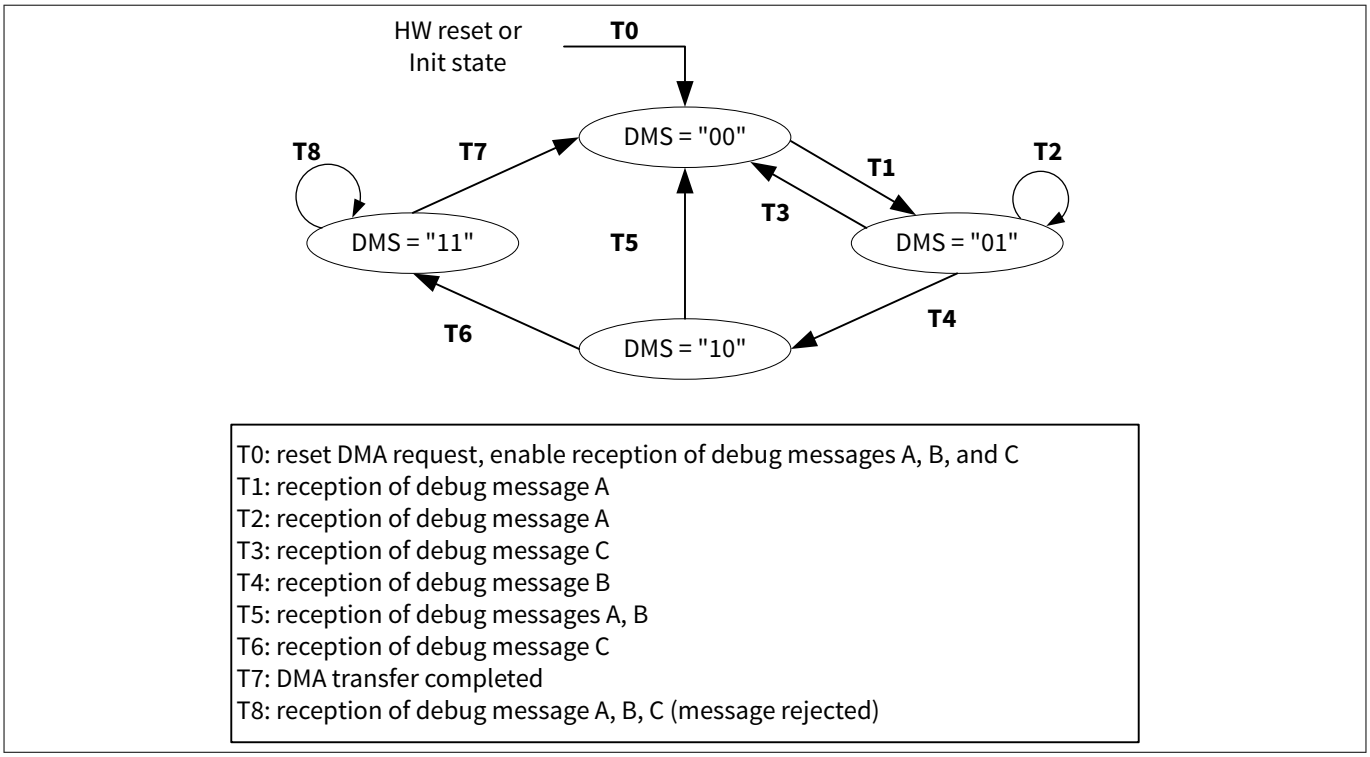


Figure 164 Debug message handling state machine

24.3.5 TX handling

The TX handler handles transmission requests for the dedicated TX buffers, TX FIFO, and TX Queue. It controls the transfer of transmit messages to the CAN Core, the Put and Get Indices, and the TX Event FIFO. Up to 32 TX buffers can be set up for message transmission. The CAN mode for transmission (classic CAN or CAN FD) can be configured separately for each TX buffer element. The TX buffer element is described in TX buffer element. Table 179 describes the possible configurations for frame transmission.

Table 179 Possible configuration for frame transmission

CANFDx_CHy_CCCR		TX buffer element		Frame transmission
BRSE	**FDOE**	**FDF**	**BRS**	
ignored	0	ignored	ignored	Classic CAN
0	1	0	ignored	Classic CAN
0	1	1	ignored	FD without bit rate switching
1	1	0	ignored	Classic CAN
1	1	1	0	FD without bit rate switching
1	1	1	1	FD with bit rate switching

The TX handler starts a TX scan to check for the highest priority pending TX request (TX buffer with lowest message ID) when the TX buffer request pending Register (CANFDx_CHy_TXBRP) is updated, or when a transmission is started.

24.3.5.1 Transmit pause

The transmit pause feature is intended for use in CAN systems where the CAN message identifiers are (permanently) assigned to specific values and cannot be changed easily. These message identifiers may have a higher CAN arbitration priority than other defined messages, while in a specific application their relative arbitration priority should be inverse. This may lead to a case where one electronic control unit (ECU) sends a burst of CAN messages that cause another ECU's CAN messages to be delayed because the other messages have a lower CAN arbitration priority.

For example, if CAN ECU-1 has the transmit pause feature enabled and is requested by the application software to transmit four messages, it will, after the first successful message transmission, wait for two nominal bit times of bus idle before it is allowed to start the next requested message. If there are other ECUs with pending messages, those messages are started in the idle time, they will not need to arbitrate with the next message of ECU-1. After having received a message, ECU-1 is allowed to start its next transmission as soon as the received message releases the CAN bus.

The transmit pause feature is controlled by the transmit pause bit (CANFDx_CHy_CCCR.TXP). If the bit is set, the M_TTCAN controller will, each time it has successfully transmitted a message, pause for two nominal bit times before starting the next transmission. This enables other CAN nodes in the network to transmit messages even if their messages have lower prior identifiers. Default is transmit pause disabled (CANFDx_CHy_CCCR.TXP = 0).

This feature loses burst transmissions coming from a single node and protects against “babbling idiot” scenarios where the application program erroneously requests too many transmissions.

24.3.5.2 Dedicated TX buffers

Dedicated TX buffers are intended for message transmission under complete control of the CPU. Each dedicated TX buffer is configured with a specific message ID. If multiple TX buffers are configured with the same message ID, then these Tx buffers shall be requested in ascending order with the lowest buffer number first. Alternatively, all Tx buffers configured with the same message ID can be requested simultaneously by a single write access to CANFDx_CHy_TXBAR.

If the data section is updated, a transmission is requested by an add request via CANFDx_CHy_TXBAR.ARn. The requested messages arbitrate internally with messages from an optional TX FIFO or TX Queue and externally with messages on the CAN bus, and are sent out according to their message ID.

Addressing dedicated TX buffers

A dedicated TX buffer allocates an element size of 32-bit words in the message RAM as shown in the Table 180. Therefore, the start address of a dedicated TX buffer in the message RAM is calculated by (transmit buffer index (0 to 31) × element size) + TX buffers start address (CANFDx_CHy_TXBC.TBSA[15:2]).

Table 180 TX buffer/FIFO/queue element size

CANFDx_CHy_TXESC.TBDS[2:0]	Data field [bytes]	Element size [RAM words]
000	8	4
001	12	5
010	16	6
011	20	7
100	24	8
101	32	10
110	48	14
111	64	18

24.3.5.3 TX FIFO

TX FIFO operation is configured by programming CANFDx_CHy_TXBC.TFQM to '0'. Messages stored in the TX FIFO are transmitted starting with the message referenced by the Get Index CANFDx_CHy_TXFQS.TFGI. After each transmission, the get index is incremented cyclically until the TX FIFO is empty. The TX FIFO enables transmission of messages with the same message ID from different TX buffers in the order these messages are written to the TX FIFO. The M_TTCAN calculates the TX FIFO Free Level CANFDx_CHy_TXFQS.TFFL as difference between get and put index. It indicates the number of available (free) TX FIFO elements.

New transmit messages must be written to the TX FIFO starting with the TX buffer referenced by the put index CANFDx_CHy_TXFQS.TFQPI. An add request increments the put index to the next free TX FIFO element. When the put index reaches the get index, TX FIFO Full (CANFDx_CHy_TXFQS.TFQF = 1) is signaled. In this case no further messages should be written to the TX FIFO until the next message is transmitted and the Get Index is incremented.

When a single message is added to the TX FIFO, the transmission is requested by writing a '1' to the CANFDx_CHy_TXBAR bit related to the TX buffer referenced by the TX FIFO's Put Index.

When multiple (n) messages are added to the TX FIFO, they are written to n consecutive TX buffers starting with the put index. The transmissions are then requested via CANFDx_CHy_TXBAR. The put Index is then cyclically incremented by n. The number of requested TX buffers should not exceed the number of free TX buffers as indicated by the TX FIFO free level.

When a transmission request for the TX buffer referenced by the get index is canceled, the get index is incremented to the next TX buffer with pending transmission request and the TX FIFO Free Level is recalculated. When transmission cancellation is applied to any other TX buffer, the get index and the FIFO free level remain unchanged.

A TX FIFO element allocates element size 32-bit words in the message RAM as shown in Table 180. Therefore, the start address of the next available (free) TX FIFO buffer is calculated by adding TX FIFO/queue put index CANFDx_CHy_TXFQS.TFQPI (0…31) • Element size to the TX buffer start address CANFDx_CHy_TXBC.TBSA.

24.3.5.4 TX queue

TX Queue operation is configured by programming CANFDx_CHy_TXBC.TFQM to '1'. Messages stored in the TX queue are transmitted starting with the message with the lowest message ID (highest priority). If multiple Tx queue buffers are configured with the same message ID, then the transmission order depends on the numbers of the buffers where the messages were stored for transmission. As these buffer numbers depend on the then current states of the PUT index, a prediction of the transmission order is not possible.

New messages must be written to the TX buffer referenced by the put index CANFDx_CHy_TXFQS.TFQPI. The put index always points to the free buffer of the Tx queue with the lowest buffer number. If the TX queue is full (CANFDx_CHy_TXFQS.TFQF = 1), the put index is not valid and no further message should be written to the TX queue until at least one of the requested messages is sent out or a pending transmission request is canceled.

The application may use the CANFDx_CHy_TXBRP Register instead of the put index and may place messages to any TX buffer without pending transmission request.

A TX queue buffer allocates element size of 32-bit words in the message RAM as shown in Table 180. Therefore, the start address of the next available (free) TX queue buffer is calculated by adding TX FIFO/queue put index CANFDx_CHy_TXFQS.TFQPI (0…31) × Element size to the TX buffer start address CANFDx_CHy_TXBC.TBSA.

24.3.5.5 Mixed dedicated TX buffers/TX FIFO

In this case, the TX buffers section in the message RAM is subdivided into a set of dedicated TX buffers and a TX FIFO. The number of dedicated TX buffers is configured by CANFDx_CHy_TXBC.NDTB. The number of TX buffers assigned to the TX FIFO is configured by CANFDx_CHy_TXBC.TFQS. If CANFDx_CHy_TXBC.TFQS is programmed to zero, only the dedicated TX buffers are used.

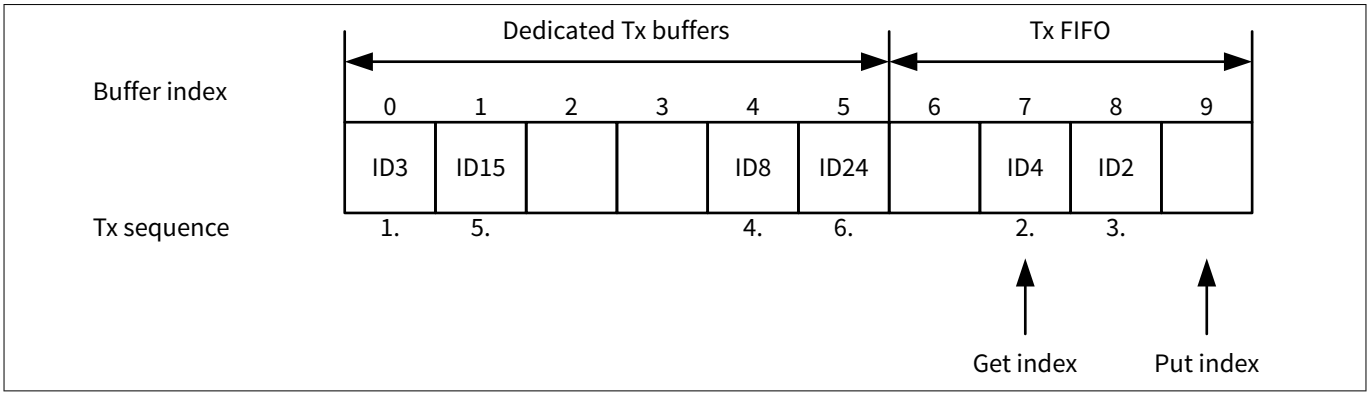


Figure 165 Example of mixed configuration dedicated TX buffers/TX FIFO

TX prioritization:

- Scan dedicated TX buffers and oldest pending TX FIFO buffer (referenced by CANFDx_CHy_TXFS.TFGI)
- Buffer with the lowest message ID gets highest priority and is transmitted next

24.3.5.6 Mixed dedicated TX buffers/TX queue

In this case the TX buffers section in the message RAM is subdivided into a set of dedicated TX buffers and a TX queue. The number of dedicated TX buffers is configured by CANFDx_CHy_TXBC.NDTB. The number of TX queue buffers is configured by CANFDx_CHy_TXBC.TFQS. In case CANFDx_CHy_TXBC.TFQS is programmed to zero, only dedicated TX buffers are used.

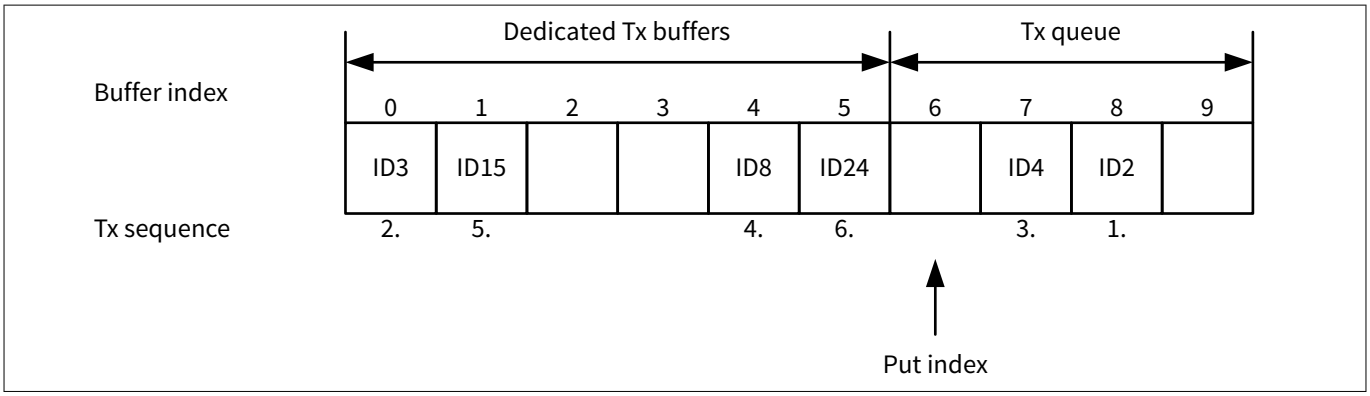


Figure 166 Example of mixed configuration dedicated TX buffers/TX queue

TX prioritization:

- Scan all TX buffers with activated transmission request
- TX buffer with the lowest message ID gets highest priority and is transmitted next

24.3.5.7 Transmit cancellation

The M_TTCAN supports transmit cancellation. This feature is especially intended for gateway applications. To cancel a requested transmission from a dedicated TX buffer or a TX queue buffer the host must write a '1' to the corresponding bit position (number of TX buffers) of the CANFDx_CHy_TXBCR Register. Transmit cancellation is not intended for TX FIFO operation.

Successful cancellation is signaled by setting the corresponding bit of the CANFDx_CHy_TXBCF Register to '1'.

In case a transmit cancellation is requested while a transmission from a TX buffer is ongoing, the corresponding CANFDx_CHy_TXBRP bit remains set as long as the transmission is in progress. If the transmission was successful, the corresponding CANFDx_CHy_TXBTO and CANFDx_CHy_TXBCF bits are set. If the transmission was not successful, it is not repeated and only the corresponding CANFDx_CHy_TXBCF bit is set.

***Note**: If a pending transmission is canceled immediately before this transmission is started, there follows a short time window where no transmission is started even if another message is also pending in this node. This may enable another node to transmit a message, which may have a lower priority than the second message in this node.*

24.3.5.8 TX event handling

To support TX event handling, the M_TTCAN has implemented a TX event FIFO. After the M_TTCAN has transmitted a message on the CAN bus, the message ID and timestamp are stored in a TX Event FIFO element. To link a TX event to a TX event FIFO element, the message marker from the transmitted TX buffer is copied into the TX event FIFO element.

The TX event FIFO can be configured to a maximum of 32 elements. The TX event FIFO element is described in TX event FIFO element.

The purpose of the TX event FIFO is to decouple handling transmit status information from transmit message handling; that is, a TX buffer holds only the message to be transmitted, while the transmit status is stored separately in the TX event FIFO. This has the advantage, especially when operating a dynamically managed transmit queue, that a TX buffer can be used for a new message immediately after successful transmission. There is no need to save transmit status information from a TX buffer before overwriting that TX buffer.

When a TX event FIFO full condition is signaled by CANFDx_CHy_IR.TEFF, no further elements are written to the TX event FIFO until at least one element is read out and the TX event FIFO Get Index is incremented. In case a TX event occurs while the TX event FIFO is full, this event is discarded and interrupt flag CANFDx_CHy_IR.TEFL is set.

To avoid a TX event FIFO overflow, the TX event FIFO watermark can be used. When the TX event FIFO fill level reaches the TX event FIFO watermark configured by CANFDx_CHy_TXEFC.EFWM, interrupt flag CANFDx_CHy_IR.TEFW is set.

When reading from the TX event FIFO, the TX Event FIFO Get Index CANFDx_CHy_TXEFS.EFGI must be added twice to the TX event FIFO start address CANFDx_CHy_TXEFC.EFSA.

24.3.6 FIFO acknowledge handling

The get indices of RX FIFO 0, RX FIFO 1, and the TX Event FIFO are controlled by the corresponding FIFO acknowledge Index.

When RX FIFO top pointer hardware logic is used, it updates the RX FIFO acknowledge Index. After CANFDx_CHy_RXFTOPn_DATA is read, the acknowledge Index (CANFDx_CHy_RXFnA.FnA) is updated automatically, which will eventually set the FIFO get index to the FIFO acknowledge index plus one and thereby updates the FIFO fill Level.

When the application does not use RX FIFO top pointer logic, the acknowledge Index must be updated. This can be done using one of the following two use cases:

- When only a single element is read from the FIFO (the one being pointed to by the get index), the get index value is written to the FIFO acknowledge index.
- When a sequence of elements is read from the FIFO, it is sufficient to write the FIFO acknowledge index only once at the end of that read sequence (value is the index of the last element read), to update the FIFO's Get Index.

Because the CPU has free access to the M_TTCAN's message RAM, take care when reading FIFO elements in an arbitrary order (get index not considered). This may be useful when reading a high-priority message from one of the two RX FIFOs. In this case the FIFO acknowledge index should not be written because this will set the get index to a wrong position and alter the FIFO's fill level. Some older FIFO elements are lost.

***Note**: The application must ensure that a valid value is written to the FIFO acknowledge index. The M_TTCAN does not check for erroneous values.*

24.3.7 Configuring the CAN bit timing

Each node in the CAN network has its own clock generator (usually a quartz oscillator). The time parameter of the bit time can be configured individually for each CAN node. Even if each CAN node's oscillator has a different period, a common bit rate can be generated.

The oscillator frequencies vary slightly because of changes in temperature or voltage, or deterioration of components. As long as the frequencies vary only within the tolerance range of the oscillators, the CAN nodes can compensate for the different bit rates by resynchronizing to the bit stream.

24.3.7.1 CAN bit timing

The CAN FD operation defines two bit times – nominal bit time and data bit time. The nominal bit time is for the arbitration phase. The data bit time has an equal or shorter length and can be used to accelerate the data phase (see CAN FD operation).

The basic construction of a bit time is shared with both the nominal and data bit times. The bit time can be divided into four segments according to the CAN specifications (see Figure 167: the synchronization segment (Sync_Seg), the propagation time segment (Prop_Seg), the phase buffer segment 1 (Phase_Seg1), and the phase buffer segment 2 (Phase_Seg2). The sample point at which the bus level is read and interpreted as the value of that respective bit, is located at the end of Phase_Seg1.

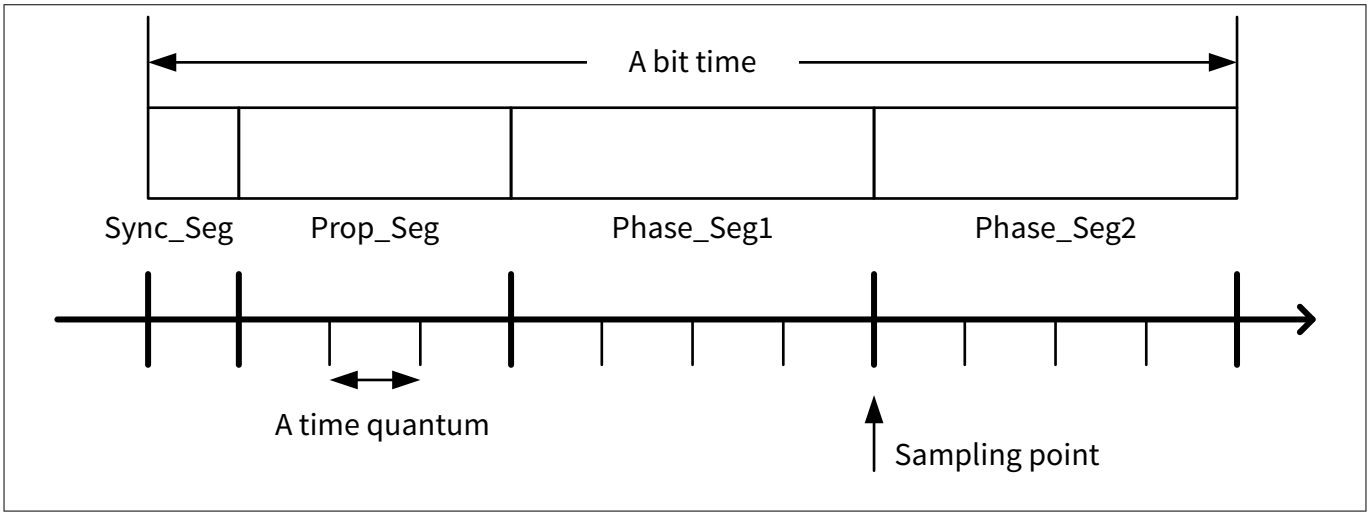


Figure 167 Bit time construction

Each segment consists of a programmable number of time quanta, which is a multiple of the time quantum that is defined by PCLK_CANFD[x]_CLOCK_CAN[y] and a prescaler. The values and prescalers used to define these parameters differ for the nominal and data bit times, and are configured by CANFDx_CHy_NBTP (Nominal Bit Timing and Prescaler Register) and CANFDx_CHy_DBTP (Data Bit Timing and Prescaler Register) as shown in Table 181.

Table 181 Bit time parameters

Parameter	Description
Time quantum tq (nominal) and tqd (data)	Time quantum. Derived by multiplying the basic unit time quanta (the PCLK_CANFD[x]_CLOCK_CAN[y] period) with the respective prescaler. The time quantum is configured by the CAN FD controller as nominal: tq = (CANFDx_CHy_NBTP.NBRP[8:0] +1) × PCLK_CANFD[x]_CLOCK_CAN[y] period data: tqd = (CANFDx_CHy_DBTP.DBRP[4:0] + 1) × PCLK_CANFD[x]_CLOCK_CAN[y] period
Sync_Seg	Sync_Seg is fixed to one time quantum as defined by the CAN specifications and is not configurable (inherently built into the CAN FD controller). nominal: 1 tq data: 1 tqd
Prop_Seg	Prop_Seg is the part of the bit time that is used to compensate for the physical delay times within the network. The CAN FD controller configures the sum of Prop_Seg and Phase_Seg1 with a single parameter: nominal: Prop_Seg + Phase_Seg1 = CANFDx_CHy_NBTP.NTSEG1[7:0] + 1 data: Prop_Seg + Phase_Seg1 = CANFDx_CHy_DBTP.DTSEG1[4:0] + 1
Phase_Seg1	Phase_Seg1 is used to compensate for edge phase errors before the sampling point. Can be lengthened by the resynchronization jump width. The sum of Prop_Seg and Phase_Seg1 is configured by the CAN FD controller as nominal: CANFDx_CHy_NBTP.NTSEG1[7:0] + 1 data: CANFDx_CHy_DBTP.DTSEG1[4:0] + 1
Phase_Seg2	Phase_Seg2 is used to compensate for edge phase errors after the sampling point. Can be shortened by the resynchronization jump width. Phase_Seg2 is configured by the CAN FD controller as nominal: CANFDx_CHy_NBTP.NTSEG2[6:0] + 1 data: CANFDx_CHy_DBTP.DTSEG2[3:0] + 1
SJW	Resynchronization Jump Width. Used to adjust the length of Phase_Seg1 and Phase_Seg2. SJW will not be longer than either Phase_Seg1 or Phase_Seg2. SJW is configured by the CAN FD controller as nominal: CANFDx_CHy_NBTP.NSJW[6:0] + 1 data: CANFDx_CHy_DBTP.DSJW[3:0] + 1

These relations result in the following equations for the nominal and data bit times:

Nominal bit time

= [Sync_Seg + Prop_Seg + Phase_Seg1 + Phase_Seg2] × tq

= [1 + (CANFDx_CHy_NBTP.NTSEG1[7:0] + 1) + (CANFDx_CHy_NBTP.NTSEG2[6:0] + 1)] × [(CANFDx_CHy_NBTP.NBRP[8:0] +1) × PCLK_CANFD[x]_CLOCK_CAN[y] period]

Data bit time

= [1 + (CANFDx_CHy_DBTP.DTSEG1[4:0] + 1) + (CANFDx_CHy_DBTP.DTSEG2[3:0] + 1)] × [(CANFDx_CHy_DBTP.DBRP[4:0] +1) × PCLK_CANFD[x]_CLOCK_CAN[y] period]

***Note**: The Information Processing Time (IPT) of the CAN FD controller is zero; this means that the data for the next bit is available at the first CAN clock edge after the sample point. Therefore, the IPT does not have to be accounted for when configuring Phase_Seg2, which is the maximum of Phase_Seg1 and the IPT.*

24.3.7.2 CAN bit rates

The bit rate is the inverse of bit time; therefore, the nominal bit rate is

1 / [{1 + (CANFDx_CHy_NBTP.NTSEG1[7:0] + 1) + (CANFDx_CHy_NBTP.NTSEG2[6:0] + 1)} × {(CANFDx_CHy_NBTP.NBRP[8:0] +1) × PCLK_CANFD[x]_CLOCK_CAN[y] period}]

and the data bit rate is

1 / [{1 + (CANFDx_CHy_DBTP.DTSEG1[4:0] + 1) + (CANFDx_CHy_DBTP.DTSEG2[3:0] + 1)} × {(CANFDx_CHy_DBTP.DBRP[4:0] +1) × PCLK_CANFD[x]_CLOCK_CAN[y] period}]

From these formulas, we can see that the bit rates of the CAN FD controller depends on the CAN clock (PCLK_CANFD[x]_CLOCK_CAN[y]) period, and the range each parameter can be configured to. The following tables list examples of the configurable bit rates at varying CAN clock frequencies. Empty boxes indicate that the desired bit rate cannot be configured at the specified input CAN clock frequency.

CAN clock frequency	8 MHz		10 MHz		16 MHz		20 MHz		32 MHz		40 MHz	
configuration / nominal bit rate	Number of tqs per bit time	CANFDx_CHy_NBTP.NBRP + 1	Number of tqs per bit time	CANFDx_CHy_NBTP.NBRP + 1	Number of tqs per bit time	CANFDx_CHy_NBTP.NBRP + 1	Number of tqs per bit time	CANFDx_CHy_NBTP.NBRP + 1	Number of tqs per bit time	CANFDx_CHy_NBTP.NBRP + 1	Number of tqs per bit time	CANFDx_CHy_NBTP.NBRP + 1
125 Kbps	64tq 32tq 16tq 8tq	1 2 4 8	80tq 40tq 20tq 10tq	1 2 4 8	128tq 64tq 32tq 16tq 8tq	1 2 4 8 16	160tq 80tq 40tq 20tq 10tq	1 2 4 8 16	256tq 128tq 64tq 32tq 16tq 8tq	1 2 4 8 16 32	320tq 160tq 80tq 40tq 20tq 10tq	1 2 4 8 16 32
250 Kbps	32tq 16tq 8tq	1 2 4	40tq 20tq 10tq	1 2 4	64tq 32tq 16tq 8tq	1 2 4 8	80tq 40tq 20tq 10tq	1 2 4 8	128tq 64tq 32tq 16tq 8tq	1 2 4 8 16	160tq 80tq 40tq 20tq 10tq	1 2 4 8 16
500 Kbps	16tq 8tq	1 2	20tq 10tq	1 2	32tq 16tq 8tq	1 2 4	40tq 20tq 10tq	1 2 4	64tq 32tq 16tq 8tq	1 2 4 8	80tq 40tq 20tq 10tq	1 2 4 8
1 Mbps	8tq	1	10tq	1	16tq 8tq	1 2	20tq 10tq	1 2	32tq 16tq 8tq	1 2 4	40tq 20tq 10tq	1 2 4

Figure 168 Example configuration for nominal bit rates

CAN clock frequency	8 MHz		10 MHz		16 MHz		20 MHz		32 MHz		40 MHz	
configuration / data bit rate	Number of tqds per bit time	CANFDx_CHy_DB TP.DBRP + 1	Number of tqds per bit time	CANFDx_CHy_DB TP.DBRP + 1	Number of tqds per bit time	CANFDx_CHy_DB TP.DBRP + 1	Number of tqds per bit time	CANFDx_CHy_DB TP.DBRP + 1	Number of tqds per bit time	CANFDx_CHy_DB TP.DBRP + 1	Number of tqds per bit time	CANFDx_CHy_DB TP.DBRP + 1
500 Kbps	16tqd 8tqd	1 2	20tqd 10tqd	1 2	32tqd 16tqd 8tqd	1 2 4	40tqd 20tqd 10tqd	1 2 4	32tqd 16tqd 8tqd	2 4 8	40tqd 20tqd 10tqd	2 4 8
1 Mbps	8tqd	1	10tqd	1	16tqd 8tqd	1 2	20tqd 10tqd	1 2	32tqd 16tqd 8tqd	1 2 4	40tqd 20tqd 10tqd	1 2 4
2 Mbps	-	-	-	-	8tqd	1	10tqd	1	16tqd 8tqd	1 2	20tqd 10tqd	1 2
4 Mbps	-	-	-	-	-	-	-	-	8tqd	1	10tqd	1
5 Mbps	-	-	-	-	-	-	-	-	-	-	8tqd	1
8 Mbps	-	-	-	-	-	-	-	-	-	-	5tqd	1

Figure 169 Example configuration for data bit rates

***Note**: The user must configure the CAN bit timings to comply with the corresponding CAN standards to ensure proper communication on the CAN bus.*

24.4 Message RAM

Message RAM (MRAM) in XMC7000 family devices is shared among multiple M_TTCAN channels present in the M_TTCAN group. Refer to the device datasheet for the supported number of M_TTCAN groups, M_TTCAN channels in each group, and the total message RAM allocated to each group. Each M_TTCAN channel in the group can configure its required message RAM according to application requirements.

The message RAM stores RX/TX messages and filter configurations.

***Note**:*
- *The message RAM should be made zero before configuration of the CAN FD controller to prevent bit errors when reading uninitialized words and ECC errors, and to avoid unexpected filter element configurations in the message RAM.*
- *Unused message RAM cannot be used for general purposes.*

24.4.1 Message RAM configuration

The message RAM has a width of 32 bits. The CAN FD controller can be configured to allocate up to 4480 words in the message RAM (note that the number of words that can be used will be limited by the size of the actual message RAM). It is not necessary to configure each of the sections listed in Figure 170, nor is there any restriction with respect to the sequence of the sections.

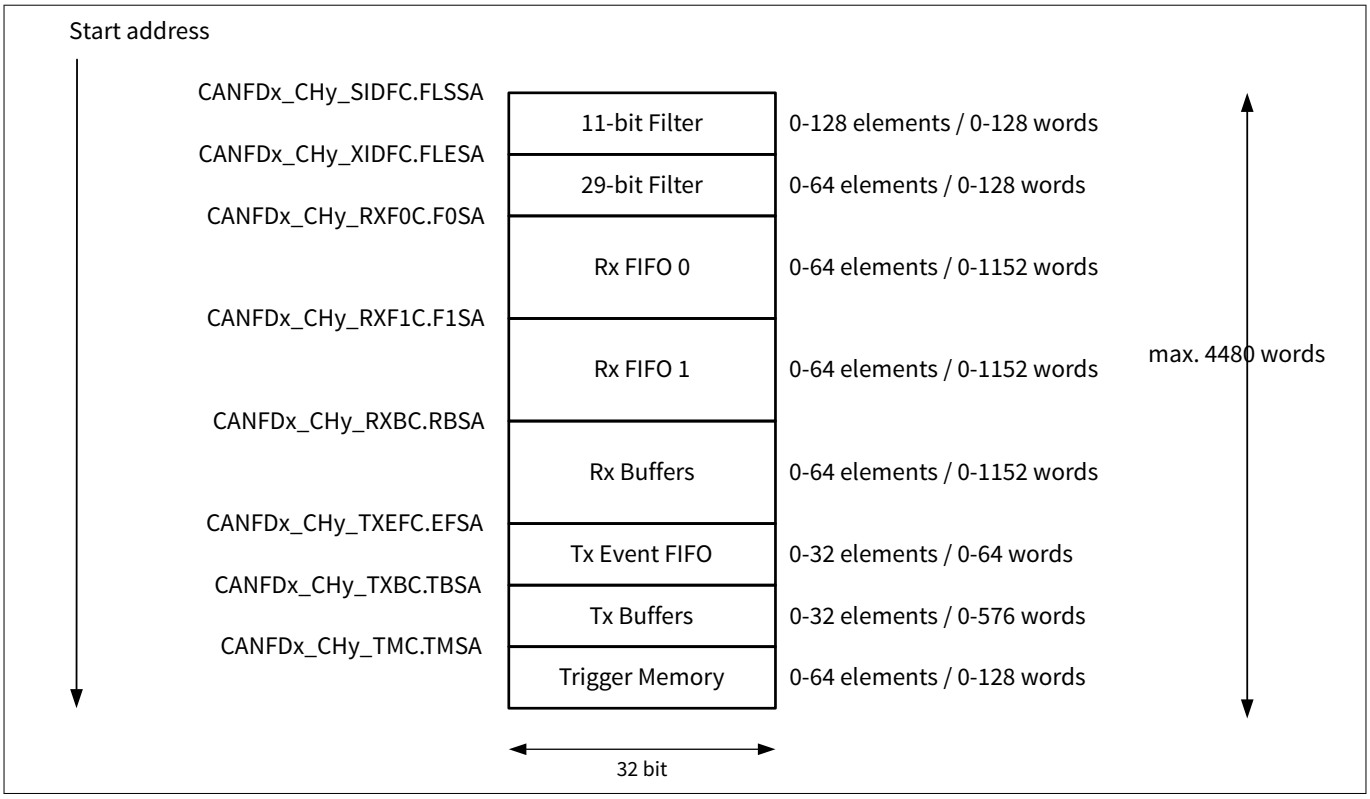


Figure 170 Message RAM configuration

The CAN FD controller addresses the message RAM in 32-bit words, not single bytes. The configurable start addresses are 32-bit word addresses – only bits 15 to 2 are evaluated, the two least significant bits are ignored.

***Note**:*

- *The CAN FD controller does not check for erroneous configuration of the message RAM. The configuration of the start addresses of different sections and the number of elements of each section should be done carefully to avoid falsification or loss of data.*
- *Message RAM is accessible by both M_TTCAN and CPU. Dynamic round-robin scheme is implemented to allocate access.*

24.4.2 RX buffer and FIFO element

An RX buffer and FIFO element is a block of 2-bit words, which holds the data and status of a received frame that was stored in the message RAM.

Up to 64 RX buffers and two RX FIFOs can be configured in the message RAM. Each RX FIFO section can be configured to store up to 64 received messages. The structure of an RX buffer and FIFO element is shown in Figure 171. The element size can be configured to store CAN FD messages with up to 64 bytes data field via Register CANFDx_CHy_RXESC (RX buffer/FIFO element size configuration).

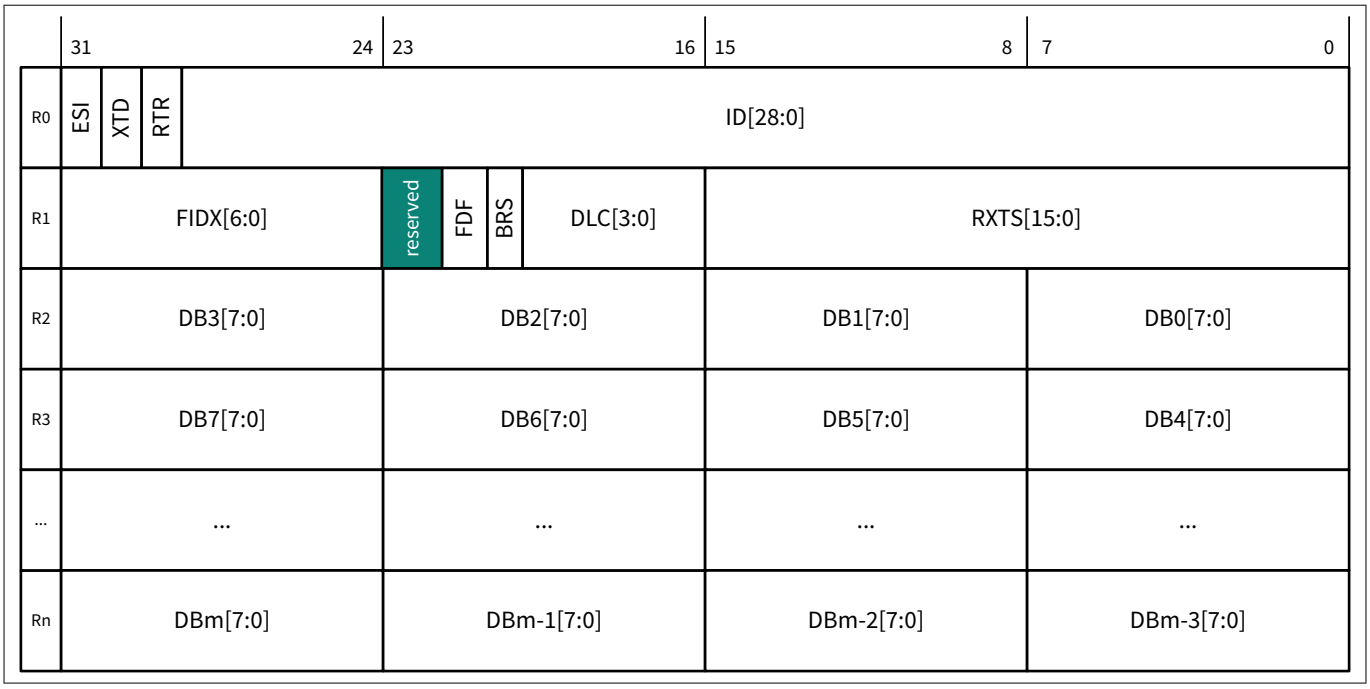


Figure 171 RX buffer and FIFO

R0 [bit31] ESI: Error state indicator

Bit	Description
0	Transmitting node is error active
1	Transmitting node is error passive

R0 [bit30] XTD: Extended identifier

Signals to the CPU whether the received frame has a standard or extended identifier.

Bit	Description
0	11-bit standard identifier
1	29-bit extended identifier

R0 [bit29] RTR: Remote transmission request

Signals to the CPU whether the received frame is a data frame or a remote frame.

Bit	Description
0	Received frame is a data frame
1	Received frame is a remote frame

***Note**: There are no remote frames in CAN FD format. In CAN FD frames (FDF = 1), the dominant RRS (remote request substitution) bit replaces bit RTR (remote transmission request).*

R0 [bit28:0] ID[28:0]: Identifier

Standard or extended identifier depending on bit XTD. A standard identifier is stored into ID[28:18].

R1 [bit31] ANMF: Accepted non-matching frame

Acceptance of non-matching frames may be enabled via CANFDx_CHy_GFC.ANFS[1:0] (accept non-matching frames standard) and CANFDx_CHy_GFC.ANFE[1:0] (accept non-matching frames extended).

Bit	Description
0	Received frame matching filter index FIDX
1	Received frame did not match any RX filter element

R1 [bit30:24] FIDX[6:0]: Filter index

FIDX[6:0]	Description
0-127	Index of matching RX acceptance filter element (invalid if ANMF = 1). Range is 0 to List Size Standard/Extended minus 1 (CANFDx_CHy_SIDFC.LSS - 1 resp. CANFDx_CHy_XIDFC.LSE - 1).

R1 [bit23:22] Reserved: Reserved bits

When writing, always write '0'. The read value is undefined.

R1 [bit21] FDF: Extended data length

Bit	Description
0	Classic CAN frame format
1	CAN FD frame format

R1 [bit20] BRS: Bit rate switch

Bit	Description
0	Frame received without bit rate switching
1	Frame received with bit rate switching

R1 [bit19:16] DLC[3:0]: Data length code

DLC[3:0]	Description
0-8	Classic CAN + CAN FD: received frame has 0-8 data bytes
9-15	Classic CAN: received frame has 8 data bytes CAN FD: received frame has 12/16/20/24/32/48/64 data bytes. See Table 175 for details.

R1 [bit15:0] RXTS[15:0]: RX timestamp

Timestamp counter value captured on start of frame reception. Resolution depending on configuration of the shared timestamp counter prescaler CANFDx_TS_CTL.PRESCALE[15:0].

R2 [bit31:24]	DB3[7:0] :	Data Byte 3
R2 [bit23:16]	DB2[7:0] :	Data Byte 2
R2 [bit15:8]	DB1[7:0] :	Data Byte 1
R2 [bit7:0]	DB0[7:0] :	Data Byte 0
R3 [bit31:24]	DB7[7:0] :	Data Byte 7

R3 [bit23:16]	DB6[7:0] :	Data Byte 6
R3 [bit15:8]	DB5[7:0] :	Data Byte 5
R3 [bit7:0]	DB4[7:0] :	Data Byte 4
…	…	…
Rn [bit31:24]	DBm[7:0]:	Data Byte m
Rn [bit23:16]	DBm-1[7:0]:	Data Byte m-1
Rn [bit15:8]	DBm-2[7:0]:	Data Byte m-2
Rn [bit7:0]	DBm-3[7:0]:	Data Byte m-3

***Notes**:*

1. *Depending on the configuration of the element size (defined by RX buffer/FIFO element size configuration (CANFDx_CHy_RXESC)), Rn will vary from n = 3 to 17.*
2. *m is a function of n, m = (n - 1) × 4 - 1.*
3. *The number of valid data bytes are defined by the data length code.*

24.4.3 TX buffer element

A TX buffer element is a block of 32-bit words stored in the message RAM that holds data and control information of a frame to be transmitted by the CAN FD controller.

The TX buffers section can be configured to hold dedicated TX buffers and a TX FIFO/TX queue. If the TX buffers section is shared by dedicated TX buffers and a TX FIFO/TX queue, the dedicated TX buffers start at the beginning of the TX buffers section followed by the buffers assigned to the TX FIFO or TX queue. The TX handler distinguishes between dedicated TX buffers and TX FIFO/TX queue by evaluating the TX buffer configuration CANFDx_CHy_TXBC.TFQS[5:0] (transmit FIFO/queue size) and CANFDx_CHy_TXBC.NDTB[5:0] (number of dedicated transmit buffers). The element size can be configured to store CAN FD messages with up to 64 bytes data field via Register CANFDx_CHy_TXESC (TX buffer element size configuration).

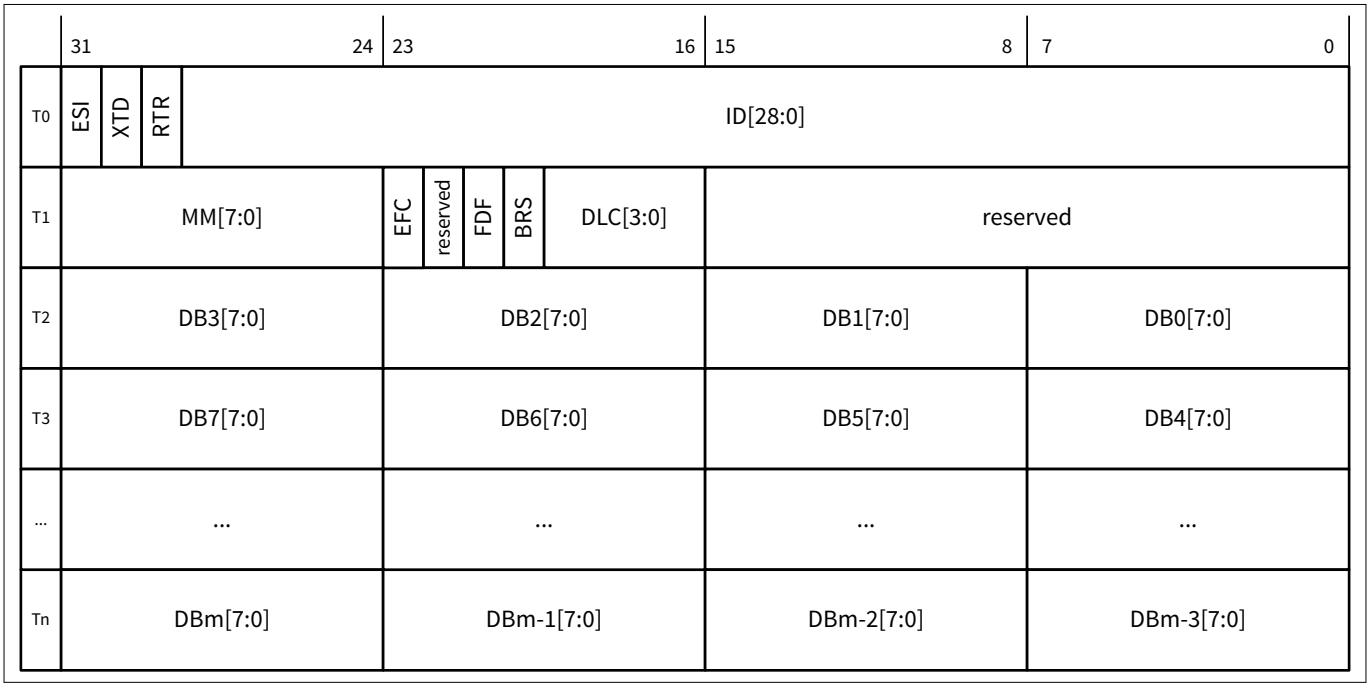


Figure 172 TX buffer element

T0 [bit31] ESI: Error state indicator

Bit	Description
0	ESI bit in CAN FD format depends only on an error passive flag
1	ESI bit in CAN FD format transmitted recessive

***Note**: The ESI bit of the transmit buffer is ORed with the error passive flag to decide the value of the ESI bit in the transmitted FD frame. As required by the CAN FD protocol specification, an error active node may optionally transmit the ESI bit recessive, but an error passive node will always transmit the ESI bit recessive.*

T0 [bit30] XTD: Extended identifier

Bit	Description
0	11-bit standard identifier
1	29-bit extended identifier

T0 [bit29] RTR: Remote transmission request

Bit	Description
0	Transmit data frame
1	Transmit remote frame

***Note**: When RTR = 1, the CAN FD controller transmits a remote frame according to ISO 11898-1, even if FD operation enable (CANFDx_CHy_CCCR.FDOE) enables the transmission in CAN FD format.*

T0 [bit28:0] ID[28:0]: Identifier

Standard or extended identifier depending on bit XTD. A standard identifier has to be written to ID[28:18].

T1 [bit31:24] MM[7:0]: Message marker

Written by CPU during TX buffer configuration. Copied into TX Event FIFO element for identification of TX message status.

T1 [bit23] EFC: Event FIFO control

Bit	Description
0	Don't store TX events
1	Store TX events

T1 [bit22] Reserved: Reserved bit

When writing, always write '0'. The read value is undefined.

T1 [bit21] FDF: FD format

Bit	Description
0	Frame transmitted in classic CAN format
1	Frame transmitted in CAN FD format

T1 [bit20] BRS: Bit rate switching

Bit	Description
0	CAN FD frames transmitted without bit rate switching
1	CAN FD frames transmitted with bit rate switching

***Note**: Bits ESI, FDF, and BRS are only evaluated when CAN FD operation is enabled CANFDx_CHy_CCCR.FDOE = 1. Bit BRS is only evaluated when in addition CANFDx_CHy_CCCR.BRSE = 1. See Table 179 for details of bits FDF and BRS.*

T1 [bit19:16] DLC[3:0]: Data length code

DLC[3:0]	Description
0-8	Classic CAN + CAN FD: transmit frame has 0-8 data bytes
9-15	Classic CAN: transmit frame has 8 data bytes

CAN FD: transmit frame has 12/16/20/24/32/48/64 data bytes.

T1 [bit15:0] Reserved: Reserved bits

When writing, always write '0'. The read value is undefined.

T2 [bit31:24]	DB3[7:0] :	Data Byte 3
T2 [bit23:16]	DB2[7:0] :	Data Byte 2
T2 [bit15:8]	DB1[7:0] :	Data Byte 1
T2 [bit7:0]	DB0[7:0] :	Data Byte 0
T3 [bit31:24]	DB7[7:0] :	Data Byte 7
T3 [bit23:16]	DB6[7:0] :	Data Byte 6
T3 [bit15:8]	DB5[7:0] :	Data Byte 5
T3 [bit7:0]	DB4[7:0] :	Data Byte 4
…	…	…
Tn [bit31:24]	DBm[7:0]:	Data Byte m
Tn [bit23:16]	DBm-1[7:0]:	Data Byte m-1
Tn [bit15:8]	DBm-2[7:0]:	Data Byte m-2
Tn [bit7:0]	DBm-3[7:0]:	Data Byte m-3

***Note**:*
- *Depending on the configuration of the element size (TXESC), Tn will vary from n = 3 to 17.*
- *m is a function of n: m = (n - 1) × 4 - 1.*

24.4.4 TX event FIFO element

Each TX Event FIFO Element stores information about transmitted messages. By reading the TX Event FIFO, the CPU gets this information in the order the messages were transmitted. Status information about the TX event FIFO can be obtained from Register CANFDx_CHy_TXEFS (TX event FIFO status).

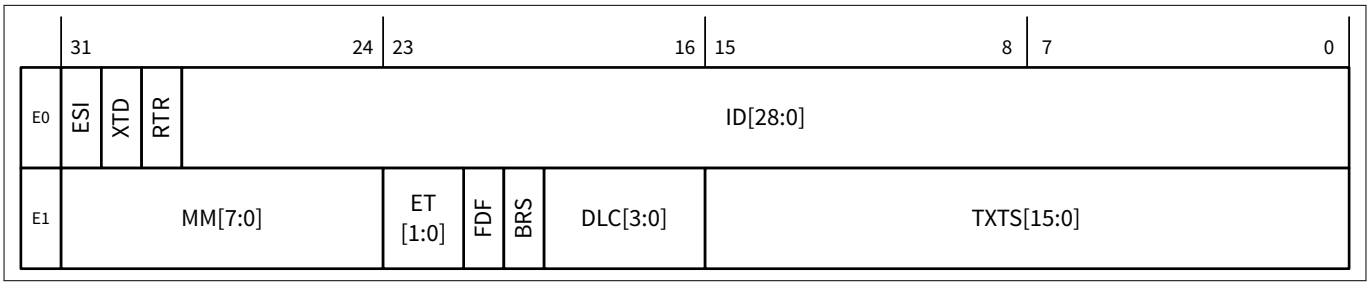


Figure 173 TX event FIFO element

E0 [bit31] ESI: Error state indicator

Bit	Description
0	Transmitting node is error active
1	Transmitting node is error passive

E0 [bit30] XTD: Extended identifier

Bit	Description
0	11-bit standard identifier
1	29-bit extended identifier

E0 [bit29] RTR: Remote transmission request

Bit	Description
0	Data frame transmitted
1	Remote frame transmitted

E0 [bit28:0] ID[28:0]: Identifier

Standard or extended identifier depending on bit XTD. A standard identifier is stored into ID[28:18].

E1 [bit31:24] MM[7:0]: Message marker

Copied from TX buffer into TX Event FIFO element for identification of TX message status.

E1 [bit23:22] ET[1:0]: Event type

ET[1:0]	Description
00	Reserved
01	TX event
10	Transmission in spite of cancellation. Always set for transmissions in DAR mode (disable automatic retransmission mode).
11	Reserved

E1 [bit21] FDF: FD format

Bit	Description
0	Classic CAN frame format
1	CAN FD frame format (new DLC-coding and CRC)

E1 [bit20] BRS: Bit rate switching

Bit	Description
0	Frame transmitted without bit rate switching
1	Frame transmitted with bit rate switching

E1 [bit19:16] DLC[3:0]: Data length code

DLC[3:0]	Description
0-8	Classic CAN + CAN FD: frame with 0-8 data bytes transmitted.
9-15	Classic CAN: frame with 8 data bytes transmitted. CAN FD: frame with 12/16/20/24/32/48/64 data bytes transmitted. See Table 175 for details.

E1 [bit15:0] TXTS[15:0]: TX timestamp

Timestamp counter value captured on start-of-frame transmission. Resolution depending on configuration of the shared timestamp counter pre-scaler CANFDx_TS_CTL.PRESCALE[15:0].

24.4.5 Standard message ID filter element

A standard message ID filter element consists of a single 32-bit word, and can be configured as a range filter, dual filter, classic bit mask filter, or filter for a single dedicated ID, for messages with 11-bit standard IDs.

Up to 128 filter elements can be configured for 11-bit standard IDs. When accessing a standard message ID filter element, its address is filter list standard start address (CANFDx_CHy_SIDFC.FLSSA[15:2]) + index of the filter element (0 to 127).

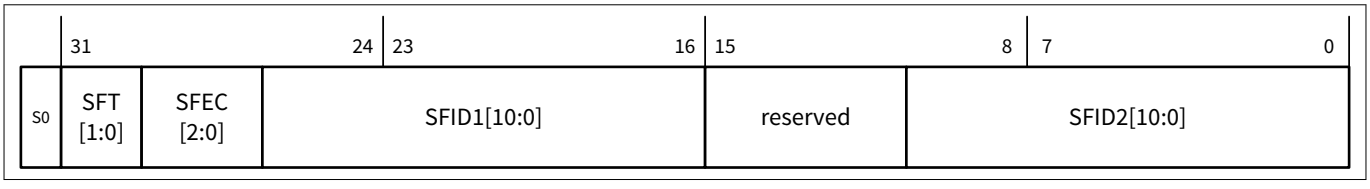


Figure 174 Standard message ID filter

S0 [bit31:30] SFT[1:0]: Standard filter type

SFT[1:0]	Description
00	Range filter from SFID1[10:0] to SFID2[10:0] (SFID2[10:0] ≥ received ID ≥ SFID1[10:0])
01	Dual ID filter for SFID1[10:0] or SFID2[10:0]
10	Classic filter: SFID1[10:0] = filter, SFID2[10:0] = mask. Only those bits of SFID1[10:0] where the corresponding SFID2[10:0] bits are 1 are relevant.

SFT[1:0]	Description
11	Filter element disabled

***Note**: With SFT = 11, the filter element is disabled and the acceptance filtering continues. (same behavior as with SFEC = 000)*

S0 [bit29:27] SFEC[2:0]: Standard filter element configuration

All enabled filter elements are used for acceptance filtering of standard frames. Acceptance filtering stops at the first matching enabled filter element or when the end of the filter list is reached.

If SFEC[2:0] = 100, 101, or 110 a match sets interrupt flag CANFDx_CHy_IR.HPM (high priority message) and, if enabled, an interrupt is generated. In this case Register CANFDx_CHy_HPMS (high priority message status) is updated with the status of the priority match.

SFEC[2:0]	Description
000	Disable filter element
001	Store in RX FIFO 0 if filter matches
010	Store in RX FIFO 1 if filter matches
011	Reject ID if filter matches
100	Set priority if filter matches
101	Set priority and store in RX FIFO 0 if filter matches
110	Set priority and store in RX FIFO 1 if filter matches
111	Store into dedicated RX buffer or as debug message, configuration of SFT[1:0] ignored.

S0 [bit26:16] SFID1[10:0]: Standard filter ID 1

This bit field has a different meaning depending on the configuration of SFEC[2:0]:

- SFEC[2:0] = 001 to 110
  Set SFID1[10:0] according to the SFT[1:0] setting.
- SFEC[2:0] = 111
  SFID1[10:0] defines the ID of a standard dedicated RX buffer or debug message to be stored. The received identifiers must match, no masking mechanism is used.

S0 [bit15:11] Reserved: Reserved bits

When writing, always write ‘0’. The read value is undefined.

S0 [bit10:0] SFID2[10:0]: Standard filter ID 2

This bit field has a different meaning depending on the configuration of SFEC[2:0]:

- SFEC[2:0] = 001 to 110

Set SFID2[10:0] according to the SFT[1:0] setting.

- SFEC[2:0] = 111

Filter for dedicated RX buffers or for debug messages.

SFID2[10:9] decides whether the received message is stored into a dedicated RX buffer or treated as message A, B, or C of the debug message sequence.

SFID2[10:9]	Description
00	Store message into a dedicated RX buffer.
01	Debug message A.
10	Debug message B.

SFID2[10:9]	Description
11	Debug message C.

SFID2[8:6] are reserved bits. When writing, always write '0'. The read value is undefined.
SFID2[5:0] defines the offset to the RX buffer start address CANFDx_CHy_RXBC.RBSA[15:2] to store a matching message.

***Note**: Debug message is used to debug on CAN feature.*

24.4.6 Extended message ID filter element

An extended message ID filter element consists of two 32-bit words, and can be configured as a range filter, dual filter, classic bit mask filter, or filter for a single dedicated ID, for messages with 29-bit extended IDs.

Up to 64 filter elements can be configured for 29-bit extended IDs. When accessing an extended message ID filter element, its address is filter list extended start address (CANFDx_CHy_XIDFC.FLESA[15:2]) + 2 × index of the filter element (0 to 63).

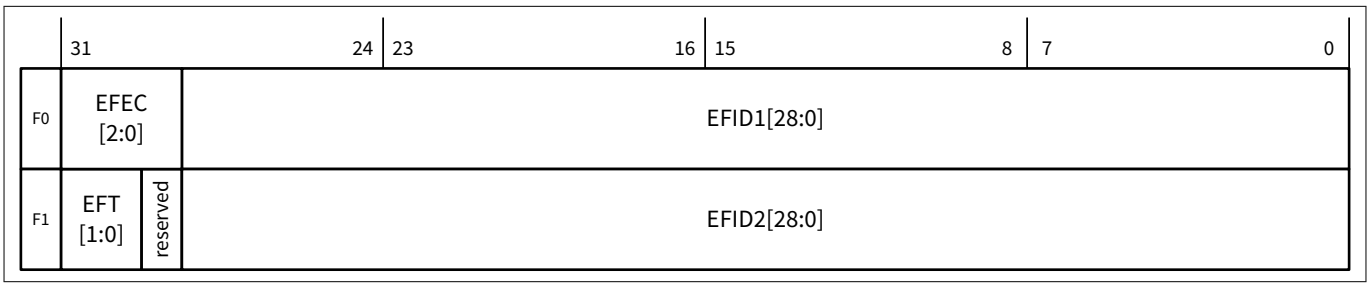


Figure 175 Extended message ID filter

F0 [bit31:29] EFEC[2:0]: Extended filter element configuration

All enabled filter elements are used for acceptance filtering of extended frames. Acceptance filtering stops at the first matching enabled filter element or when the end of the filter list is reached.

If EFEC[2:0] = 100, 101, or 110 a match sets interrupt flag CANFDx_CHy_IR.HPM (high priority message) and, if enabled, an interrupt is generated. In this case Register CANFDx_CHy_HPMS (high priority message status) is updated with the status of the priority match.

EFEC[2:0]	Description
000	Disable filter element.
001	Store in RX FIFO 0 if filter matches.
010	Store in RX FIFO 1 if filter matches.
011	Reject ID if filter matches.
100	Set priority if filter matches.
101	Set priority and store in RX FIFO 0 if filter matches.
110	Set priority and store in RX FIFO 1 if filter matches.
111	Store into dedicated RX buffer or as debug message, configuration of EFT[1:0] ignored.

F0 [bit28:0] EFID1[28:0]: Extended filter ID 1

This bit field has a different meaning depending on the configuration of EFEC[2:0].

- EFEC[2:0] = 001 to

Set EFID1[28:0] according to the EFT[1:0] setting.

- EFEC[2:0] = 11

EFID1[28:0] defines the ID of an extended dedicated RX buffer or debug message to be stored. The received identifiers must match, only XIDAM masking mechanism is used.

F1 [bit31:30] EFT[1:0]: Extended filter type

EFT[1:0]	Description
00	Range filter from EFID1[28:0] to EFID2[28:0] (EFID2[28:0] ≥ received ID ANDed with XIDAM ≥ EFID1[28:0]).
01	Dual ID filter Matches when EFID1[28:0] or EFID2[28:0] is equal to received ID ANDed with XIDAM.
10	Classic filter: EFID1[28:0] = filter, EFID2[28:0] = mask. Only those bits of EFID1[28:0] where the corresponding EFID2[28:0] bits are 1 are relevant. Matches when the received ID ANDed with XIDAM is equal to EFID1[28:0] masked by EFID2[28:0].
11	Range filter from EFID1[28:0] to EFID2[28:0] (EFID2[28:0] ≥ EFID1[28:0]), XIDAM mask not applied.

F1 [bit29] Reserved: Reserved bit

When writing, always write '0'. The read value is undefined.

F1 [bit28:0] EFID2[28:0]: Extended filter ID 2

This bit field has a different meaning depending on the configuration of EFEC[2:0]:

- EFEC[2:0] = 001 to 110

Set EFID2[28:0] according to the EFT[1:0] setting

- EFEC[2:0] = 111

EFID2[28:0] is used to configure this filter for dedicated RX buffers or for debug messages

EFID2[28:11] are reserved bits. When writing, always write '0'. The read value is undefined.

EFID2[10:9] decides whether the received message is stored into a dedicated RX buffer or treated as message A, B, or C of the debug message sequence.

EFID2[10:9]	Description
00	Store message into a dedicated RX buffer
01	Debug message A
10	Debug message B
11	Debug message C

EFID2[8:6] are reserved bits. When writing, always write '0'. The read value is undefined.

EFID2[5:0] defines the offset to the RX buffer start address CANFDx_CHy_RXBC.RBSA[15:2] to store a matching message.

***Note**: Debug message is used to debug on CAN feature.*

24.4.7 Trigger memory element

Up to 64 trigger memory elements can be configured. When accessing a trigger memory element, its address is the trigger memory start address CANFDx_CHy_TTTMC.TMSA plus the index of the trigger memory element (0…63).

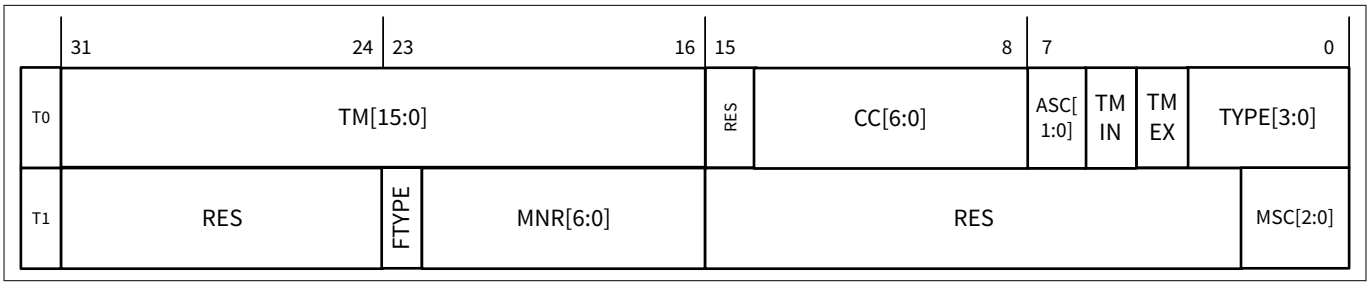


Figure 176 Trigger memory element

T0 Bit 31:16 TM[15:0]: Time mark

Cycle time for which the trigger becomes active.

T0 Bit 14:8 CC[6:0]: Cycle code

Cycle count for which the trigger is valid. Ignored for trigger types Tx_Ref_Trigger, Tx_Ref_Trigger_Gap, Watch_Trigger, Watch_Trigger_Gap, and End_of_List.

CC[6:0]	Description
0b000000x	Valid for all cycles
0b000001c	Valid every second cycle at cycle count mod2 = c
0b00001cc	Valid every fourth cycle at cycle count mod4 = cc
0b0001ccc	Valid every eighth cycle at cycle count mod8 = ccc
0b001cccc	Valid every sixteenth cycle at cycle count mod16 = cccc
0b01ccccc	Valid every thirty-second cycle at cycle count mod32 = ccccc
0b1cccccc	Valid every sixty-fourth cycle at cycle count mod64 = cccccc

T0 Bit 7:6 ASC[1:0]: Asynchronous serial communication

ASC[1:0]	Description
00	No ASC operation
01	Reserved, do not use
10	Node is ASC receiver
11	Node is ASC transmitter

***Note**: ASC functionality is not supported in any XMC7000 device*

T0 Bit 5 TMIN: Time mark event internal

TMIN	Description
0	No action
1	CANFDx_CHy_TTIR.TTMI is set when trigger memory element becomes active

T0 Bit 4 TMEX: Time mark event external

TMEX	Description
0	No action
1	Pulse at output of trigger time mark with the length of one PCLK_CANFD[x]_CLOCK_CAN[y] period is generated when the time mark of the trigger memory element becomes active and CANFDx_CHy_TTOCN.TTMIE = 1

T0 Bit 3:0 TYPE[3:0]: Trigger type

TYPE [3:0]	Description
0000	Tx_Ref_Trigger - valid when not in gap
0001	Tx_Ref_Trigger_Gap - valid when in gap
0010	Tx_Trigger_Single - starts a single transmission in an exclusive time window
0011	Tx_Trigger_Continuous - starts continuous transmission in an exclusive time window
0100	Tx_Trigger_Arbitration - starts a transmission in an arbitrating time window
0101	Tx_Trigger_Merged - starts a merged arbitration window
0110	Watch_Trigger - valid when not in gap
0111	Watch_Trigger_Gap - valid when in gap
1000	Rx_Trigger - check for reception
1001	Time_Base_Trigger - only control TMIN, TMEX, and ASC
1010 … 1111	End_of_List - illegal type, causes configuration error

***Note**:*
- *For ASC operation (ASC = 10, 11) only trigger types Rx_Trigger and Time_Base_Trigger should be used.*
- *ASC operation is not supported in this device.*

T1 Bit 23 FTYPE: Filter type

FTYPE	Description
0	11-bit standard message ID
1	29-bit extended message ID

T1 Bit 22:16 MNR[6:0]: Message number

- Transmission: Trigger is valid for configured TX buffer number. Valid values are 0 to 31.
- Reception: Trigger is valid for standard/extended message ID filter element number. Valid values are 0 to 63 and 0 to 127.

T1 Bits 2:0 MSC[2:0]: Message status count

Counts scheduling errors for periodic messages in exclusive time windows. It has no function for arbitrating messages and in event-driven CAN communication (ISO 11898-1:2015).

***Note**:*
- *The trigger memory elements should be written when the M_TTCAN is in INIT state. Write access to the trigger memory elements outside INIT state is not allowed.*
- *There is an exception for TMIN and TMEX when they are defined as part of a trigger memory element of TYPE Tx_Ref_Trigger. In this case they become active at the time mark modified by the actual Reference Trigger Offset (CANFDx_CHy_TTOST.RTO).*

24.4.8 ECC for message RAM

The error correcting code (ECC) function of the message RAM enables detection and correction of the data errors in message RAM. Code uses a 7-bit parity for a 32-bit data word for the ECC functionality. It has the following features:

- Single error correction and double error detection (SECDED)
  - Single-bit error correction in memory data word
  - Detection of single- and double-bit errors in memory data word
  - Detection of error in message RAM address decoding
- Stopping CAN FD function upon detecting a double-bit (non-correctable) error
- Error injection while data transfer

The following sections describe types of ECC errors.

24.4.8.1 Correctable ECC error

When a correctable ECC error is detected, the following will happen:

- The corrected data is returned to the read access master
  - No error signal is returned to the master
- The corrected data is written back to the message RAM, unless that write is canceled by another write to the same address.
- The ECC error is reported in the fault structures as a correctable error with the following information:
  - DATA0[15:0]: Violating address
  - DATA0[22:16]: ECC syndrome[6:0]
  - DATA0[27:24]: Master ID: 0-7 = CAN channel ID, 8 = AHB I/F

    Note that for security reasons the violating data is not reported in the fault structure.

In the unlikely event of two correctable ECC errors too close together (before fault reporting and correction write back are both complete) the second ECC error is neither corrected nor reported in the fault structure. If later the same address is read again the correction and fault reporting will be done at that time.

More details are available in Fault subsystem.

24.4.8.2 Non-correctable ECC error

When data is read from message RAM and upon ECC check double-bit error is detected, the following actions are taken:

- An error is reported to master
  - In case of AHB master, bus error will occur
  - In case of M_TTCAN channel, channel will shut down immediately (CANFDx_CHy_CCCR.INIT = 1)
- Interrupt (BEU) is raised
- ECC error is reported to the fault structure as non-correctable error with the following information:
  - DATA0[15:0]: Violating address

- DATA0[22:16]: ECC syndrome[6:0]
- DATA0[27:24]: Master ID: 0-7 = CAN channel ID, 8 = AHB I/F
  For security reasons, data is not reported to the fault structure

Unlike single-bit (correctable) error, double-bit error is reported at both the master and fault structures, because read access master needs to know that the read data is not correct and fault structure needs to know all ECC errors.

ECC errors on the address bits are always non-correctable.

24.4.8.3 Address error

An address error is detected when either M_TTCAN channel or MCU is trying to access an out of range message RAM address (address >= MRAM_SIZE). This feature is added to make software debugging easier. Address error is independent of ECC; that is, it works even if ECC is disabled. When such an address error is detected the following will happen:

- For writes, the error is not reported back to the master
  - Writes are posted and both the AHB interface and the CAN channels ignore error signaling
- For reads from a M_TTCAN channel, the error is reported back to the channel as if it is a non-correctable ECC error; this will result in the following:
  - To prevent corrupt data from being sent, channel will be shutdown (CANFDx_CHy_CCCR.INIT=1) immediately
  - An interrupt is raised (BEU)
- For reads from the AHB interface the address error results in a bus error
- For any case, read, write, and any master, the address error will be reported in the fault structure as a non-correctable error with the following information:
  - DATA0[15:0]: Violating address.
  - DATA0[27:24]: Master ID: 0-7 = CAN channel ID, 8 = AHB I/F
  - DATA0[30]: Set for a write access and cleared for a read access
  - DATA0[31]: Set to flag an address error

24.4.8.4 ECC error injection

For safety of the functionality, the ECC error injection feature is added to enable the software to write the ECC bits. With this feature the software can trigger correctable or non-correctable ECC errors to verify that all related hardware and software are functioning properly.

Using this feature, software is able to inject ECC error in the background while data is being fetched from the message RAM.

This feature consists of:

- An enable bit (CANFDx_ECC_ERR_INJ.ERR_EN) to enable the ECC error injection logic
- A 7-bit error parity (CANFDx_ECC_ERR_INJ.ERR_PAR) to be written instead of the generated ECC bits
- An address (CANFDx_ECC_ERR_INJ.ERR_ADDR) to specify at which message RAM address the ECC error injection is done

When a write is done to the specified error injection address, the specified error parity will be used instead of the ECC parity generated by the ECC logic. By limiting this to just one address the software can run this functional test without affecting any other message RAM accesses.

If a write back is done when a correctable error occurs in the specified error injection address, the ECC parity generated by the ECC logic will be used.

As described in the preceding sections, detection of a non-correctable error will result in either shutting down a CAN channel or shutting down a CPU (bus error). Therefore, reporting a non-correctable error back to the master will be suppressed for the targeted error injection address. This feature is necessary to allow non-correctable errors to be verified without affecting the running application.

Note that this error suppression applies to both ECC non-correctable errors and address errors. The software can easily disable this error suppression by disabling the error injection logic (CANFD_ECC_ERR_INJ.ERR_EN = 0).

24.4.8.5 ECC parity generation by software

To inject the ECC error for fault generation, ECC parity must be generated by software. Follow this procedure to generate a 7-bit ECC parity.

```
CODEWORD_SW[63:0] = 64{1'b0};
CODEWORD_SW[31:0] = ACTUALWORD[31:0];
CODEWORD_SW[((x-1)+32):32] = ADDR[(x-1):0];  //where x = MRAM_ADDR_WIDTH

ECC_P0_SW = 64b00000011_01111111_00110110_11011011_00100010_01010100_00101010_10101011;
ECC_P1_SW = 64b00000101_10111101_11101011_01011010_01000100_10011001_01001101_00110101;
ECC_P2_SW = 64b00001001_11011101_11011100_11101110_00001000_11100010_01110001_11000110;
ECC_P3_SW = 64b00010001_11101110_10111011_10101001_10001111_00000011_10000001_11111000;
ECC_P4_SW = 64b00100001_11110110_11010111_01110101_11110000_00000011_11111110_00000000;
ECC_P5_SW = 64b01000001_11111011_01101101_10110100_11111111_11111100_00000000_00000000;
ECC_P6_SW = 64b10000001_00000011_11111111_11111000_00010001_00101100_10010110_01011111;
```

As shown here, reduction XOR of the ANDed result of CODEWORD_SW[63:0] and the respective ECC constants will give a single parity bit.

```
parity[0] = ^ (CODEWORD_SW[63:0] & ECC_P0_SW)
parity[1] = ^ (CODEWORD_SW[63:0] & ECC_P1_SW)
…
parity[6] = ^ (CODEWORD_SW[63:0] & ECC_P6_SW)
```

parity[6:0] gives seven-bit parity for 32 bits ACTUALWORD[31:0].

MRAM_ADDR_WIDTH defined in the above procedure depends on the total message RAM allocated for one M_TTCAN group. The following table specifies the MRAM_ADDR_WIDTH for each message RAM size.

Parameter	Combinations											
Message RAM size [KB]	4	8	10	16	20	24	32	36	40	48	56	64
MRAM_ADDR_WIDTH	10	11	12	12	13	13	13	14	14	14	14	14

24.4.9 Message RAM OFF

Message RAM can be turned off to save power by setting CANFDx_CTL.MRAM_OFF bit. Default value of this bit is '0' and message RAM is retained in this configuration during deep sleep power mode.

All the M_TTCAN channels must be powered down before setting CANFDx_CTL.MRAM_OFF bit. See Power down (sleep mode) to power down the M_TTCAN channels. When message RAM is OFF, any access to message RAM may raise address error (MRAM_SIZE = 0).

After switching the message RAM on again, software needs to allow a certain power-up time before message RAM can be used, that is, before STOP_REQ can be de-asserted. Check the RAM_PWR_DELAY_CTL Register to see the required time for message RAM power up process.

24.4.10 RAM watchdog (RWD)

The RAM watchdog monitors the READY output of the message RAM. A message RAM access starts the message RAM watchdog counter with the value configured by CANFDx_CHy_RWD.WDC. The counter is reloaded with CANFDx_CHy_RWD.WDC when the message RAM signals successful completion by activating its READY output. In case there is no response from the message RAM until the counter has counted down to zero, the counter stops and interrupt flag CANFDx_CHy_IR.WDI is set. The RAM Watchdog Counter is clocked by the Host clock (CLK_SYS). Refer to the Registers TRM for more information about the CANFDx_CHy_RWD Register.

24.5 TTCAN operation

24.5.1 Reference message

A reference message is a data frame characterized by a specific CAN identifier. It is received and accepted by all nodes except the time master (sender of the reference message).

For level 1, the data length must be at least one. For level 0 and level 2, the data length must be at least four; otherwise, the message is not accepted as a reference message. The reference message may be extended by other data up to the sum of eight CAN data bytes. All bits of the identifier except the three LSBs characterize the message as a reference message. The last three bits specify the priorities of up to eight potential time masters. Reserved bits are transmitted as logical 0 and are ignored by the receivers. The reference message is configured using the CANFDx_CHy_TTRMC Register.

The time master transmits the reference message. If the reference message is disturbed by an error, it is retransmitted immediately. In a retransmission, the transmitted Master_Ref_Mark is updated. The reference message is sent periodically, but it is allowed to stop the periodic transmission (Next_is_Gap bit). It can initiate event-synchronized transmission at the start of the next basic cycle by the current time master or by one of the other potential time masters.

The node transmitting the reference message is the current time master. The time master is allowed to transmit other messages. If the current time master fails, its function is replicated by the potential time master with the highest priority. Nodes that are neither time master nor potential time master are time-receiving nodes.

24.5.1.1 Level 1

Level 1 operation is configured via CANFDx_CHy_TTOCF.OM = 01 and CANFDx_CHy_TTOCF.GEN. External clock synchronization is not available in Level 1.

The information related to the reference message is stored in the first data byte as shown in Table 182. Cycle_Count is optional.

Table 182 First byte of level 1 reference message

Bits	7	6	5	4	3	2	1	0
First byte	Next_is_Gap	Reserved	Cycle_Count [5:0]					

24.5.1.2 Level 2

Level 2 operation is configured via CANFDx_CHy_TTOCF.OM = 10 and CANFDx_CHy_TTOCF.GEN.

The information related to the reference message is stored in the first four data bytes as shown in Table 183. Cycle_Count and the lower four bits of NTU_Res are optional. The M_TTCAN does not evaluate NTU_Res[3:0] from received reference messages, it always transmits these bits as zero.

Table 183 First four bytes of level 2 reference message

Bits	7	6	5	4	3	2	1	0
First byte	Next_is_Gap	Reserved	Cycle_Count[5:0]					
Second byte	NTU_Res[6:4]			NTU_Res[3:0]				Disc_Bit
Third byte	Master_Ref_Mark[7:0]							
Fourth byte	Master_Ref_Mark[15:8]							

24.5.1.3 Level 0

Level 0 operation is configured via CANFDx_CHy_TTOCF.OM = 11. External event-synchronized time-triggered operation is not available in Level 0.

The information related to the reference message is stored in the first four data bytes as shown in Table 184. In level 0, Next_is_Gap is always zero. Cycle_Count and the lower four bits of NTU_Res are optional. The M_TTCAN does not evaluate NTU_Res[3:0] from received reference messages; it always transmits these bits as zero.

Table 184 First four bytes of level 0 reference message

Bits	7	6	5	4	3	2	1	0
First byte	Next_is_Gap	Reserved	Cycle_Count[5:0]					
Second byte	NTU_Res[6:4]			NTU_Res[3:0]				Disc_Bit
Third byte	Master_Ref_Mark[7:0]							
Fourth byte	Master_Ref_Mark[15:8]							

24.5.2 TTCAN configuration

24.5.2.1 TTCAN timing

The network time unit (NTU) is the unit in which all times are measured. The NTU is a constant of the whole network and is defined by the network system designer. In TTCAN Level 1 the NTU is the nominal CAN bit time. In TTCAN Level 0 and Level 2 the NTU is a fraction of the physical second.

The NTU is the time base for the local time. The integer part of the local time (16-bit value) is incremented once for each NTU. Cycle time and global time are both derived from local time. The fractional part (3-bit value) of local time, cycle time, and global time is not readable.

In TTCAN Level 0 and Level 2, the length of the NTU is defined by the time unit ratio (TUR). The TUR is a non-integer number given by the formula TUR = CANFDx_CHy_TURNA.NAV/CANFDx_CHy_TURCF.DC. The length of the NTU is given by the formula NTU = CAN clock period × TUR.

The TUR numerator configuration NC is an 18-bit number, CANFDx_CHy_TURCF.NCL[15:0] can be programmed in the range 0x0000-0xFFFF. CANFDx_CHy_TURCF.NCH[17:16] is hard-wired to 0b01. When the number 0xnnnn is written to CANFDx_CHy_TURCF.NCL[15:0], CANFDx_CHy_TURNA.NAV starts with the value 0x10000 + 0x0nnnn = 0x1nnnn. The TUR denominator configuration CANFDx_CHy_TURCF.DC is a 14-bit number. CANFDx_CHy_TURCF.DC may be programmed in the range 0x0001 - 0x3FFF; 0x0000 is an illegal value.

In Level 1, NC must be ≥ 4 × CANFDx_CHy_TURCF.DC. In Level 0 and Level 2 NC must be ≥ 8 × CANFDx_CHy_TURCF.DC to allow the 3-bit resolution for the internal fractional part of the NTU.

A hardware reset presets CANFDx_CHy_TURCF.DC to 0x1000 and CANFDx_CHy_TURCF.NCL to 0x10000, resulting in an NTU consisting of 16 CAN clock periods. Local time and application watchdog are not started before either the CANFDx_CHy_CCCR.INIT is reset or CANFDx_CHy_TURCF.ELT is set. CANFDx_CHy_TURCF.ELT may not be set before the NTU is configured. Setting CANFDx_CHy_TURCF.ELT to '1' also locks the write access to Register CANFDx_CHy_TURCF.

At startup CANFDx_CHy_TURNA.NAV is updated from NC (= CANFDx_CHy_TURCF.NCL + 0x10000) when CANFDx_CHy_TURCF.ELT is set. In TTCAN Level 1 there is no drift compensation. CANFDx_CHy_TURNA.NAV does not change during operation, it always equals NC.

In TTCAN level 0 and level 2, there are two possibilities for CANFDx_CHy_TURNA.NAV to change. When operating as time slave or backup time master, and when CANFDx_CHy_TTOCF.ECC is set, CANFDx_CHy_TURNA.NAV is updated automatically to the value calculated from the monitored global time speed, as long as the M_TTCAN is in synchronization states In_Schedule or In_Gap. When it loses synchronization, it returns to NC. When operating as the actual time master, and when CANFDx_CHy_TTOCF.EECS is set, the host may update CANFDx_CHy_TURCF.NCL. When the host sets CANFDx_CHy_TTOCN.ECS, CANFDx_CHy_TURNA.NAV will be updated from the new value of NC at the next reference message. The status flag CANFDx_CHy_TTOST.WECS is set when CANFDx_CHy_TTOCN.ECS is set and is cleared when CANFDx_CHy_TURNA.NAV is updated. CANFDx_CHy_TURCF.NCL is write-locked while CANFDx_CHy_TTOST.WECS is set.

In TTCAN Level 0 and Level 2, the clock calibration process adapts CANFDx_CHy_TURNA.NAV in the range of the synchronization deviation limit (SDL) of NC ± 2 (CANFDx_CHy_TTOCF.LDSDL + 5). CANFDx_CHy_TURCF.NCL should be programmed to the largest applicable numerical value to achieve the best accuracy in the calculation of CANFDx_CHy_TURNA.NAV.

The synchronization deviation (SD) is the difference between NC and CANFDx_CHy_TURNA.NAV (SD = |NC - CANFDx_CHy_TURNA.NAV|). It is limited by the SDL, which is configured by its dual logarithm CANFDx_CHy_TTOCF.LDSDL (SDL = 2 (CANFDx_CHy_TTOCF.LDSDL + 5)) and should not exceed the clock tolerance given by the CAN bit timing configuration. SD is calculated at each new basic cycle. When the calculated CANFDx_CHy_TURNA.NAV deviates by more than SDL from NC, or if the Disc_Bit in the reference message is set, the drift compensation is suspended, CANFDx_CHy_TTIR.GTE is set, and CANFDx_CHy_TTOSC.QCS is reset; if Disc_Bit = '1', CANFDx_CHy_TTIR.GTD is set.

TUR configuration examples are shown in Table 185.

Table 185 TUR configuration examples

TUR	8	10	24	50	510	125000	32.5	100/12	529/17
NC	0x1FFF8	0x1FFFE	0x1FFF8	0x1FFEA	0x1FFFE	0x1FFE0	0x1FFE0	0x19000	0x10880
CANFDx_CHy_TURCF.DC	0x3FFF	0x3333	0x1555	0x0A3D	0x0101	0x0001	0x0FC0	0x3000	0x0880

CANFDx_CHy_TTOCN.ECS schedules NC for activation by the next reference message. CANFDx_CHy_TTOCN.SGT schedules CANFDx_CHy_TTGTP.TP for activation by the next reference message. Setting of CANFDx_CHy_TTOCN.ECS and CANFDx_CHy_TTOCN.SGT requires CANFDx_CHy_TTOCF.EECS to be set (external clock synchronization enabled) while the M_TTCAN is actual time master.

The M_TTCAN module provides an application watchdog to verity the function of the application program. The host has to serve this watchdog regularly; otherwise, all CAN bus activity is stopped. The application watchdog limit CANFDx_CHy_TTOCF.AWL specifies the number of NTUs the watchdog has to be served. The maximum number of NTUs is 256. The application watchdog is served by reading Register CANFDx_CHy_TTOST. CANFDx_CHy_TTOST.AWE indicates whether the watchdog is served in time. In case the application failed to serve the application watchdog, interrupt flag CANFDx_CHy_TTIR.AW is set. For software development, the

application watchdog may be disabled by programming CANFDx_CHy_TTOCF.AWL to 0x00 (see Application watchdog).

24.5.2.2 Message scheduling

CANFDx_CHy_TTOCF.TM controls whether the M_TTCAN operates as a potential time master or as a time slave. If it is a potential time master, the three LSBs of the reference message identifier CANFDx_CHy_TTRMC.RID define the master priority, 0 being the highest and 7 the lowest priority. Two nodes in the network may not use the same master priority. CANFDx_CHy_TTRMC.RID is used for recognition of reference messages. CANFDx_CHy_TTRMC.RMPS is not relevant for time slaves.

The initial reference trigger offset CANFDx_CHy_TTOCF.IRTO is a 7-bit-value that defines (in NTUs) how long a backup time master waits before it starts the transmission of a reference message, when a reference message is expected but the bus remains idle. The recommended value for CANFDx_CHy_TTOCF.IRTO is the master priority multiplied with a factor depending on the expected clock drift between the potential time masters in the network. The sequential order of the backup time masters, when one of them starts the reference message if the current time master fails, should correspond to their master priority, even with maximum clock drift.

CANFDx_CHy_TTOCF.OM decides whether the node operates in TTCAN level 0, level 1, or level 2. In one network, all potential time masters should operate on the same level. Time slaves may operate on level 1 in a level 2 network, but not vice versa. The configuration of the TTCAN operation mode via CANFDx_CHy_TTOCF.OM is the last step in the setup. When CANFDx_CHy_TTOCF.OM = 00 (event-driven CAN communication), the M_TTCAN operates according to ISO 11898-1:2015, without time triggers. When CANFDx_CHy_TTOCF.OM = 01 (level 1), the M_TTCAN operates according to ISO 11898-4, but without the possibility to synchronize the basic cycles to external events, the Next_is_Gap bit in the reference message is ignored. When CANFDx_CHy_TTOCF.OM = 10 (level 2), the M_TTCAN operates according to ISO 11898-4, including the event-synchronized start of a basic cycle. When CANFDx_CHy_TTOCF.OM = 11 (level 0), the M_TTCAN operates as event-driven CAN but maintains a calibrated global time base similar to level 2.

CANFDx_CHy_TTOCF.EECS enables the external clock synchronization, allowing the application program of the current time master to update the TUR configuration during time-triggered operation, to adapt the clock speed and (in level 0, 2 only) the global clock phase to an external reference.

CANFDx_CHy_TTMLM.ENTT in the TT matrix limits register specifies the number of expected Tx_Triggers in the system matrix. This is the sum of Tx_Triggers for exclusive single arbitrating and merged arbitrating windows, excluding the Tx_Ref_Triggers. Note that this is usually not the number of Tx_Trigger memory elements; the number of basic cycles in the system matrix and the trigger's repeat factors must be taken into account. An inaccurate configuration of CANFDx_CHy_TTMLM.ENTT will result in either a TX count underflow (CANFDx_CHy_TTIR.TXU = 1 and CANFDx_CHy_TTOST.EL = 01, severity 1) or in a TX count overflow (CANFDx_CHy_TTIR.TXO = 1 and CANFDx_CHy_TTOST.EL = 10, severity 2).

***Note**: In case the first reference message seen by a node does not have Cycle_Count zero, this node may finish its first matrix cycle with its TX count resulting in a TX count underflow condition. As long as a node is in state, synchronizing its Tx_Triggers will not lead to transmissions.*

CANFDx_CHy_TTMLM.CCM specifies the number of the last basic cycle in the system matrix. The counting of basic cycles starts at 0. In a system matrix consisting of eight basic cycles CANFDx_CHy_TTMLM.CCM would be 7. CANFDx_CHy_TTMLM.CCM is ignored by time slaves, a receiver of a reference message considers the received cycle count as the valid cycle count for the actual basic cycle.

CANFDx_CHy_TTMLM.TXEW specifies the length of the TX enable window in NTUs. The TX enable window is the period at the beginning of a time window where a transmission may be started. If the sample point of the first bit of a transmit message is not inside the TX enable window, the transmission cannot be started in that time window at all. An example is because of an overlap from the previous time window's message. CANFDx_CHy_TTMLM.TXEW should be chosen based on the network's synchronization quality and the relation between the length of the time windows and the length of messages.

24.5.2.3 Trigger memory

The trigger memory is part of the message RAM. It stores up to 64 trigger elements. A trigger memory element consists of time mark TM, cycle code CC, trigger type TYPE, filter type FTYPE, message number MNR, message status count MSC, time mark event internal TMIN, time mark event external TMEX, and asynchronous serial communication ASC (see Trigger memory element).

The time mark defines at which cycle time a trigger becomes active. The trigger elements in the trigger memory must be sorted by their time marks. The trigger element with the lowest time mark is written to the first trigger memory word. Message number and cycle code are ignored for triggers of type Tx_Ref_Trigger, Tx_Ref_Trigger_Gap, Watch_Trigger, Watch_Trigger_Gap, and End_of_List.

When the cycle time reaches the time mark of the actual trigger, the FSE switches to the next trigger and starts to read it from the trigger memory. For a transmit trigger, the TX handler starts to read the message from the message RAM as soon as the FSE switches to its trigger. The RAM access speed defines the minimum time step between a transmit trigger and its preceding trigger, the TX handler should be able to prepare the transmission before the transmit trigger's time mark is reached. The RAM access speed also limits the number of non-matching (with regard to their cycle code) triggers between two matching triggers, the next matching trigger must be read before its time mark is reached. If the reference message is n NTU long, a trigger with a time mark less than n will never become active and will be treated as a configuration error.

The starting point of cycle time is the sample point of the reference message's start-of-frame bit. The next reference message is requested when cycle time reaches the Tx_Ref_Trigger's time mark. The M_TTCAN reacts to the transmission request at the next sample point. A new Sync_Mark is captured at the start-of-frame bit, but the cycle time is incremented until the reference message is successfully transmitted (or received) and the Sync_Mark is taken as the new Ref_Mark. At that point, cycle time is restarted. As a consequence, cycle time can never (with the exception of initialization) be seen at a value less than n, with n being the length of the reference message measured in NTU.

Length of a basic cycle: Tx_Ref_Trigger time mark + 1 NTU + 1 CAN bit time

The trigger list will be different for all nodes in the TTCAN network. Each node knows only the Tx_Triggers for its own transmit messages, the Rx_Triggers for the receive messages that are processed by this node, and the triggers concerning the reference messages.

Trigger types

Tx_Ref_Trigger (TYPE = 0000) and Tx_Ref_Trigger_Gap (TYPE = 0001) cause the transmission of a reference message by a time master. A configuration error (CANFDx_CHy_TTOST.EL = 11, severity 3) is detected when a time slave encounters a Tx_Ref_Trigger(_Gap) in its trigger memory. Tx_Ref_Trigger_Gap is only used in external event-synchronized time-triggered operation mode. In that mode, Tx_Ref_Trigger is ignored when the M_TTCAN synchronization state is In_Gap (CANFDx_CHy_TTOST.SYS = 10).

Tx_Trigger_Single (TYPE = 0010), Tx_Continous (TYPE = 0011), Tx_Trigger_Arbitration (TYPE = 0100), and Tx_Trigger_Merged (TYPE = 0101) cause the start of a transmission. They define the start of a time window.

Tx_Trigger_Single starts a single transmission in an exclusive time window when the message buffer's transmission request pending bit is set. After successful transmission, the transmission request pending bit is reset.

Tx_Trigger_Continuous starts a transmission in an exclusive time window when the message buffer's transmission request pending bit is set. After successful transmission, the transmission request pending bit remains set, and the message buffer is transmitted again in the next matching time window.

Tx_Trigger_Arbitration starts an arbitrating time window, Tx_Trigger_Merged a merged arbitrating time window. The last Tx_Trigger of a merged arbitrating time window must be of type Tx_Trigger_Arbitration. A configuration error (CANFDx_CHy_TTOST.EL = 11, severity 3) is detected when a trigger of type Tx_Trigger_Merged is followed by any other Tx_Trigger than one of type Tx_Trigger_Merged or Tx_Trigger_Arbitration. Several Tx_Triggers may be defined for the same TX message buffer. Depending on their cycle code, they may be ignored in some basic cycles. The cycle code should be considered when the expected number of Tx_Triggers (CANFDx_CHy_TTMLM.ENTT) is calculated.

Watch_Trigger (TYPE = 0110) and Watch_Trigger_Gap (TYPE = 0111) check for missing reference messages. They are used by both time masters and time slaves. Watch_Trigger_Gap is only used in external event-synchronized time-triggered operation mode. In that mode, a Watch_Trigger is ignored when the M_TTCAN synchronization state is In_Gap (CANFDx_CHy_TTOST.SYS = 10).

Rx_Trigger (TYPE = 1000) is used to check for the reception of periodic messages in exclusive time windows. Rx_Triggers are not active until state In_Schedule or In_Gap is reached. The time mark of an Rx_Trigger should be placed after the end of that message transmission, independent of time window boundaries. Depending on their cycle code, Rx_Triggers may be ignored in some basic cycles. At the Rx_Trigger time mark, it is checked whether the last received message before this time mark and after start of cycle or previous Rx_Trigger matches the acceptance filter element referenced by MNR. Accepted messages are stored in one of two receive FIFOs, according to the acceptance filtering, independent of the Rx_Trigger. Acceptance filter elements that are referenced by Rx_Triggers should be placed at the beginning of the filter list to ensure that the filtering is finished before the Rx_Trigger time mark is reached.

Time_Base_Trigger (TYPE = 1001) is used to generate internal/external events depending on the configuration of ASC, TMIN, and TMEX.

End_of_List (TYPE = 1010…1111) is an illegal trigger type, a configuration error (CANFDx_CHy_TTOST.EL = 11, severity 3) is detected when an End_of_List trigger is encountered in the trigger memory before the Watch_Trigger or Watch_Trigger_Gap.

Restrictions for the node's trigger list

Two triggers may not be active at the same cycle time and cycle count, but triggers that are active in different basic cycles (different cycle code) may share the same time mark.

Rx_Triggers and Time_Base_Triggers may not be placed inside the TX enable windows of Tx_Trigger_Single/continuous/arbitration, but they may be placed after Tx_Trigger_Merged.

Triggers that are placed after the Watch_Trigger (or the Watch_Trigger_Gap when CANFDx_CHy_TTOST.SYS = 10) will never become active. The watch triggers themselves will not become active when the reference messages are transmitted on time.

All unused trigger memory words (after the Watch_Trigger or after the Watch_Trigger_Gap when CANFDx_CHy_TTOST.SYS = 10) must be set to trigger type End_of_List.

A typical trigger list for a potential time master will begin with a number of Tx_Triggers and Rx_Triggers followed by the Tx_Ref_Trigger and Watch_Trigger. For networks with external event-synchronized time-triggered communication, this is followed by the Tx_Ref_Trigger_Gap and the Watch_Trigger_Gap. The trigger list for a time slave will be the same but without the Tx_Ref_Trigger and the Tx_Ref_Trigger_Gap.

At the beginning of each basic cycle, that is at each reception or transmission of a reference message, the trigger list is processed starting with the first trigger memory element. The FSE looks for the first trigger with a cycle code that matches the current cycle count. The FSE waits until cycle time reaches the trigger's time mark and activates the trigger. Later, the FSE looks for the next trigger in the list with a cycle code that matches the current cycle count.

Special consideration is needed for the time around Tx_Ref_Trigger and Tx_Ref_Trigger_Gap. In a time master competing for master ship, the effective time mark of a Tx_Ref_Trigger may be decremented to be the first node to start a reference message. In backup time masters the effective time mark of a Tx_Ref_Trigger or Tx_Ref_Trigger_Gap is the sum of its configured time mark and the reference trigger offset CANFDx_CHy_TTOCF.IRTO. If error level 2 is reached (CANFDx_CHy_TTOST.EL = 10), the effective time mark is the sum of its time mark and 0x127. No other trigger elements should be placed in this range; otherwise, the time marks may appear out of order and are flagged as a configuration error. Trigger elements that are coming after Tx_Ref_Trigger may never become active as long as the reference messages come in time.

There are interdependencies between the following parameters:

- Host clock frequency
- Speed and waiting time for trigger RAM accesses
- Length of the acceptance filter list
- Number of trigger elements
- Complexity of cycle code filtering in the trigger elements
- Offset between time marks of the trigger elements

Examples of trigger handling

The following example shows how the trigger list is derived from a node's system matrix. Assume that node A is a first time master; a section of the system matrix shown in Table 186.

Table 186 System matrix node A

Cycle count	Time mark1	Time mark2	Time mark3	Time mark4	Time mark5	Time mark6	Time mark7
0	Tx7	–	–	–	–	TxRef	Error
1	Rx3	–	Tx2, Tx4		–	TxRef	Error
2	–	–	–	–	–	TxRef	Error
3	Tx7	–	Rx5	–	–	TxRef	Error
4	Tx7	–	–	Rx6	–	TxRef	Error

The cycle count starts with 0 – 0, 1, 3, 7, 15, 31, 63 (the number of basic cycles in the system matrix is 1, 2, 4, 8, 16, 32, 64). The maximum cycle count is configured by CANFDx_CHy_TTMLM.CCM. The cycle code (CC) is composed of repeat factor (value of most significant '1') and the number of the first basic cycle in the system matrix (bit field after most significant '1').

Example: When CC is 0b0010011 (repeat factor: 16, first basic cycle: 3) and maximum cycle count of CANFDx_CHy_TTMLM.CCM = 0x3F, matches occur at cycle counts 3, 19, 35, 51.

A trigger element consists of time mark (TM), cycle code (CC), trigger type (TYPE), and message number (MNR). For transmission, MNR references the TX buffer number (0..31). For reception, MNR references the number of the filter element (0..127) that matched during acceptance filtering. Depending on the configuration of the Filter Type FTYPE, the 11-bit or 29-bit message ID filter list is referenced.

In addition, a trigger element can be configured for asynchronous serial communication (ASC), generation of time mark event internal (TMIN), and time mark event external (TMEX). The message status count (MSC) holds the counter value (0..7) for scheduling errors for periodic messages in exclusive time windows when the time mark of the trigger element becomes active.

Table 187 Trigger list node A

Trigger	Time mark TM[15:0]	Cycle code CC[6:0]	Trigger type TYPE[3:0]	Mess. No. MNR[6:0]
0	Mark1	0b0000100	Tx_Trigger_Single	7
1	Mark1	0b1000000	Rx_Trigger	3
2	Mark1	0b1000011	Tx_Trigger_Single	7
3	Mark3	0b1000001	Tx_Trigger_Merged	2
4	Mark3	0b1000011	Rx_Trigger	5
5	Mark4	0b1000001	Tx_Trigger_Arbitration	4
6	Mark4	0b1000100	Rx_Trigger	6
7	Mark6	n.a.	Tx_Ref_Trigger	0 (Ref)
8	Mark7	n.a.	Watch_Trigger	n.a.
9	n.a.	n.a.	End_of_List	n.a.

Tx_Trigger_Single, Tx_Trigger_Continous, Tx_Trigger_Merged, Tx_Trigger_Arbitration, Rx_Trigger, and Time_Base_Trigger are only valid for the specified cycle code. For all other trigger types the cycle code is ignored.

The FSE starts the basic cycle by scanning the trigger list starting from zero until a trigger with time mark that is greater than the cycle time is reached, CC matches the actual cycle count, or a trigger of type Tx_Ref_Trigger, Tx_Ref_Trigger_Gap, Watch_Trigger, or Watch_Trigger_Gap is encountered.

When the cycle time reaches TM, the action defined by TYPE and MNR is started. There is an error in the configuration when it reaches End_of_List.

At Mark6, the reference message (always TxRef) is transmitted. After transmission, the FSE returns to the beginning of the trigger list. When it reaches watch trigger at Mark7, the node is unable to transmit the reference message; error treatment is then started.

Detection of configuration errors

A configuration error is signaled via CANFDx_CHy_TTOST.EL = 11 (severity 3) when:

- The FSE comes to a trigger in the list with a cycle code that matches the current cycle count but with a time mark that is less than the cycle time.
- The previous active trigger was a Tx_Trigger_Merged and the FSE comes to a trigger in the list with a cycle code that matches the current cycle count but that is neither a Tx_Trigger_Merged nor a Tx_Trigger_Arbitration nor a Time_Base_Trigger nor an Rx_Trigger.
- The FSE of a node with CANFDx_CHy_TTOCF.TM = 0 (time slave) encounters a Tx_Ref_Trigger or a Tx_Ref_Trigger_Gap.
- Any time mark placed inside the TX enable window (defined by CANFDx_CHy_TTMLM.TXEW) of a Tx_Trigger with a matching cycle code.
- A time mark is placed near the time mark of a Tx_Ref_Trigger and the Reference Trigger Offset CANFDx_CHy_TTOST.RTO causes a reversal of their sequential order measured in cycle time.

24.5.2.4 TTCAN schedule initialization

The synchronization to the M_TTCAN message schedule starts when CANFDx_CHy_CCCR.INIT is reset. The M_TTCAN can operate time-triggered (CANFDx_CHy_TTOCF.GEN = 0) or external event-synchronized time-triggered (CANFDx_CHy_TTOCF.GEN = 1). All nodes start with cycle time zero at the beginning of their trigger list with CANFDx_CHy_TTOST.SYS = 00 (out of synchronization); no transmission is enabled with the exception of the reference message. Nodes in external event-synchronized time-triggered operation mode will ignore Tx_Ref_Trigger and Watch_Trigger and use Tx_Ref_Trigger_Gap and Watch_Trigger_Gap instead until the first reference message decides whether a gap is active.

Time slaves

After configuration, a time slave will ignore its Watch_Trigger and Watch_Trigger_Gap when it does not receive any message before reaching the Watch_Triggers. When it reaches Init_Watch_Trigger, interrupt flag CANFDx_CHy_TTIR.IWT is set, the FSE is frozen, and the cycle time will become invalid. However, the node will still be able to take part in CAN bus communication (to give acknowledge or to send error flags). The first received reference message will restart the FSE and the cycle time.

***Note**: Init_Watch_Trigger is not part of the trigger list. It is implemented as an internal counter that counts up to 0xFFFF = maximum cycle time.*

When a time slave receives any message but the reference message before reaching the Watch_Triggers, it will assume a fatal error (CANFDx_CHy_TTOST.EL = 11, severity 3), set interrupt flag CANFDx_CHy_TTIR.WT, switch off its CAN bus output, and enter the bus monitoring mode (CANFDx_CHy_CCCR.MON set to '1'). In the bus monitoring mode, it is still able to receive messages, but cannot send any dominant bits and therefore, cannot acknowledge.

***Note**: To leave the fatal error state, the host must set CANFDx_CHy_CCCR.INIT = 1. After reset of CANFDx_CHy_CCCR.INIT, the node restarts TTCAN communication.*

When no error is encountered during synchronization, the first reference message sets CANFDx_CHy_TTOST.SYS = 01 (synchronizing), the second sets the TTCAN synchronization state (depending on its Next_is_Gap bit) to

CANFDx_CHy_TTOST.SYS = 11 (In_Schedule) or CANFDx_CHy_TTOST.SYS = 10 (In_Gap), enabling all Tx_Triggers and Rx_Triggers.

Potential time master

After configuration, a potential time master will start the transmission of a reference message when it reaches its Tx_Ref_Trigger (or its Tx_Ref_Trigger_Gap when in external event-synchronized time-triggered operation). It will ignore its Watch_Trigger and Watch_Trigger_Gap when it does not receive any message or transmit the reference message successfully before reaching the Watch_Triggers (the reason assumed is that all other nodes still in reset or configuration and does not acknowledge). When it reaches Init_Watch_Trigger, the attempted transmission is aborted, interrupt flag CANFDx_CHy_TTIR.IWT is set, the FSE is frozen, and the cycle time will become invalid, but the node will still be able to take part in CAN bus communication (to acknowledge or send error flags). Resetting CANFDx_CHy_TTIR.IWT will re-enable the transmission of reference messages until the next time Init_Watch_Trigger condition is met, or another CAN message is received. The FSE will be restarted by the reception of a reference message.

When a potential time master reaches the Watch_Triggers after it has received any message but the reference message, it will assume a fatal error (CANFDx_CHy_TTOST.EL = 11, severity 3), set interrupt flag CANFDx_CHy_TTIR.WT, switch off its CAN bus output, and enter the bus monitoring mode (CANFDx_CHy_CCCR.MON set to '1'). In bus monitoring mode, it is still able to receive messages, but it cannot send any dominant bits and therefore, cannot acknowledge.

When no error is detected during initialization, the first reference message sets CANFDx_CHy_TTOST.SYS = 01 (synchronizing), the second sets the TTCAN synchronization state (depending on its Next_is_Gap bit) to CANFDx_CHy_TTOST.SYS = 11 (In_Schedule) or CANFDx_CHy_TTOST.SYS = 10 (In_Gap), enabling all Tx_Triggers and Rx_Triggers.

A potential time master is current time master (CANFDx_CHy_TTOST.MS = 11) when it is the transmitter of the last reference message; otherwise, it is the backup time master (CANFDx_CHy_TTOST.MS = 10).

When all potential time masters have finished configuration, the node with the highest time master priority in the network will become the current time master.

24.5.3 TTCAN gap control

All functions related to gap control apply only when the M_TTCAN is operated in external event-synchronized time-triggered mode (CANFDx_CHy_TTOCF.GEN = 1). In this operation mode the TTCAN message schedule may be interrupted by inserting gaps between the basic cycles of the system matrix. All nodes connected to the CAN network should be configured for external event-synchronized time-triggered operation.

During a gap, all transmissions are stopped and the CAN bus remains idle. A gap is finished when the next reference message starts a new basic cycle. The gap starts at the end of a basic cycle that was started by a reference message with bit Next_is_Gap = '1'; for example, gaps are initiated by the current time master.

The current time master has two options to initiate a gap. A gap can be initiated under software control when the application program writes CANFDx_CHy_TTOCN.NIG = 1. The Next_is_Gap bit will be transmitted as '1' with the next reference message. A gap can also be initiated under hardware control when the application program writes CANFDx_CHy_TTOCN.GCS = 1. When a reference message is started and CANFDx_CHy_TTOCN.GCS is set, Next_is_Gap = '1' will be set.

As soon as that reference message is completed, the CANFDx_CHy_TTOST.WFE bit will announce the gap to the time master and slaves. The current basic cycle will continue until its last time window. The time after the last time window is the gap time.

For the actual time master and the potential time masters, CANFDx_CHy_TTOST.GSI will be set when the last basic cycle has finished and the gap time starts. In nodes that are time slaves, the CANFDx_CHy_TTOST.GSI bit will remain at '0'.

When a potential time master is in synchronization state In_Gap (CANFDx_CHy_TTOST.SYS = 10), it has four options to intentionally finish a gap:

- Under software control by writing CANFDx_CHy_TTOCN.FGP = 1.

- Under hardware control (CANFDx_CHy_TTOCN.GCS = 1), CANFDx_CHy_TTOCN.FGP will automatically be set when an edge from HIGH to LOW at the internal event trigger input pin is detected and restarts the schedule.
- The third option is a time-triggered restart. When CANFDx_CHy_TTOCN.TMG = 1, the next register time mark interrupt (CANFDx_CHy_TTIR.RTMI = 1) will set CANFDx_CHy_TTOCN.FGP and start the reference message.
- Any potential time master will finish a gap when it reaches its Tx_Ref_Trigger_Gap, assuming that the event to synchronize to did not occur on time.

None of these options can cause a basic cycle to be interrupted with a reference message.

Setting CANFDx_CHy_TTOCN.FGP after the gap time has started will start the transmission of a reference message immediately and will thereby synchronize the message schedule. When CANFDx_CHy_TTOCN.FGP is set before the gap time has started (while the basic cycle is still in progress), the next reference message is started at the end of the basic cycle, at the Tx_Ref_Trigger – there will be no gap time in the message schedule.

In time-triggered operation, bit Next_is_Gap = ‘1’ in the reference message will be ignored, as well as the CANFDx_CHy_TTOCN.NIG, CANFDx_CHy_TTOCN.FGP, and CANFDx_CHy_TTOCN.TMG bits.

24.5.4 Stop watch

The stop watch function enables capturing of M_TTCAN internal time values (local time, cycle time, or global time) triggered by an external event.

To enable the stop watch function, the application program must first define local time, cycle time, or global time as stop watch source via CANFDx_CHy_TTOCN.SWS. When CANFDx_CHy_TTOCN.SWS is not equal to ‘00’ and TT Interrupt Register flag CANFDx_CHy_TTIR.SWE is ‘0’, the actual value of the time selected by CANFDx_CHy_TTOCN.SWS will be copied into CANFDx_CHy_TTCPT.SWV on the next rising/falling edge (as configured via CANFDx_CHy_TTOCN.SWP) on stop watch trigger. This will set interrupt flag CANFDx_CHy_TTIR.SWE. After the application program has read CANFDx_CHy_TTCPT.SWV, it may enable the next stop watch event by resetting CANFDx_CHy_TTIR.SWE to '0'.

24.5.5 Local time, cycle time, global time, and external clock synchronization

There are two possible levels in time-triggered CAN: level 1 and level 2. Level 1 provides only time-triggered operation using cycle time. Level 2 additionally provides increased synchronization quality, global time, and external clock synchronization. In both levels, all timing features are based on a local time base – the local time.

The local time is a 16-bit cyclic counter, it is incremented once each NTU. Internally the NTU is represented by a 3-bit counter, which can be regarded as a fractional part (three binary digits) of the local time. Generally, the 3-bit NTU counter is incremented eight times each NTU. If the length of the NTU is shorter than eight CAN clock periods (as may be configured in Level 1, or as a result of clock calibration in Level 2), the length of the NTU fraction is adapted, and the NTU counter is incremented only four times each NTU.

Figure 177 describes the synchronization of the cycle time and global time, performed in the same manner by all TTCAN nodes, including the time master. Any message received or transmitted invokes a capture of the local time taken at the message's frame synchronization event. This frame synchronization event occurs at the sample point of each start-of-frame (SoF) bit and causes the local time to be stored as Sync_Mark. Sync_Marks and Ref_Marks are captured including the 3-bit fractional part.

Whenever a valid reference message is transmitted or received, the internal Ref_Mark is updated from the Sync_Mark. The difference between Ref_Mark and Sync_Mark is the cycle sync mark (Cycle sync mark = Sync_Mark – Ref_Mark) stored in Register CANFDx_CHy_TTCSM. The most significant 16 bits of the difference between Ref_Mark and the actual value of the local time is the cycle time (Cycle time = Local time – Ref_Mark).

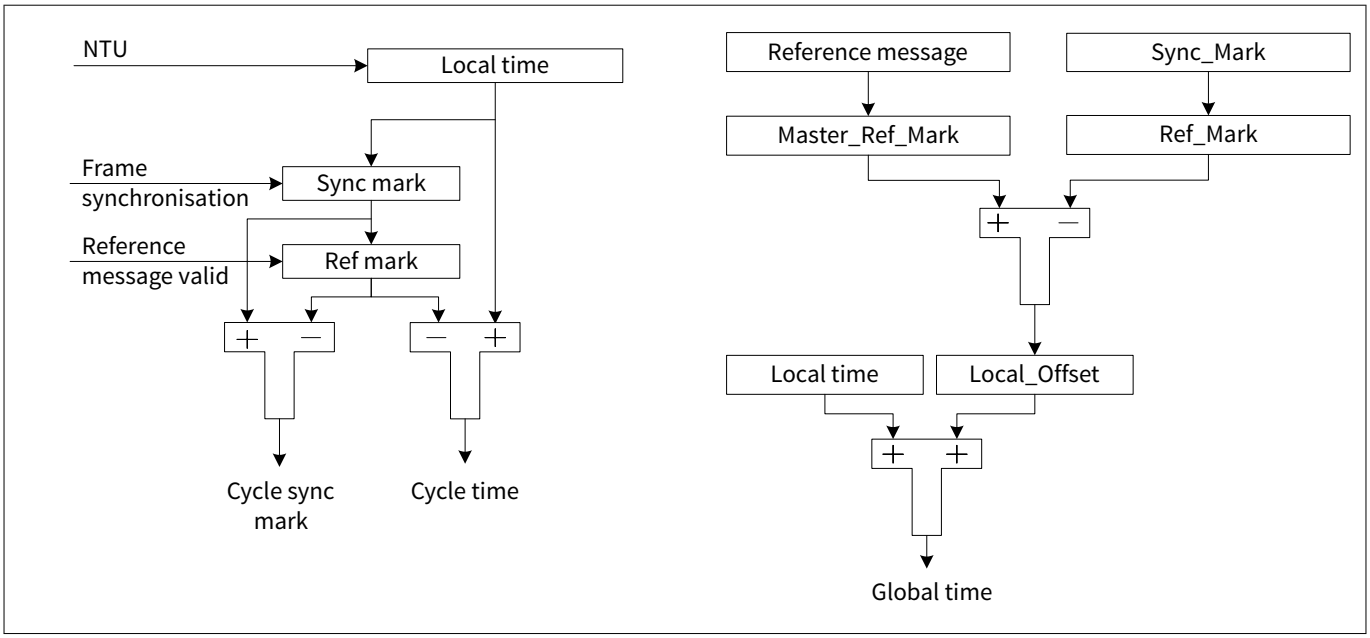


Figure 177 Cycle time and global time synchronization

The cycle time that can be read from CANFDx_CHy_TTCTC.CT is the difference of the node's local time and Ref_Mark, both synchronized into the host clock domain and truncated to 16 bits.

The global time exists for TTCAN level 0 and level 2 only, in level 1 it is invalid. The node's view of the global time is the local image of the global time in (local) NTUs. After configuration, a potential time master will use its own local time as global time. This is done by transmitting its own Ref_Marks as Master_Ref_Marks in the reference message (bytes 3 and 4). The global time that can be read from CANFDx_CHy_TTLGT.GT is the sum of the node's local time and its local offset, both synchronized into the host clock domain and truncated to 16 bit. The fractional part is used for clock synchronization only.

A node that receives a reference message calculates its local offset to the global time by comparing its local Ref_Mark with the received Master_Ref_Mark (see Figure 177). The node's view of the global time is local time + local offset. In a potential time master that has never received another time master's reference message, Local_Offset will be zero. When a node becomes the current time master after having received other reference messages first, Local_Offset will be frozen at its last value. In the time receiving nodes, Local_Offset may be subject to small adjustments, due to clock drift, when another node becomes time master, or when there is a global time discontinuity, signaled by Disc_Bit in the reference message. With the exception of global time discontinuity, the global time provided to the application program by Register CANFDx_CHy_TTLGT is smoothed by a low-pass filtering to have a continuous monotonic value.

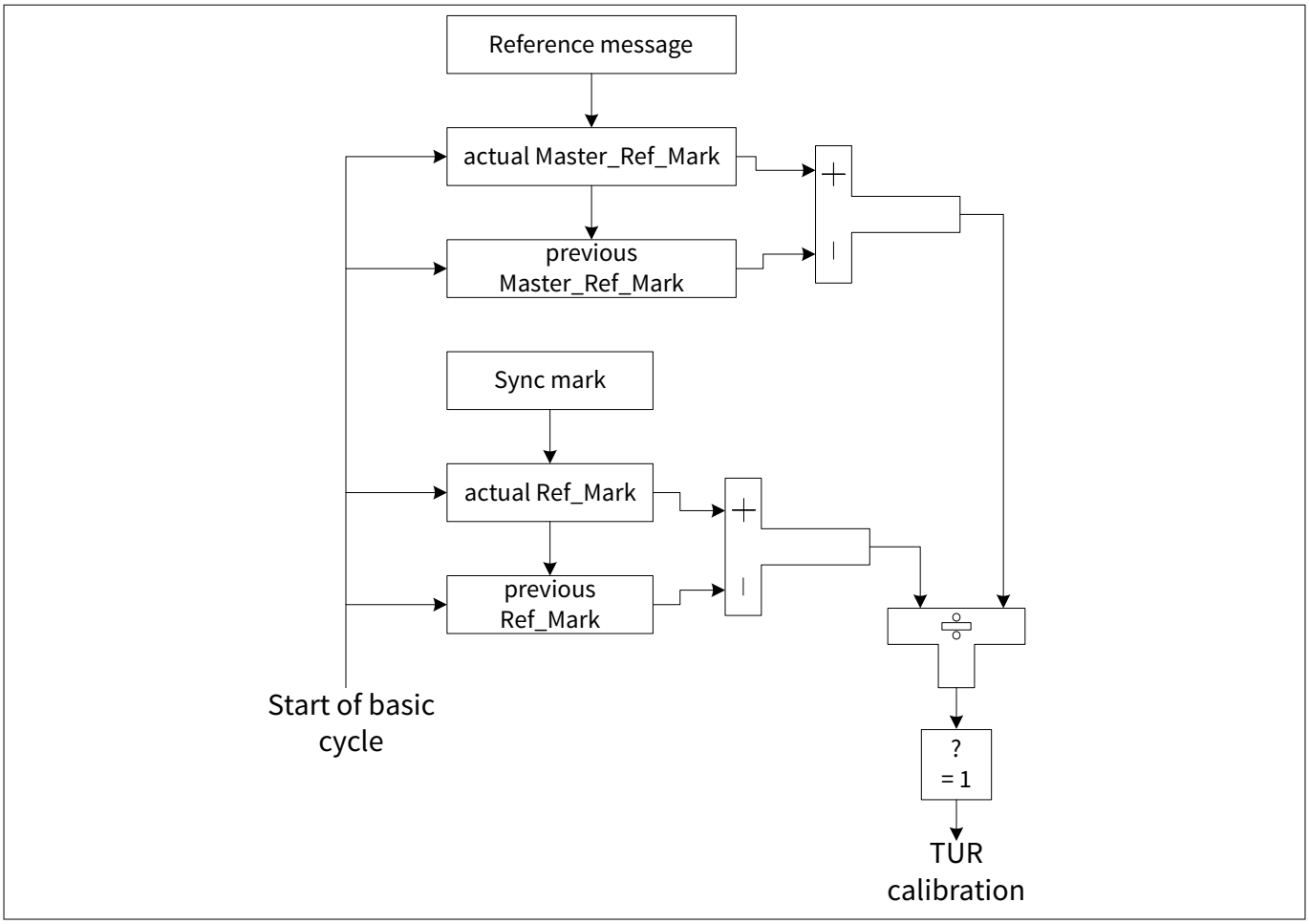


Figure 178 TTCAN level 0 and level 2 drift compensation

Figure 178 illustrates how in TTCAN Level 0 and Level 2 the receiving node compensates the drift between its own local clock and the time master's clock by comparing the length of a basic cycle in local time and in global time. If there is a difference between the two values, and the Disc_Bit in the reference message is not set, a new value for CANFDx_CHy_TURNA.NAV is calculated. If the synchronization deviation (SD) = |NC – CANFDx_CHy_TURNA.NAV| ≤ SDL, the new value for CANFDx_CHy_TURNA.NAV takes effect. Otherwise, the automatic drift compensation is suspended.

In TTCAN Level 0 and Level 2, CANFDx_CHy_TTOST.QCS indicates whether the automatic drift compensation is active or suspended. In TTCAN Level 1, CANFDx_CHy_TTOST.QCS is always '1'.

The current time master may synchronize its local clock speed and the global time phase to an external clock source. This is enabled by bit CANFDx_CHy_TTOCF.EECS.

The stop watch function (see Stop watch) may be used to measure the difference in clock speed between the local clock and the external clock. The local clock speed is adjusted by first writing the newly calculated numerator configuration Low to CANFDx_CHy_TURCF.NCL (CANFDx_CHy_TURCF.DC cannot be updated during operation). The new value takes effect by writing CANFDx_CHy_TTOCN.ECS to '1'.

The global time phase is adjusted by first writing the phase offset into the TT global time preset Register (CANFDx_CHy_TTGTP). The new value takes effect by writing CANFDx_CHy_TTOCN.SGT to '1'. The first reference message transmitted after the global time phase adjustment will have the Disc_Bit set to '1'.

CANFDx_CHy_TTOST.QGTP shows whether the node's global time is in phase with the time master's global time. CANFDx_CHy_TTOST.QGTP is permanently '0' in TTCAN Level 1 and when the SDL is exceeded in TTCAN Level 0,2 (CANFDx_CHy_TTOST.QCS = 0). It is temporarily '0' while the global time is low-pass filtered to supply the application with a continuous monotonic value. There is no low-pass filtering when the last reference message contains a Disc_Bit = 1 or when CANFDx_CHy_TTOST.QCS = 0.

24.5.6 Synchronization triggers

One of the benefits of TTCAN is that it can make communication latency deterministic. To maintain this property across multiple CAN networks these networks must be synchronized. M_TTCAN includes several trigger inputs and outputs to enable this synchronization.

Each M_TTCAN channel has trigger input and trigger output connected to trigger multiplexer. Using trigger functionality each channel has the possibility to not just synchronize with any other M_TTCAN channel, but also to other working network. For more information refer to the Trigger multiplexer.

Stop watch and event trigger inputs for the M_TTCAN channel are connected through the CANx_TT_TR_INy[3] signal coming from the trigger multiplexer. Output trigger from the channel such as Time Mark Trigger and Register Time Mark triggers are connected through CANx_TT_TR_OUTy[3] to the trigger multiplexer.

Using this infrastructure, synchronously running networks are achievable.

24.5.7 TTCAN error level

The ISO 11898-4 specifies four levels of error severity:

- S0 - No error
- S1 - Warning

  Only notification of application, reaction application-specific.
- S2 error

  Notification of application. All transmissions in exclusive or arbitrating time windows are disabled (that is, no data or remote frames may be started). Potential time masters still transmit reference messages with the reference trigger offset CANFDx_CHy_TTOST.RTO set to the maximum value of 127.
- S3 - Severe error

  Notification of application. All CAN bus operations are stopped; that is, transmission of dominant bits is not allowed and CANFDx_CHy_CCCR.MON is set. The S3 error condition remains active until the application updates the configuration (sets CANFDx_CHy_CCCR.CCE).

If several errors are detected at the same time, the highest severity prevails. When an error is detected, the application is notified by CANFDx_CHy_TTIR.ELC. The error level is monitored by CANFDx_CHy_TTOST.EL.

The M_TTCAN signals the following error conditions as required by ISO 11898-4:

Config_Error (S3)

Sets error level CANFDx_CHy_TTOST.EL to ‘11’ when a merged arbitrating time window is not properly closed or when there is a Tx_Trigger with a time mark beyond the Tx_Ref_Trigger.

Watch_Trigger_Reached (S3)

Sets error level CANFDx_CHy_TTOST.EL to ‘11’ when a watch trigger is reached because the reference message is missing.

Application_Watchdog (S3)

Sets error level CANFDx_CHy_TTOST.EL to ‘11’ when the application fails to serve the application watchdog. The application watchdog is configured via CANFDx_CHy_TTOCF.AWL. It is served by reading the CANFDx_CHy_TTOST Register. When the watchdog is not served in time, bit CANFDx_CHy_TTOST.AWE and interrupt flag CANFDx_CHy_TTIR.AW are set, all TTCAN communication is stopped, and the M_TTCAN is set into bus monitoring mode (CANFDx_CHy_CCCR.MON set to '1').

[3] x:CAN instance, y: channel of instance

CAN_Bus_Off (S3)

Entering CAN_Bus_Off state sets error level CANFDx_CHy_TTOST.EL to '11'. CAN_Bus_Off state is signaled by CANFDx_CHy_PSR.BO = 1 and CANFDx_CHy_CCCR.INIT = 1.

Scheduling_Error_2 (S2)

Sets error level CANFDx_CHy_TTOST.EL to '10' if the MSC of one Tx_Trigger has reached 7. In addition, interrupt flag CANFDx_CHy_TTIR.SE2 is set. CANFDx_CHy_TTOST.EL is reset to 00 at the beginning of a matrix cycle when no Tx_Trigger has an MSC of 7 in the preceding matrix cycle.

Tx_Overflow (S2)

Sets error level CANFDx_CHy_TTOST.EL to '10' when the TX count is equal or higher than the expected number of Tx_Triggers CANFDx_CHy_TTMLM.ENTT and a Tx_Trigger event occurs. In addition, interrupt flag CANFDx_CHy_TTIR.TXO is set. CANFDx_CHy_TTOST.EL is reset to 00 when the TX count is no more than CANFDx_CHy_TTMLM.ENTT at the start of a new matrix cycle.

Scheduling_Error_1 (S1)

Sets error level CANFDx_CHy_TTOST.EL to '01' if within one matrix cycle the difference between the maximum MSC and the minimum MSC for all trigger memory elements (of exclusive time windows) is larger than 2, or if one of the MSCs of an exclusive Rx_Trigger has reached 7. In addition, interrupt flag CANFDx_CHy_TTIR.SE1 is set. If within one matrix cycle none of these conditions is valid, CANFDx_CHy_TTOST.EL is reset to 00.

Tx_Underflow (S1)

Sets error level CANFDx_CHy_TTOST.EL to '01' when the TX count is less than the expected number of Tx_Triggers CANFDx_CHy_TTMLM.ENTT at the start of a new matrix cycle. In addition, interrupt flag CANFDx_CHy_TTIR.TXU is set. CANFDx_CHy_TTOST.EL is reset to 00 when the TX count is at least CANFDx_CHy_TTMLM.ENTT at the start of a new matrix cycle.

24.5.8 TTCAN message handling

24.5.8.1 Reference message

For potential time masters, the identifier of the reference message is configured via CANFDx_CHy_TTRMC.RID. No dedicated TX buffer is required for transmission of the reference message. When a reference message is transmitted, the first data byte (TTCAN Level 1) and the first four data bytes (TTCAN level 0 and level 2) will be provided by the FSE.

If the Payload Select reference message CANFDx_CHy_TTRMC.RMPS is set, the rest of the reference message's payload (Level 1: bytes 2-8, level 0 and level 2: bytes 5-6) is taken from TX buffer 0. In this case, the data length DLC code from message buffer 0 is used.

Table 188 Number of data bytes transmitted with a reference messages

CANFDx_CHy_TTRMC.RMPS	CANFDx_CHy_TXBRP.TRP0	Level 0	Level 1	Level 2
0	0	4	1	4
0	1	4	1	4
1	0	4	1	4
1	1	4 + MB0	1 + MB0	4 + MB0

To send additional payload with the reference message in level 1, a DLC > 1 should be configured. For level 0 and level 2 a DLC > 4 is required. In addition, the transmission request pending bit CANFDx_CHy_TXBRP.TRP0 of

message buffer 0 must be set (see Table 188). If CANFDx_CHy_TXBRP.TRP0 is not set when a reference message is started, the reference message is transmitted with the data bytes supplied by the FSE only.

For acceptance filtering of reference messages the reference identifier CANFDx_CHy_TTRMC.RID is used.

24.5.8.2 Message reception

Message reception is done via the two RX FIFOs in the same way as for event-driven CAN communication.

The MSC is part of the corresponding trigger memory element and must be initialized to zero during configuration. It is updated while the M_TTCAN is in synchronization states In_Gap or In_Schedule. The update happens at the message's Rx_Trigger. At this point, it is checked at which acceptance filter element the latest message received in this basic cycle is matched. The matching filter number is stored as the acceptance filter result. If this is the same as the filter number defined in this trigger memory element, the MSC is decremented by one. If the acceptance filter result is not the same filter number as defined for this filter element, or if the acceptance filter result is cleared, the MSC is incremented by one. At each Rx_Trigger and at each start of cycle, the last acceptance filter result is cleared.

The time mark of an Rx_Trigger should be set to a value that ensures reception and acceptance filtering for the targeted message is completed. This should consider the RAM access time and the order of the filter list. It is recommended, that filters used for Rx_Triggers are placed at the beginning of the filter list. It is not recommended to use an Rx_Trigger for the reference message.

24.5.8.3 Message transmission

For time-triggered message transmission, the M_TTCAN supplies 32 dedicated TX buffers (see TTCAN configuration). A TX FIFO or TX queue is not available when the M_TTCAN is configured for time-triggered operation (CANFDx_CHy_TTOCF.OM = 01 or 10).

Each Tx_Trigger in the trigger memory points to a particular TX buffer containing a specific message. There may be more than one Tx_Trigger for a given TX buffer if that TX buffer contains a message that is to be transmitted more than once in a basic cycle or matrix cycle.

The application program must update the data regularly and on time, synchronized to the cycle time. The host CPU should ensure that no partially updated messages are transmitted. To assure this the host should proceed in the following way:

- Tx_Trigger_Single/Tx_Trigger_Merged/Tx_Trigger_Arbitration:
  - Check whether the previous transmission has completed by reading TXBTO
  - Update the TX buffer's configuration and/or payload
  - Issue an Add Request to set the TX buffer request pending bit
- Tx_Trigger_Continous:
  - Issue a cancellation request to reset the TX buffer request pending bit
  - Check whether the cancellation has finished by reading CANFDx_CHy_TXBCF
  - Update TX buffer configuration and/or payload
  - Issue an add request to set the TX buffer request pending bit

The message MSC stored with the corresponding Tx_Trigger provides information on the success of the transmission.

The MSC is incremented by one when the transmission cannot be started because the CAN bus was not idle within the corresponding transmit enable window or when the message was started but could not be completed successfully. The MSC is decremented by one when the message is transmitted successfully or when the message could have been started within its transmit enable window but was not started because transmission was disabled (M_TTCAN in error level S2 or host has disabled this particular message).

The TX buffers may be managed dynamically – several messages with different identifiers may share the same TX buffer element. In this case the host must ensure that no transmission request is pending for the TX buffer element to be reconfigured by checking CANFDx_CHy_TXBRP.

If a TX buffer with pending transmission request should be updated, the host must first issue a cancellation request and check whether the cancellation has completed by reading CANFDx_CHy_TXBCF before it starts updating.

The TX handler will transfer a message from the message RAM to its intermediate output buffer at the trigger element, which becomes active immediately before the Tx_Trigger element that defines the beginning of the transmit window. During and after transfer time, the transmit message may not be updated and its CANFDx_CHy_TXBRP bit may not be changed. To control this transfer time, an additional trigger element may be placed before the Tx_Trigger. An example is a Time_Base_Trigger, which does not cause any other action. The difference in time marks between the Tx_Trigger and the preceding trigger should be large enough to guarantee that the TX handler can read four words from the message RAM even at high RAM access load from other modules.

Transmission in exclusive time windows

A transmission is started time-triggered when the cycle time reaches the time mark of a Tx_Trigger_Single or Tx_Trigger_Continous. There is no arbitration on the bus with messages from other nodes. The MSC is updated according the result of the transmission attempt. After successful transmission started by a Tx_Trigger_Single, the respective TX buffer request pending bit is reset. After successful transmission started by a Tx_Trigger_Continous the respective TX buffer request pending bit remains set. When the transmission is not successful due to disturbances, it will be repeated the next time one of its Tx_Triggers becomes active.

Transmission in arbitrating time windows

A transmission is started time-triggered when the cycle time reaches the time mark of a Tx_Trigger_Arbitration. Several nodes may start to transmit at the same time. In this case the message has to arbitrate with the messages from other nodes. The MSC is not updated. When the transmission is not successful (lost arbitration or disturbance), it will be repeated the next time one of its Tx_Triggers becomes active.

Transmission in merged arbitrating time windows

The purpose of a merged arbitrating time window is to enable multiple nodes to send a limited number of frames, which are transmitted in immediate sequence, the order given by CAN arbitration. It is not intended for burst transmission by a single node. Because the node does not have exclusive access within this time window, all requested transmissions may not be successful.

Messages that have lost arbitration or were disturbed by an error, may be retransmitted inside the same merged arbitrating time window. The retransmission will not be started if the corresponding transmission request pending flag was reset by a successful TX cancellation.

In single transmit windows, the TX handler transmits the message indicated by the message number of the trigger element. In merged arbitrating time windows, it can handle up to three message numbers from the trigger list. Their transmissions will be attempted in the sequence defined by the trigger list. If the time mark of a fourth message is read before the first is transmitted (or canceled by the host), the fourth request will be ignored.

The transmission inside a merged arbitrating time window is not time-triggered. The transmission of a message may start before its time mark, or after the time mark if the bus was not idle.

The messages transmitted by a specific node inside a merged arbitrating time window will be started in the order of their Tx_Triggers. Therefore, a message with low CAN priority may prevent the successful transmission of a following message with higher priority, if there is competing bus traffic. This should be considered for the configuration of the trigger list. Time_Base_Triggers may be placed between consecutive Tx_Triggers to define the time until the data of the corresponding TX buffer needs to be updated.

24.5.9 TTCAN interrupt and error handling

The TT Interrupt Register CANFDx_CHy_TTIR consists of four segments. Each interrupt can be enabled separately by the corresponding bit in the TT Interrupt Enable Register CANFDx_CHy_TTIE. The flags remain set until the host clears them. A flag is cleared by writing a '1' to the corresponding bit position.

The first segment consists of flags CER, AW, WT, and IWT. Each flag indicates a fatal error condition where the CAN communication is stopped. With the exception of IWT, these error conditions require a reconfiguration of the M_TTCAN module before the communication can be restarted.

The second segment consists of flags ELC, SE1, SE2, TXO, TXU, and GTE. Each flag indicates an error condition where the CAN communication is disturbed. If they are caused by a transient failure, such as by disturbances on the CAN bus, they will be handled by the TTCAN protocol's failure handling and do not require intervention by the application program.

The third segment consists of flags GTD, GTW, SWE, TTMI, and RTMI. The first two flags are controlled by global time events (level 0 and level 2 only) that require a reaction by the application program. With a stop watch event, internal time values are captured. The trigger time mark interrupt notifies the application that a specific Time_Base_Trigger is reached. The register time mark interrupt signals that the time referenced by CANFDx_CHy_TTOCN.TMC (cycle, local, or global) equals time mark CANFDx_CHy_TTTMK.TM. It can also be used to finish a gap.

The fourth segment consists of flags SOG, CSM, SMC, and SBC. These flags provide a means to synchronize the application program to the communication schedule.

24.5.10 Level 0

TTCAN Level 0 is not part of ISO 11898-4. This operation mode makes the hardware, that in TTCAN level 2 maintains the calibrated global time base, also available for event-driven CAN according to ISO 11898-1:2015.

Level 0 operation is configured via CANFDx_CHy_TTOCF.OM = 11. In this mode, M_TTCAN operates in event-driven CAN communication; there is no fixed schedule, the configuration of CANFDx_CHy_TTOCF.GEN is ignored. External event-synchronized operation is not available in Level 0. A synchronized time base is maintained by transmission of reference messages.

In Level 0 the trigger memory is not active and need not be configured. The time mark interrupt flag (CANFDx_CHy_TTIR.TTMI) is set when the cycle time has reached CANFDx_CHy_TTOCF.IRTO × 0x200. It reminds the host to set a transmission request for message buffer 0. The Watch_Trigger interrupt flag (CANFDx_CHy_TTIR.WT) is set when the cycle time has reached 0xFF00. These values were chosen to have enough margin for a stable clock calibration. There are no further TT-error-checks.

Register time mark interrupts (CANFDx_CHy_TTIR.RTMI) are also possible.

The reference message is configured as for Level 2 operation. Received reference messages are recognized by the identifier configured in Register CANFDx_CHy_TTRMC. For the transmission of reference messages only message buffer 0 may be used. The node transmits reference messages any time the host sets a transmission request for message buffer 0; there is no reference trigger offset.

- Level 0 operation is configured via:
  - CANFDx_CHy_TTRMC
  - CANFDx_CHy_TTOCF except EVTP, AWL, GEN
  - CANFDx_CHy_TTMLM except ENTT, TXEW
  - CANFDx_CHy_TURCF
- Level 0 operation is controlled via:
  - CANFDx_CHy_TTOCN except NIG, TMG, FGP, GCS, TTMIE
  - CANFDx_CHy_TTGTP
  - CANFDx_CHy_TTTMK

- CANFDx_CHy_TTIR excluding bits CER, AW, IWT SE2, SE1, TXO, TXU, SOG (no function)
- CANFDx_CHy_TTIR – the following bits have changed function:
  1. TTMI not defined by trigger memory - activated at cycle time CANFDx_CHy_TTOCF.IRTO × 0x200
  2. WT not defined by trigger memory - activated at cycle time 0xFF00

- Level 0 operation is signaled via:
  - CANFDx_CHy_TTOST excluding bits AWE, WFE, GSI, GFI, RTO (no function)

24.5.10.1 Synchronizing

Figure 179 describes the states and state transitions in TTCAN level 0 operation. Level 0 has no In_Gap state.

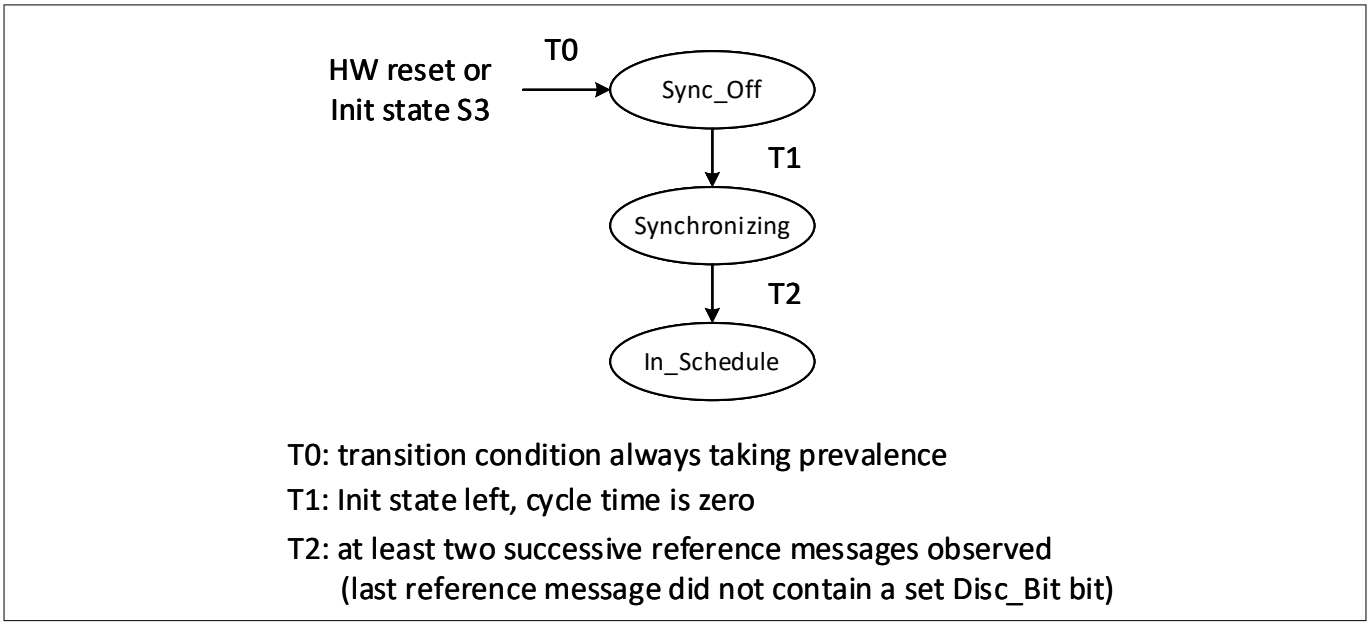


Figure 179 Level 0 schedule synchronization state machine

24.5.10.2 Handling error levels

During Level 0 operation only the following error conditions may occur:

- Watch_Trigger_Reached (S3), reached cycle time 0xFF00
- CAN_Bus_Off (S3)

Because S1 and S2 errors are not possible, the error level can only switch between S0 (no error) and S3 (severe error). In TTCAN level 0 an S3 error is handled differently. When S3 error is reached, both CANFDx_CHy_TTOST.SYS and CANFDx_CHy_TTOST.MS are reset, and interrupt flags CANFDx_CHy_TTIR.GTE and CANFDx_CHy_TTIR.GTD are set.

When S3 (CANFDx_CHy_TTOST.EL = 11) is entered, bus monitoring mode is, contrary to TTCAN level 1 and level 2, not entered. S3 error level is left automatically after transmission (time master) or reception (time slave) of the next reference message.

24.5.10.3 Master slave relation

Figure 180 describes the master slave relation in TTCAN Level 0. In case of an S3 error, the M_TTCAN returns to state Master_Off.

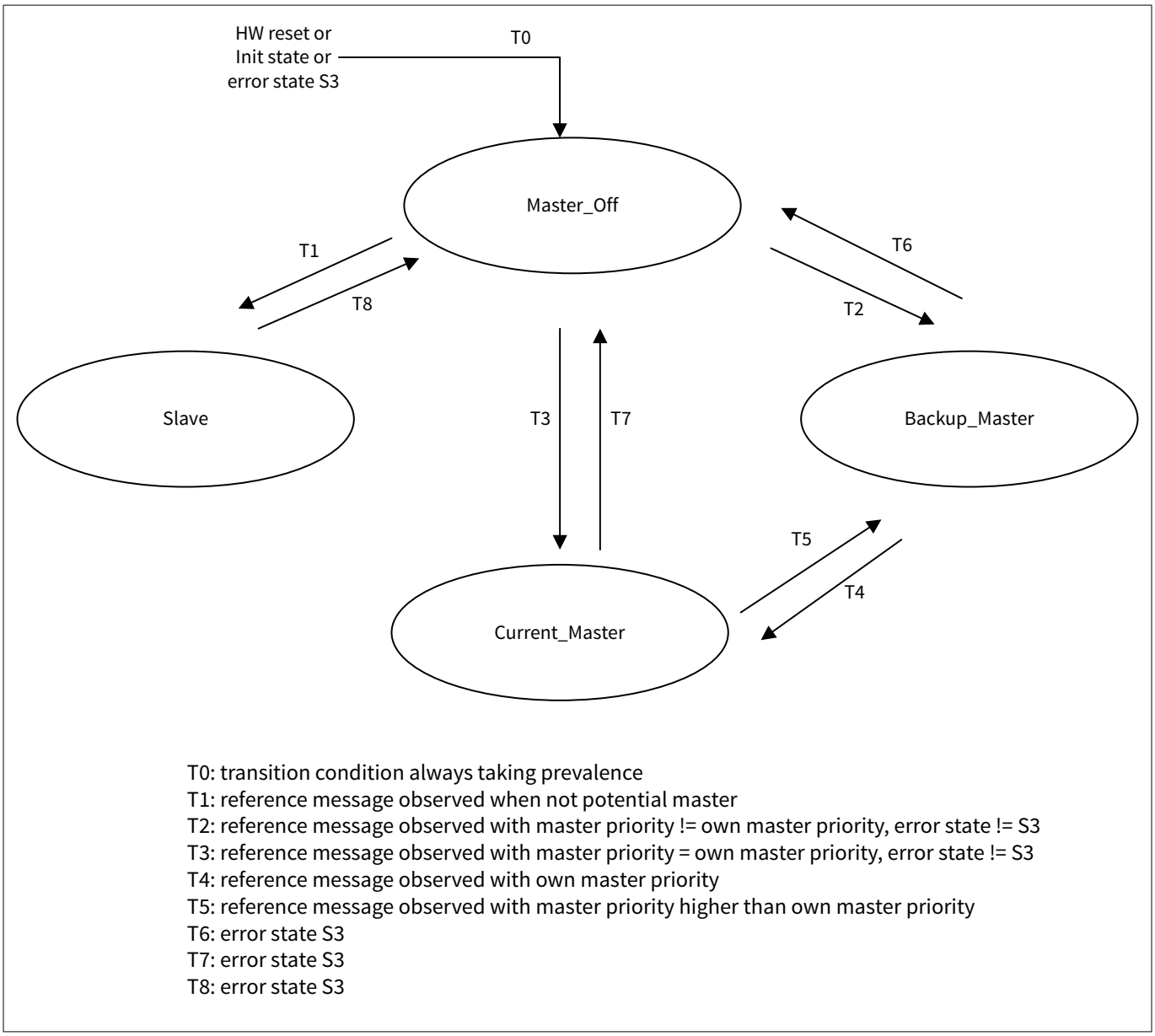


Figure 180 Level 0 master to slave relation

24.5.11 Synchronization to external time schedule

This feature can be used to synchronize the phase of the M_TTCAN's schedule to an external schedule (for example, that of a second TTCAN network). It is applicable only when the M_TTCAN is current time master (CANFDx_CHy_TTOST.MS = 11).

External synchronization is controlled by the CANFDx_CHy_TTOCN.ESCN bit. If CANFDx_CHy_TTOCN.ESCN is set, at rising edge of the internal event trigger pin, the M_TTCAN compares its actual cycle time with the target phase value configured by CANFDx_CHy_TTGTP.CTP.

Before setting CANFDx_CHy_TTOCN.ESCN, the host should adapt the phases of the two time schedules, for example, by using the TTCAN gap control (see TTCAN gap control). When the host sets CANFDx_CHy_TTOCN.ESCN, CANFDx_CHy_TTOST.SPL is set.

If the difference between the cycle time and target phase value CANFDx_CHy_TTGTP.CTP at the trigger is greater than 9 NTU, the phase lock bit CANFDx_CHy_TTOST.SPL is reset, and interrupt flag CANFDx_CHy_TTIR.CSM is set. CANFDx_CHy_TTOST.SPL is also reset (and CANFDx_CHy_TTIR.CSM is set), when another node becomes time master.

If both CANFDx_CHy_TTOST.SPL and CANFDx_CHy_TTOCN.ESCN are set, and if the difference between the cycle time and the target phase value CANFDx_CHy_TTGTP.CTP is less or equal 9 NTU, the phase lock bit CANFDx_CHy_TTOST.SPL remains set, and the measured difference is used as reference trigger offset value to adjust the phase at the next transmitted reference message.

***Note**: The rising edge detection at the internal pin is enabled at the start of each basic cycle. The first rising edge triggers the compare of the actual cycle time with CANFDx_CHy_TTGTP.CTP. All further edges until the beginning of the next basic cycle are ignored.*

24.6 Setup procedures

This section provides example procedures for configurations of M_TTCAN group and flow for respective M_TTCAN channels.

24.6.1 General program flow

This is a general flow to configure the M_TTCAN module.

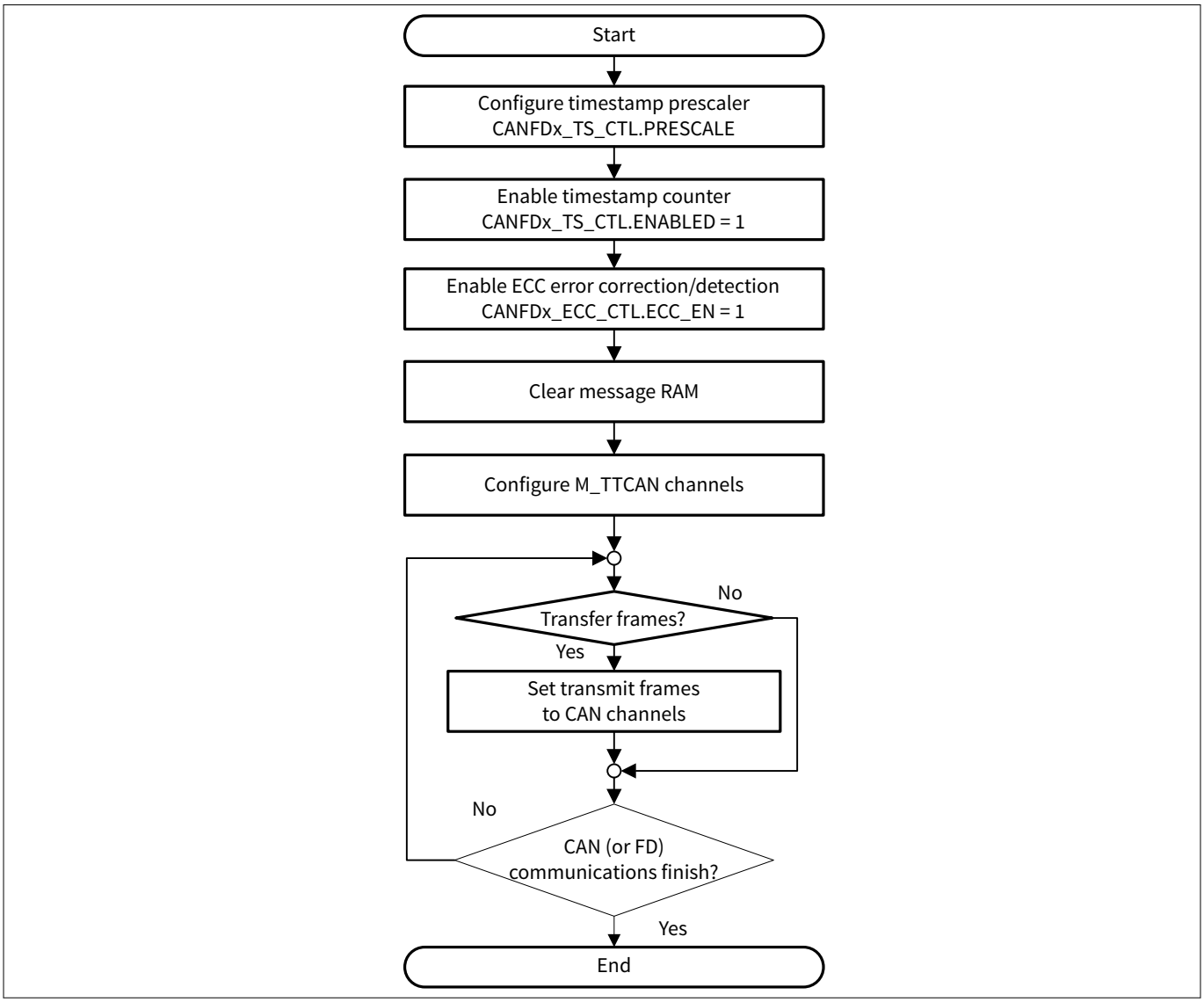


Figure 181 General program flow

24.6.2 Clock stop request

To save power, the application can stop providing clock to unused M_TTCAN channels by following these steps.

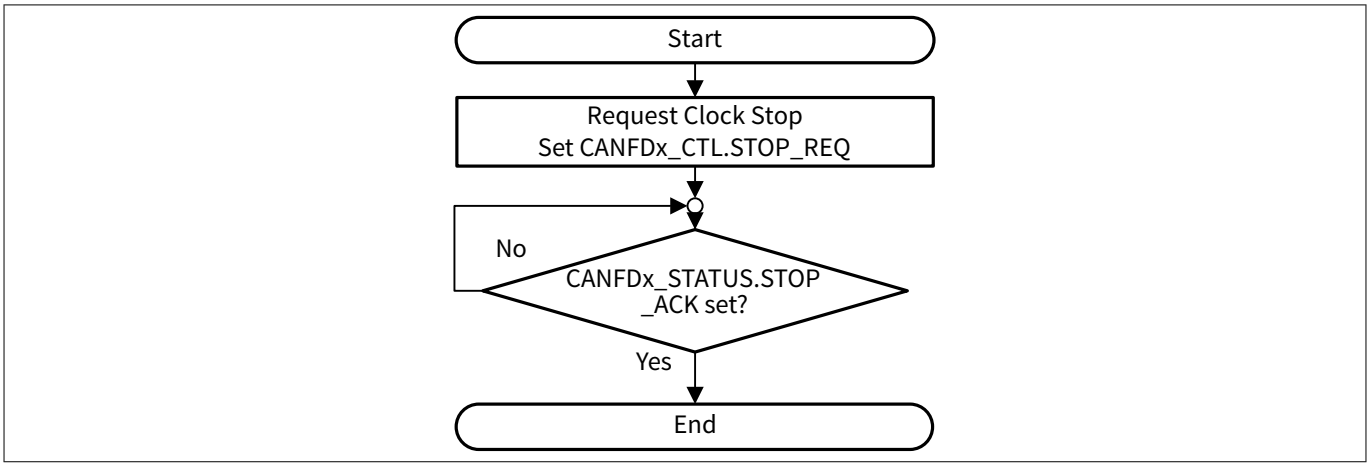


Figure 182 Clock stop request procedure

To resume providing clock, the CANFDx_CTL.STOP_REQ bit should be reset.

24.6.3 Message RAM OFF operation

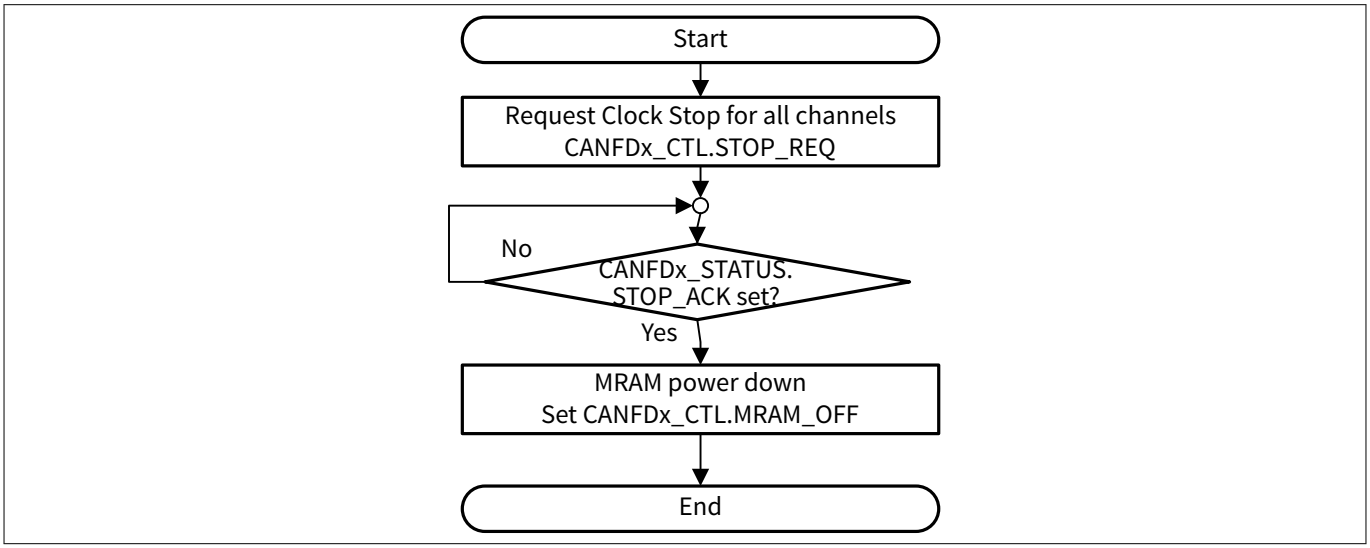


Figure 183 Message RAM OFF operation

24.6.4 Message RAM ON operation

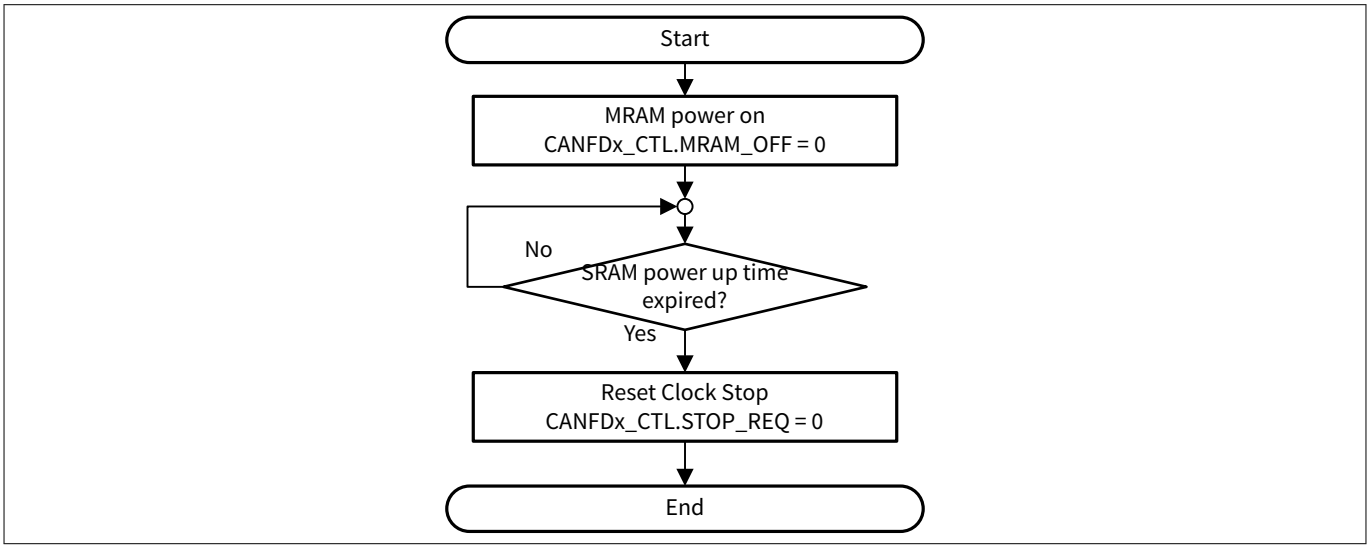


Figure 184 Message RAM on procedure

After switching message RAM ON again, software needs to allow a certain power-up time before message RAM can be used; that is, before STOP_REQ can be de-asserted. The power-up time is equivalent to the system SRAM power-up time specified in the CPUSS.RAM_PWR_DELAY_CTL Register.

24.6.5 Consolidated interrupt handling

When using consolidated interrupt for the M_TTCAN group, follow the procedure given in Figure 185.

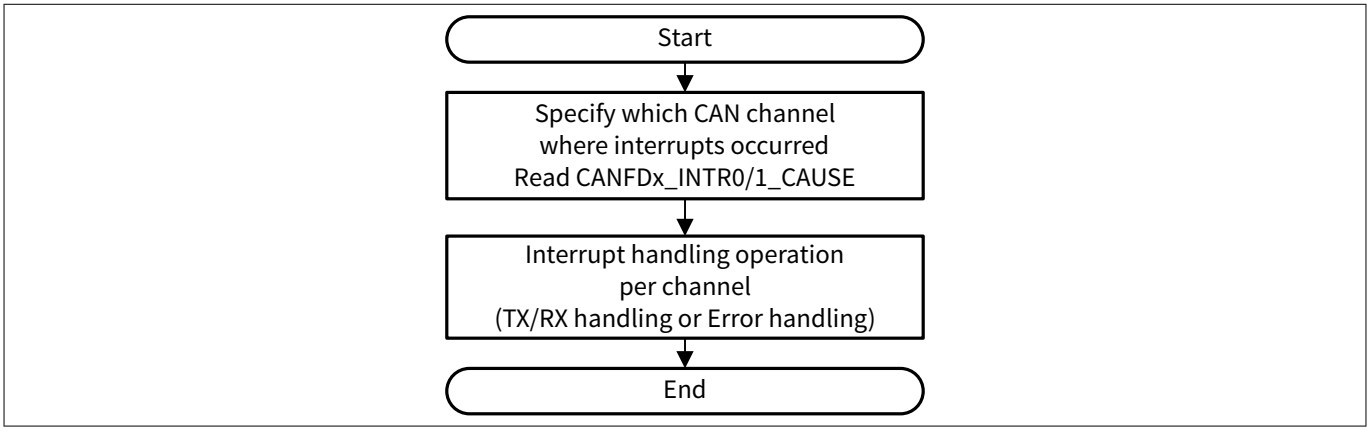


Figure 185 Consolidated interrupt processing

24.6.6 Procedures specific to M_TTCAN channel

This section describes sample procedures per channel. If several M_TTCAN channels are used, the application should configure each channel as shown in Figure 186. The figure shows the general program flow (per channel).

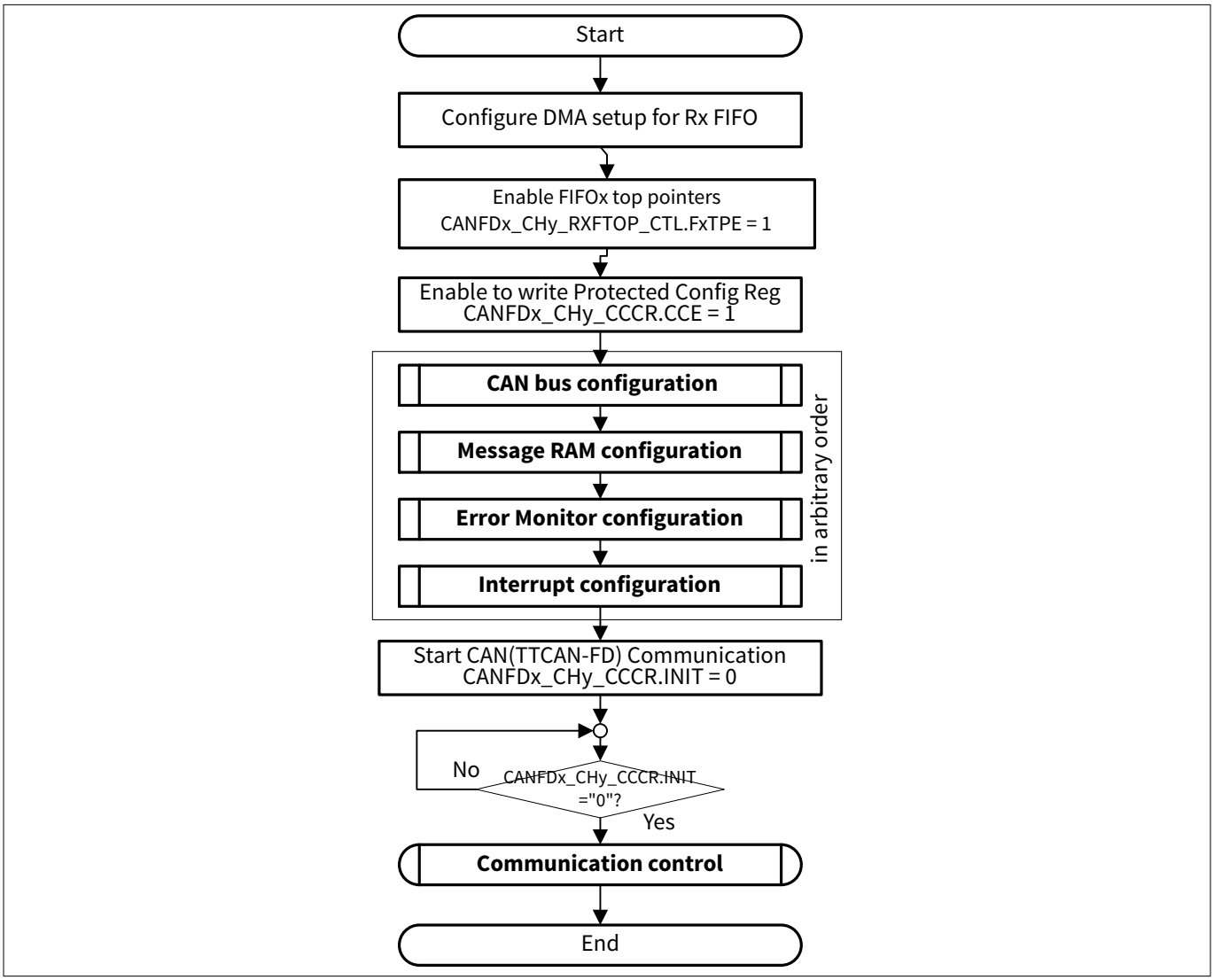


Figure 186 Configuration sequence specific to channel

24.6.6.1 CAN bus configuration

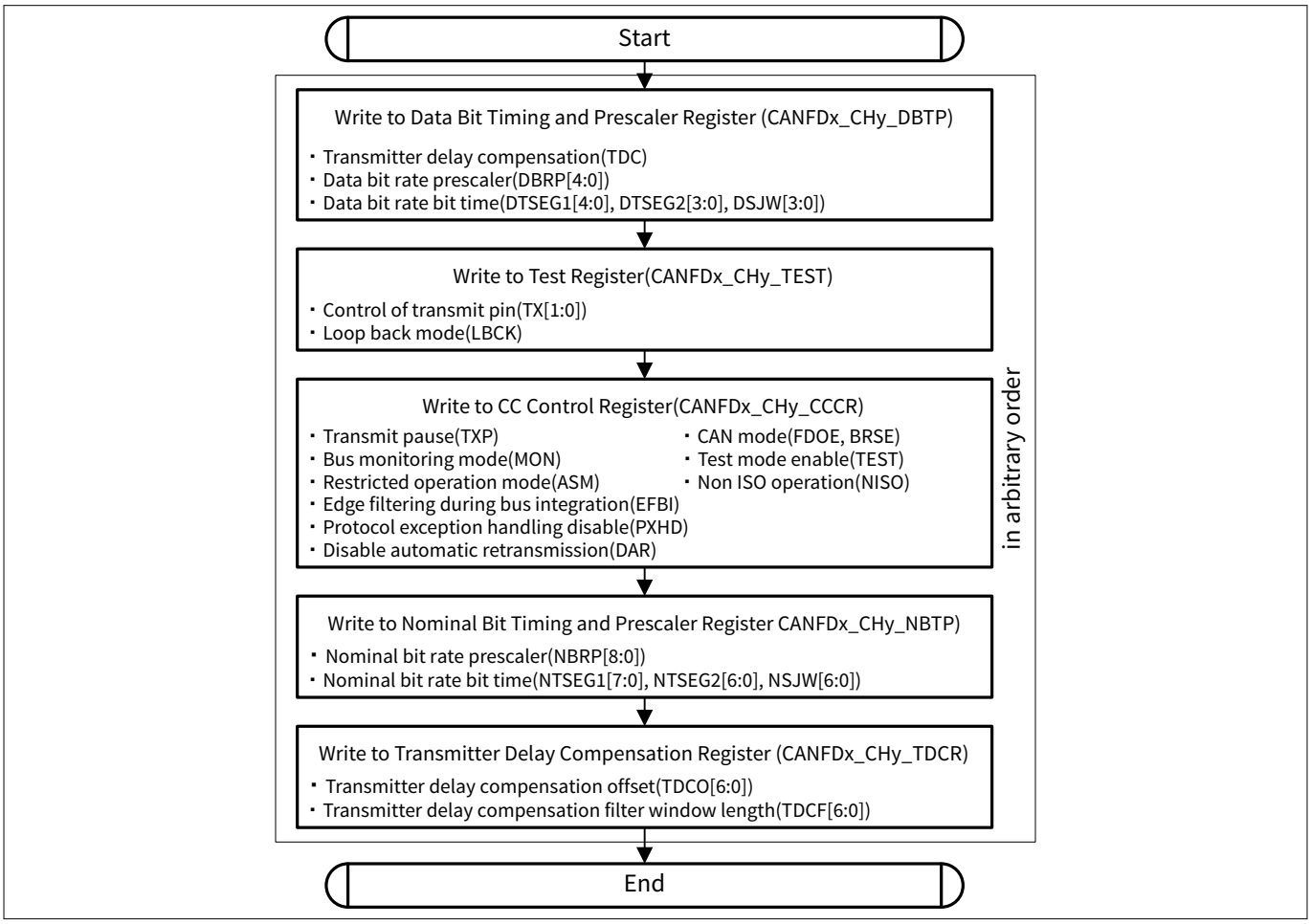


Figure 187 Configuration required for CAN bus

24.6.6.2 Message RAM configuration

The following flow chart shows an overview of the message RAM configuration.

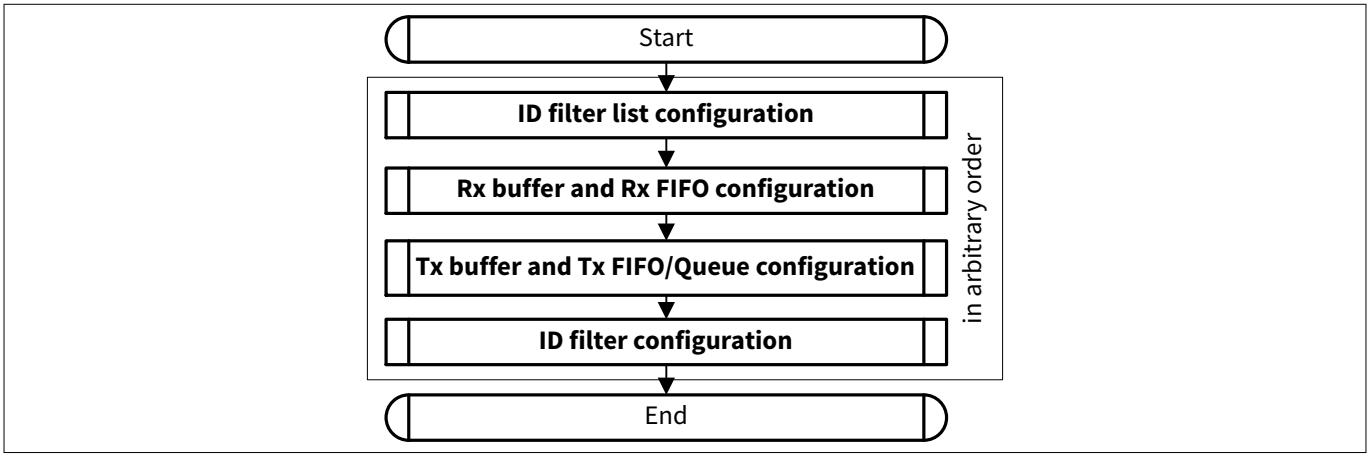


Figure 188 Message RAM configuration overview

Each configuration mentioned in the overview is detailed in the following figures.

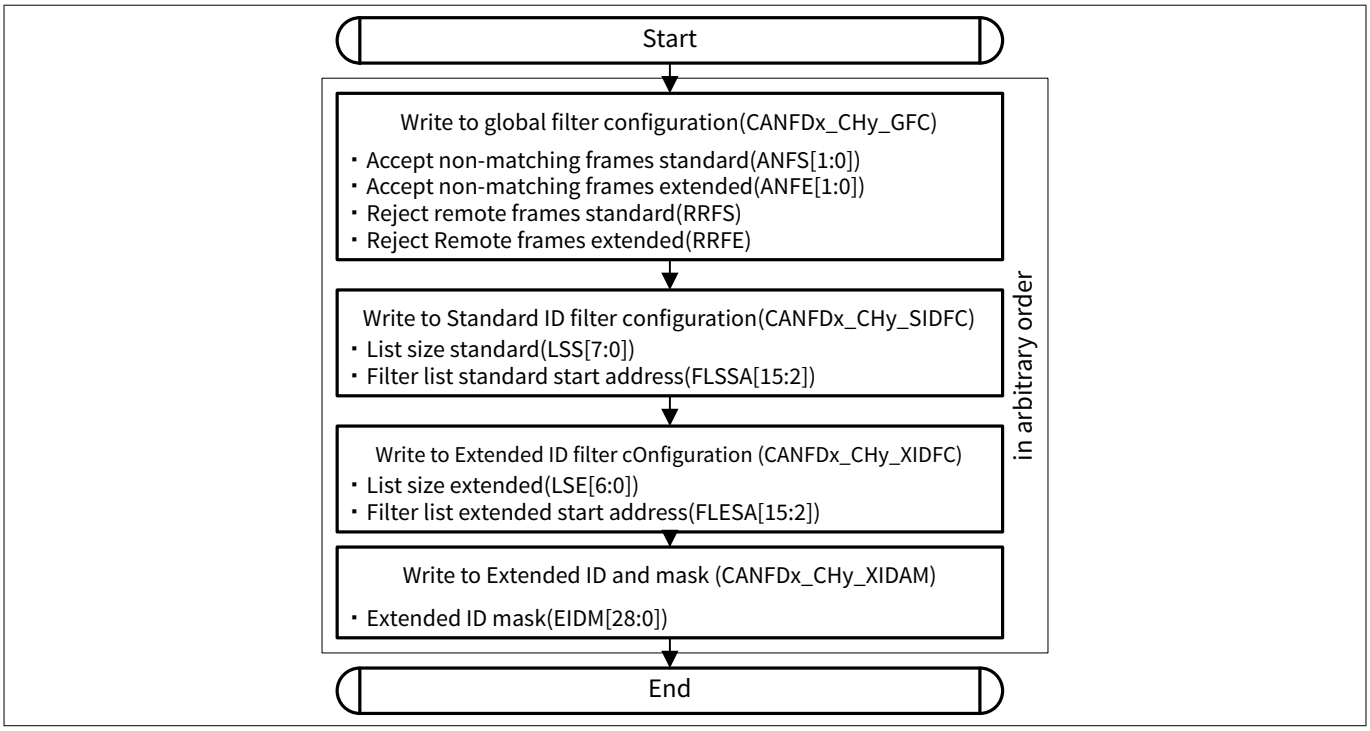


Figure 189 ID filter list configuration

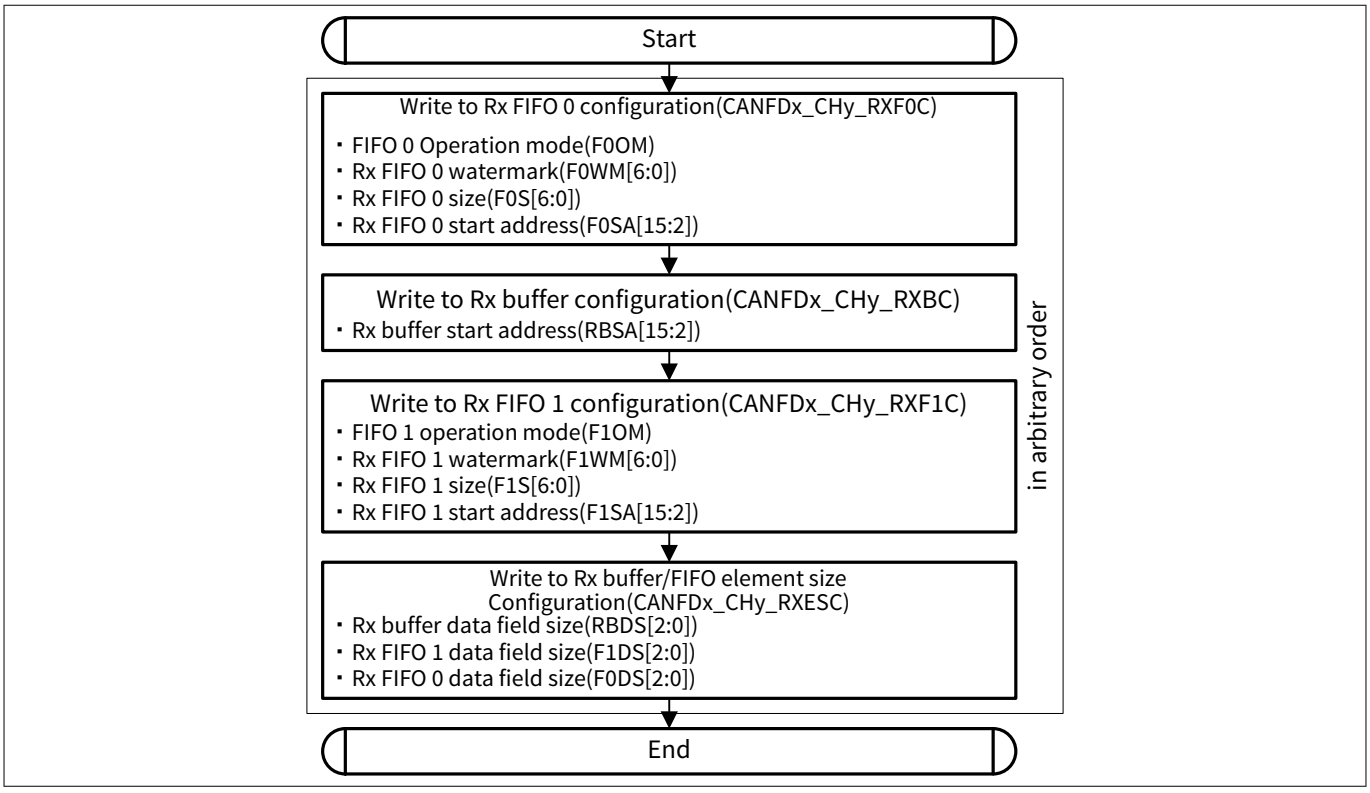


Figure 190 RX FIFO and RX buffer configuration

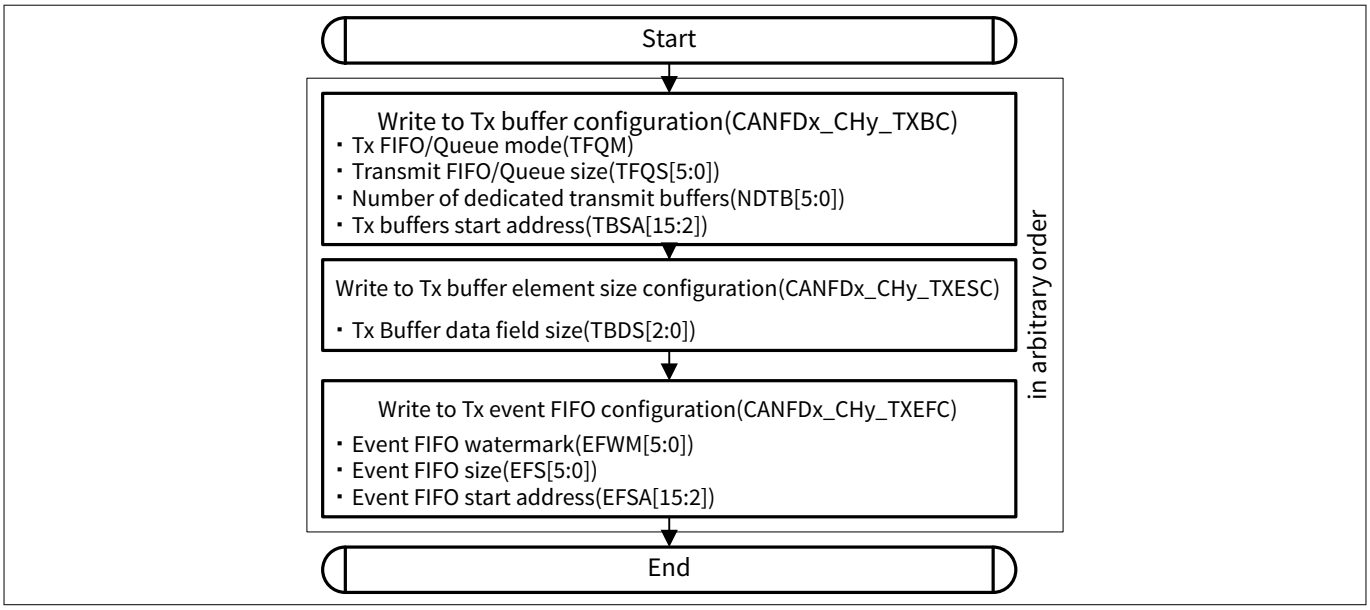


Figure 191 TX buffer and TX FIFO/queue configuration

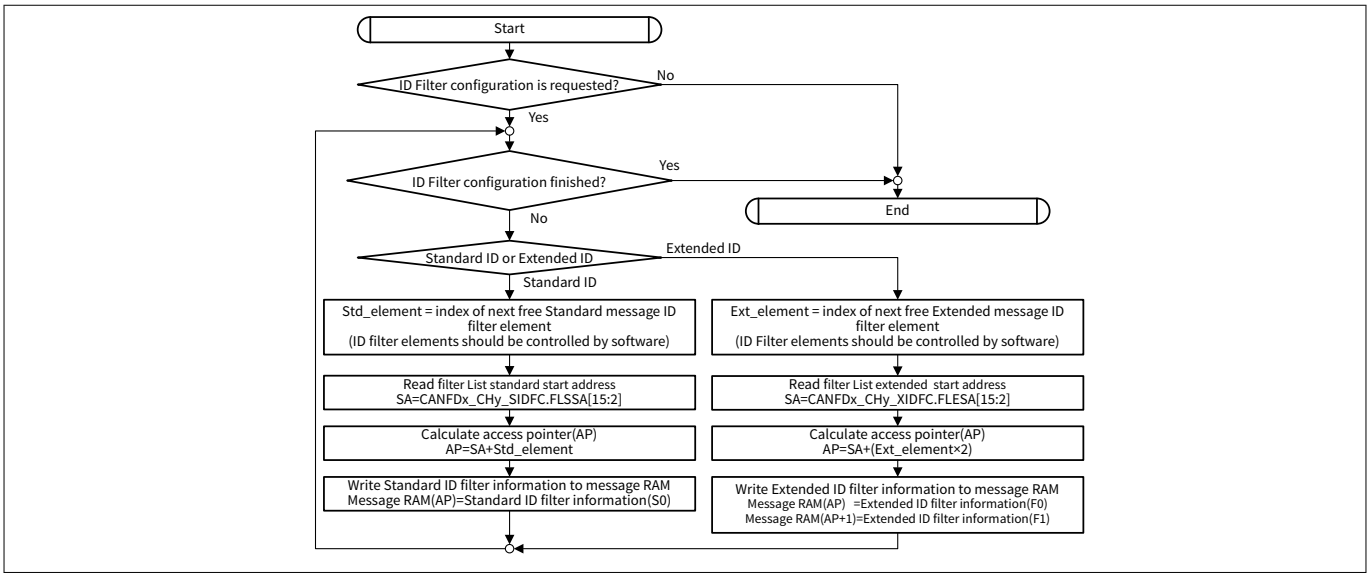


Figure 192 ID filter configuration

24.6.6.3 Interrupt configuration

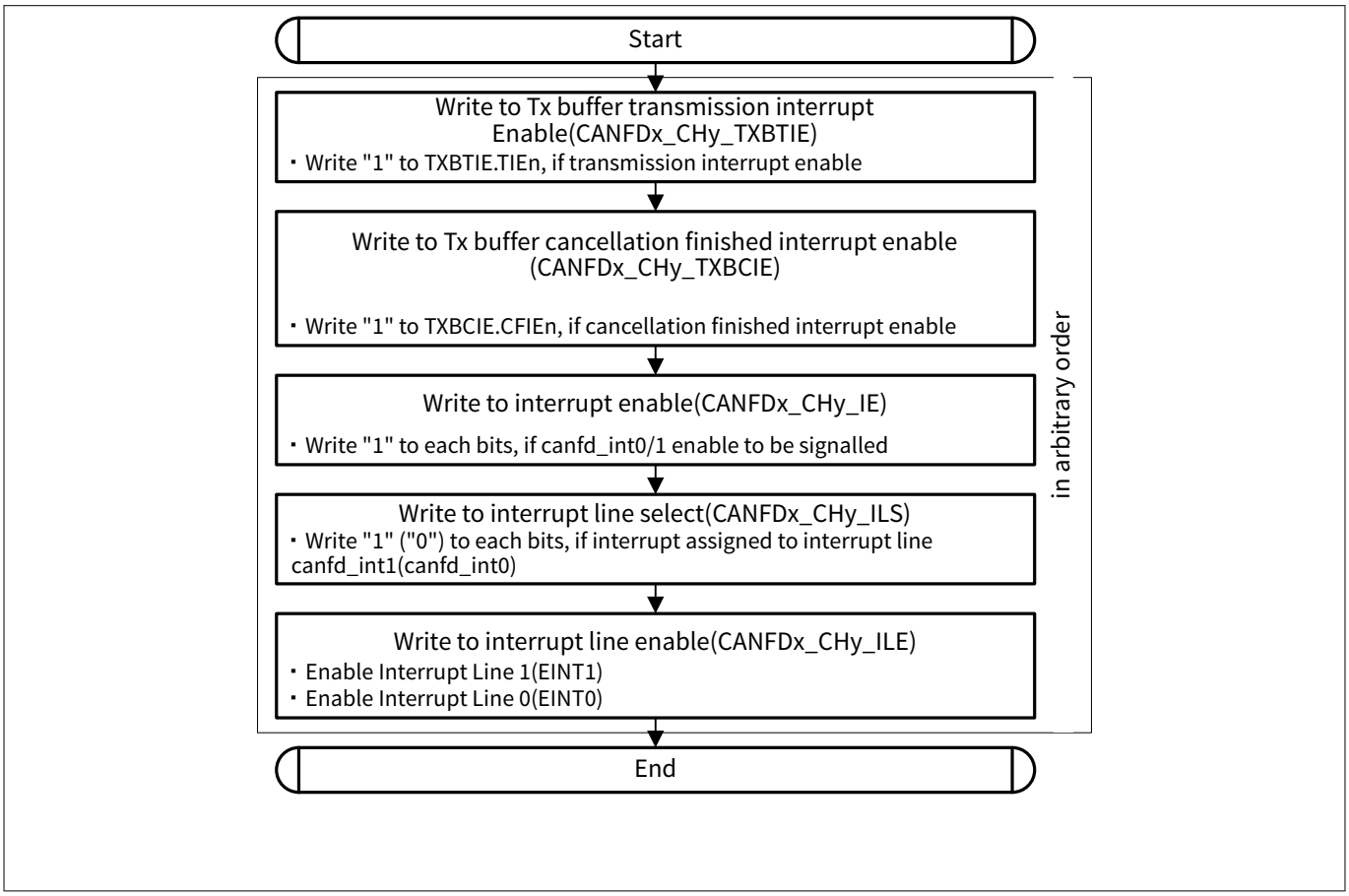


Figure 193 Interrupt configuration

24.6.6.4 Transmit frame configuration

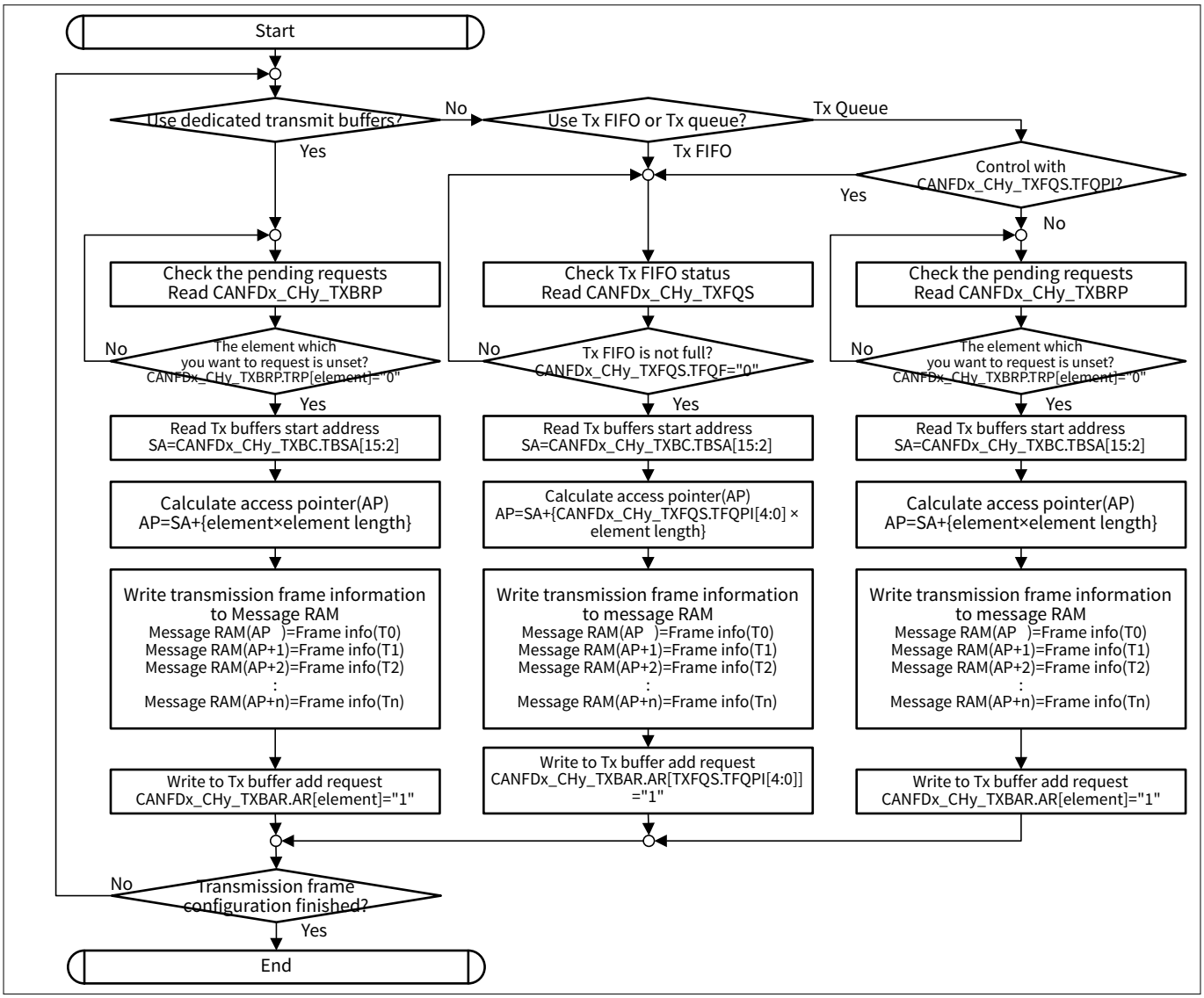


Figure 194 Transmit frame configuration

24.6.6.5 Interrupt handling

When consolidated interrupts are configured, INTR0/1_CAUSE Register will be read to find out the source M_TTCAN channel for the triggered interrupt. Figure 195 shows a general interrupt handling flow chart.

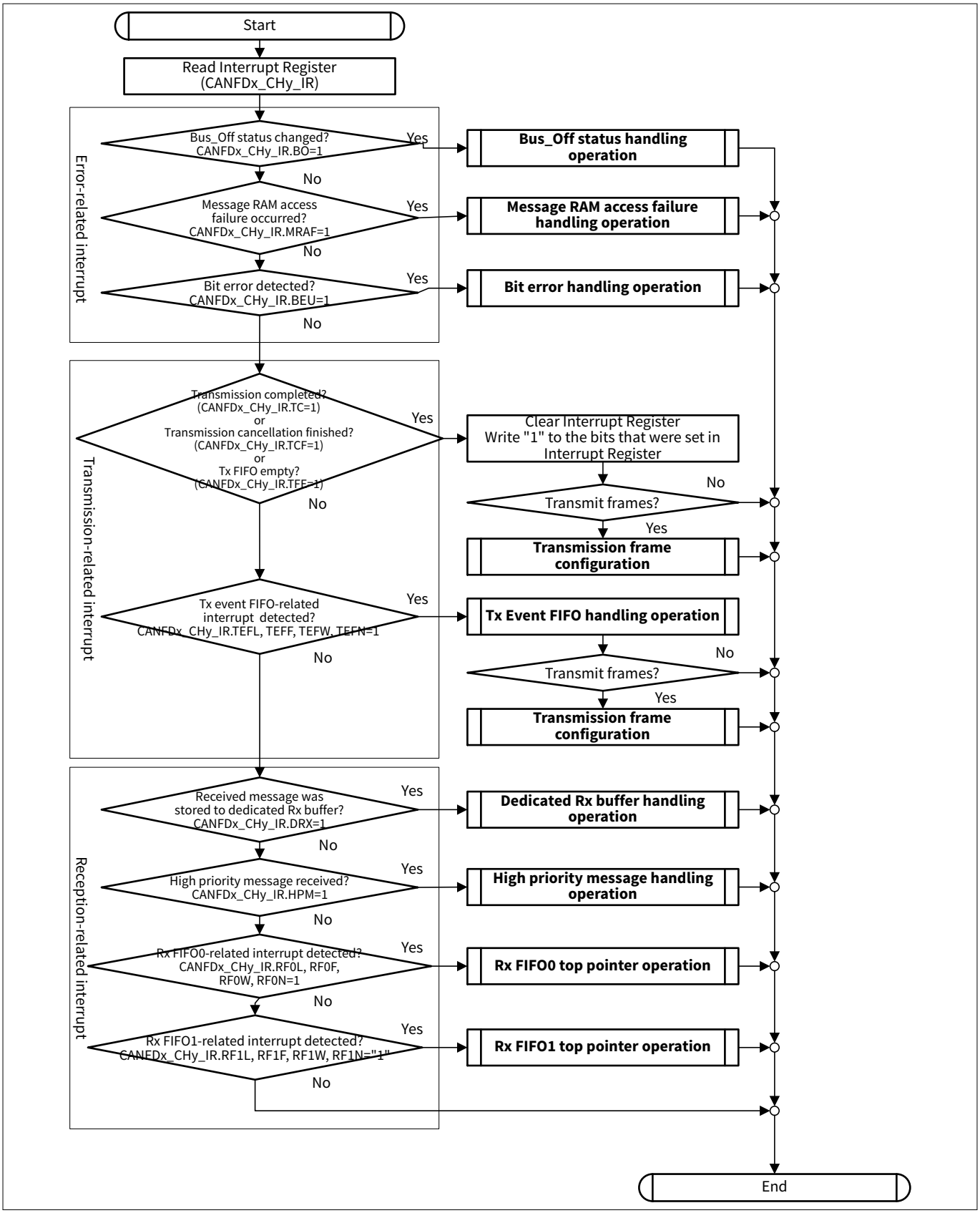


Figure 195 Interrupt handling

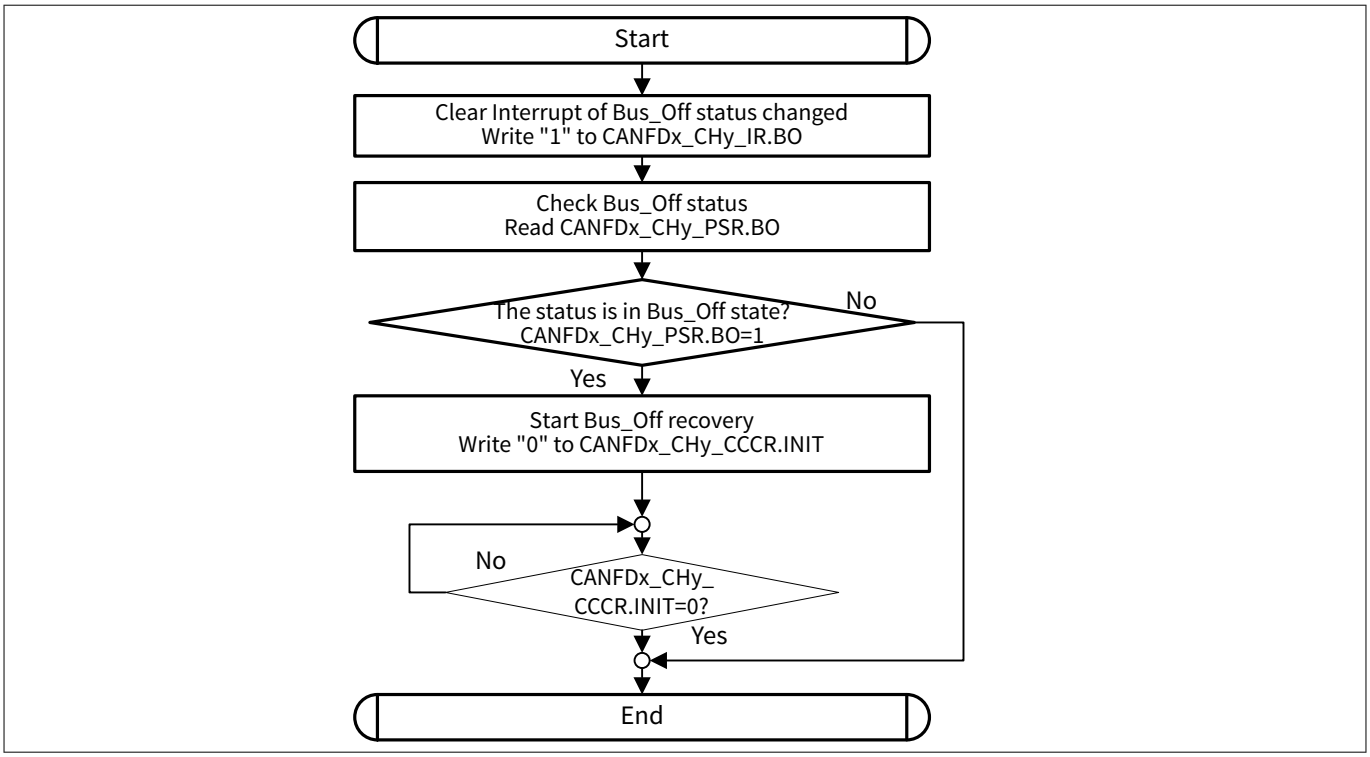


Figure 196 Bus OFF error handling

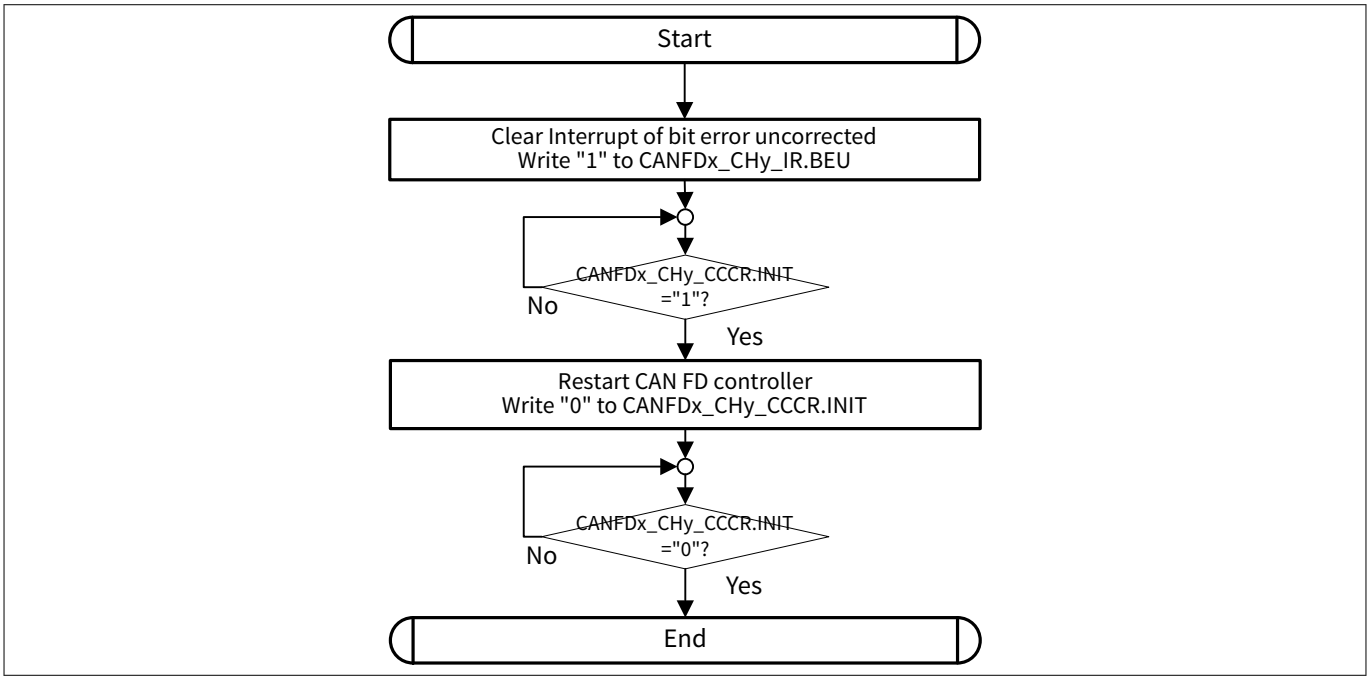


Figure 197 Bit error handling

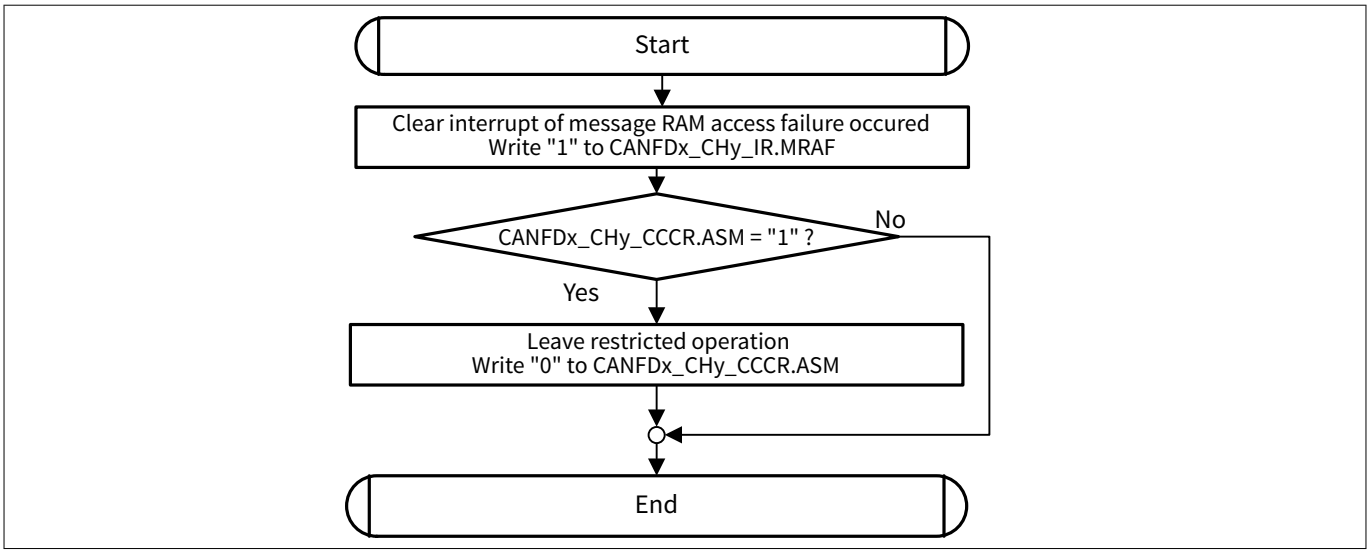


Figure 198 Message RAM access failure handling

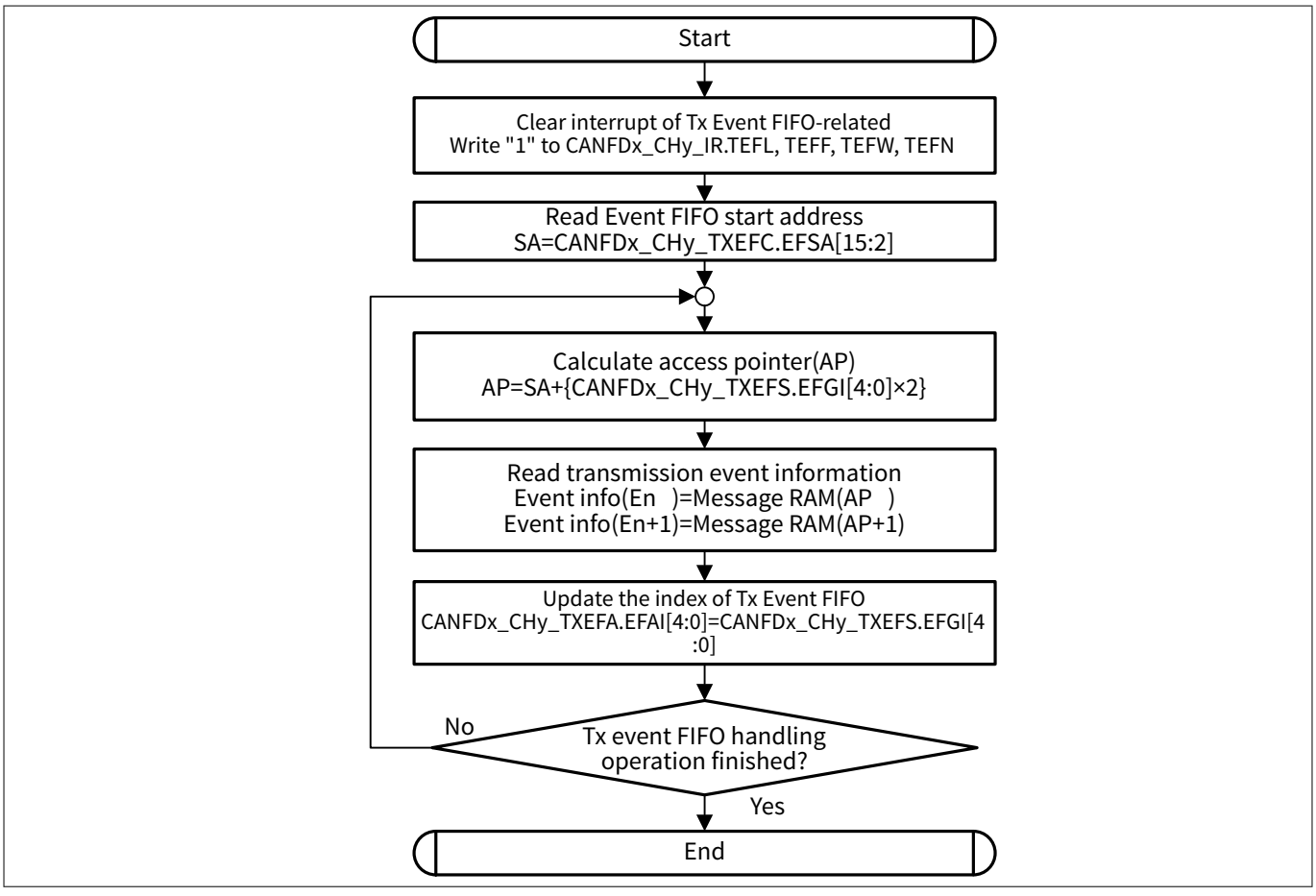


Figure 199 TX event FIFO handling

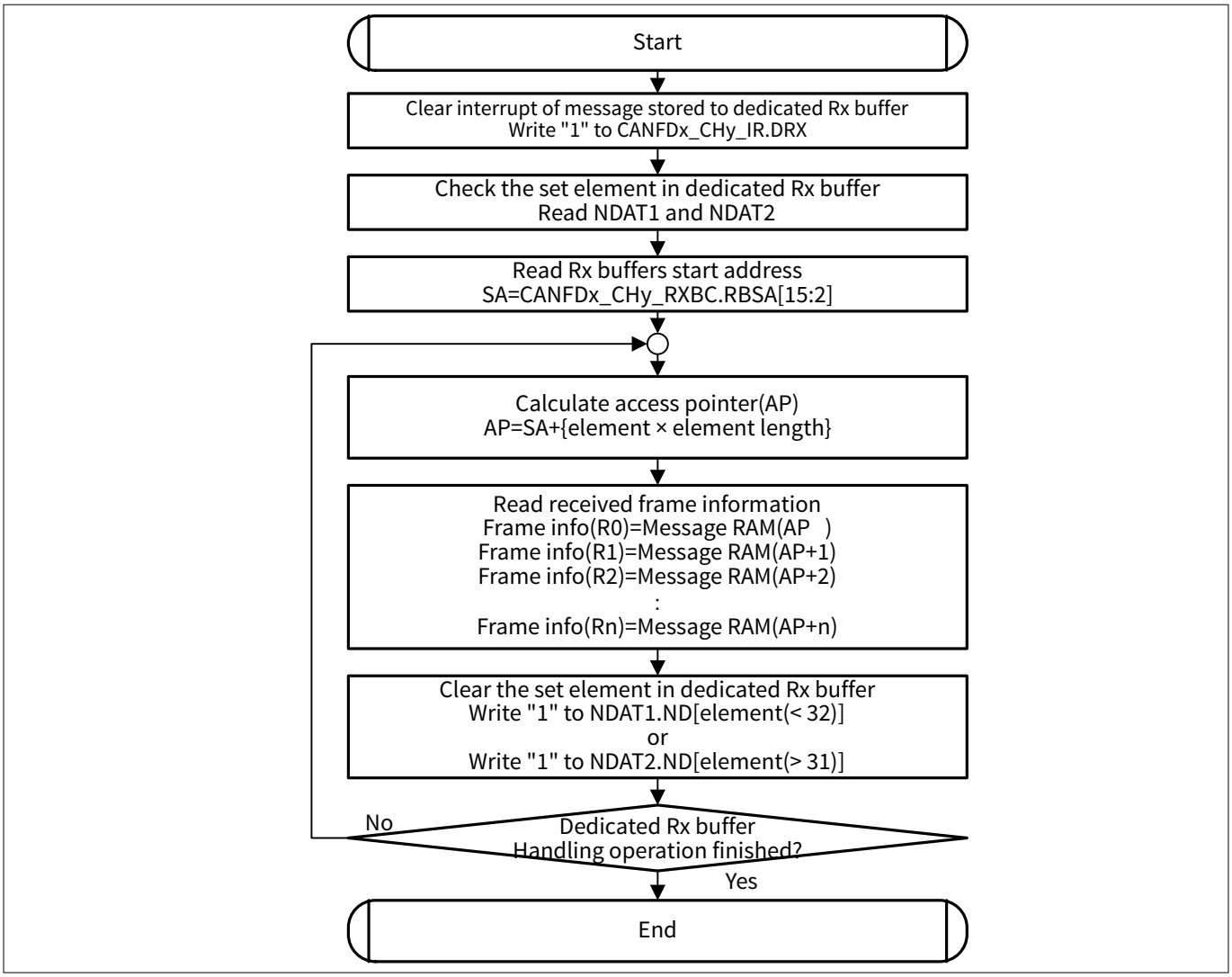


Figure 200 Dedicated RX buffer handling

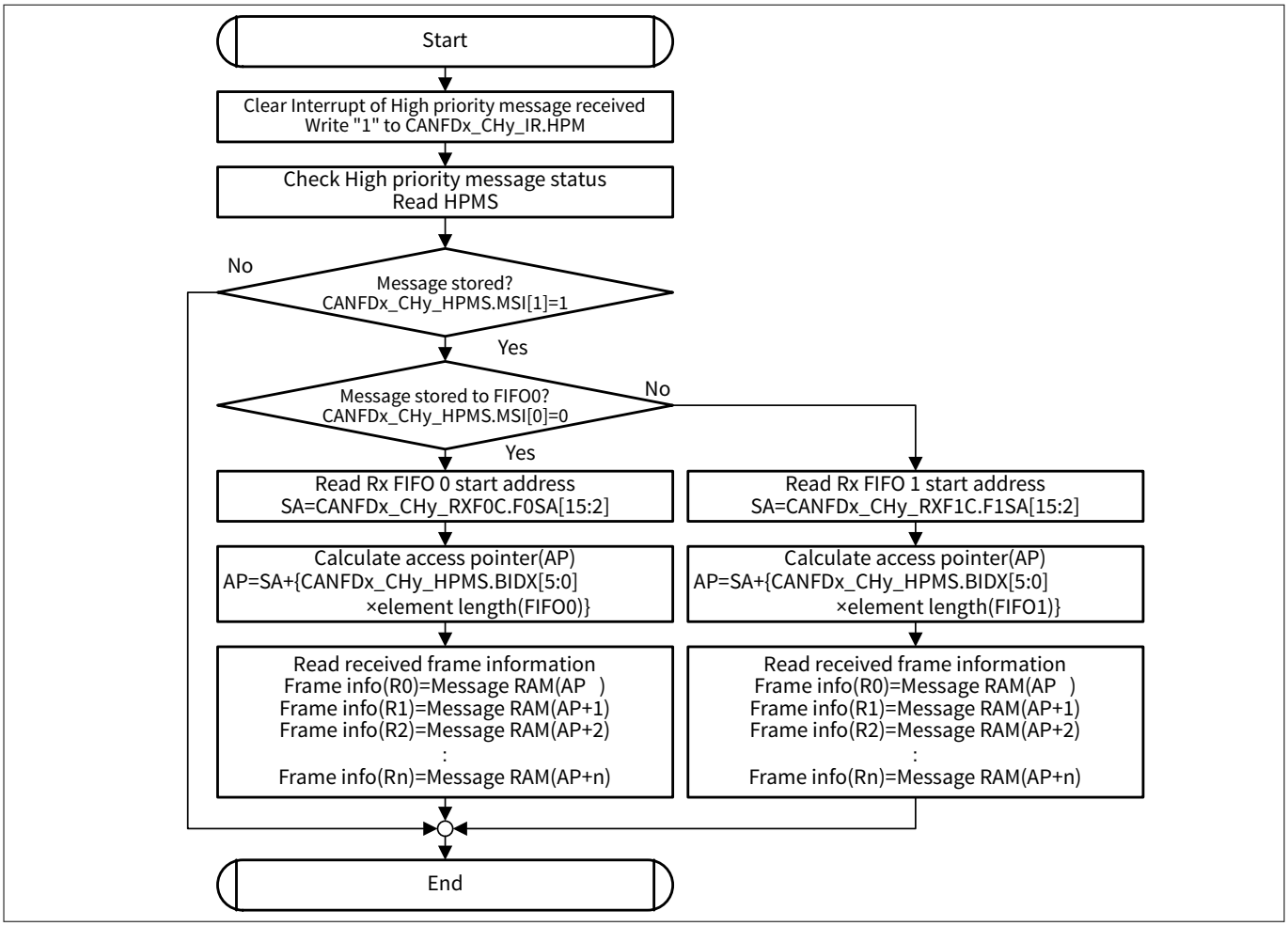


Figure 201 High priority message handling

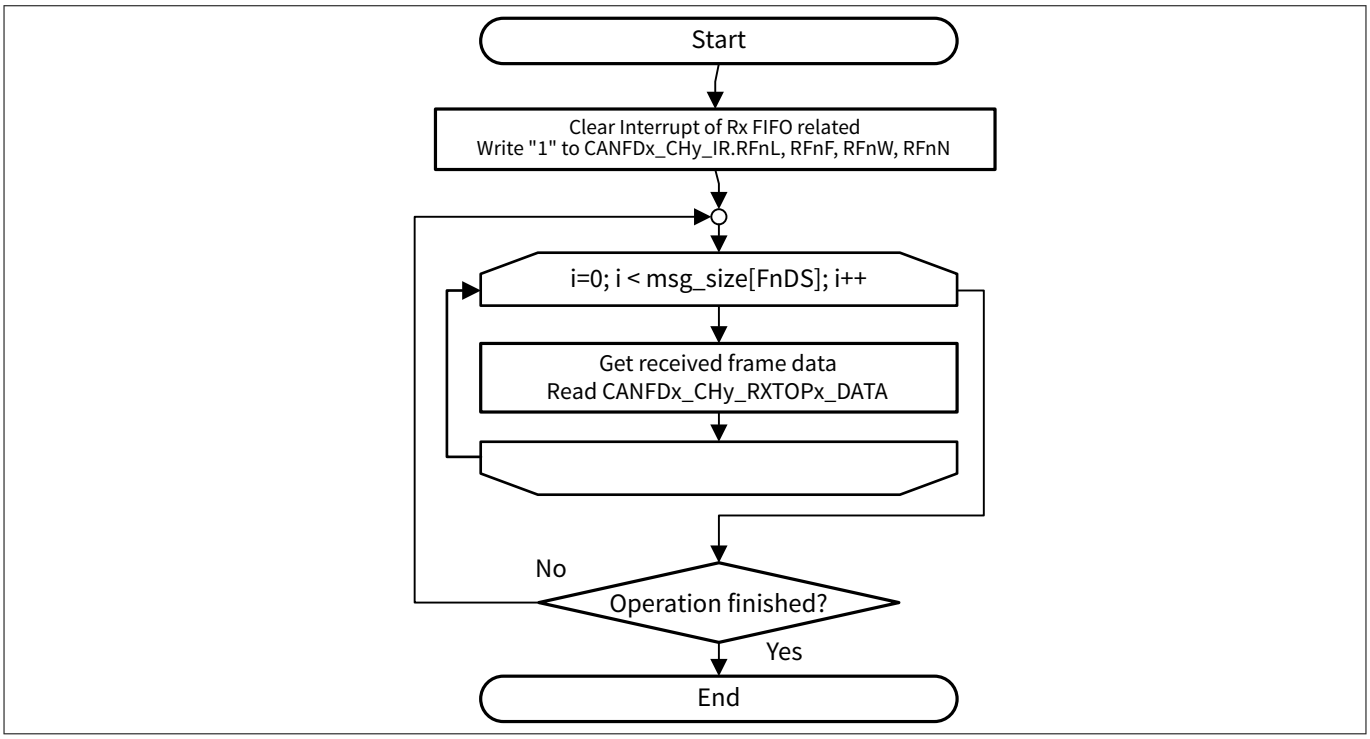


Figure 202 RX FIFO top pointer handling

24.7 Registers

Table 189 Register list

Register	Name	Description
CANFDx_CHy_CREL	Core Release Register	Displays the revision of the CAN FD controller.
CANFDx_CHy_ENDN	Endian Register	Checks the endianness of the CAN FD controller when accessed by the CPU.
CANFDx_CHy_DBTP	Data Bit Timing and Prescaler register	Configures the data bit time and enables Transmitter Delay Compensation.
CANFDx_CHy_TEST	Test register	Monitors the CANx_y_RX/CANx_y_TX pins. It is also used to enable the Loop Back modes.
CANFDx_CHy_RWD	RAM Watchdog	Monitors the message RAM to see if it is ready to be accessed.
CANFDx_CHy_CCCR	CC Control register	Configures various operating modes of the CAN FD controller.
CANFDx_CHy_NBTP	Nominal Bit Timing and Prescaler register	Configures the nominal bit time of the CAN FD controller.
CANFDx_CHy_TSCC	Timestamp Counter Configuration	Holds the settings for timestamp generation.
CANFDx_CHy_TOCC	Timeout Counter Configuration	Holds the settings for the timeout counter.
CANFDx_CHy_TOCV	Timeout Counter Value	Holds the value of the timeout counter.
CANFDx_CHy_ECR	Error counter register	Holds the values of the error counters
CANFDx_CHy_PSR	Protocol Status register	Displays the CAN protocol status of the CAN FD controller.
CANFDx_CHy_TDCR	Transmitter Delay Compensation register	Configures the offset value and the filter window length for transmitter delay compensation
CANFDx_CHy_IR	Interrupt register	Holds the flags that are set when one of the listed conditions is detected (edge-sensitive).
CANFDx_CHy_IE	Interrupt Enable	The settings in this register determine which status changes in the Interrupt Register (IR) will be signaled on an interrupt line
CANFDx_CHy_ILS	Interrupt Line Select	Assigns an interrupt generated by a specific interrupt flag from the Interrupt Register (IR) to one of the two CAN FD controller interrupt lines (canfd_int0/1).

(table continues...)

Table 189 (continued) Register list

Register	Name	Description
CANFDx_CHy_ILE	Interrupt Line Enable	This register can separately enable/ disable each of the two interrupt lines to the CPU.
CANFDx_CHy_GFC	Global Filter Configuration	Global settings for message ID filtering.
CANFDx_CHy_SIDFC	Standard ID Filter Configuration	Settings for 11-bit standard message ID filtering.
CANFDx_CHy_XIDFC	Extended ID Filter Configuration	Settings for 29-bit extended message ID filtering.
CANFDx_CHy_XIDAM	Extended ID and Mask	Defines the valid bits of a 29-bit ID for acceptance filtering.
CANFDx_CHy_HPMS	High Priority Message Status	This register is updated every time a message ID filter element configured to generate a priority event matches.
CANFDx_CHy_NDAT1	New Data 1	Holds flags that are set when the respective dedicated RX buffer receives a frame.
CANFDx_CHy_NDAT2	New Data 2	Holds flags that are set when the respective dedicated RX buffer receives a frame.
CANFDx_CHy_RXF0C	RX FIFO 0 Configuration	Settings for the RX FIFO 0.
CANFDx_CHy_RXF0S	RX FIFO 0 Status	Status of the RX FIFO 0.
CANFDx_CHy_RXF0A	RX FIFO 0 Acknowledge	Acknowledges that the CPU has read a message or a sequence of messages from the RX FIFO 0 to indicate to the CAN FD controller that the corresponding message RAM area may be released.
CANFDx_CHy_RXBC	RX Buffer Configuration	Defines the start address of the RX buffer section in the message RAM.
CANFDx_CHy_RXF1C	RX FIFO 1 Configuration	Settings for the RX FIFO 1.
CANFDx_CHy_RXF1S	RX FIFO 1 Status	Status of the RX FIFO 1.
CANFDx_CHy_RXF1A	RX FIFO 1 Acknowledge	Acknowledges that the CPU has read a message or a sequence of messages from the RX FIFO 1 to indicate to the CAN FD controller that the corresponding message RAM area may be released.
CANFDx_CHy_RXESC	RX buffer/FIFO Element Size Configuration	Configures the number of data bytes belonging to an RX buffer and FIFO element.
CANFDx_CHy_TXBC	TX Buffer Configuration	Settings for TX buffers stored in the message RAM

(table continues...)

Table 189 (continued) Register list

Register	Name	Description
CANFDx_CHy_TXFQS	TX FIFO/Queue Status	Related to the pending TX requests listed in the TX buffer request pending Register (CANFDx_CHy_TXBRP).
CANFDx_CHy_TXESC	TX Buffer Element Size Configuration	Configures the number of data bytes belonging to a TX buffer element
CANFDx_CHy_TXBRP	TX Buffer Request Pending	Holds the status of the transmission requests of each corresponding TX buffer
CANFDx_CHy_TXBAR	TX Buffer Add Request	Requests the transmission of each corresponding TX buffer.
CANFDx_CHy_TXBCR	TX Buffer Cancellation Request	Cancels transmission requests of each corresponding TX buffer.
CANFDx_CHy_TXBTO	TX Buffer Transmission Occurred	Displays whether the corresponding TX buffer is transmitted.
CANFDx_CHy_TXBCF	TX Buffer Cancellation Finished	Signals whether the cancellation request of the corresponding TX buffer is successful.
CANFDx_CHy_TXBTIE	TX Buffer Transmission Interrupt Enable	The settings in this register determine which TX buffer will assert an interrupt upon transmission.
CANFDx_CHy_TXBCIE	TX Buffer Cancellation Finished Interrupt Enable	The settings in this register determine which TX buffer will assert an interrupt upon completion of a transmission cancellation request.
CANFDx_CHy_TXEFC	TX Event FIFO Configuration	Settings for the TX Event FIFO.
CANFDx_CHy_TXEFS	TX Event FIFO Status	Status of the TX Event FIFO.
CANFDx_CHy_TXEFA	TX Event FIFO Acknowledge	Acknowledges that the CPU has read an event from the TX Event FIFO to indicate to the CAN FD controller that the corresponding message RAM area may be released.
CANFDx_CHy_TTTMC	TT Trigger Memory Configuration	Configures memory element and memory start address.
CANFDx_CHy_TTRMC	TT Reference Message Configuration	Configures the reference message such as reference identifier, reference payload type, and type of identifier.
CANFDx_CHy_TTOCF	TT Operation Configuration	Configures fundamentals for time-triggered operations such as TTCAN operation level, time master, and clock calibration.

(table continues...)

Table 189 (continued) Register list

Register	Name	Description
CANFDx_CHy_TTMLM	TT Matrix Limits	Configures cycle counts and synchronization for the clock start, and enables TX Window.
CANFDx_CHy_TURCF	TUR Configuration	Configures numerator and denominator for time unit configuration.
CANFDx_CHy_TTOCN	TT Operation Control	Controls main TTCAN operation.
CANFDx_CHy_TTGTP	TT Global Time Preset	Sets preset value and defines target of cycle time when a rising edge of TTCAN event is expected.
CANFDx_CHy_TTTMK	TT Time Mark	Configures number of cycles in which time mark will be valid.
CANFDx_CHy_TTIR	TT Interrupt Register	Flags in the TTIR is set when particular conditions are met.
CANFDx_CHy_TTIE	TT Interrupt Enable	Provides possibility to enable interrupt for several status changes.
CANFDx_CHy_TTILS	TT Interrupt Line Select	User can select dedicated Interrupt0 or Interrupt1 line for specific interrupt source.
CANFDx_CHy_TTOST	TT Operation Status	Status Register for TT operation.
CANFDx_CHy_TURNA	TUR Numerator Actual	Shows actual numerator value for time unit configuration.
CANFDx_CHy_TTLGT	TT Local and Global time	Shows non-fractional part of the global and local time.
CANFDx_CHy_TTCTC	TT Cycle Time and Count	Read-only register that shows current cycle count and non-fraction part of the cycle time.
CANFDx_CHy_TTCPT	TT Capture Time	Read-only register that shows current cycle count and stop watch value.
CANFDx_CHy_TTCSM	TT Cycle Sync Mark	Read-only register that shows cycle sync mark in terms of cycle time.
CANFDx_CHy_RXFTOP_CTL	Receive FIFO Top Control	Enables receive FIFO top control logic for both FIFOs.
CANFDx_CHy_RXFTOP0_STAT	Receive FIFO 0 Top Status	This is a pointer to the next word in the message buffer defined by FIFO start address.
CANFDx_CHy_RXFTOP0_DATA	Receive FIFO 0 Top Data	Data placed at the address by CANFDx_CHy_RXFTOP0_STAT.
CANFDx_CHy_RXFTOP1_STAT	Receive FIFO 1 Top Status	This is a pointer to the next word in the message buffer defined by FIFO start address.

(table continues...)

Table 189 (continued) Register list

Register	Name	Description
CANFDx_CHy_RXFTOP1_DATA	Receive FIFO 1 Top Data	Data placed at the address by CANFDx_CHy_RXFTOP1_STAT.
CANFDx_CTL	Global CAN Control Register	Provides clock control to the respective TTCAN channels.
CANFDx_STATUS	Global CAN Status Register	Read-only register that shows the acknowledge from the respective TTCAN channel for the clock stop request.
CANFDx_INTR0_CAUSE	Consolidated Int0 Cause Register	Shows pending interrupt0 for each TTCAN channel.
CANFDx_INTR1_CAUSE	Consolidated Int1 Cause Register	Shows pending interrupt1 for each TTCAN channel.
CANFDx_TS_CTL	Time Stamp Control Register	Configuration for the timestamp prescaler and counter enable is done in this register.
CANFDx_TS_CNT	Time Stamp Count Register	Shows timestamp counter value.
CANFDx_ECC_CTL	ECC Control Register	Configures ECC for message RAM.
CANFDx_ECC_ERR_INJ	ECC Error Injection Register	ECC error can be injected to a particular word address in the message RAM using this Register.

***Note**: 'x' in CANFDx signifies the CAN macro instance and 'y' in CANFDx_CHy signifies the channel under the CAN instance.*

25 Timer, counter, and PWM

The timer, counter, and pulse width modulator (TCPWM) block in XMC7000 implements a 16- or 32-bit timer, counter, pulse width modulator (PWM), pseudo random PWM, Shift Register, and quadrature decoder functionality. TCPWM includes up to four counter groups where each group can include up to 256 counters. The counter can be used to measure the period and pulse width of an input signal (timer), find the number of times an event occurs (counter), generate PWM signals, or decode quadrature signals. The TCPWM block works in active and sleep modes.

This chapter explains the features, implementation, and operational modes of the TCPWM block.

25.1 Features

The TCPWM block has the following features:

- Supports up to four counter groups (device specific)
- Each counter group consists up to 256 counters (counter group specific)
- Each counter can run in one of seven function modes:
  - Timer-counter with compare
  - Timer-counter with capture
  - Quadrature decoding
  - PWM with dead time/three-phase motor control (brushless-DC, BLDC)
  - Pseudo-random PWM
  - Shift Register mode
- 16-bit or 32-bit counters (counter group specific)
- Up, down, and up/down counting modes
- Clock prescaling (division by 1, 2, 4, ... 64, 128)
- Up to two capture and compare functions (counter group specific)
- Double buffering of all Compare/Capture and Period Registers
- Two output trigger signals for each counter to indicate underflow, overflow, and capture/compare events; they can also directly be connected with the line output signal
- Supports interrupt on:
  - Terminal count - Depends on the mode; typically occurs on overflow or underflow
  - Capture/compare - The count is captured in the Capture Registers or the counter value equals the value in the compare Register
- Line out selection feature for stepper motor application including two complementary output lines with dead time insertion
  - PWM output can select 0, 1, PWM, Inv PWM, and Hi-Z
- Selectable start, reload, stop, count, and two capture event signals for each TCPWM with rising edge, falling edge, both edges, and level trigger options
- Each counter with up to 254 (device specific) synchronized input trigger signals and two constant input signals: '0' and '1'.
- Two types of input triggers for each counter:
  - General-purpose triggers used by all counters
  - One-to-one triggers for specific counter
- Synchronous operation of multiple counters
- Debug mode support

25.2 Block diagram

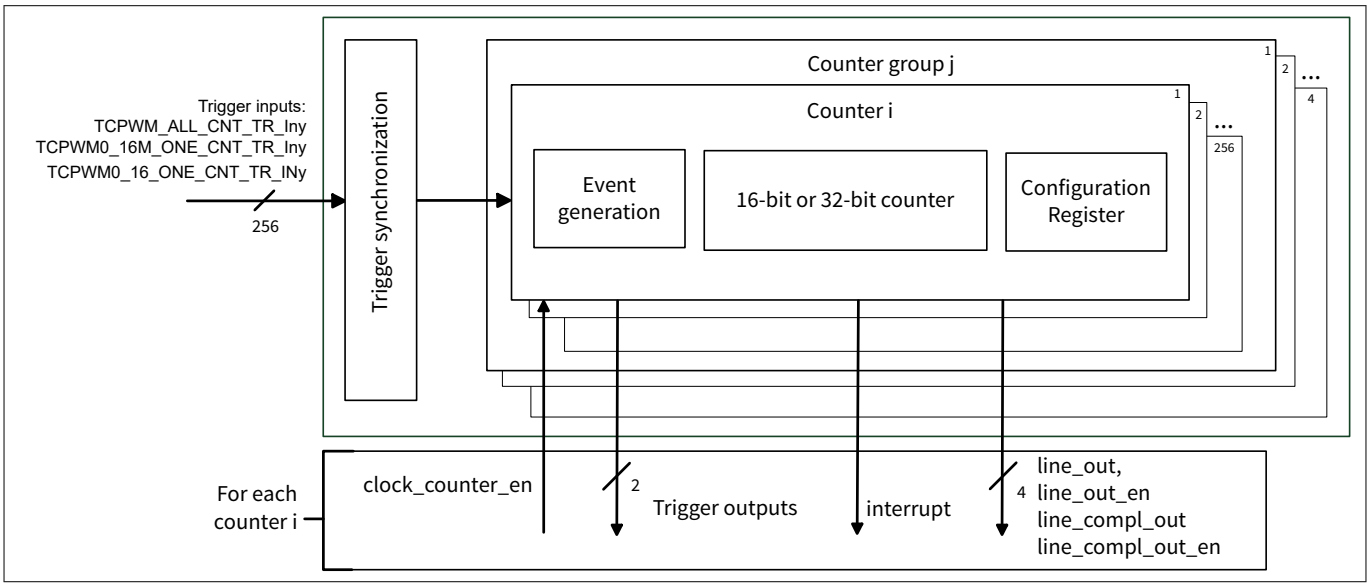


Figure 203 TCPWM block diagram

In the XMC7000 device, there are up to four TCPWM counter groups each supporting up to 256 counters; they can have a counter width of 16-bit or 32-bit. In addition, counter groups can also include a second capture and compare function. Refer to the device datasheet to find dedicated counter group configurations.

***Note**: This document does not discuss the specific counter group configuration in detail. If a second capture/compare feature is mentioned, refer to the device datasheet to know if these functions are available in the particular device.*

All register names and related bit fields are related to one counter example. Find the register prefixes for dedicated counters in the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual.

Each counter can have 254 input trigger signals and two constant input signals, '0' and '1'; all of them are synchronized with CLK_PERI clock.

The TCPWM block has these interfaces:

- Bus interface: Connects the block to the CPU subsystem via AHB-Lite interface.
- I/O signal interface: Consists of input triggers (such as reload, start, stop, count, and capture0/1) and output signals (such as LINE_OUT, LINE_COMPL_OUT, TR_OUT0, and TR_OUT1).
- Interrupts: Provides interrupt request signals from each counter, based on terminal count (TC), Compare/Capture CC0_match, or Compare/Capture CC1_match event.
- System interface: Consists of control signals such as clock and reset from the system resources subsystem (SRSS).

The TCPWM block can be configured by writing to the TCPWM Registers. See Registers for more information on all registers required for this block.

25.2.1 Enabling and disabling counters in TCPWM block

A counter can be enabled by writing '1' to the corresponding ENABLE bit of the CTRL Register; it can be disabled by writing '0' to the same bit.

***Note**: The counter must be configured before enabling it. Disabling the counter retains the values in the registers.*

25.2.2 Clocking

The TCPWM receives a single clock, CLK_PERI. Furthermore, it receives a system clock enable signal clock_sys_en to generate internal CLK_SYS and a counter clock enable signal clock_counter_en for PCLK_TCPWM[x]_CLOCKS[y] of each counter.

Each TCPWM counter can have its own clock source. The only source for the clock is from the configurable peripheral clock dividers generated by the clocking system; see the Clocking system for details. To select a clock divider for a particular counter inside a TCPWM, use the CLOCK_CTL Register from the PERI Register space. In this section the clock to the counter will be called PCLK_TCPWM[x]_CLOCKS[y]. Event generation is performed on the PCLK_TCPWM[x]_CLOCKS[y]. Another clock, CLK_SYS, is used for the pulse width of the output triggers. CLK_SYS is synchronous to CLK_PERI, but can be divided using CLOCK_CTL from the PERI_GROUP_STRUCT Register.

25.2.2.1 Clock prescaling

PCLK_TCPWM[x]_CLOCKS[y] can be further divided inside each counter, with values of 1, 2, 4, 8…64, 128. This division is called prescaling. The prescaling is set in the DT_LINE_OUT_L [7:0] field of the DT Register. The lower three bits of this field determine prescaling of the selected counter clock.

***Note**: Clock prescaling is not available in quadrature mode and pulse width modulation mode with dead time.*

25.2.2.2 Count event

The counter functionality is performed on an “active count” prescaled clock, which is gated by a “count event” signal. For example, a counter increments or decrements by '1' every counter clock cycle in which a count event is detected.

***Note**: Count events are not supported in quadrature and pulse-width modulation pseudo-random modes; the PCLK_TCPWM[x]_CLOCKS[y] is used in these cases instead of the active count prescaled clock.*

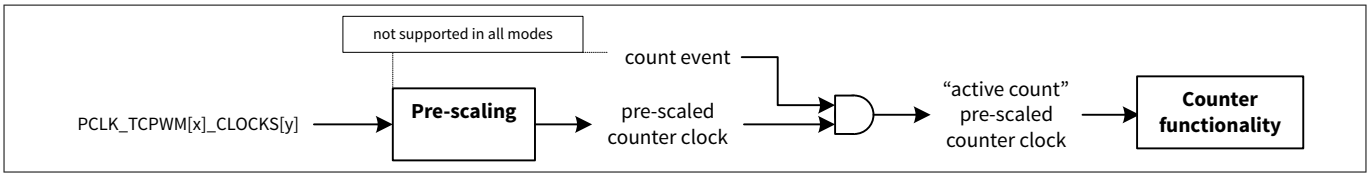


Figure 204 Counter clock generation

All status or output change can only happen at active count prescaled counter clock. In the other words, if a count event is inactive, counter, status, interrupt, and all outputs will not change value. For example, if a count event in pass-through mode becomes low when counter goes to ‘0’ in down count mode, the tc event and underflow event will be generated at the next prescaled counter clock after count event goes high. The only exception is immediate kill mode. Kill input will suppress the PWM output immediately regardless of active count prescaled counter clock.

25.2.3 Trigger inputs

Each TCPWM block has 254 Trigger_In signals and constant '0' and '1' signals, which come from other on-chip resources such as other TCPWMs, SCBs, and DMA. The Trigger_In signals are shared with all counters inside one TCPWM block.

Two types of trigger signals are synchronized and can be used by the counters to generate events.

- General-purpose triggers. These can be used by all counters. These triggers are generated by different blocks in the system and are distributed by the trigger infrastructure (peripheral trigger multiplexers). The

'Trigger multiplexer' section of the device-specific datasheet lists the possible options in which signals can be routed to any general-purpose trigger. E.g.HSIOM_IO _INPUT[0:31] signals can be routed via a specific multiplexer structure to aTCPWM_ALL_CNT_TR_INy signal, which can be used in any TCPWM counter as a trigger input for different trigger events (start, stop, capture, and so on).

- One-to-one triggers. A separate set exists for each counter, only connected to that counter. These triggers are used for direct trigger connections from trigger sources (such as ADC channels) to associated TCPWM counters. The 'Triggers one-to-one' section in the device-specific datasheet lists all the specific one-to-one trigger signals to the TCPWM trigger inputs and related trigger outputs coming from a different module. For example, PASS0_CH_RANGEVIO_TR_OUT[0] is connected directly to the TCPWM0_16M_ONE_CNT_TR_IN[0] trigger input to connect SAR0 ch#0 to TCPWM Group #1Counter #00.

Use the trigger mux registers TR_IN_SEL0 and TR_IN_SEL1 to configure which signals get routed to the Trigger_In for each TCPWM block. See Table 190 for all possible multiplexer settings selecting an input trigger event for a TCPWM block. For each event two constant trigger inputs are available. Input trigger 0 is always constant '0' and input trigger 1 is always constant '1'.

Each counter can select any of the 256 trigger signals to be the source for any of the following events:

- Capture 0 and capture 1
- Count
- Reload
- Stop/kill
- Start

***Note**: The TR_CMD Register can be used to trigger the reload, stop, start, and capture0/1 respectively from software.*

Table 190 Multiplexer selection for input trigger events

Input trigger selection Register	Bit field	Bits	Description
TR_IN_SEL0	CAPTURE0_SEL	7:0	Selects one of the up to 256 input triggers as a capture0 trigger. In the PWM, PWM_DT, and PWM_PR modes this trigger is used to switch the values if the Compare and Period Registers with their buffer counterparts.
	COUNT_SEL	15:8	Selects one of the 256 input triggers as a count trigger. In Quad mode, this is the first phase (phi A)
	RELOAD_SEL	23:16	Selects one of the 256 input triggers as a reload trigger. In Quad mode, this is the index or revolution pulse
	STOP_SEL	31:24	Selects one of the 256 input triggers as a stop trigger. In PWM, PWM_DT, and PWM_PR modes, this is the kill trigger
TR_IN_SEL1	START_SEL	7:0	Selects one of the 256 input triggers as a start trigger. In Quad mode, this is the second phase (phi B)
	CAPTURE1_SEL	15:8	Selects one of the up to 256 input triggers as a capture1 trigger

The following sections describe each TCPWM mode and the function of each input event in detail.

Typical operation uses the reload event once to initialize and start the counter and the stop event to stop the counter. When the counter is stopped, the start event can be used to start the counter with its counter value unmodified from when it was stopped.

If stop, reload, and start events coincide, the following precedence relationship holds:

- A stop event has higher priority than a reload event.
- A reload event has higher priority that a start event.

As a result, when a reload or start event coincides with a stop event, the reload or start event has no effect.

Before going to the counter each Trigger_IN can pass through a positive edge detector, negative edge detector, both edge detector, or pass straight through to the counter. This is controlled using TR_IN_EDGE_SEL Register.

Multiple detected events are treated as follows:

- In the rising edge and falling edge modes, multiple events are effectively reduced to a single event. As a result, events may be lost.
- In the rising/falling edge mode, an even number of events are not detected and an odd number of events are reduced to a single event. This is because the rising/falling edge mode is typically used for capture events to determine the width of a pulse. The current functionality will ensure that the alternating pattern of rising and falling is maintained.

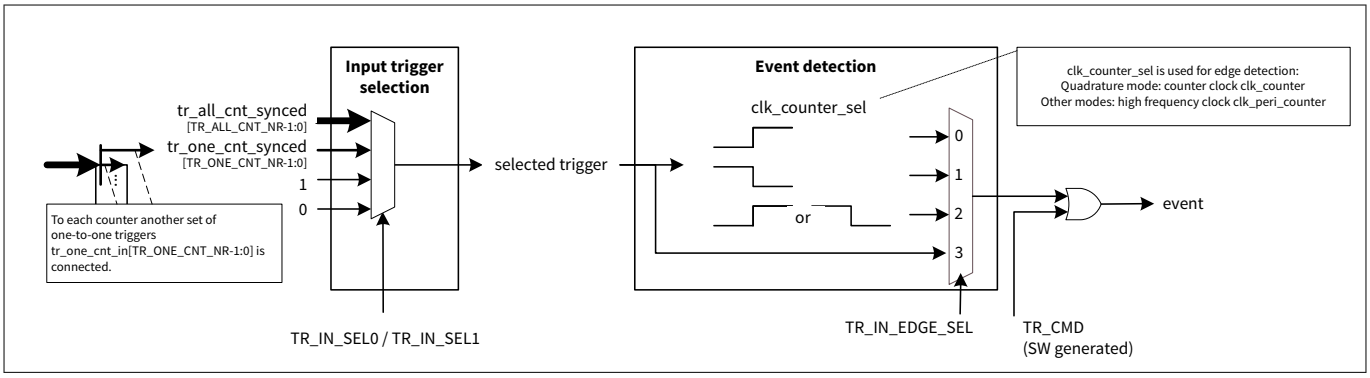


Figure 205 TCPWM input events

According to Table 190, a dedicated input trigger signal for trigger event generation can be defined by the TR_IN_SEL0 and TR_IN_SEL1 Registers. The selection can be done between two constant signals (constant '0' and constant '1'), specific one-to-one trigger input signals, or general-purpose input trigger signals. Figure 205 shows how the input trigger source is selected.

The number of one-to-one (tr_one_cnt_synced) and general-purpose (tr_all_cnt_synced) input triggers are device specific, but the following assignment order is used for the input trigger selection multiplexer:

- Constant '0' (fix TR_IN_SEL value = 0)
- Constant '1' (fix TR_IN_SEL value = 1)
- Specific one-to-one input triggers (TR_IN_SEL value = 2 to (TR_ONE_CNT_NR + 1))
- General-purpose input triggers (TR_IN_SEL value = (TR_ONE_CNT_NR + 2) to (TR_ALL_CNT_NR) + (TR_ONE_CNT_NR + 1))

While the general-purpose input triggers are connected to all counters, the specific one-to-one input triggers are assigned to dedicated counters. There is a large number of port pin input signals (tr_one_cnt_in[x]) used for specific one-to-one triggers. The related mapping of input triggers to GPIO pins is available in the datasheet.

The routing to the multiplexer inputs is calculated using the following relationship:

- Constants (valid for all counters)
  - trigger[0] is constant “0”
  - trigger[1] is constant “1”
- Specific one-to-one input triggers:

Each counter group has 256 × TR_ONE_CNT_NR bits tr_one_cnt_in[] input, and each counter has TR_ONE_CNT_NR bits tr_one_cnt_in[] input as triggers. The mapping is done as follows:

- group[A].counter[B].trigger[TR_ONE_CNT_NR+1:2] is connected to:
- tr_one_cnt_in[256 × A × TR_ONE_CNT_NR + (B+1) × TR_ONE_CNT_NR - 1: 256 × A × TR_ONE_CNT_NR+B × TR_ONE_CNT_NR]

As an example: TR_ONE_CNT_NR = 3 (this value is also valid for first XMC7000 device)

- group[A].counter[B].trigger[4:2]
  is connected to:
- tr_one_cnt_in[256*A*2 + (B+1)*2 - 1 : 256*A*2+B*2]
  tr_one_cnt_in[0]' group[0] counter[0] trigger[2]
  tr_one_cnt_in[1]' group[0] counter[0] trigger[3]
  tr_one_cnt_in[2]' group[0] counter[0] trigger[4]
  tr_one_cnt_in[3]' group[0] counter[1] trigger[2]
  tr_one_cnt_in[4]' group[0] counter[1] trigger[3]
  tr_one_cnt_in[5]' group[0] counter[1] trigger[4]
  tr_one_cnt_in[6]' group[0] counter[2] trigger[2]
  tr_one_cnt_in[7]' group[0] counter[2] trigger[3]
  tr_one_cnt_in[8] ' group[0] counter[2] trigger[4]
  tr_one_cnt_in[9]' group[0] counter[3] trigger[2]
  tr_one_cnt_in[10]' group[0] counter[3] trigger[3]
  tr_one_cnt_in[11]' group[0] counter[3] trigger[4]
  ...
  tr_one_cnt_in[512] ' group[1] counter[0] trigger[2]
  tr_one_cnt_in[513] ' group[1] counter[0] trigger[3]
  tr_one_cnt_in[514] ' group[1] counter[0] trigger[4]
  tr_one_cnt_in[515] ' group[1] counter[1] trigger[2]
  tr_one_cnt_in[516] ' group[1] counter[1] trigger[3]
  tr_one_cnt_in[517] ' group[1] counter[1] trigger[4]

- General-purpose triggers (valid for all counters): General-purpose input triggers are connected to all counters, each on the same trigger position. The mapping is done as follows:
  - group[A].counter[B].trigger[TR_ONE_CNT_NR+1+ TR_ALL_CNT_NR:TR_ONE_CNT_NR+2]
    is connected to:
  - tr_all_cnt_in[TR_ALL_CNT_NR : 0]
    As an example: TR_ONE_CNT_NR = 2, TR_ALL_CNT_NR = 4
  - group[A].counter[B].trigger[7:4]
    tr_all_cnt_in[0]' group[A] counter[B] trigger[4]
    tr_all_cnt_in[1] ' group[A] counter[B] trigger[5]
    tr_all_cnt_in[2] ' group[A] counter[B] trigger[6]
    tr_all_cnt_in[3] ' group[A] counter[B] trigger[7]

***Note**:*
- *A: Number of counter group*
- *B: Number of counter*
- *TR_ONE_CNT_NR: Number of input triggers per counter only routed to one counter.*
- *TR_ALL_CNT_NR: Number of input triggers per counter routed to all counter.*

Table 191 shows how the multiplexer should be handled for the input trigger event generation. The XMC7000 supports the following input triggers:

- Number of specific one-to-one trigger inputs: 3
- Number of general-purpose trigger inputs: 27

Table 191 Handling input trigger multiplexers

Input trigger selection	Input trigger	Input trigger source
0	constant ‘0’	constant '0'
1	constant ‘1’	constant '1'
2	HSIOM column ACT#2	Refer to the “Alternate pin function” section in the device datasheet
3	HSIOM column ACT#3	Refer to the “Alternate pin function” section in the device datasheet
4	PASS (programmable analog subsystem), through 1:1 trigger mux #0,	Refer to the product sheet tab “triggersOnetoOne”. Not all counters will have this input trigger.
5	tr_all_cnt_in[0]	Refer to the trigger mux block.
...		
31	tr_all_cnt_in[26]	Refer to the trigger mux block.

***Note**: The input triggers can be generated by different sources. While the general-purpose trigger inputs (tr_all_cnt_in[0] to tr_all_cnt_in[26])] are only from the trigger multiplexer block (see the Trigger multiplexer), the one-to-one input triggers can also be generated by external GPIO input pins.*

All trigger inputs are synchronized to PERI_CLK. 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 prescaled (divided) counter_clock is used.

The following figure illustrates the timing on how input triggers are detected by counter.

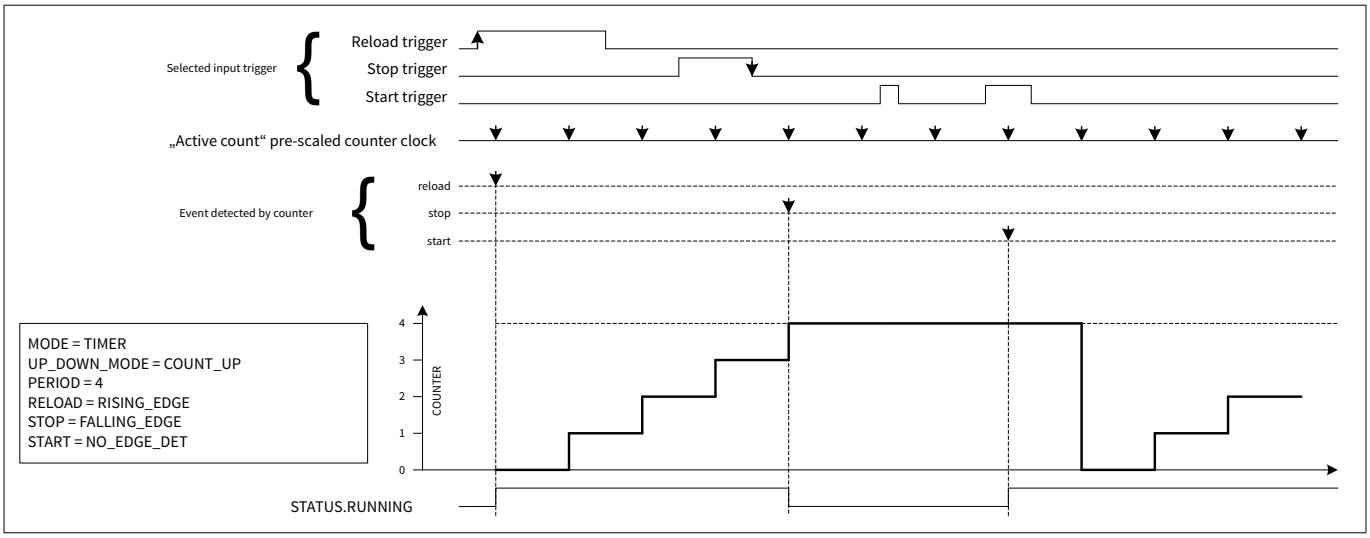


Figure 206 Input trigger detection by “active count” prescaled counter clock

***Note**: The arrows in the figure depict the events that are detected by the counter.*

Two examples explain how edge detection event works on active count prescaled counter clocks:

- In PWM mode, if PWM_IMM_KILL = 0, the rising edge kill asserts while count event is inactive, line output will not be suppressed until the next prescaled counter clock after the count event becomes active.
- In capture mode, if rising edge capture0 inputs while count event is inactive, CC0 and CC0_BUFF will get updated at the next prescaled counter clock after count event becomes active.

Typically, the count event is a constant '1' and prescaling is off. In this case, the active count prescaled counter clock is the same as the counter clock. In other cases, edge detection may detect multiple events (on the counter clock) before the next active count prescaled counter clock on which the detected event is used. Multiple detected events are treated as follows:

- In the rising edge and falling edge modes, multiple events are effectively reduced to a single event. As a result, events may be lost.
- In the rising/falling edge mode, an even number of events is not detected and an odd number of events is reduced to a single event. This is because the rising/falling edge mode is typically used for capture events to determine the width of a pulse. The current functionality will ensure that the alternating pattern of rising, falling, rising, falling, and so on is maintained.

A pass-through event will not be remembered by CLK_PERI; it will affect the functionality if it lasts and can be detected by a counter operation clock. If the pulse width of a pass-through event is less than a counter operation clock cycle, it may get lost. Pass-through detection may result in an event that is active for multiple counter clocks. This may result in undesirable behavior of the counter and its associated trigger outputs. Pass-through event detection should only be used for stop and count event types in most function modes. Pass-through mode can also be used in switch events in the PWM/PWM_DT/PWM_PR mode, if it selects the constant high as the source. In quadrature mode, both start and count event is used with pass through in X1/X2/X4 mode.

25.2.4 Synchronization of multiple counters

The previous sections described hardware-based event generation. In addition, software-based event generation is supported: the reload, start, stop, capture0, and capture1 events can be generated by writing to the TR_CMD Registers. These are counter specific registers and allow software-based event generation only for a single counter.

Synchronized software-based event generation (such as starting multiple counters synchronously) is possible by selecting the same trigger signal in all desired counters (via TR_IN_SEL0 and TR_IN_SEL1 Registers) and generating a trigger by the TR_CMD Register in the PERI block.

The following figure illustrates an example of how to synchronously start multiple counters with TR_CMD of the PERI block.

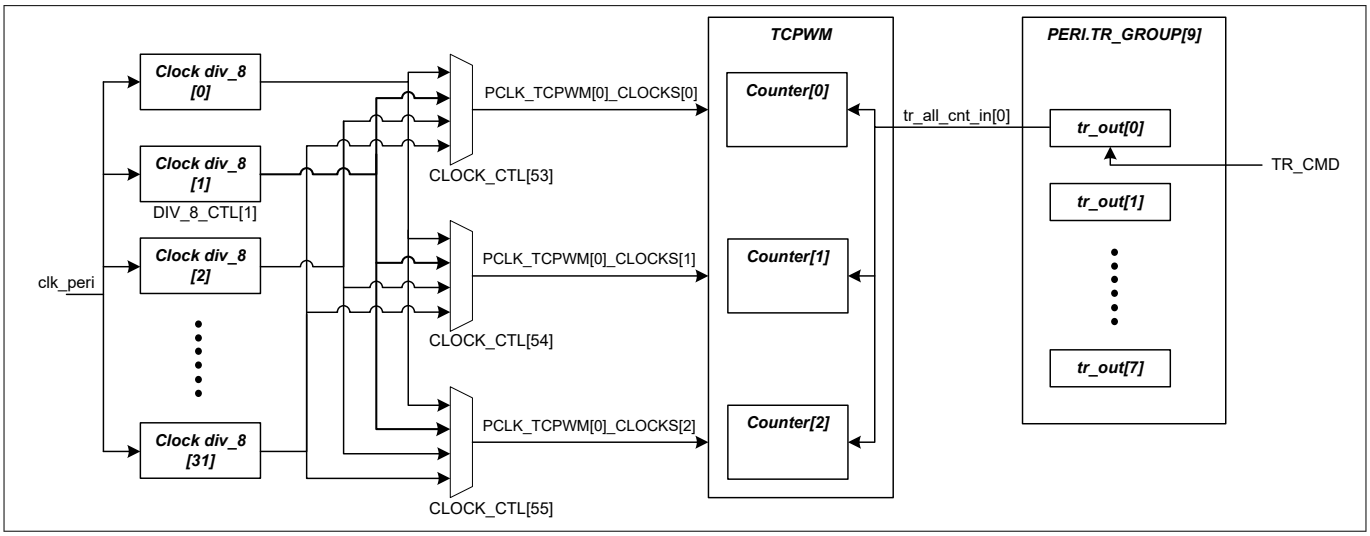


Figure 207 PERI TR_CMD synchronously starts counters

The following example describes the required steps to start counters synchronously:

- Configure CLOCK_CTL[55]/[54]/[53] to select same clock divider clock_div_8[1].
- Configure DIV_8_CTL[1] and DIV_CMD to generate clock divider enable signal for TCPWM. Hence, three counters will have the same clock divider enable signal; in other words, their clocks are synchronous.
- Enable the three counters one by one - counter is enabled but will not run until start or reload event is detected.
- Configure the three counters' start event, all selecting tr_all_cnt_in[0]; this means the three counters will run synchronously when tr_all_cnt_in[0] asserts.
- Use TR_CMD to generate trigger pulse on TR_GROUP[9].TR_OUT[0]. All three counters will run synchronously.

***Note**: All registers listed here belongs to the PERI block (see the Trigger multiplexer).*

- A software-based event is set after writing TR_CMD respective bit to '1', and cleared by hardware on the next "active count" prescaled counter clock. For some events in a specific mode, it is cleared on the next "active count" prescaled counter clocks that have a tc event. The event detection setting (TR_IN_EDGE_SEL) does not have an effect on a software-based event.

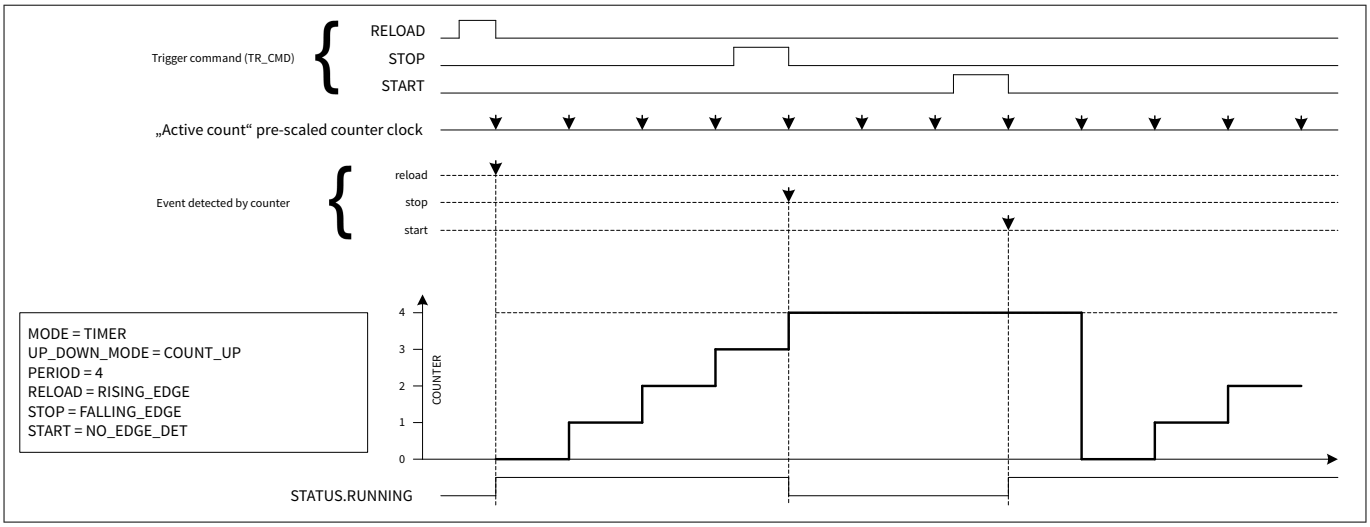


Figure 208 Software trigger command detection by active count prescaled counter clock

25.2.5 Trigger outputs

Each counter has two trigger output signals (TR_OUT0 and TR_OUT1) to indicate following events. They can be routed through the trigger mux to other peripherals on the device. The bit field OUT0 in TR_OUT_SEL Register is used to select one of the internal events to generate output trigger 0 (TR_OUT0), respectively the bit field OUT1 is selecting one of the internal events to generate TR_OUT1. It allows also to disable the output triggers.

- Overflow (OV): An overflow event indicates that in up counting mode, COUNTER equals PERIOD Register, and is changed to a different value.
- Underflow (UN): An underflow event indicates that in a down counting mode, COUNTER equals 0, and is changed to a different value.
- Terminal count (TC): A TC event is the logical OR of the underflow and overflow events
- CC0/1_MATCH: This event is generated when the counter is running and one of the following conditions occur:
  - Counter equals the compare value. This event is either generated when the match is about to occur (COUNTER does not equal CC0/1 and is changed to CC0/1) or when the match is about to not occur (COUNTER equals CC0/1 and is changed to a different value).
  - A capture event has occurred and the CC0 (CC1) and CC0_BUFF (CC1_BUFF) Registers are updated.
- LINE_OUT: A PWM output signal
- DISABLED: Output trigger is disabled

The selection of the events for the output trigger generation is done by the TR_OUT_SEL Register. It also allows disabling the output triggers.

***Note**: These signals only remain high for two cycles of CLK_SYS. For reliable operation, the condition that causes this trigger should be a maximum of one quarter of the CLK_SYS. For example, if the CLK_SYS is running at 24 MHz, the condition causing the trigger should occur at a frequency equal to or less than 6 MHz.*

When LINE_OUT is selected for output triggers, output trigger will bypass two cycle pulses generation logic and directly output LINE_OUT.

The generated triggers have two main uses:

- Initiating a DW/DMA data transfer. For example, in PWM mode with an up counting timer, the overflow can be used to transfer new period and compare values from memory to the counters' PERIOD_BUFF and CC0_BUFF Registers.
- Reconstruction of a PWM signal in a programmable digital component. As documented in Pulse width modulation (PWM) mode, the PWM line output signal is derived from the cc0_match (cc1_match), underflow, and overflow internal events. By making these internal events available as output triggers, other components can reconstruct and potentially modify the PWM signal (note the mentioned frequency restrictions).

25.2.6 Internal events

25.2.6.1 Underflow event

An underflow event indicates that in down counting, COUNTER equals zero, and is changed to a different value. Reload will also generate underflow event in some specific mode. Table 192 summarizes the underflow generation of each function mode.

Table 192 Underflow generation

MODE	UP	DOWN	UPDN1	UPDN2
TIMER	Counter is decrementing and changes from a state in which COUNTER equals ‘0’. Reload event in DOWN, UPDN1, and UPDN2 modes.			
CAPTURE	Counter is decrementing and changes from a state in which COUNTER equals ‘0’. Reload event in DOWN, UPDN1, and UPDN2 modes.			
QUAD	QUAD_RANGE0: Not used QUAD_RANGE0_CMP: Not used QUAD_RANGE1_CMP: Counter value COUNTER equals ‘0’ and is decrementing. QUAD_RANGE1_CAPT: Counter value COUNTER equals ‘0’ and is decrementing.			
PWM	Counter is decrementing and changes from a state in which COUNTER equals ‘0’. Reload event in DOWN, UPDN1, and UPDN2 modes.			
PWM_DT	Counter is decrementing and changes from a state in which COUNTER equals ‘0’ Reload event in DOWN, UPDN1, and UPDN2 modes.			
PWM_PR	Not used			
SR	Not used			

25.2.6.2 Overflow event

An overflow event indicates that in up counting, COUNTER equals PERIOD, and is changed to a different value. Reload will also generate overflow event in some specific mode. Table 193 summarizes the overflow generation of each function mode.

Table 193 Overflow generation

MODE	UP	DOWN	UPDN1	UPDN2
TIMER	Counter is incrementing and changes from a state in which COUNTER equals 'PERIOD'. Reload event in UP count mode.			
CAPTURE	Counter is incrementing and changes from a state in which COUNTER equals 'PERIOD'. Reload event in UP count mode.			
QUAD	QUAD_RANGE0: Not used QUAD_RANGE0_CMP: Not used QUAD_RANGE1_CMP: Counter value COUNTER equals 'PERIOD' and is incrementing. QUAD_RANGE1_CAPT: Counter value COUNTER equals 'PERIOD' and is incrementing.			
PWM	Counter is incrementing and changes from a state in which COUNTER equals 'PERIOD'. Reload event in UP count mode.			
PWM_DT	Counter is incrementing and changes from a state in which COUNTER equals 'PERIOD'. Reload event in UP count mode.			
PWM_PR	Not used			
SR	Not used			

25.2.6.3 TC event

A tc (terminal count) event is the logical OR of the underflow and overflow events. An exception is that reload event will generate an underflow or overflow, but not a tc event. In quadrature mode, index will generate a tc event. Table 194 summarizes the tc generation of each function mode.

Table 194 TC generation

MODE	UP	DOWN	UPDN1	UPDN2
TIMER	Overflow	Underflow	Underflow	Logic OR of overflow and underflow
CAPTURE	Overflow	Underflow	Underflow	Logic OR of overflow and underflow
QUAD	QUAD_RANGE0: • Index event QUAD_RANGE0_CMP: • Counter value COUNTER equals 0 or 0xFFFF/0xFFFFFFFF in "wraparound capture" mode. • Index or capture0 event in "index capture" mode. QUAD_RANGE1_CMP: • Counter value COUNTER equals 0 and decrementing (underflow), or PERIOD and incrementing (overflow). • Index event QUAD_RANGE1_CAPT: • Same as QUAD_RANGE1_CMP			

(table continues...)

Table 194 (continued) TC generation

MODE	UP	DOWN	UPDN1	UPDN2
PWM	Overflow	Underflow	Underflow	Logic OR of overflow and underflow
PWM_DT	Overflow	Underflow	Underflow	Logic OR of overflow and underflow
PWM_PR	Counter changes from a state in which COUNTER equals PERIOD.			
SR	Not used			

25.2.6.4 cc0_match (cc1_match) event

A cc0_match event indicates that the COUNTER equals CC0. This event is either generated when COUNTER is about to change to CC0, or when COUNTER equals CC0 and is about to change to a different value. A special case is for 0 or 100 percent duty cycle generation in PWM mode; for more details, see the Pulse width modulation (PWM) mode. In other specific operation modes, the event is used to indicate that the CC0/ CC0_BUFF Registers are updated. cc1_match is generated per state of COUNTER and CC1, other behavior is same as cc0_match. Table 195 and Table 196 summarize the cc0/1_match generation of each function mode. Compare match events can be enabled/disabled independently by the CC0/1_MATCH_UP_EN and CC0/1_MATCH_DOWN_EN bits in the CTRL register while the counter is up or down counting, in the COUNT_UPDN1/2 counting mode of PWM/PWMDT for 'Advanced motor' Control counter groups which support CC1 (second Capture / Compare unit) feature.

Table 195 cc0_match generation

MODE	UP	DOWN	UPDN1	UPDN2
Timer	Counter changes from a state in which COUNTER equals ‘CC0’.			
CAPTURE	Capture0 event			
QUAD	QUAD_RANGE0: • Counter value COUNTER equals 0 or 0xFFFF • Index event QUAD_RANGE0_CMP: • Counter changes to a state in which COUNTER equals ‘CC0’. QUAD_RANGE1_CMP: • Same as QUAD_RANGE0_CMP QUAD_RANGE1_CAPT: • Capture0 event			
PWM	Counter changes to a state in which COUNTER equals ‘CC0’.		COUNT_UPDN1/2: counter changes from a state in which COUNTER equals ‘CC0’. If a second compare function is present in a counter group, CC0_MATCH_DOWN_EN/CC0_MATCH_UP_EN will enable/disable cc0_match generation.	

(table continues...)

Table 195 (continued) cc0_match generation

MODE	UP	DOWN	UPDN1	UPDN2
PWM_DT	Counter changes to a state in which COUNTER equals ‘CC0’.		COUNT_UPDN1/2: counter changes from a state in which COUNTER equals ‘CC0’. If a second compare function is present in a counter group, CC0_MATCH_DOWN_EN/CC0_MATCH_UP_EN will enable/disable cc0_match generation.	
PWM_PR	Counter changes from a state in which COUNTER equals ‘CC0’.			
SR	Counter changes to a state in which COUNTER equals ‘CC0’.			

Table 196 cc1_match generation

MODE	UP	DOWN	UPDN1	UPDN2
Timer	Counter changes from a state in which COUNTER equals ‘CC1’.			
CAPTURE	Capture1 event			
QUAD	QUAD_RANGE0: • Not used QUAD_RANGE0_CMP: • Counter changes to a state in which COUNTER equals ‘CC1’. QUAD_RANGE1_CMP: • Same as QUAD_RANGE0_CMP QUAD_RANGE1_CAPT: • Capture1 event			
PWM	Counter changes to a state in which COUNTER equals ‘CC1’.		COUNT_UPDN1/2: counter changes from a state in which COUNTER equals ‘CC1’. If a second compare function is present in a counter group, CC1_MATCH_DOWN_EN/CC1_MATCH_UP_EN will enable/disable cc1_match generation.	
PWM_DT	Counter changes to a state in which COUNTER equals ‘CC1’.		COUNT_UPDN1/2: counter changes from a state in which COUNTER equals ‘CC1’. If a second compare function is present in a counter group, CC1_MATCH_DOWN_EN/CC1_MATCH_UP_EN will enable/disable cc1_match generation.	
PWM_PR	Counter changes from a state in which COUNTER equals ‘CC1’.			
SR	Counter changes to a state in which COUNTER equals ‘CC1’.			

25.2.7 Interrupts

The TCPWM block provides a dedicated interrupt output for each counter. Interrupts are counter mode specific and can be generated for a terminal count (TC) or compare/capture0/1 (CC0/1) event. A TC is the logical OR of the OV and UN events.

Four registers are used to handle interrupts in this block, as shown in Table 197.

Table 197 Interrupt Register

Interrupt Registers	Bits	Name	Description
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	CC0_MATCH	This bit is set to '1' when the counter value matches capture/ compare0 (CC0) register value. Write '1' to clear this bit.
	2	CC1_MATCH	This bit is set to '1' when the counter value matches capture/ compare1 (CC1) register value. Write '1' to clear this bit.
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	CC0_MATCH	Write '1' to set the corresponding bit in the Interrupt Request Register. When read, this register reflects the interrupt request register status.
	2	CC1_MATCH	Write '1' to set the corresponding bit in the Interrupt Request Register. When read, this register reflects the interrupt request register status.
INTR_MASK (Interrupt Mask Register)	0	TC	Mask bit for the corresponding TC bit in the Interrupt Request Register.
	1	CC0_MATCH	Mask bit for the corresponding CC_MATCH0 bit in the Interrupt Request Register.
	2	CC1_MATCH	Mask bit for the corresponding CC_MATCH1 bit in the Interrupt Request Register.
INTR_MASKED (Interrupt Masked Request Register)	0	TC	Logical AND of the corresponding TC request and mask bits.
	1	CC0_MATCH	Logical AND of the corresponding CC_MATCH0 request and mask bits.
	2	CC1_MATCH	Logical AND of the corresponding CC_MATCH1 request and mask bits.

25.2.8 Debug mode

The TCPWM counters support debugging. It can be configured per counter if the counter operation continues or pauses in debug state (for example, after running to a break point). This feature is especially intended when using a TCPWM counter as an OS timer. It is realized by gating the PCLK_TCPWM[x]_CLOCKS[y] when entering debug state by setting the DBG_FREEZE_EN bit to ‘1’ in the CTRL Register and asserting a debug pause trigger.

In a multicore environment ‘debug state’ means that at least one of the CPUs is in the debug state. In cases where only one CPU is debugged but another or multiple other CPUs are continuously running, the user can configure the counter via the debugger to continue or pause depending on which CPU is using the counter.

***Note**: The trigger input cannot be asserted when the counter is in debug state.*

25.2.9 PWM outputs

The PWM, PWM_DT, PWM_PR, and SR operation modes produce two output signals:

- A PWM LINE_OUT output signal
- A complementary PWM LINE_COMPL_OUT output signal (inverted version of LINE_OUT)

Note that in PWM and PWM_DT modes the CC0_match, CC1_match, underflow, and overflow internal event conditions are used to drive LINE_OUT and LINE_COMPL_OUT, by configuring the TR_PWM_CTRL Register (Table 198). In PWM_PR and SR modes, line output is not controlled by TR_PWM_CTRL.

Table 198 Configuring output line for OV, UN, and CC0/1 conditions

Field	Bit	Value	Event	Description
CC0_MATCH_MODE Default Value = 3	1:0	0	Set LINE_OUT to '1'	Configures output line on a compare match (CC0) 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 an 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 an underflow (UN) event
		1	Clear LINE_OUT to '0'	
		2	Invert LINE_OUT	
		3	No change	
CC1_MATCH_MODE Default Value = 3	7:6	0	Set LINE_OUT to '1'	Configures output line on a compare match (CC1) event
		1	Clear LINE_OUT to '0'	
		2	Invert LINE_OUT	
		3	No change	

The generation of PWM output signals is a multi-step process. Both LINE_OUT and LINE_COMPL_OUT are generated from the PWM signal line. The PWM signal line is generated as per the state of cc0_match, cc1_match, underflow, and overflow internal events, as specified by the counter's TR_PWM_CTRL Register. For each internal event, the TR_PWM_CTRL Register specifies how the event affects the output LINE_OUT

- The output is set to '0'
- The output is set to '1'
- The output is inverted
- The output is not affected

In case the internal cc0_match event generates at the same time when internal underflow or overflow event generates, cc0_match will take effect after LINE_OUT changes state per settings of underflow/overflow. cc1_match will take effect after cc0_match. Table 199 lists some examples to show the mechanism.

Table 199 LINE_OUT construction example

Coincide case	Overflow	Underflow	CC0_match	CC1_match	LINE_OUT
CC0_match and overflow	CLEAR	Don't care	INVERT	Don't care	1 (SET)
CC0_match and underflow	Don't care	SET	INVERT	Don't care	0 (CLEAR)
CC0_match and CC1_match	Don't care	Don't care	SET	CLEAR	CLEAR
CC0_match and CC1_match and overflow	INVERT	Don't care	INVERT	INVERT	INVERT

The following figure illustrates the process of LINE_OUT and LINE_COMPL_OUT generation.

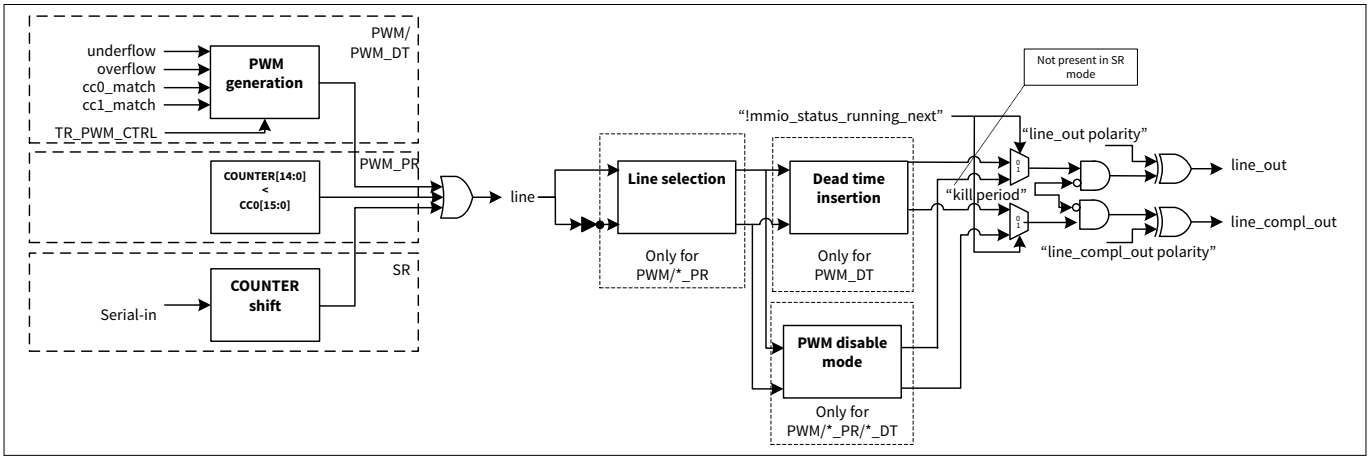


Figure 209 PWM output generation process in PWM/PWM_DT/PWM_PR/SR mode

Generally, LINE_OUT output reflects the state of PWM signal line and LINE_COMPL_OUT output reflects the inverted version of line. The line behavior depends on different function modes. Furthermore, some other factors will impact LINE_OUT and LINE_COMPL_OUT according to different function modes; they include 'line selection', 'dead time insertion', 'kill function', and 'line polarity'.

- PWM signal line generation
  - In PWM/PWM_DT mode, line is constructed by internal events underflow, overflow, cc0_match, and cc1_match per settings of TR_PWM_CTL.
  - In PWM_PR mode, line reflects the state of comparison b/w COUNTER and CC0.
  - In SR mode, line is the shift output of Shift Register (COUNTER).
- Line selection (available in counter groups supporting advanced motor control)
  - LINE_OUT and LINE_COMPL_OUT can individually select different output according to the LINE_SEL.OUT_SEL and LINE_SEL.COMPL_SEL Register settings. This functionality only works in PWM and PWM_PR modes.
  - LINE_OUT and LINE_COMPL_OUT can individually selects low, high, line, inverted line, and Hi-Z. When it selects Hi-Z, line_out_en and line_compl_out_en will be low.
- Dead time insertion
  - Dead time insertion functionality is mutually exclusive with line selection functionality, it only works in PWM_DT mode.
  - Dead time works on both line signal and inverted version of line signal.
- PWM disable mode

Specifies the behavior of the line_out and line_out_compl_out PWM outputs while the TCPWM counter is disabled (ENABLED bit set to '0' in the CTRL Register) or stopped. The PWM output behavior is determined by the PWM_DISABLE_MODE bit field in the CTRL Register. There are four options:

- Z (PWM_DISABLE_MODE = 0)

  When the counter is disabled the line_out and line_compl_out PWM outputs are not driven by the TCPWM. Instead, the port default level configuration applies, for example, "Z" (high impedance). When the counter is stopped on a stop event, the PWM outputs are deactivated and the polarity is defined by the QUAD_ENCODING_MODE bit field in the CTRL Register.
- Retain (PWM_DISABLE_MODE = 1)

  When the counter is disabled or stopped on a stop event, the PWM outputs are retained (keep their previous levels).
- Low (PWM_DISABLE_MODE = 2)

  When the counter is disabled or stopped on a stop event, the line_out PWM output is driven as a fixed '0' and the line_compl_out PWM output is driven as a fixed '1'.
- High (PWM_DISABLE_MODE = 3)

  When the counter is disabled or stopped on a stop event, the line_out PWM output is driven as a fixed '1' and the line_compl_out PWM output is driven as a fixed '0'.

- Kill function
  - Kill function works in PWM, PWM_DT, and PWM_PR modes. It does not work in SR mode.
  - Kill works on both line and inverted version of line, and there are several kill function modes supported.
- Polarity for LINE_OUT and LINE_COMPL_OUT
  - Polarity inversion is used to determine the LINE_OUT and LINE_COMPL_OUT output signal values.
  - CTRL.QUADRATURE_ENCODING_MODE[0] is for LINE_OUT polarity and CTRL.QUADRATURE_ENCODING_MODE[1] is for LINE_COMPL_OUT polarity.
  - When the counter is not enabled in reset state or not running (temporarily stopped or killed), the PWM output signals values are determined by their respective polarity settings.

Details of PWM line signal generation is described in separate function mode sections later in the document.

Besides LINE_OUT and LINE_COMPL_OUT, each counter provides line_out_en and line_compl_out_en, which reflect the counter enable state. These two output enable signals can be used to disable GPIO output after counter is disabled.

TCPWM block has four ports LINE_OUT[counter group number*256-1:0], LINE_COMPL_OUT[counter group number*256-1:0], line_out_en[counter group number*256-1:0], and line_compl_out_en[counter group number*256-1:0].

25.2.10 Power modes

The TCPWM block works in active and sleep modes. The TCPWM block is powered from VCCACT. The retentioned MMIO Register are powered in deep sleep with VCCRET, but unpowered in hibernate mode. The Configuration Registers and other logic are powered in deep sleep mode to keep the states of configuration registers. See Table 200 for details.

Table 200 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	The CPU is in sleep but the block is still functional in this mode. All counter clocks are on.

(table continues...)

Table 200 (continued) Power modes in TCPWM block

Power mode	Block status
Deep sleep	Both power and clocks to the block are turned off, but Configuration Registers retain their states.
Hibernate	In this mode, the power to this block is switched off. Configuration Registers will lose their state. No CLK_PERI is provided.

25.3 Operation modes

The counter block can function in seven operational modes, as shown in Table 201. The MODE [26:24] field of the Counter Control Register (CTRL) configures the counter in the specific operational mode.

Table 201 Operational mode configuration

Mode	MODE field [26:24]	Description
Timer	000	The counter increments or decrements by '1' at every counter clock cycle in which a count event is detected. The Compare/Capture Register is used to compare the count.
Capture	010	The counter increments or decrements by '1' at every counter clock cycle in which a count event is detected. A capture event copies the counter value into the Capture Register.
Quadrature	011	Quadrature decoding. The counter is decremented or incremented based on two phase inputs according to an X1, X2, and X4 decoding scheme or to the rotary count mode.
PWM	100	Pulse width modulation.
PWM_DT	101	Pulse width modulation with dead time insertion.
PWM_PR	110	Pseudo-random PWM using a 16- or 32-bit Linear Feedback Shift Register (LFSR) with programmable length to generate pseudo-random noise.
SR	111	Shift Register mode

The counter can be configured to count up, down, and up/down by setting the UP_DOWN_MODE[17:16] field in the CTRL Register, as shown in Table 202.

Table 202 Counting mode configuration (except quadrature mode)

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 the counter changes from 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 changes from a value of '0'.
UP/DOWN counting mode 1	10	Increments the counter until the period value is reached, and then decrements the counter until '0' is reached. A TC condition is generated only when the counter changes from a value of '0'.

(table continues...)

Table 202 (continued) Counting mode configuration (except quadrature mode)

Counting modes	UP_DOWN_MODE[17:16]	Description
UP/DOWN counting mode 2	11	Similar to up/down counting mode 1 but a TC condition is generated when the counter changes from '0' and when the counter value changes from the period value.

In quadrature mode this field acts as QUAD_RANGE_MODE field selecting between different counter ranges, reload value, and compare/capture behavior.

Table 203 Counting mode configuration for quadrature mode

Counting modes	UP_DOWN_MODE[17:16]	Description
UP counting mode	00	Increments the counter until the period value is reached. A TC condition is generated when the counter changes from the period value.
DOWN counting mode	01	Decrements the counter from the period value until 0 is reached. A TC condition is generated when the counter changes from a value of '0'.
UP/DOWN counting mode 1	10	Increments the counter until the period value is reached, and then decrements the counter until '0' is reached. A TC condition is generated only when the counter changes from a value of '0'.
UP/DOWN counting mode 2	11	Similar to up/down counting mode 1 but a TC condition is generated when the counter changes from '0' and when the counter value changes from the period value.

25.3.1 Timer mode

The timer mode is commonly used to measure the time of occurrence of an event or to measure the time difference between two events. The timer functionality increments/decrements a counter between 0 and the value stored in the PERIOD Register. When the counter is running, the count value stored in the COUNTER Register is compared with the compare/capture Register (CC0 and CC1). When COUNTER equals CC0, the cc0_match event is generated, even-handedly when COUNTER equals CC1, the cc1_match event is generated.

Timer functionality is typically used for one of the following:

- Timing a specific delay - the count event is a constant '1'.
- Counting the occurrence of a specific event - the event should be connected as an input trigger and selected for the count event.

Table 204 Timer mode trigger input description

Trigger inputs	Usage
Reload	Initializes and starts the counter. Behavior is dependent on UP_DOWN_MODE: • COUNT_UP: The counter is set to ‘0’ and count direction is set to ‘up’ • COUNT_DOWN: The counter is set to PERIOD and count direction is set to ‘down’ • COUNT_UPDN1/2: The counter is set to ‘1’ and count direction is set to ‘up’ Can be used when the counter is running or not running.

(table continues...)

Table 204 (continued) Timer mode trigger input description

Trigger inputs	Usage
Start	Starts the counter. The counter is not initialized by hardware. The current counter value is used. Behavior is dependent on UP_DOWN_MODE. When the counter is not running: • COUNT_UP: The count direction is set to 'up' • COUNT_DOWN: The count direction is set to 'down' • COUNT_UPDN1/2: The count direction is not modified Note that when the counter is running, the start event has no effect. Can be used when the counter is running or not running.
Stop	Stops the counter
Count	Count event increments/decrements the counter.
Capture0	Not used
Capture1	Not used

Incrementing and decrementing the counter is controlled by the count event and the counter clock, PCLK_TCPWM[x]_CLOCKS[y]. Typical operation will use a constant '1' count event and PCLK_TCPWM[x]_CLOCKS[y] without prescaling. Advanced operations are also possible; for example, the counter event configuration can decide to count the rising edges of a synchronized input trigger.

Table 205 Timer mode supported features

Supported Features	Description
Clock prescaling	Prescales the PCLK_TCPWM[x]_CLOCKS[y].
One shot	Counter is stopped by hardware, on a tc event. In COUNT_UPDN2, counter is stopped on tc event when underflow.
Auto reload CC	CC0 and CC0_BUFF are exchanged on a cc0_match event (when specified by CTRL.AUTO_RELOAD_CC0, no input event is required). CC1 and CC1_BUFF are exchanged on a cc1_match event (when specified by CTRL.AUTO_RELOAD_CC1).
Up/down modes	Specified by UP_DOWN_MODE: • COUNT_UP: The counter counts from 0 to PERIOD. • COUNT_DOWN: The counter counts from PERIOD to 0. • COUNT_UPDN1/2: The counter counts from 1 to PERIOD and back to 0.

Table 206 lists the trigger outputs and the conditions when they are triggered.

Table 206 Timer mode trigger outputs

Trigger outputs	Description
cc0_match	Counter changes from a state in which COUNTER equals ‘CC0’.
cc1_match	Counter changes from a state in which COUNTER equals ‘CC1’.
Underflow (UN)	Counter is decrementing and changes from a state in which COUNTER equals ‘0’.
Overflow (OV)	Counter is incrementing and changes from a state in which COUNTER equals ‘PERIOD.
TC	Specified by UP_DOWN_MODE: • COUNT_UP: tc event is the same as the overflow event. • COUNT_DOWN: tc event is the same as the underflow event. • COUNT_UPDN1: tc event is the same as the underflow event. • COUNT_UPDN2: tc event is the same as the logical OR of the overflow and underflow events. Reload will generate underflow/overflow, but not generate tc.

Table 207 Timer mode PWM outputs

PWM outputs	Description
LINE_OUT	Not used
LINE_COMPL_OUT	Not used

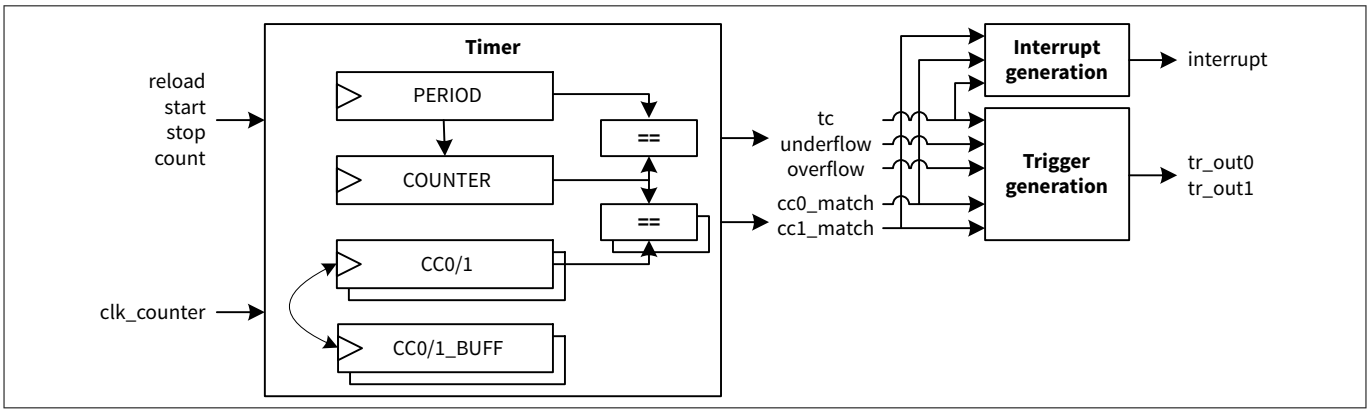


Figure 210 Timer functionality

***Note**:*
- *The triggers tr_out0 and tr_out1 are generated based on the internal events cc0_match, cc1_match, underflow, overflow, and tc respectively (selection is done by the TR_OUT_SEL Register).*
- *The timer functionality only uses PERIOD (and not PERIOD_BUFF).*
- *It is not recommended to write to COUNTER when the counter is running.*

Figure 211 illustrates a timer in up-counting mode. The counter is initialized (to 0) and started with a software-based reload event.

Note:
- *When the counter changes from a state in which COUNTER is 4, an overflow and tc event are generated.*
- *When the counter changes from a state in which COUNTER is 2, a cc0_match event is generated.*
- *PERIOD is 4, resulting in an effective repeating counter pattern of 4+1 = 5 counter clock periods. The CC0 Register is 2, and sets the condition for a cc0_match event.*

A constant count event of '1' and PCLK_TCPWM[x]_CLOCKS[y] without prescaling is used in the following scenarios. If the count event is '0' and a reload event is triggered, the reload will only be registered on the first clock edge when the count event is '1'.

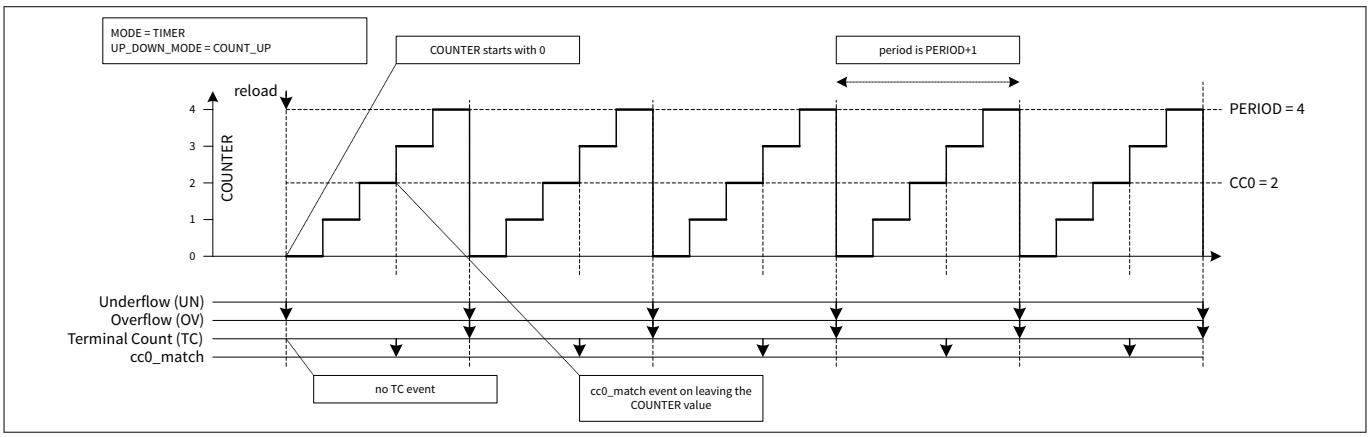


Figure 211 Timer in up-counting mode

Figure 212 illustrates a timer in “one-shot” operation mode. Note that the counter is stopped on a tc event.

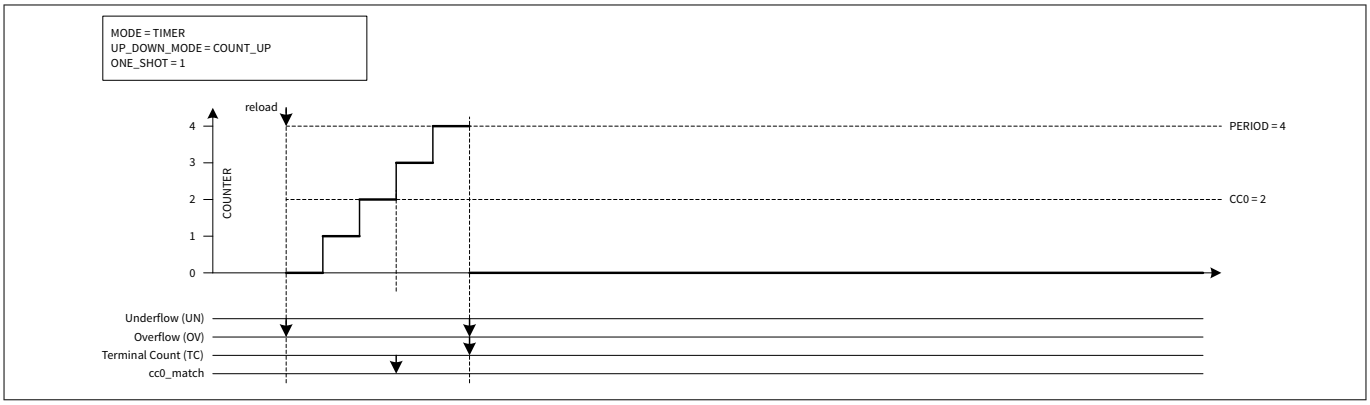


Figure 212 Timer in one-shot mode

Figure 213 illustrates clock prescaling. Note that the counter is only incremented every other counter cycle.

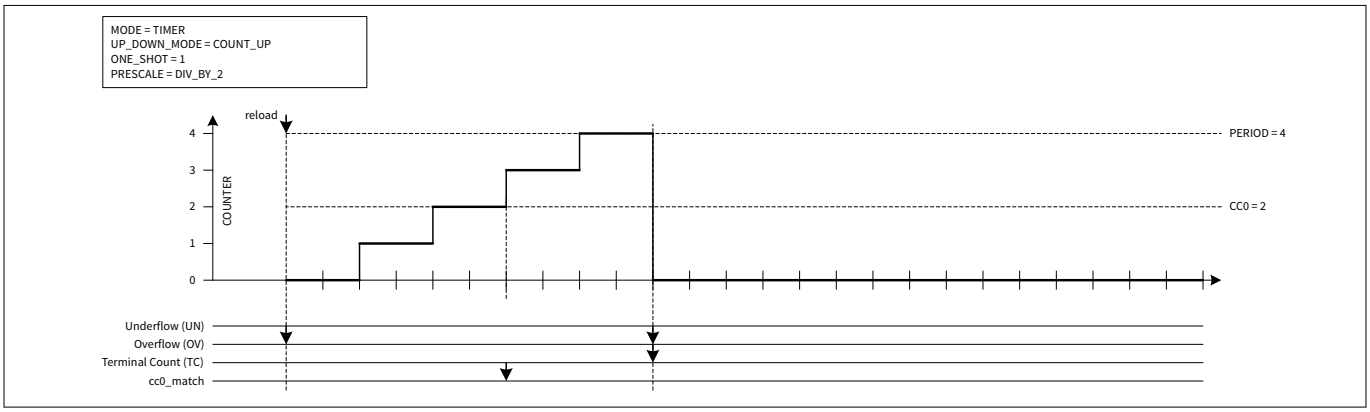


Figure 213 Timer clock prescaling

Figure 214 illustrates a counter that is initialized and started (reload event), stopped (stop event), and continued/started (start event). Note that the counter does not change value when it is not running (STATUS.RUNNING).

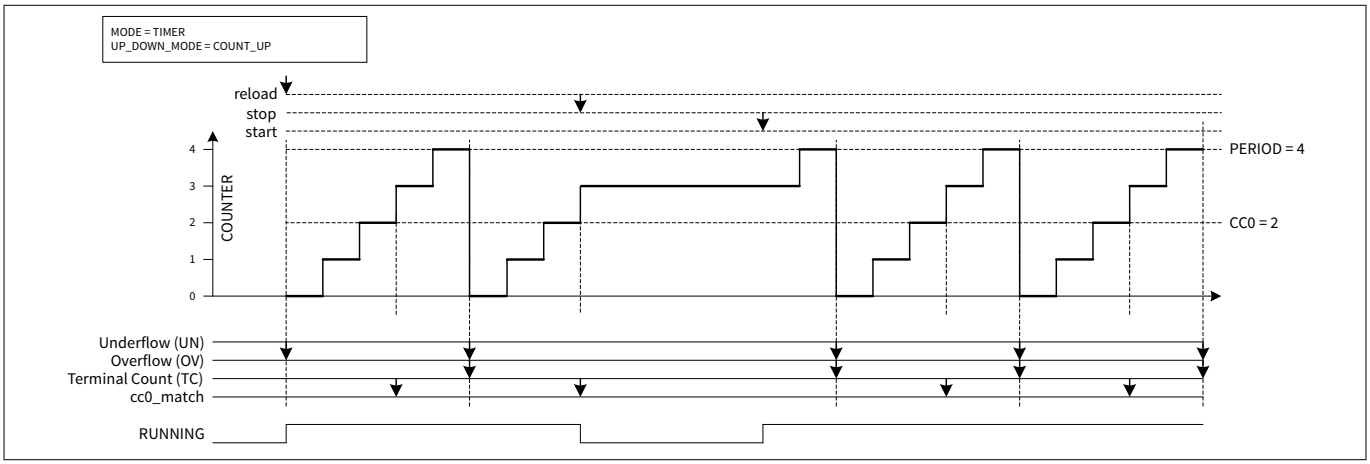


Figure 214 Counter start/stopped/continued

Figure 215 illustrates a timer that uses CC0/1 and CC0/1_BUFF Registers. Note that CC0/1 and CC0/1_BUFF Register contents are exchanged on a cc0/1_match event.

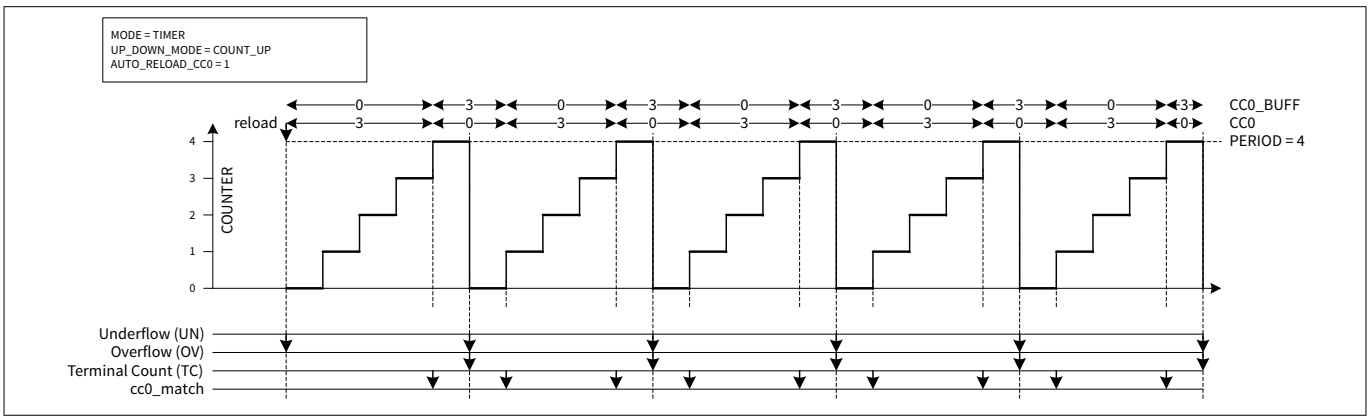


Figure 215 Use of CC0 and CC0_BUFF Register bits

Figure 216 illustrates a timer in down-counting mode. The counter is initialized (to PERIOD) and started with a software-based reload event.

***Note**:*
- *When the counter changes from a state in which COUNTER is '0', an underflow and tc event are generated.*
- *When the counter changes from a state in which COUNTER is '2', a cc0_match event is generated.*
- *PERIOD is 4, resulting in an effective repeating counter pattern of 4+1 = 5 counter clock periods.*

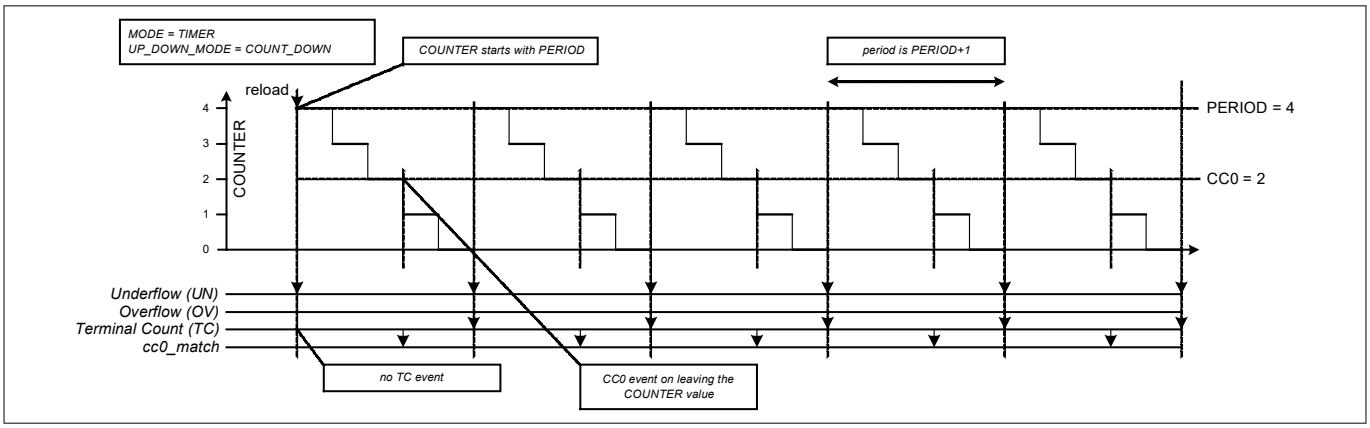


Figure 216 Timer in down-counting mode

Figure 217 illustrates a timer in up/down counting mode 1. The counter is initialized (to 1) and started with a software-based reload event.

***Note**:*
- *When the counter changes from a state in which COUNTER is '4', an overflow is generated.*
- *When the counter changes from a state in which COUNTER is '0', an underflow and tc event are generated.*
- *When the counter changes from a state in which COUNTER is '2', a cc0_match event is generated.*
- *PERIOD is 4, resulting in an effective repeating counter pattern of 2 × 4 = 8 counter clock periods.*

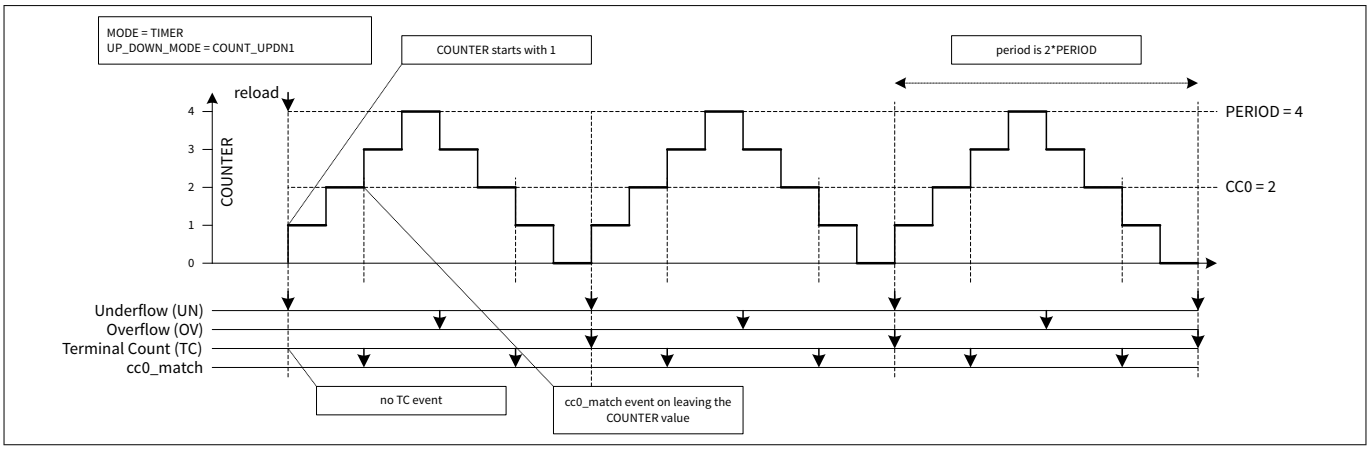


Figure 217 Timer in up/down counting mode 1

Figure 218 illustrates a timer in up/down counting mode 1, with different CC values.

***Note**:*

- *When CC0 is '0', the cc0_match event is generated at the start of the period (when the counter changes from a state in which COUNTER is '0').*
- *When CC0 is PERIOD, the cc0_match event is generated at the middle of the period (when the counter changes from a state in which COUNTER is 'PERIOD').*

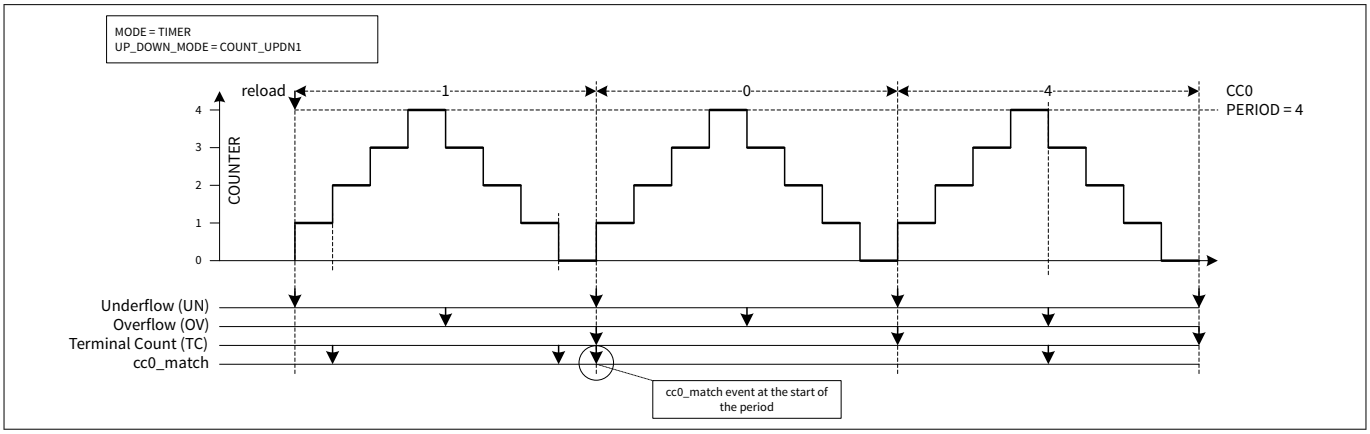


Figure 218 Up/down counting mode with different CC values

Figure 219 illustrates a timer in up/down counting mode 2. This mode is same as up/down counting mode 1, except for the TC event, which is generated when either underflow or overflow event occurs.

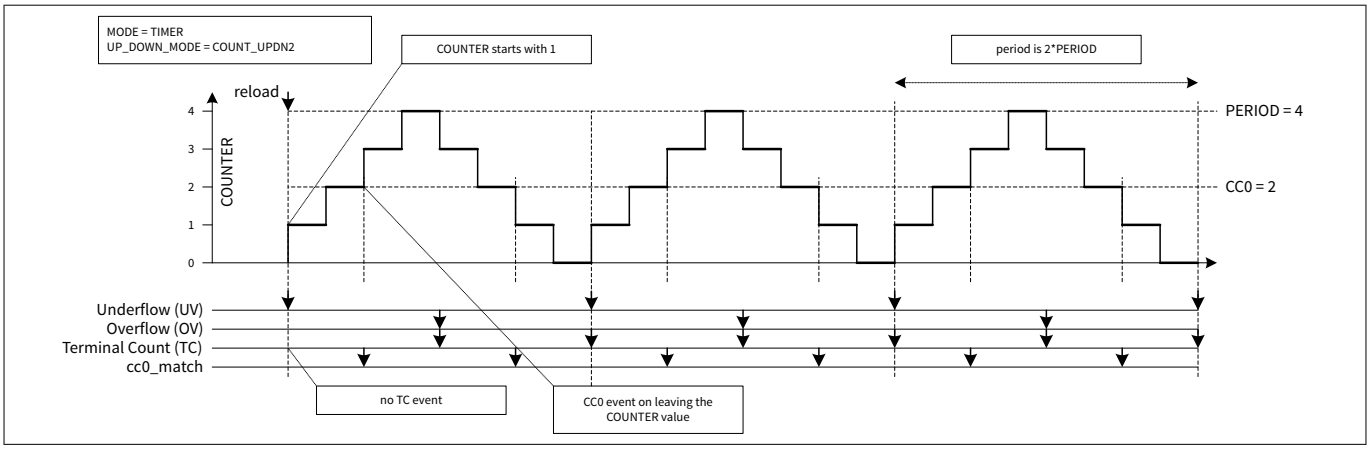


Figure 219 Up/down counting mode 2

25.3.1.1 Configuring counter for timer mode

The steps to configure the counter for Timer mode of operation and the affected register bits are as follows.

1. Disable the counter by writing '0' to the ENABLE bit of the CTRL Register.
2. Select Timer mode by writing '000' to the MODE[26:24] field of the CTRL Register.
3. Set the required 16- or 32-bit period in the PERIOD Register.
4. Set the 16- or 32-bit compare value in the CC0 Register and the buffer compare value in the CC0_BUFF Register.
5. Set AUTO_RELOAD_CC0 field of the CTRL Register, if required to switch values at every CC condition.
6. Set clock prescaling by writing to the DT_LINE_OUT_L[7:0] field of the DT Register.
7. Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of the CTRL Register.

8. The timer can be configured to run either in continuous mode or one-shot mode by writing 0 or 1, respectively to the ONE_SHOT[18] field of the CTRL Register.
9. Set the TR_IN_SEL0 or TR_IN_SEL1 Register to select the trigger that causes the event (reload, start, stop, capture0/1, and count).
10. Set the TR_IN_EDGE_SEL Register to select the edge of the trigger that causes the event (reload, start, stop, capture0/1, and count).
11. If required, set the interrupt upon TC or CC0_MATCH or CC1_MATCH condition.
12. Enable the counter by writing '1' to ENABLED bit of the CTRL Register. A start trigger must be provided through firmware (START bit in the TR_CMD Register) to start the counter if the hardware start signal is not enabled.

25.3.2 Capture mode

The capture functionality increments and decrements a counter between 0 and PERIOD. When the capture event is activated the counter value COUNTER is copied to CC0/1 (and CC0/1 is copied to CC0/1_BUFF).

The capture functionality can be used to measure the width of a pulse (connected as one of the input triggers and used as capture event).

The capture event can be triggered through the capture trigger input or through a firmware write to CAPTURE0/1 bit in the TR_CMD Command Register.

Table 208 Capture mode trigger input description

Generated events	Usage
Reload	Sets the counter value and starts the counter. Behavior is dependent on UP_DOWN_MODE: • COUNT_UP: The counter is set to '0' and count direction is set to 'up' • COUNT_DOWN: The counter is set to PERIOD and count direction is set to 'down'. • COUNT_UPDN1/2: The counter is set to '1' and count direction is set to 'up'. Can be used when the counter is running or not running.
Start	Starts the counter. The counter is not initialized by hardware. The current counter value is used. Behavior is dependent on UP_DOWN_MODE: • COUNT_UP: The count direction is set to 'up' • COUNT_DOWN: The count direction is set to 'down'. • COUNT_UPDN1/2: The count direction is not modified Note that when the counter is running, the start event has no effect. Can be used when the counter is running or not running.
Stop	Stops the counter
Count	Count event increments/decrements the counter
Capture0	Copies the counter value to CC0 and copies CC0 to CC0_BUFF
Capture1	Copies the counter value to CC1 and copies CC1 to CC1_BUFF

Table 209 Supported features of CAPTURE

Supported features	Description
Clock prescaling	Prescales the PCLK_TCPWM[x]_CLOCKS[y].

(table continues...)

Table 209 (continued) Supported features of CAPTURE

Supported features	Description
One shot	Counter is stopped by hardware, after a single period of the counter: • COUNT_UP: on an overflow event. • COUNT_DOWN, COUNT_UPDN1/2: on an underflow event
Up/down modes	Specified by UP_DOWN_MODE: • COUNT_UP: The counter counts from 0 to PERIOD. • COUNT_DOWN: The counter counts from PERIOD to 0 • COUNT_UPDN1/2: The counter counts from 1 to PERIOD and back to 0

Table 210 Internal events of CAPTURE

Internal events	Description
CC0_match	CC0 is copied to CC0_BUFF and counter value is copied to CC0 (cc0_match equals capture event).
CC1_match	CC1 is copied to CC1_BUFF and counter value is copied to CC1 (cc1_match equals capture event).
Underflow (UN)	Counter is decrementing and changes from a state in which COUNTER equals 0.
Overflow (OV)	Counter is incrementing and changes from a state in which COUNTER equals PERIOD.
TC	Specified by UP_DOWN_MODE: • COUNT_UP: tc event is the same as the overflow event • COUNT_DOWN: tc event is the same as the underflow event • COUNT_UPDN1: tc event is the same as the underflow event • COUNT_UPDN2: tc event is the same as the logical OR of the overflow and underflow events. Reload will generate underflow/overflow, but not generate tc.

Table 211 Capture mode PWM outputs

PWM outputs	Description
LINE_OUT	Not used
LINE_COMPL_OUT	Not used

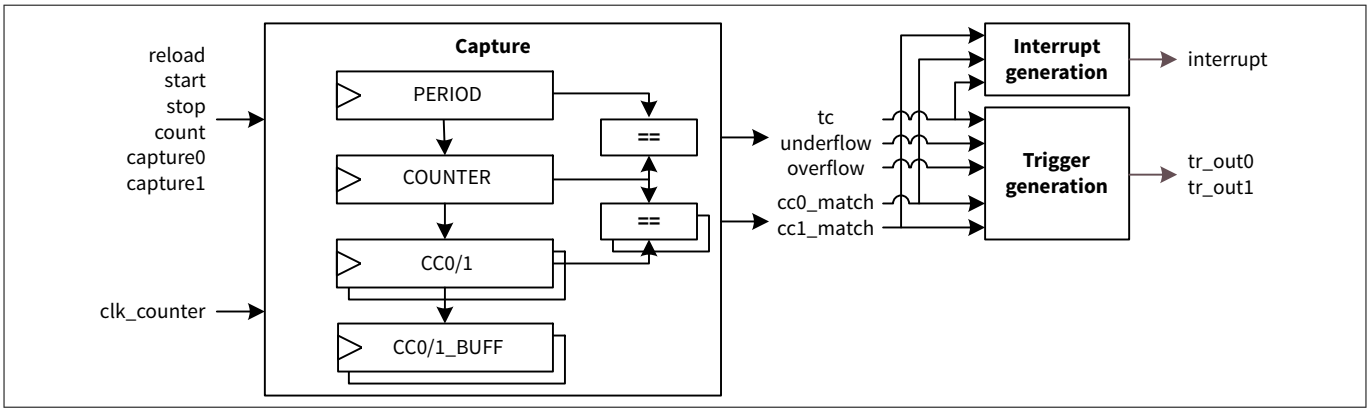


Figure 220 Capture functionality

Figure 221 illustrates capture behavior in the up-counting mode.

***Note**:*

- *The capture event detection uses rising edge detection. As a result, the capture event is remembered until the next active count prescaled counter clock.*
- *When a capture event occurs, COUNTER is copied into CC0/1. CC0/1 is copied to CC0/1_BUFF Register.*
- *A cc_match event is generated when the counter value is captured.*

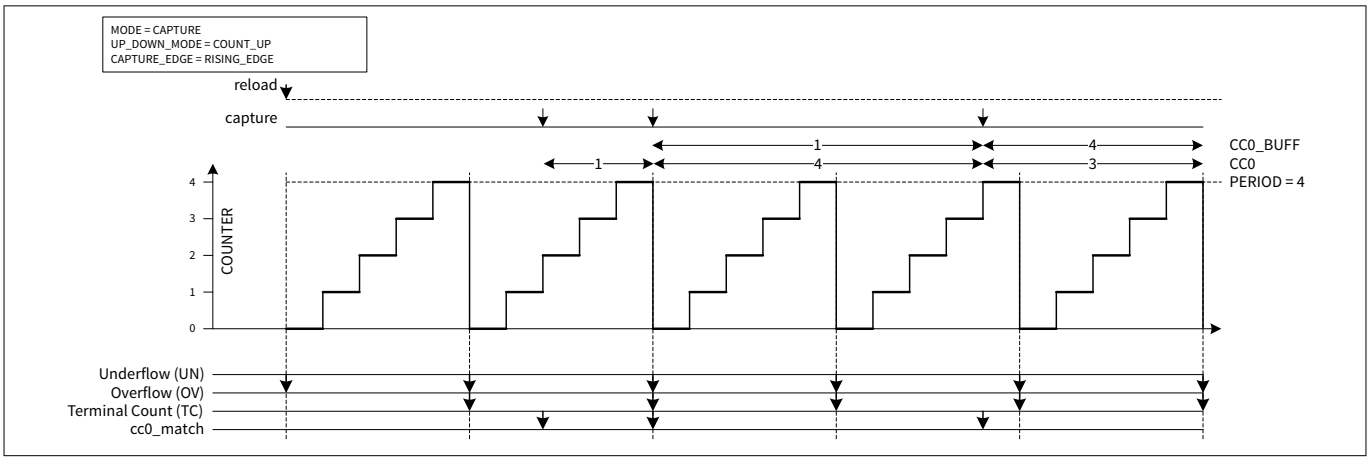


Figure 221 Capture in up-counting mode

When multiple capture events are detected before the next active count prescaled counter clock, capture events are treated as follows:

- In the rising edge and falling edge modes, multiple events are effectively reduced to a single event.
- In the rising/falling edge mode, an even number of events is not detected and an odd number of events is reduced to a single event.

This behavior is illustrated by Figure 222, in which a prescaler by a factor of 4 is used.

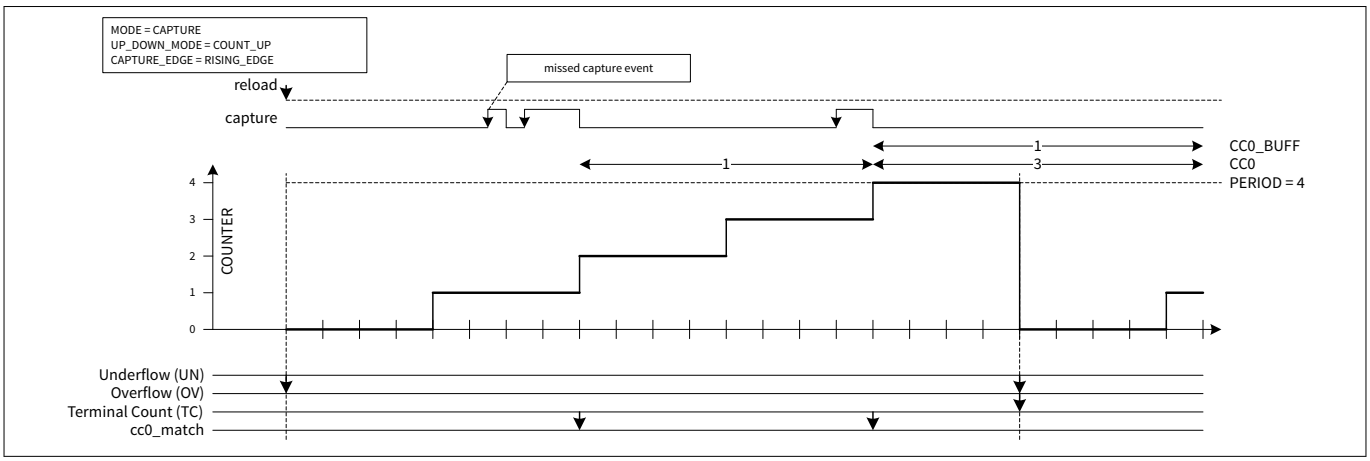


Figure 222 Multiple events detected before active-count

25.3.2.1 Configuring counter for capture mode

The steps to configure the counter for capture mode operation and the affected Register bits are as follows.

1. Disable the counter by writing '0' to the ENABLE bit of the CTRL Register.
2. Select capture mode by writing '010' to the MODE[26:24] field of the CTRL Register.
3. Set the required 16-bit period in the PERIOD Register.
4. Set clock prescaling by writing to the DT_LINE_OUT_L[17:16] field of the DT Register.
5. Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of the CTRL Register.
6. Counter can be configured to run either in continuous mode or one-shot mode by writing '0' or '1', respectively to the ONE_SHOT[18] field of the CTRL Register.
7. Set the TR_IN_SEL0 or TR_IN_SEL1 Register to select the trigger that causes the event (reload, start, stop, capture0/1, and count).
8. Set the TR_IN_EGE_SEL Register to select the edge that causes the event (reload, start, stop, capture0/1, and count).
9. If required, set the interrupt upon TC or CC0_MATCH or CC1_MATCH condition.
10. Enable the counter by writing '1' to the ENABLED bit in CTRL Register. A start trigger must be provided through firmware (START bit in TR_CMD Register) to start the counter if the hardware start signal is not enabled.

25.3.3 Quadrature decoder mode

Quadrature functionality increments and decrements a counter between 0 and 0xFFFF or 0xFFFFFFFF (32-bit mode) or PERIOD (depending on QUAD_RANGE_MODE). Counter updates are under control of quadrature signal inputs: index, phiA, and phiB. The index input is used to indicate an absolute position. The phiA and phiB inputs are used to determine a change in position (the rate of change in position can be used to derive speed).

Table 212 shows an overview of supported range modes, which varies between different maximum counter values uses capture and compare functionalities.

Table 212 Quadrature mode functionality overview

Supported range modes (QUAD_RANGE_MODE)	Description
QUAD_RANGE0	Counter range is between 0x0000 and 0xFFFF/0xFFFFFFFF (32-bit mode).
QUAD_RANGE0_CMP	Counter range is between 0x0000 and 0xFFFF/0xFFFFFFFF (32-bit mode). In this mode a compare function is supported during quadrature decoding using the CC0/CC0_BUFF (CC1/CC1_BUFF) Registers and the cc0_match (cc1_match) event.
QUAD_RANGE1_CMP	The compare functionality is the same as for QUAD_RANGE0_CMP mode. The counter range can be set between 0x0000 and PERIOD.
QUAD_RANGE1_CAPT	Counter range is between 0x0000 and PERIOD. Quadrature functionality in QUAD_RANGE1_CAPT mode provides the same functionality as the QUAD_RANGE1_CMP mode with the only difference that 1 or 2 capture functions are available instead of 1 or 2 compare functions.

The quadrature inputs are mapped onto triggers (as described in Table 213).

Table 213 Quadrature mode trigger input description

Trigger input	Usage
reload/index	This event acts as a quadrature index input. It initializes the counter to the counter midpoint 0x8000 (16-bit) or 0x8000000 (32-bit mode) and starts the quadrature functionality. Rising edge event detection or falling edge detection mode should be used. • QUAD_RANGE0: initialize counter to 0x8000/0x80000000 (midpoint) • QUAD_RANGE0_CMP: initialize counter to 0x8000/0x80000000 (midpoint) • QUAD_RANGE1_CMP: initialize counter to'0' • QUAD_RANGE1_CAPT: initialize counter to '0'
start/phiB	This event acts as a quadrature phiB input. Pass-through (no edge detection) event detection mode should be used for X1, X2, or X4.
stop	Stops the quadrature functionality. When quadrature stops, reload must be used to start the quadrature.
count/phiA	This event acts as a quadrature phiA input. Pass-through (no edge detection) event detection mode should be used for X1, X2, or X4.
Capture0	• QUAD_RANGE0: Not used • QUAD_RANGE0_CMP: Second index • QUAD_RANGE1_CMP: Not used • QUAD_RANGE1_CAPT: Capture event to copy COUNTER to CC0 and CC0 to CC0_BUFF
Capture1	Available only in counter groups with second capture function. • QUAD_RANGE0: Not used • QUAD_RANGE0_CMP: Not used • QUAD_RANGE1_CMP: Not used • QUAD_RANGE1_CAPT: Second capture event to copy COUNTER to CC1 and CC1 to CC1_BUFF

Table 214 Quadrature mode supported features

Supported features	Description
Supported encoding modes (QUAD_ENCODING_MODE)	Four encoding schemes for the phiA and phiB inputs are supported (as specified by QUAD_ENCODING_MODE [21:20] bit field in the CTRL Register): • X1 encoding • X2 encoding • X4 encoding Up/down rotary count mode

***Note**: Clock prescaling is not supported and the count event is used as a quadrature input phiA. Thus, the quadrature functionality operates on the counter clock (PCLK_TCPWM[x]_CLOCKS[y]), rather than on an active count prescaled counter clock.*

Table 215 summarize the trigger outputs dependent on different QUAD range modes.

Table 215 Quadrature mode trigger output description

Trigger outputs	QUAD range mode	Description
cc0_match	QUAD_RANGE0	Counter value COUNTER equals 0 or 0xFFFF/0xFFFFFFFF (32-bit mode) reload/index event
	QUAD_RANGE0_CMP	Counter changes to a state in which COUNTER equals CC0
	QUAD_RANGE1_CMP	Same as QUAD_RANGE0_CMP
	QUAD_RANGE1_CAPT	Capture0 event
cc1_match	QUAD_RANGE0	Not used
	QUAD_RANGE0_CMP	Counter changes to a state in which COUNTER equals CC1
	QUAD_RANGE1_CMP	Same as QUAD_RANGE0_CMP
	QUAD_RANGE1_CAPT	Capture1 event
underflow	QUAD_RANGE0	Not used
	QUAD_RANGE0_CMP	Not used
	QUAD_RANGE1_CMP	Counter value COUNTER equals 0 and is decrementing
	QUAD_RANGE1_CAPT	Counter value COUNTER equals 0 and is decrementing
overflow	QUAD_RANGE0	Not used
	QUAD_RANGE0_CMP	Not used
	QUAD_RANGE1_CMP	Counter value COUNTER equals PERIOD and is incrementing
	QUAD_RANGE1_CAPT	Counter value COUNTER equals PERIOD and is incrementing
tc	QUAD_RANGE0	Index event
	QUAD_RANGE0_CMP	Counter value COUNTER equals 0 or 0xFFFF/0xFFFFFFFF Index or capture on index event (specified by AUTO_RELOAD_PERIOD in the CTRL Register)
	QUAD_RANGE1_CMP	Counter value COUNTER equals 0 and is decrementing (underflow) or PERIOD and is incrementing (overflow) Index event
	QUAD_RANGE1_CAPT	Same as QUAD_RANGE1_CMP

Table 216 Quadrature mode PWM outputs

PWM outputs	Description
LINE_OUT	Not used
LINE_COMPL_OUT	Not used

Counter increments (incr1 event) and decrements (decr1 event) are determined by the quadrature encoding scheme as illustrated by Figure 223.

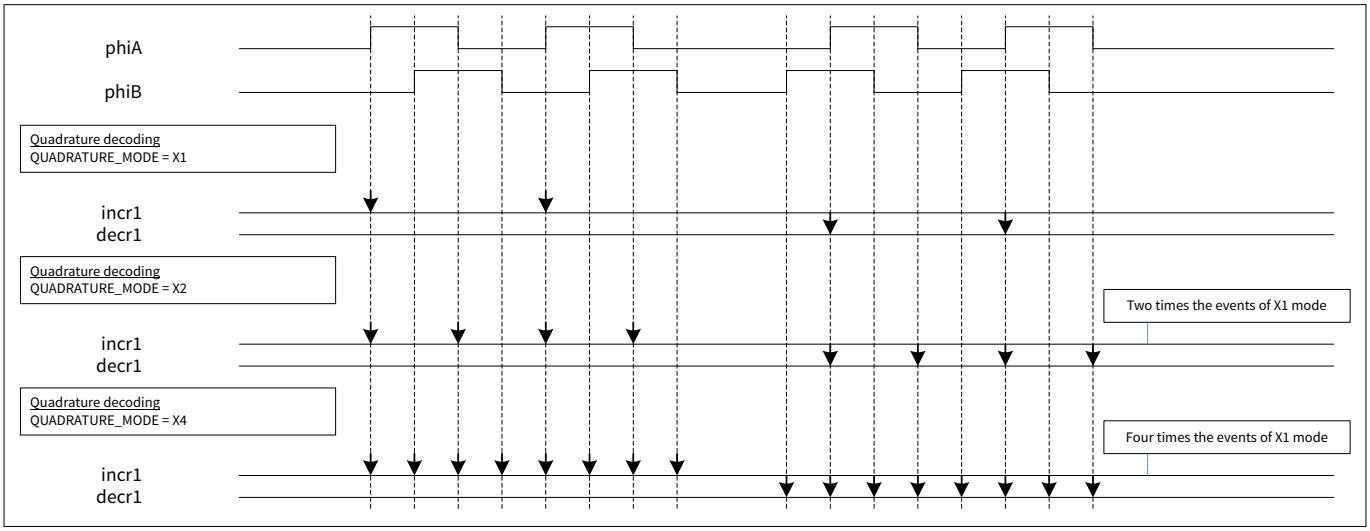


Figure 223 Quadrature mode waveforms (X1, X2, and X4 mode)

***Note**: The x1 encoding scheme is identical to the up/down counting functionality as follows: Rising edges of input phiA increment or decrement the counter depending on the state of input phiB (direction input).*

With UP_DOWN encoding (up/down rotary count mode) the counter is incremented by phiA and decremented by phiB as illustrated by Figure 224. In UP_DOWN encoding mode, phiA and phiB can be in the pass through or edge detection mode.

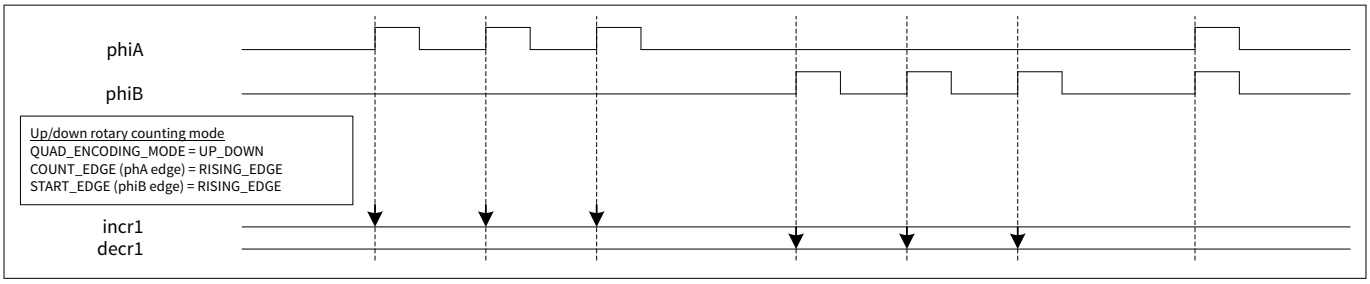


Figure 224 Up/down rotary mode

The state of phiA/phiB determines the increment/decrement according to the setting of the encoding mode; the phiA/phiB is detected by counter clock “clk_counter”. The increment/decrement occurs when the edge of phiA or phiB is detected by the “clk_counter”. To get correct quadrature encoding as shown in Figure 25-22, only one edge can happen within one “clk_counter” cycle. Figure 224 illustrates phase inputs edge detection in the X4 encoding mode. Figure 225 also shows the phiA/phiB detection at a different counter clock cycle.

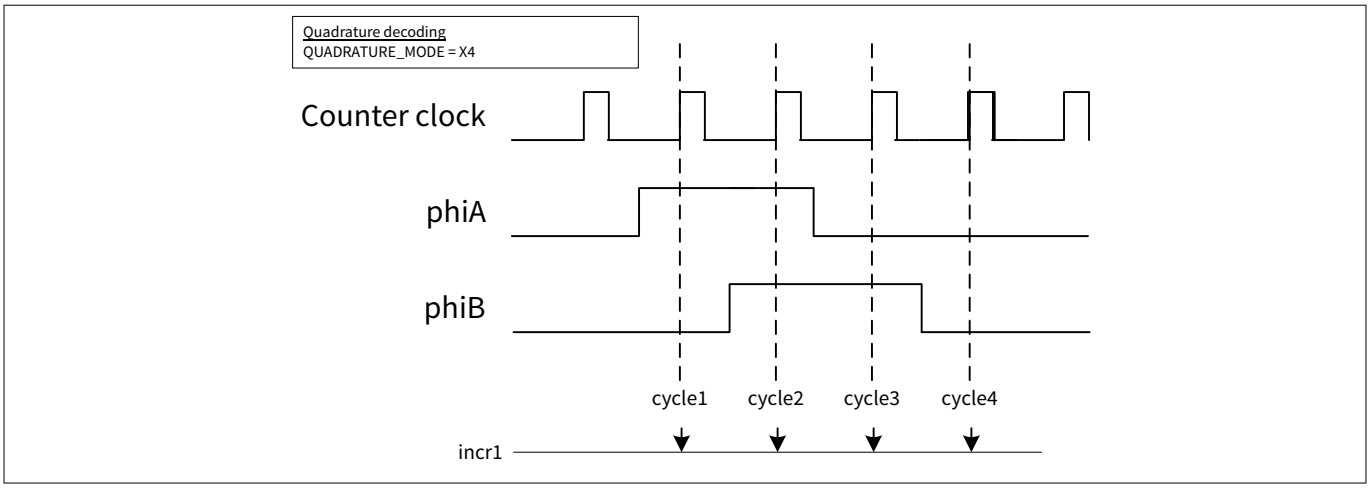


Figure 225 phiA/phiB detection at counter clock

25.3.3.1 Quadrature QUAD_RANGE0 mode

In this mode the counter range is between 0x0000 and 0xFFFF/0xFFFFFFFF (32-bit mode)

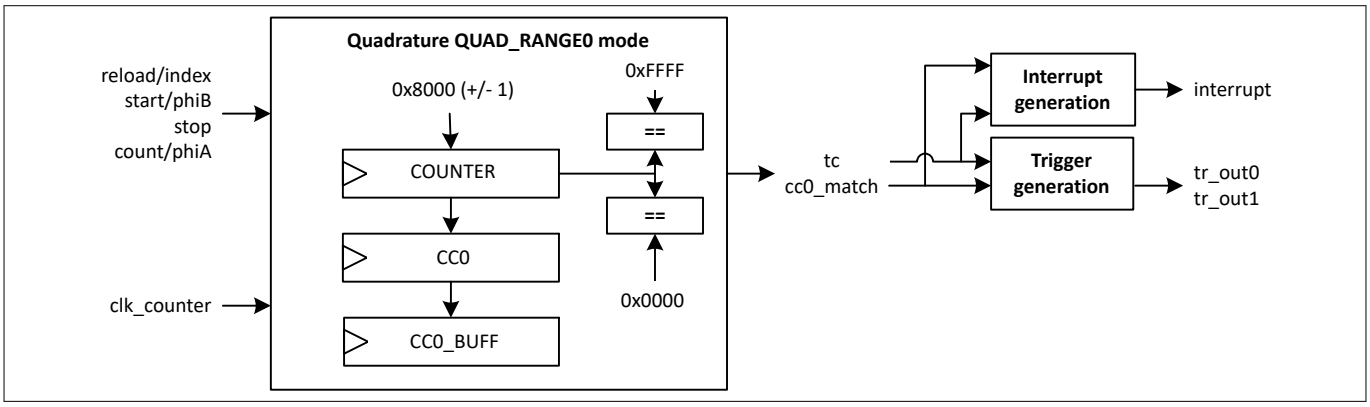


Figure 226 Quadrature (QUAD_RANGE0 mode) function diagram

Quadrature functionality in QUAD_RANGE0 mode (16-bit example) is described as a software-generated reload event starts quadrature operation. As a result, COUNTER is set to 0x8000, which is the counter midpoint (the COUNTER is set to 0x7FFF if the reload event coincides with a decrement event; the COUNTER is set to 0x8001 if the reload event coincides with an increment event). Note that a software-generated reload event is typically generated only once, when the counter is not running. All other reload/index events are hardware-generated reload events as a result of the quadrature index signal.

During quadrature operation:

- The counter value COUNTER is incremented or decremented based on the specified quadrature encoding scheme.
- On a reload/index event, CC0 is copied to CC0_BUFF, COUNTER is copied to CC0, and COUNTER is set to 0x8000. In addition, the tc and cc0_match events are generated.
- When the counter value COUNTER is 0x0000, CC0 is copied to CC0_BUFF, COUNTER (0x0000) is copied to CC0, and COUNTER is set to 0x8000. In addition, the cc0_match event is generated.
- When the counter value COUNTER is 0xFFFF, CC0 is copied to CC0_BUFF, COUNTER (0xFFFF) is copied to CC0, and COUNTER is set to 0x8000. In addition, the cc0_match event is generated.

The software interrupt handler uses the tc and cc0_match interrupt cause fields to distinguish between a reload/index event and a situation in which a minimum/maximum counter value was reached (about to wrap around). The CC0 and CC0_BUFF Registers are used to determine when the interrupt causing event occurred.

Note that a counter increment/decrement can coincide with a reload/index/tc event or with a situation cc0_match event. Under these circumstances, the counter value is set to either 0x8000+1 (increment) or 0x8000-1 (decrement).

Figure 227 illustrates quadrature functionality as a function of the reload/index, incr1, and decr1 events. Note that the first reload/index event copies the counter value COUNTER to CC0.

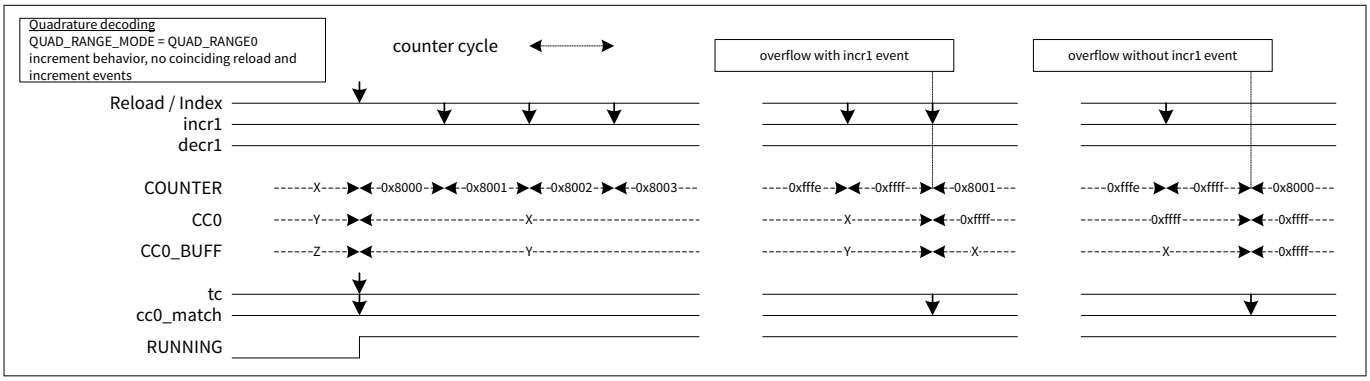


Figure 227 Overflow coincides with increment

Figure 228 to Figure 230 illustrate quadrature functionality for different event scenarios (including scenarios with coinciding events). In all scenarios, the first reload/index event is generated by software when the counter is not yet running.

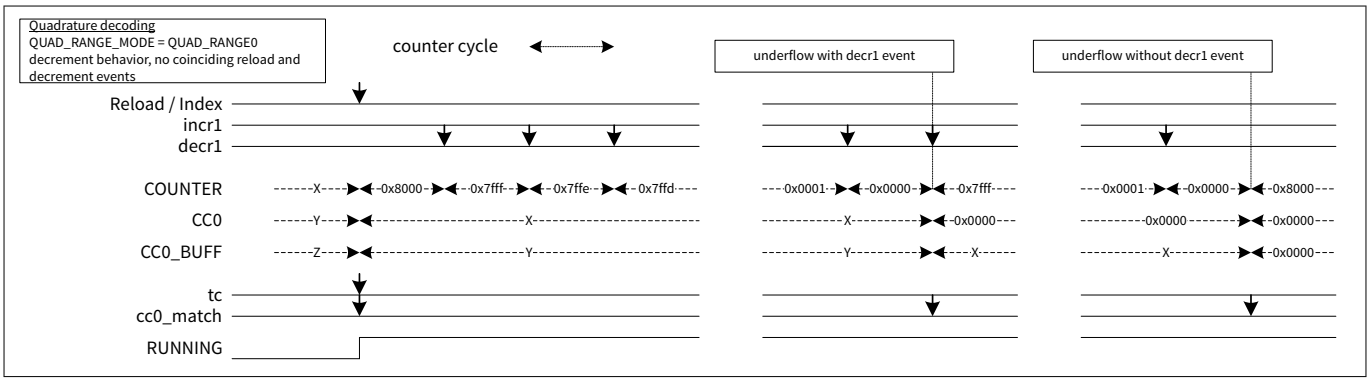


Figure 228 Underflow coincides with decrement

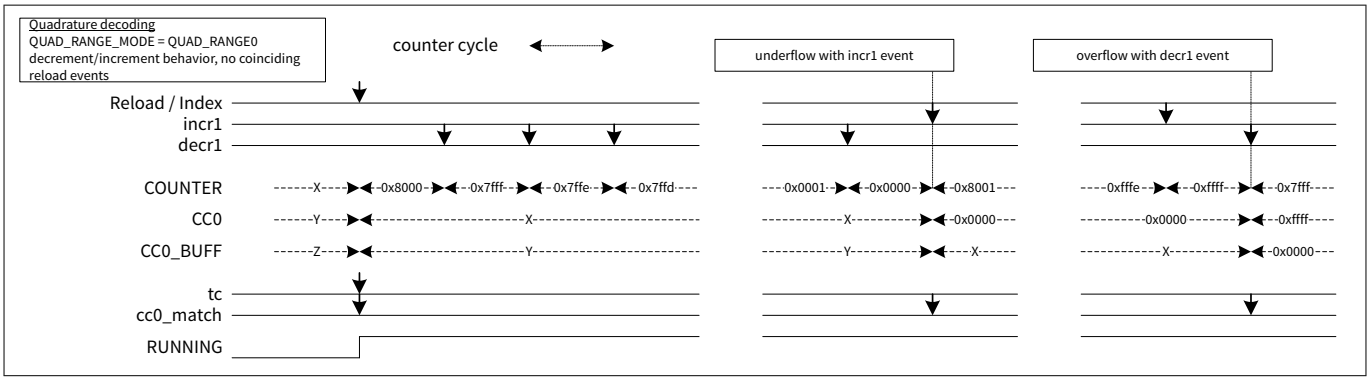


Figure 229 Underflow coincides with increment and overflow coincides with decrement

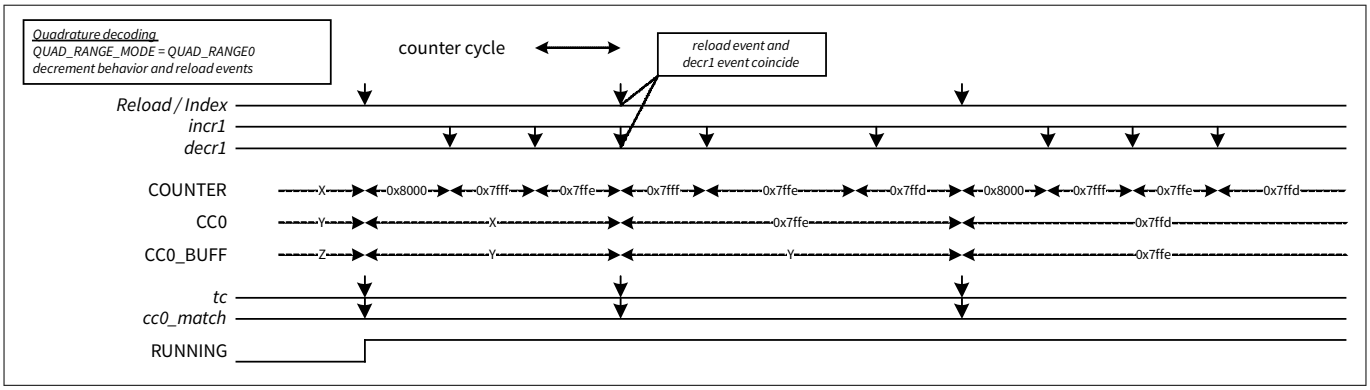


Figure 230 Index, decrement (increment), and underflow (overflow) coincides

The QUAD_RANGE0 functionality has the advantage that the interrupts when reaching minimum/maximum values are far apart in time, so such interrupts are unlikely to get lost. This mode is preferred when interrupts are used for example to implement a higher range counter in software. Because the hardware and software counters are not updated in an atomic operation, this is not recommended for applications with real-time requirements (such as motor control).

A disadvantage of this mode is that a physical angle position of the quadrature encoder can have multiple counter representations, so software needs to do module and subtract operations to calculate the absolute angle position.

```
x = COUNTER;                                    // read COUNTER register
if (x>=0x8000)
        pos = (x-0x8000) mod NR_COUNTS;     // NR_COUNTS = encoders number of counts for one
revolution
else
        pos = NR_COUNTS - ((0x8000-x) mod NR_COUNTS);
```

25.3.3.2 Configuring counter for quadrature mode (QUAD_RANGE0 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 ENABLE bit of the CTRL Register.
2. Select quadrature mode by writing '011' to the MODE[26:24] field of the CTRL Register.
3. Set the required encoding mode by writing to the QUADRATURE_MODE[21:20] field of the CTRL Register.
4. Set the TR_IN_SEL0 or TR_IN_SEL1 Register to select the trigger that causes the event (index and stop).
5. Set the TR_IN_EDGE_SEL Register to select the edge that causes the event (index and stop).
6. Set the quadrature mode QUAD_RANGE0 by writing with the value '0' to the UP_DOWN_MODE [17:16] field in the CTRL RegisterRegister.
7. If required, set the interrupt upon TC or CC0_MATCH condition.
8. Enable the counter by writing '1' to the ENABLE bit of the CTRL Register. A start trigger must be provided through firmware (START bit in TR_CMD Register) to start the counter if the hardware start signal is not enabled.

25.3.3.3 Quadrature QUAD_RANGE0_CMP mode

In this mode the counter range is also between 0x0000 and 0xFFFF/0xFFFFFFFF (32-bit mode). It allows a compare function during quadrature decoding using the CC0/CC0_BUFF Registers and the cc0_match event.

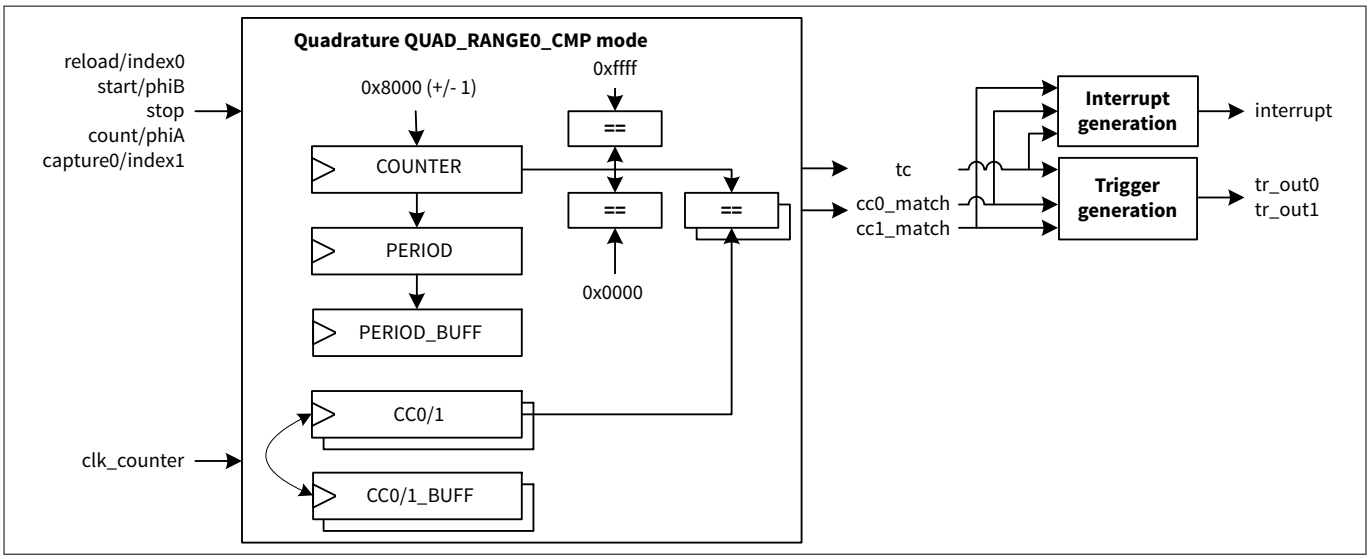


Figure 231 Quadrature (QUAD_RANGE0_CMP mode) function diagram

Quadrature functionality in QUAD_RANGE0_CMP mode provides the same functionality as the QUAD_RANGE0 mode, except for the following differences:

- PERIOD and PERIOD_BUFF are used instead of CC0 and CC0_BUFF to capture the counter value when a reload/index event occurs or the minimum/maximum value is reached (about to wrap around).
  - When the 'capture on index' function is selected (via overloaded AUTO_RELOAD_PERIOD bit) and a reload/index event occurs, PERIOD is copied to PERIOD_BUFF, COUNTER is copied to PERIOD, and COUNTER is set to 0x8000. In addition, the tc event is generated.
  - When the 'capture on wrap-around' function is selected (via overloaded AUTO_RELOAD_PERIOD bit) and the counter value COUNTER is 0x0000 or 0xFFFF, PERIOD is copied to PERIOD_BUFF, COUNTER (0x0000 or 0xFFFF) is copied to PERIOD, and COUNTER is set to 0x8000. In addition, the tc event is generated.
- Capture0 can be used as the second index event. This event acts as a second quadrature index input. It has the same function as the reload/index0 event. Both events are OR combined.
- CC0 (CC1) and CC0_BUFF (CC1_BUFF) are used for compare functionality.
  - A cc0_match (cc1_match) event is generated when the counter changes to a state in which COUNTER equals CC0 (CC1).
  - CC0 (CC1) and CC0_BUFF (CC1_BUFF) are exchanged on a cc0_match (cc1_match) event (when specified by AUTO_RELOAD_CC bit).

Note that 'capture on index' and 'capture on wraparound' functions are separated to prevent PERIOD and PERIOD_BUFF from being overwritten before software has read them in case a wraparound is followed by multiple index events in a short time (quadrature encoder is moved back and forth around its index point). If both functions are needed, the two counters can be used synchronously (or a counter group, which includes two compare functions) in QUAD_RANGE0_CMP mode, one with 'capture on index', the other with 'capture on wraparound' behavior selected. Note also that multiple compare values can be realized by multiple synchronous counters in QUAD_RANGE0_CMP mode with different CC0 values. Except the differences mentioned above, the QUAD_RANGE0_CMP mode behaves as the QUAD_RANGE0 mode including behavior with coinciding events. Figure 232 illustrates an example scenario with decrementing counter and additional compare functionality.

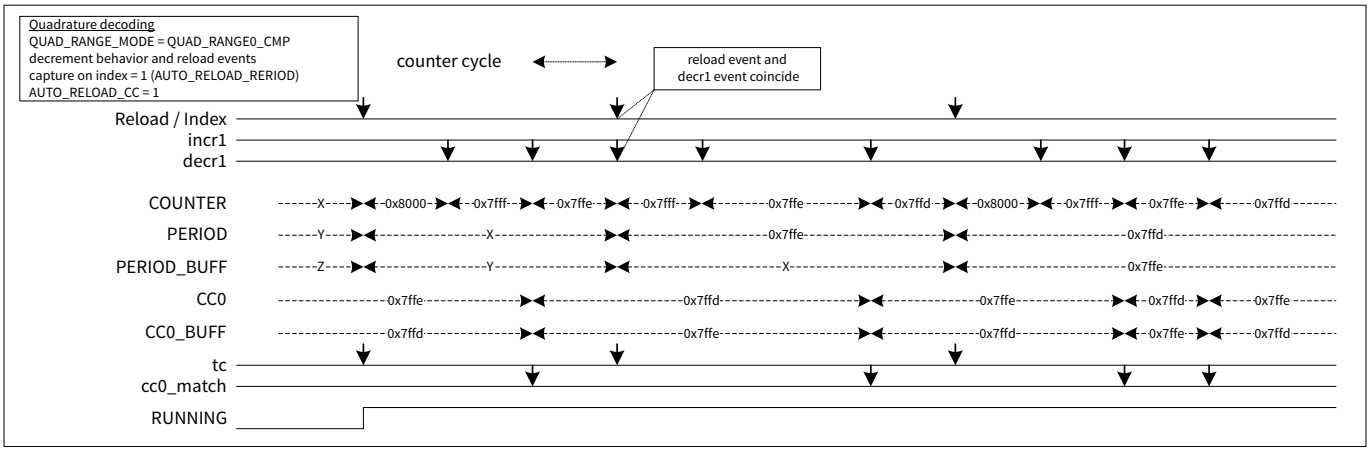


Figure 232 Quadrature (QUAD_RANGE0_CMP) operation

The QUAD_RANGE0_CMP functionality still allows a similar interrupt usage as in QUAD_RANGE0 mode. Additionally it supports one or two compare functions. More compare functions can be reached with multiple synchronously running counters in QUAD_RANGE0_CMP mode. These compare functions can be for example used for a position compare. As in QUAD_RANGE0 mode there is the disadvantage that a physical angle position of the quadrature encoder can have multiple counter representations, so software needs to do module and subtract operations to calculate the absolute angle position. In QUAD_RANGE0_CMP mode this software operation can be simplified using two compare functions; for example, by setting CC0 = 0x8000 + NR_COUNTS and CC1 = 0x8000 - NR_COUNTS, and feeding back the cc0/1_match events back into the TCPWM counter as index0/1 events using peripheral trigger multiplexers. This sets the counter back to its midpoint when reaching CC0 or CC1 value (after up to three CLK_PERI cycles for synchronization). This way the module operation can be saved in software when calculating the absolute angle position:

```
x = COUNTER;            // read COUNTER register
if (x>=0x8000)
        pos = x-0x8000;
else
        pos = x - CC1;
```

This is much less software overhead than in QUAD_RANGE0 mode; however, software is still involved. A DMA copy of the absolute angle position, for example, to send a buffer of CAN/UART/field bus interface to synchronize with other devices is not possible. This can only be supported when the counter represents the absolute angle position, as done in the QUAD_RANGE1_CAPT and QUAD_RANGE1_CMP modes.

25.3.3.4 Quadrature QUAD_RANGE1_CMP mode

In this mode the counter range is between 0x0000 and PERIOD.

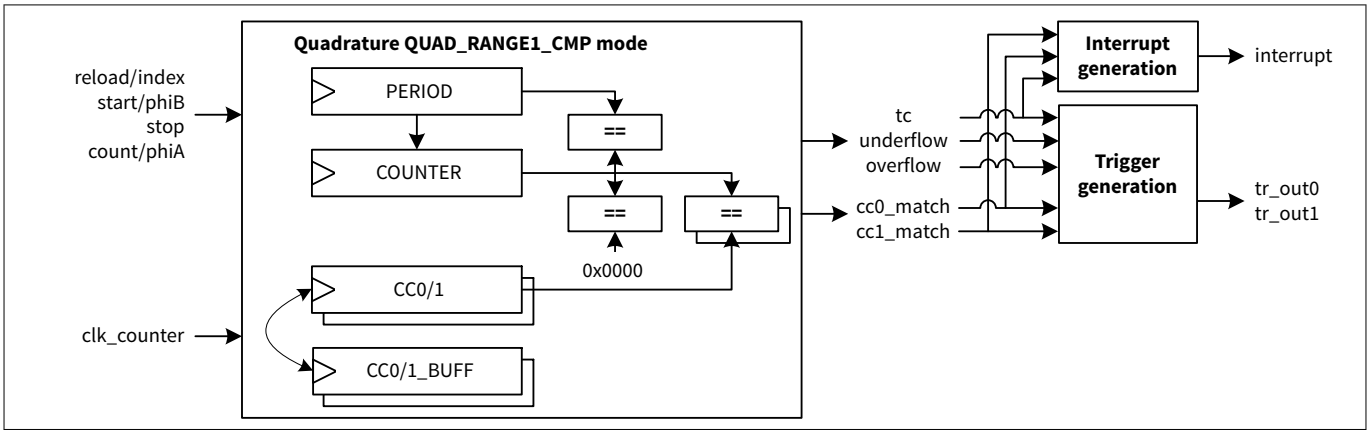


Figure 233 Quadrature (QUAD_RANGE1_CMP) function diagram

Quadrature functionality in QUAD_RANGE1_CMP mode is described as a software-generated reload event starts quadrature operation. As a result, COUNTER is set to 0x0000 (the COUNTER is set to PERIOD if the reload event coincides with a decrement event; the COUNTER is set to 0x0001 if the reload event coincides with an increment event). Note that a software-generated reload event is generated only once, when the counter is not running. All other reload/index events are hardware-generated reload events as a result of the quadrature index signal.

During quadrature operation:

- The counter value COUNTER is incremented or decremented based on the specified quadrature encoding scheme.
- On a reload/index event, COUNTER is set to 0x0000. In addition, the tc event is generated.
- When COUNTER is 0x0000 and decrementing, COUNTER is set to PERIOD. In addition, the tc event and underflow event are generated.
- When COUNTER equals PERIOD and is incrementing, COUNTER is set to 0x0000. In addition, the tc event and overflow event are generated.

CC0 and CC0_BUFF are used for compare functionality.

- A cc0/1_match event is generated when the counter changes to a state in which COUNTER equals CC0/1.
- CC0/1 and CC0/1_BUFF are exchanged on a cc0/1_match event (when specified by AUTO_RELOAD_CC bit in the CTRL Register).

Note that a counter increment/decrement can coincide with a reload/index/tc event. In this case, the counter value is set to either 0x0000+1 (increment) or PERIOD (decrement). The following figure illustrates quadrature functionality as a function of the reload/index, incr1 and decr1 events.

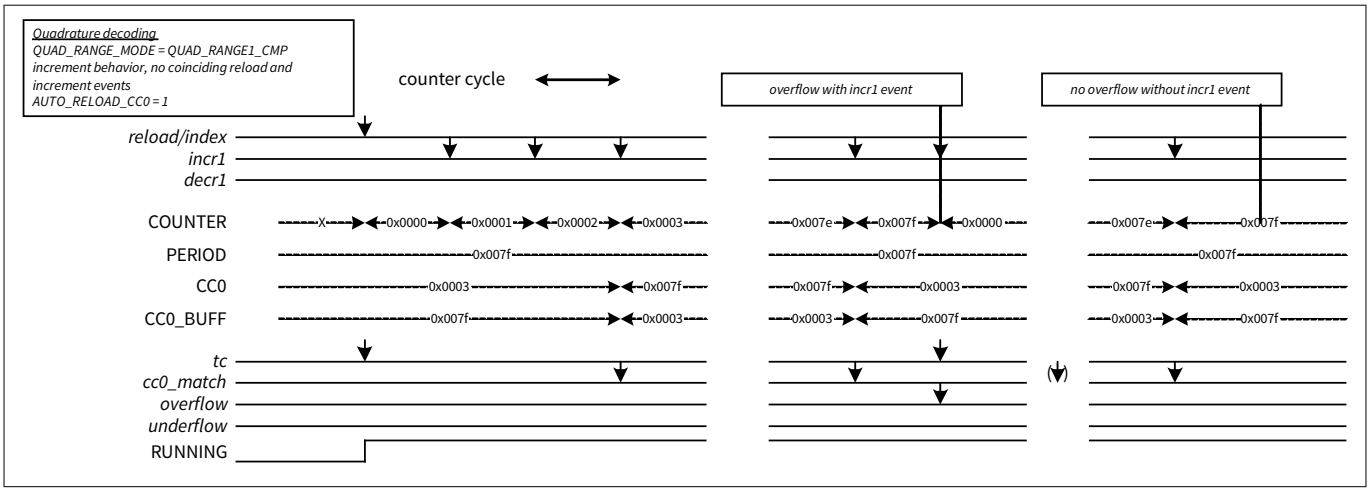


Figure 234 Quadrature Index, incr1 and tc (overflow) generation

The following figures illustrate quadrature functionality for different event scenarios (including scenarios with coinciding events). In all scenarios, the first reload/index event is generated by software when the counter is not yet running.

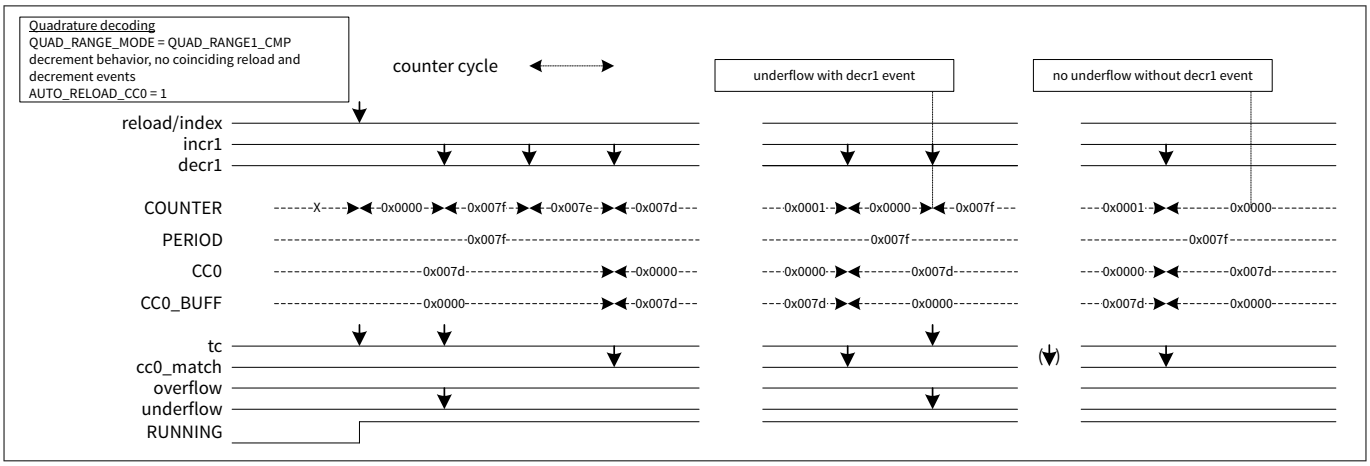


Figure 235 Quadrature Index, decr1 and tc (underflow) generation

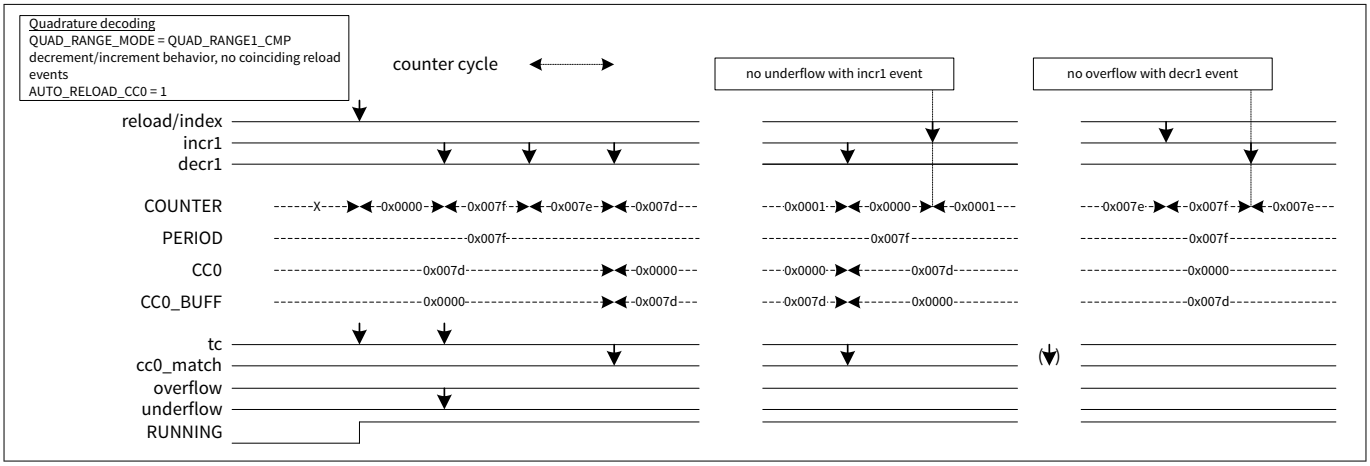


Figure 236 No tc (underflow) after COUNTER = 0x0000 and no tc (overflow) after COUNTER = PERIOD

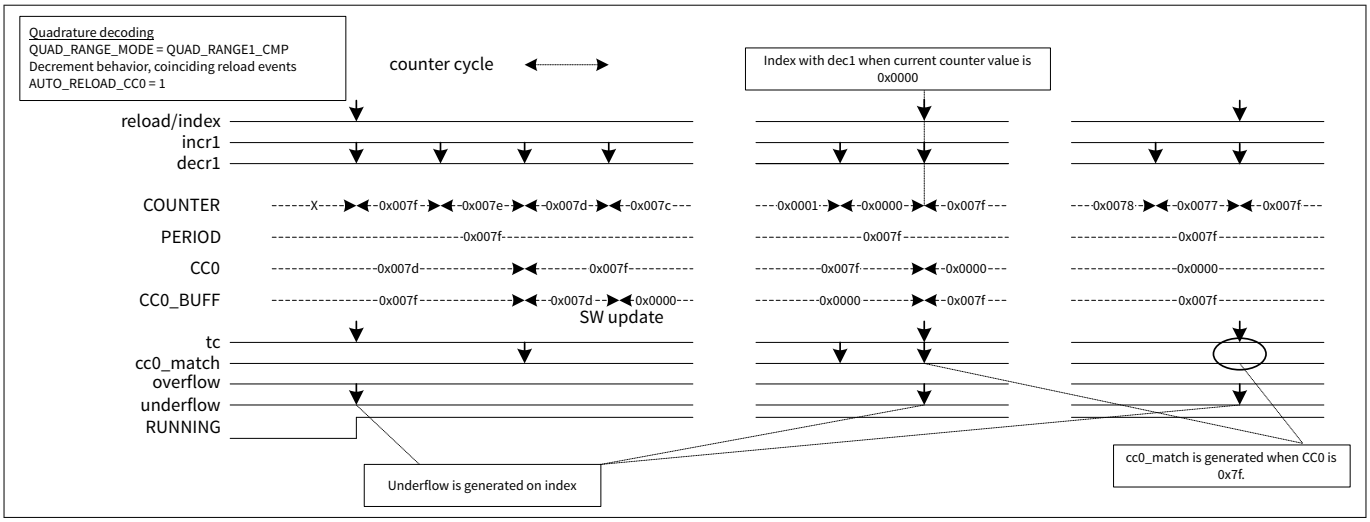


Figure 237 Index, decrement, underflow, and cc0_match coincide

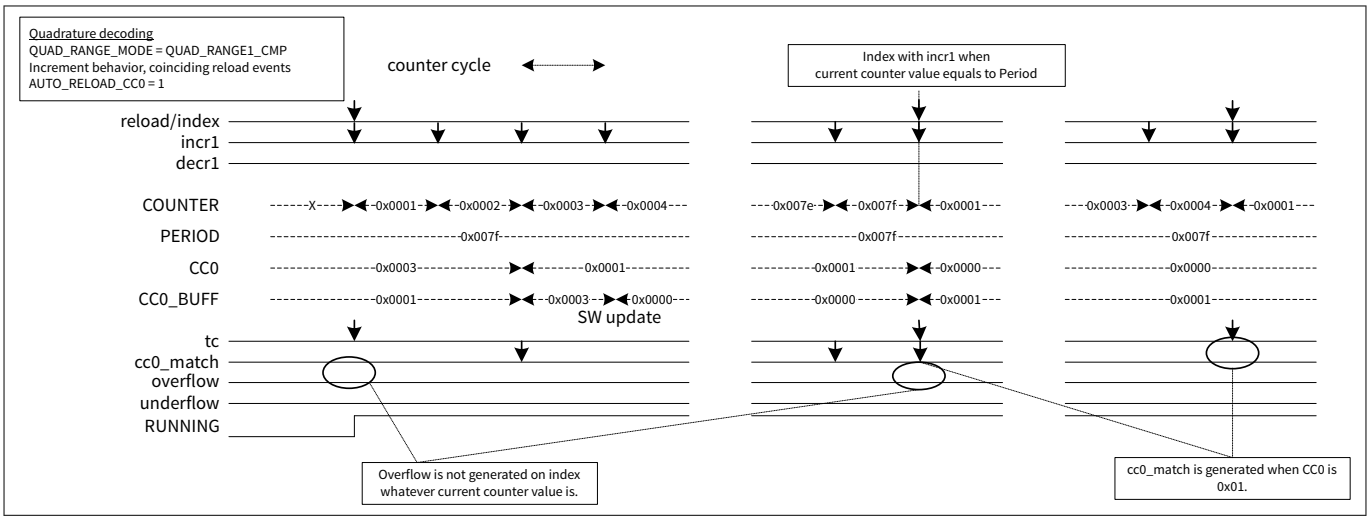


Figure 238 Index, increment, overflow, and cc0_match coincide

The QUAD_RANGE1_CMP functionality allows the COUNTER Register to reflect the current angle position of the rotary encoder; that is, no MOD or SUB calculations need to be done in software on the COUNTER value to get the current angle position. This allows a DMA copy of the current angle position from the COUNTER Register; for example, to send a buffer of CAN/UART/field bus interface to synchronize with other devices. However, a disadvantage of this mode is that fast sequences of tc interrupts can occur (when encoder moves back and forth around start position). It is recommended to not use the tc interrupt in this mode.

25.3.3.5 Quadrature QUAD_RANGE1_CAPT mode

In this mode the counter range is also between 0x0000 and PERIOD. Quadrature functionality in QUAD_RANGE1_CAPT mode provides the same functionality as the QUAD_RANGE1_CMP mode with the only difference that one or two capture functions are available instead of one or two compare functions.

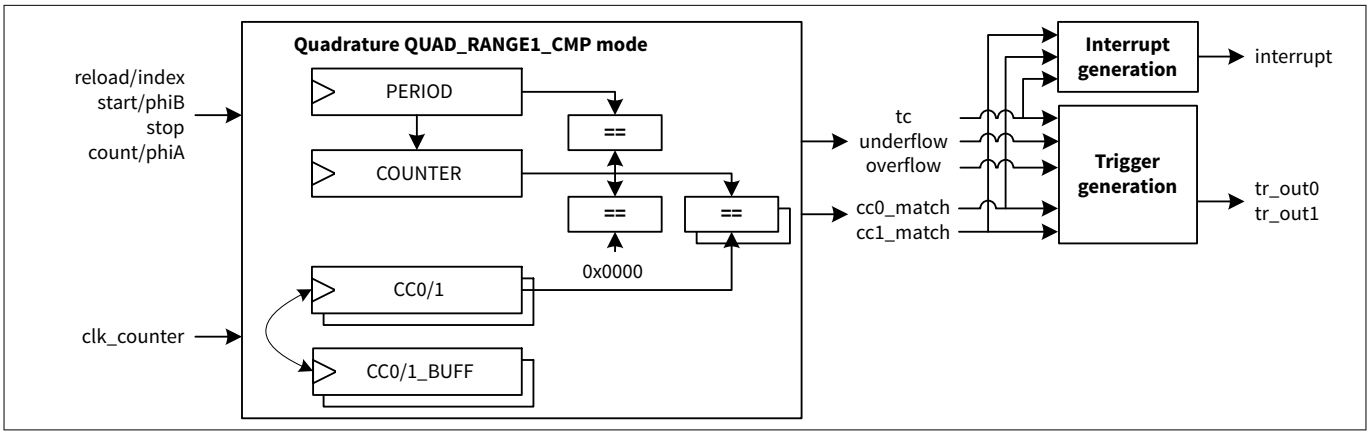


Figure 239 Quadrature (QUAD_RANGE1_CAPT) function diagram

Figure 240 illustrates an example scenario with decrementing counter and capture functionality.

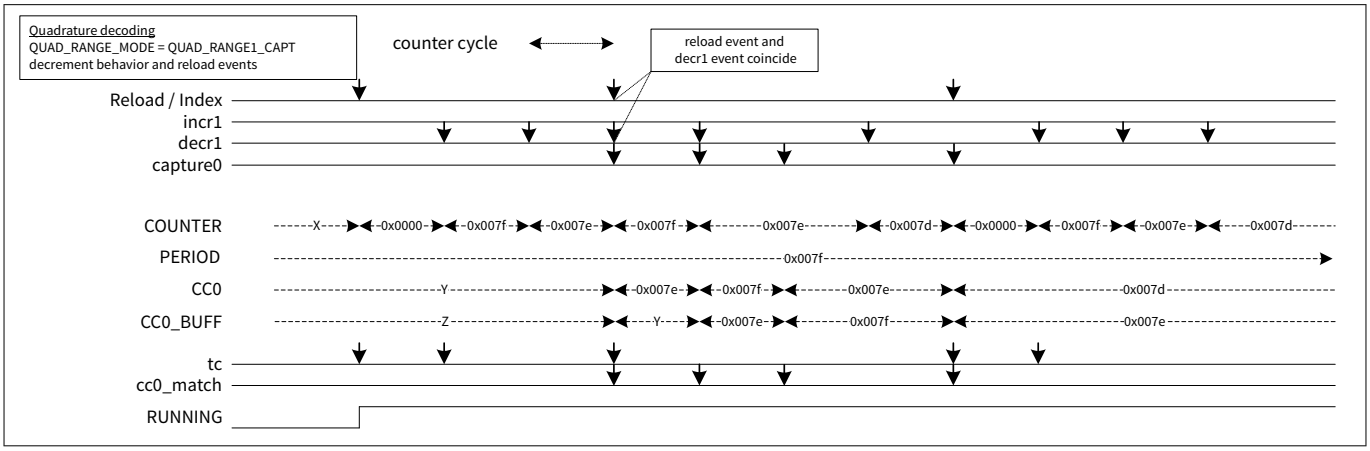


Figure 240 Quadrature (QUAD_RANGE1_CAPT) capture operation

25.3.4 Pulse width modulation (PWM) mode

PWM functionality increments/decrements a counter between 0 and PERIOD. When the counter is running, the counter value COUNTER is compared with CC0 (CC1). When COUNTER equals CC0 (CC1), the cc0_match (cc1_match) event is generated. Additionally, on a counter overflow and counter underflow, the overflow and underflow events are generated. Combined, the cc0_match, cc1_match, overflow, and underflow events are used to generate a pulse-width modulated signal on the PWM LINE_OUT and LINE_COMPL_OUT output signals. Left-aligned, right-aligned, and center-aligned PWM signals can be generated. Asymmetric PWM signals can be generated using the COUNT_UPDN2 mode. The current PWM output level can be read. A special case of 0 or 100 percent duty cycle is supported. The PERIOD_BUFF Register is used for duty cycle update and becomes active by a tc event.

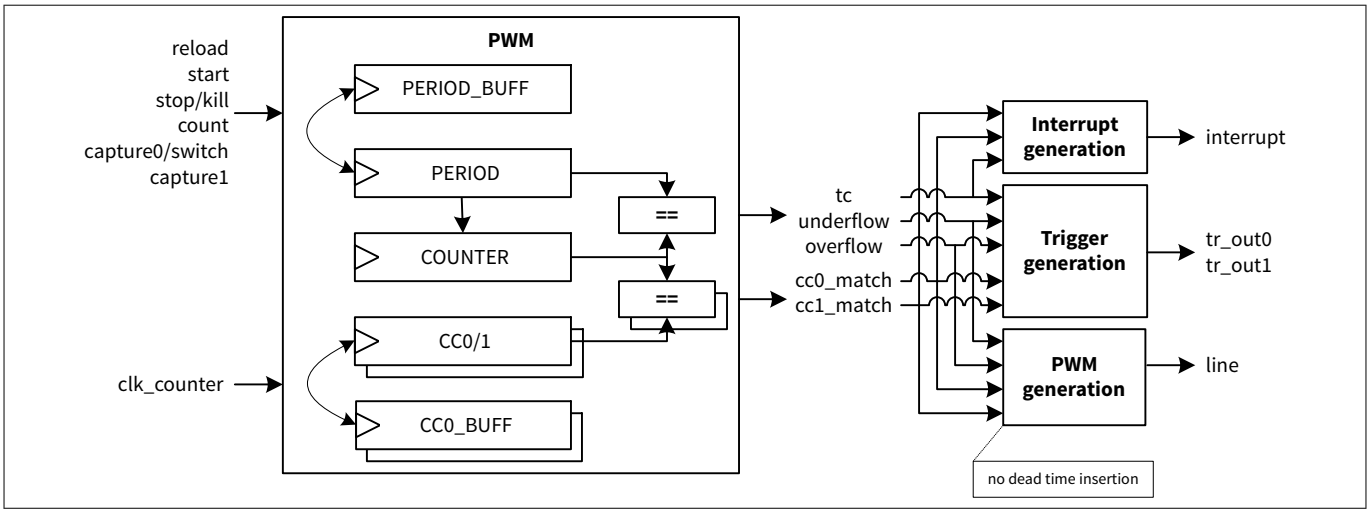


Figure 241 PWM function diagram

Table 217 PWM mode trigger input description

Trigger inputs	Usage
reload	Sets the counter value and starts the counter. Behavior is dependent on UP_DOWN_MODE: • COUNT_UP: The counter is set to '0' and count direction is set to 'up' • COUNT_DOWN: The counter is set to PERIOD and count direction is set to 'down' • COUNT_UPDN1/2: The counter is set to '1' and count direction is set to 'up' Can only be used when the counter is not running.
start	Starts the counter. The counter is not initialized by hardware. The current counter value is used. Behavior is dependent on UP_DOWN_MODE: • COUNT_UP: The count direction is set to 'up' • COUNT_DOWN: The count direction is set to 'down' • COUNT_UPDN1/2: The count direction is set to 'up' Note that when the counter is running, the start event has no effect. Can be used when the counter is running or not running.
stop/kill	Stops the counter. Different stop/kill modes exist.
count	Count event increments/decrements the counter.
Capture0	This event acts as a switch event. When this event is active, the CC0/CC0_BUFF, CC1/CC1_BUFF, PERIOD/PERIOD_BUFF, and LINE_SEL/LINE_SEL_BUFF Registers are exchanged on a tc event (when specified by AUTO_RELOAD_CC bit, AUTO_RELOAD_PERIOD bit, and AUTO_RELOAD_LINE_SEL bit in the CTRL Register). A switch event requires rising, falling, or rising/falling edge event detection mode. Pass-through mode is not supported, unless the selected event is a constant '0' or '1'. When a switch event is detected and the counter is running, the event is kept pending until the next tc event. The switch event will be cleared and has no effect if it is detected when counter is not running.

(table continues...)

Table 217 (continued) PWM mode trigger input description

Trigger inputs	Usage
Capture1 (stop1/ kill1)	This event acts as a second stop/kill event. It has the same function as the stop0/kill0 event. Both events are OR combined. ***Note**: Having two stop/kill events for a PWM allows selecting a common trigger for one stop/kill event from a PERI trigger multiplexer (allowing synchronous stop/ kill operation of multiple PWMs) while selecting a dedicated ADC out-of-range trigger for the other stop/kill event (allowing real-time hardware stop of a PWM when current PWM driven signal is out of range).*

Table 218 PWM mode supported features

Supported features	Description
Clock prescaling	Prescales the PCLK_TCPWM[x]_CLOCKS[y].
One shot	Counter is stopped by hardware, after a single period of the counter: • COUNT_UP: on an overflow event. • COUNT_DOWN and COUNT_UPDN1/2: on an underflow event.
Auto reload CC	CC0/1 and CC0/1_BUFF are exchanged on a switch event and tc event (when specified by AUTO_RELOAD_CC bit in CTRL Register).
Auto reload PERIOD	PERIOD and PERIOD_BUFF are exchanged on a switch event and tc event (when specified by CTRL.AUTO_RELOAD_PERIOD). ***Note**: When COUNT_UPDN2 mode exchanges PERIOD and PERIOD_BUFF at a tc event that coincides with an overflow event, software should ensure that the PERIOD and PERIOD_BUFF values are the same.*
Auto reload LINE_SEL	LINE_SEL and LINE_SEL_BUFF are exchanged on a switch event and tc event (when specified by the AUTO_RELOAD_LINE_SEL bit in the CTRL Register).
Up/down modes	Specified by UP_DOWN_MODE: • COUNT_UP: The counter counts from 0 to PERIOD • COUNT_DOWN: The counter counts from PERIOD to 0 • COUNT_UPDN1/2: The counter counts from 1 to PERIOD and back to 0
Kill modes	Specified by PWM_IMM_KILL, PWM_SYNC_KILL, and PWM_STOP_ON_KILL.

Note that the PWM mode does not support dead time insertion. This functionality is supported by the separate PWM_DT mode.

Table 219 PWM mode trigger output description

Trigger output	Description
cc0_match	Specified by UP_DOWN_MODE: • COUNT_UP and COUNT_DOWN: The counter changes to a state in which COUNTER equals CC0 • COUNT_UPDN1/2: counter changes from a state in which COUNTER equals CC0

(table continues...)

Table 219 (continued) PWM mode trigger output description

Trigger output	Description
cc1_match	Specified by UP_DOWN_MODE: • COUNT_UP and COUNT_DOWN: The counter changes to a state in which COUNTER equals CC1 • COUNT_UPDN1/2: counter changes from a state in which COUNTER equals CC1
Underflow (UN)	Counter is decrementing and changes from a state in which COUNTER equals 0. Reload event generate underflow in COUNT_DOWN, COUNT_UPDN1, or COUNT_UPDN2 mode.
Overflow (OV)	Counter is incrementing and changes from a state in which COUNTER equals PERIOD. Reload event generate overflow in COUNT_UP mode.
tc	Specified by UP_DOWN_MODE: • COUNT_UP: tc event is the same as the overflow event • COUNT_DOWN: tc event is the same as the underflow event • COUNT_UPDN1: tc event is the same as the underflow event • COUNT_UPDN2: tc event is the same as the logical OR of the overflow and underflow events Reload will generate underflow/overflow, but not the tc output trigger.

Table 220 PWM mode PWM outputs

PWM outputs	Description
LINE_OUT	PWM line output
LINE_COMPL_OUT	Complementary PWM line output

Note that the cc0_match event generation in COUNT_UP and COUNT_DOWN modes are different from the generation in other functional modes or counting modes. This is to ensure that 0 percent and 100 percent duty cycles can be generated.

PWM behavior depends on the PERIOD and CC0 Registers. Software can update the PERIOD_BUFF and CC0_BUFF Registers, without affecting the PWM behavior. The switch/capture event can be used to switch the values of the Compare and Buffered Compare Registers. It also switches the values of the Period and Buffered Period Registers. This is the main rationale for double buffering these registers. Table 221 summarizes the kill mode supported in PWM mode.

Table 221 Kill modes of PWM

Kill mode	Settings	Kill-behavior
No-IMM-Async	PWM_IMM_KILL = 0; PWM_SYNC_KILL = 0; PWM_STOP_ON_KILL = 0; STOP_EDGE = NO_EDGE_DET	PWM output is suppressed: At next active count clock after kill input is active. PWM output suppress is removed: At next active count clock after kill input is inactive
IMM-Async	PWM_IMM_KILL = 1; PWM_SYNC_KILL = 0; PWM_STOP_ON_KILL = 0; STOP_EDGE = NO_EDGE_DET	PWM output is suppressed: Immediately after kill input is active. PWM output suppress is removed: At next active count clock after kill input is inactive.

(table continues...)

Table 221 (continued) Kill modes of PWM

Kill mode	Settings	Kill-behavior
No-IMM-Sync	PWM_IMM_KILL = 0; PWM_SYNC_KILL = 1; PWM_STOP_ON_KILL = 0; STOP_EDGE = RISING	PWM output is suppressed: At next active count clock after kill input is active. PWM output suppress is removed: At next tc event after kill input is inactive.
IMM-Sync	PWM_IMM_KILL = 1; PWM_SYNC_KILL = 1; PWM_STOP_ON_KILL = 0; STOP_EDGE = RISING	PWM output is suppressed: Immediately after kill input is active. PWM output suppress is removed: At next tc event after kill input is inactive.
No-IMM-Stop	PWM_IMM_KILL = 0; PWM_SYNC_KILL = Don't care; PWM_STOP_ON_KILL = 1; STOP_EDGE= RISING_EDGE/ FALLING_EDGE/BOTH_EDGES	PWM output is suppressed: At next active count clock after kill input is active. PWM output suppress is removed: Counter restart after kill input is inactive.
IMM-Stop	PWM_IMM_KILL = 1; PWM_SYNC_KILL = Don't care; PWM_STOP_ON_KILL = 1; STOP_EDGE= RISING_EDGE/ FALLING_EDGE/BOTH_EDGES	PWM output is suppressed: Immediately after kill input is active. PWM output suppress is removed: Counter restart after kill input is inactive.

25.3.4.1 PWM mode functionalities

***Note**: One-shot mode and clock prescaling are the same as in timer mode.*

Up/down count modes

Up/down count modes control the counting direction (increment or decrement) while counter is running.
Figure 242 illustrates a PWM in COUNT_UP mode. The counter is initialized (to '0') and started with a software-based reload event.

***Note**:*
- *When the counter changes from a state in which COUNTER is '4', an overflow and tc event are generated.*
- *When the counter changes to a state in which COUNTER equals '2', a cc0_match event is generated.*
- *PERIOD is 4, resulting in an effective repeating counter pattern of 4+1 = 5 counter clock periods.*

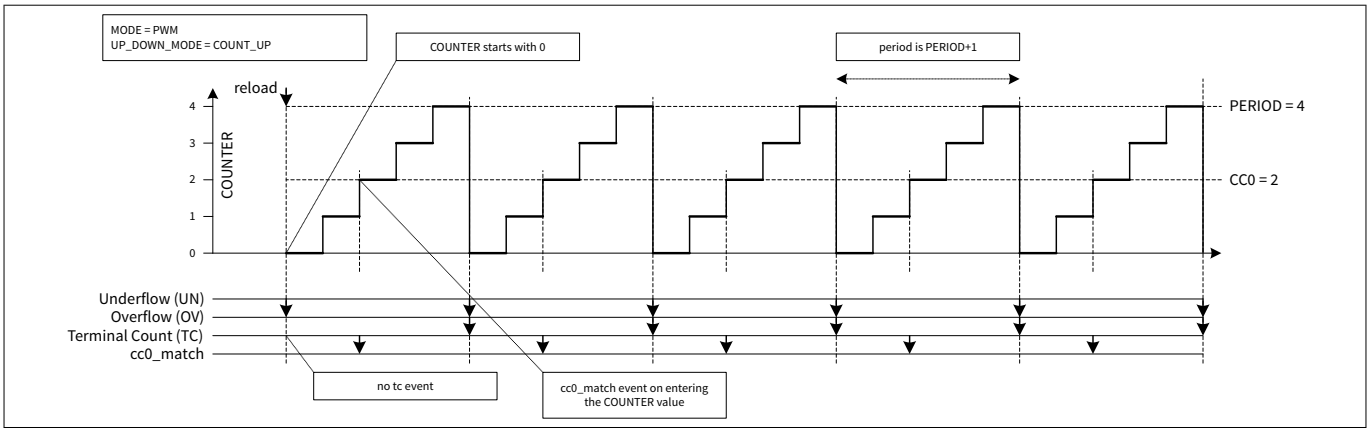


Figure 242 PWM in up counting mode

Figure 243 illustrates a PWM in down counting mode. The counter is initialized (to' PERIOD') and started with a software-based reload event.

Note:
- *When the counter changes from a state in which COUNTER is '0', an underflow and tc event are generated.*
- *When the counter changes to a state in which COUNTER is '2', a cc0_match event is generated.*
- *PERIOD is 4, resulting in an effective repeating counter pattern of 4+1 = 5 counter clock periods.*

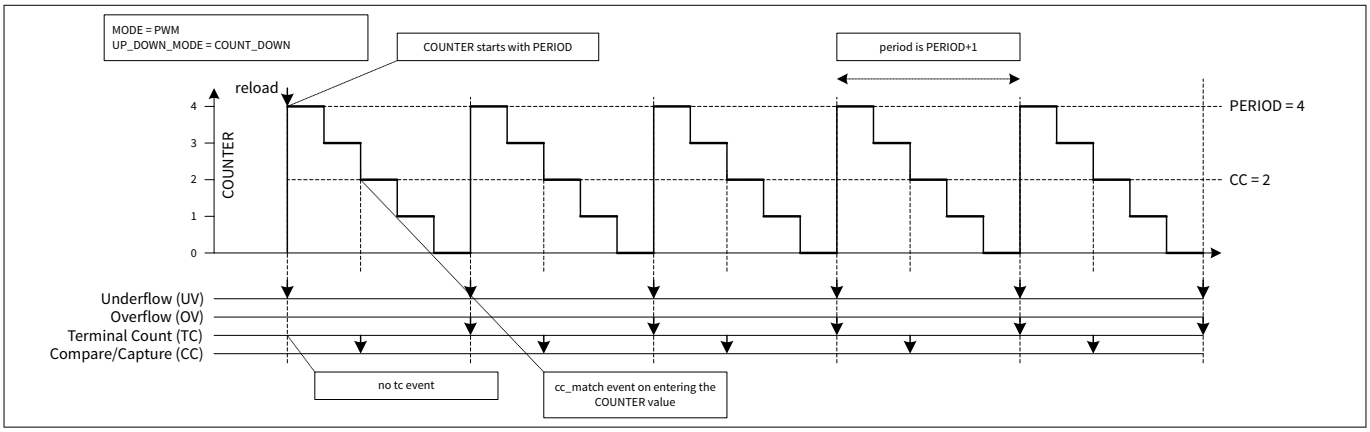


Figure 243 PWM in down counting mode

Figure 244 illustrates a PWM in up/down counting mode. The counter is initialized (to '1') and started with a software-based reload event.

Note:
- *When the counter changes from a state in which COUNTER is '4', an overflow is generated.*
- *When the counter changes from a state in which COUNTER is '0', an underflow and tc event are generated.*
- *When the counter changes from a state in which COUNTER is '2', a cc0_match event is generated. Note that the actual counter value COUNTER from before the reload event is not used, instead the counter value before the reload event is considered to be 0.*
- *PERIOD is 4, resulting in an effective repeating counter pattern of 2 × 4 = 8 counter clock periods.*

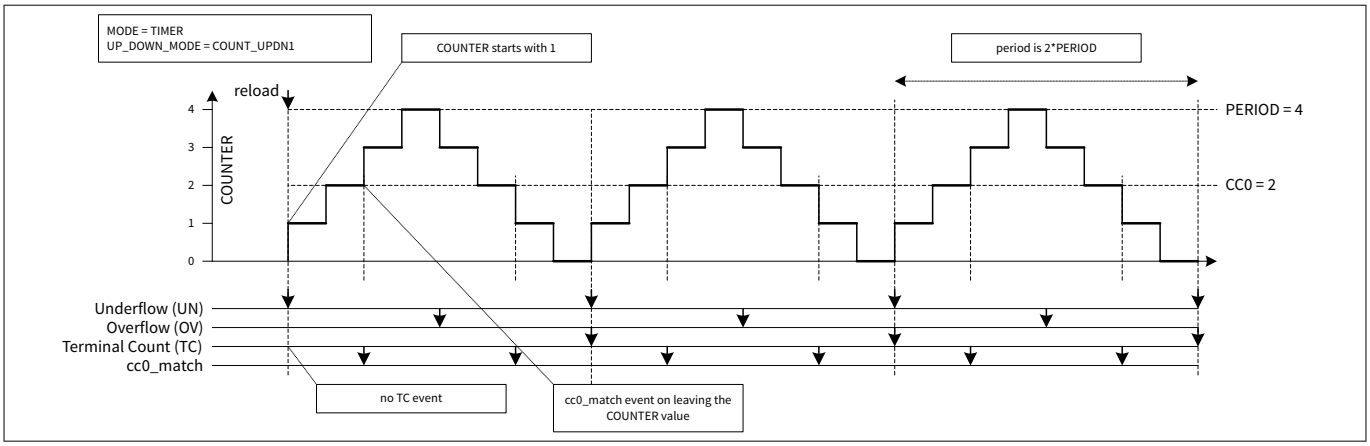


Figure 244 Up/down counting PWM

PWM output lines

PWM provides two output lines

- A PWM LINE_OUT output signal.
- A complementary PWM LINE_COMPL_OUT output signal.

The generation of the PWM output signals is a multi-step process and is illustrated as follows.

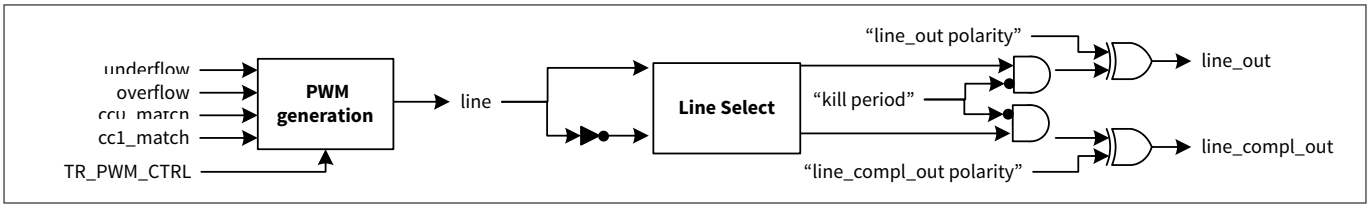


Figure 245 Line generation logic

TR_PWM_CTRL is to control the line state change per four internal events.

Dead time insertion is not supported in PWM mode, only the PWM_DT mode has this feature.

Kill input will disable both LINE_OUT and LINE_COMPL_OUT. The kill mode is specified by PWM_IMM_KILL, PWM_STOP_ON_KILL, and PWM_SYNC_KILL.

The polarity of both LINE_OUT signals can be configured in the CTRL Register. The QUAD_ENCODING_MODE [0] bit sets the polarity of LINE_OUT; QUAD_ENCODING_MODE [1] bit can be used to set the polarity of LINE_COMPL_OUT. The value '1' inverts the corresponding LINE_OUT signal. Note that the polarity configuration must be done only when the counter is not enabled or running.

CC0 and PERIOD auto reload with switch event

Auto CC reload and auto PERIOD reload will provide dynamic PWM duty cycle change and count period length change. The active switch event (capture0) is required for switch activity:

- At TC event, if switch event is active and AUTO_RELOAD_CC bit of the CTRL Register is set to '1', CC0/1 and CC0/1_BUFF will exchange value.
- At TC event, if switch event is active and AUTO_RELOAD_PERIOD bit of CTRL Register is set to '1', PERIOD and PERIOD_BUFF will exchange value.

The following figures illustrate the update of period value in COUNT_UP mode and COUNT_DOWN mode resulting in different period times after each switch event.

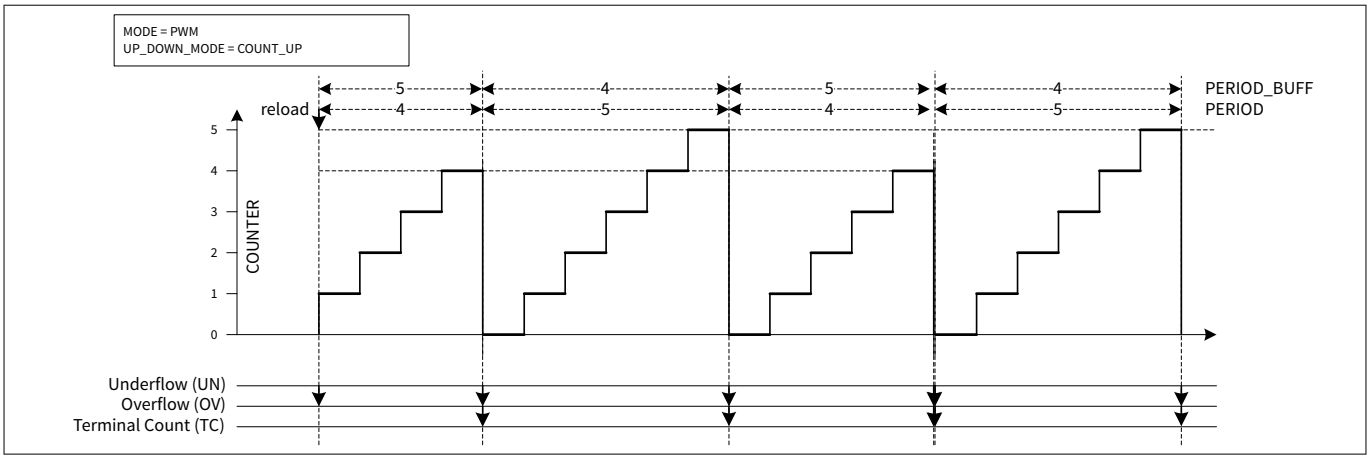


Figure 246 PERIOD/PERIOD_BUFF exchange in COUNT_UP Mode by a switch event

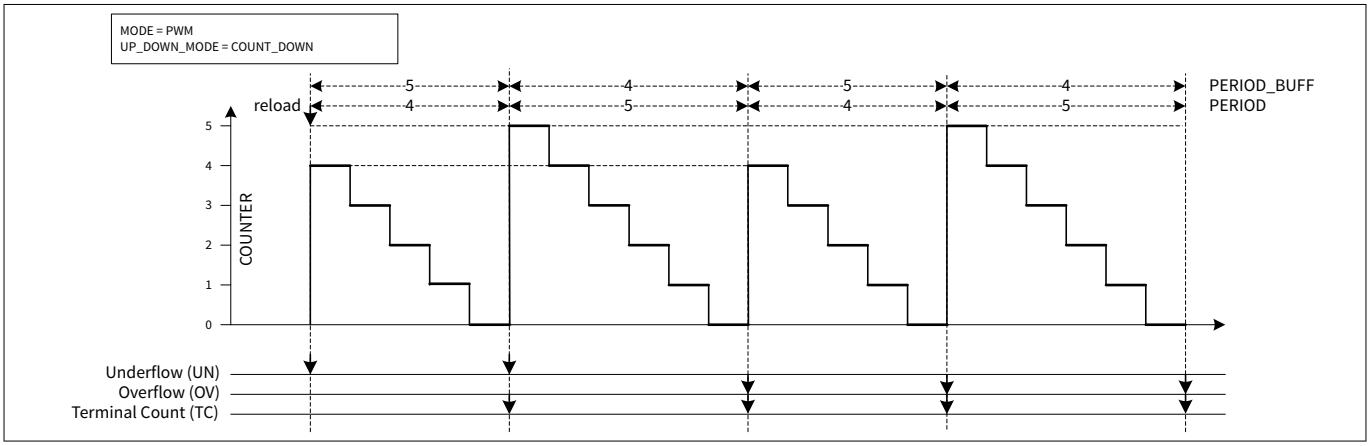


Figure 247 PERIOD/PERIOD_BUFF exchange in COUNT_DOWN mode by a switch event

A potential problem arises when software updates are not completed before the next tc event with an active pending switch event. For example, if software updates PERIOD_BUFF before the tc event and CC0_BUFF after the tc event, switching does not reflect the CC0_BUFF Register update. To prevent this from happening, the switch event should be generated by software through a MMIO Register write after both the PERIOD_BUFF and CC0_BUFF Registers are updated. The switch event is kept pending by the hardware until the next tc event occurs.

Left/right/center align PWM with CC0/CC0_BUFF auto reload

PWM can generate left-align, right-align. and center-align with the following features supported in PWM mode:

- Up/down count mode must be used to generate different phase aligned PWM.
- Line state is changed per underflow/overflow/cc0_match/cc1_match internal event and can be configured in TR_PWM_CTRL Register.

The required settings for left-aligned, right-aligned, and center-aligned PWM are:

- Left-align:
  - Write the value '0' to UP_DOWN_MODE [17:16] field in the CTRL Register to set the counter direction to COUNT_UP mode
  - Write the value '0' (SET) to OVERFLOW_MODE [3:2] field of the TR_PWM_CTRL Register to set the LINE_OUT signal to '1' when the COUNTER reaches PERIOD value.
  - Write the value '1' (CLEAR) to CC0_MATCH_MODE [1:0] field in the TR_PWM_CTRL Register to set the LINE_OUT signal to '0' when the COUNTER equals CC0 value.

- Right-align:
  - Write the value '1' to UP_DOWN_MODE [17:16] field in the CTRL Register to set the counter direction to COUNT_DOWN mode
  - Write the value '0' (SET) to CC0_MATCH_MODE [1:0] field in the TR_PWM_CTRL Register to set the LINE_OUT signal to '1' when the COUNTER equals CC0 value.
  - Write the value '1' (CLEAR) to UNDERFLOW_MODE [5:4] field in the TR_PWM_CTRL Register to set the LINE_OUT signal to '0' when the COUNTER reaches '0'.
- Center-align:
  - Write the value '2' to UP_DOWN_MODE [17:16] field in the CTRL Register to set the counter direction to COUNT_UPDN1 mode
  - Write the value '0' (SET) to OVERFLOW_MODE [3:2] field in the TR_PWM_CTRL Register to set the LINE_OUT signal to '1' when the COUNTER reaches PERIOD value.
  - Write the value '1' (CLEAR) to UNDERFLOW_MODE [5:4] field in the TR_PWM_CTRL Register to set the LINE_OUT signal to '0' when the COUNTER reaches '0'.
  - Write the value '2' (INVERT) to CC0_MATCH_MODE [1:0] field in the TR_PWM_CTRL Register to invert the LINE_OUT signal when the COUNTER equals CC0 value.

Figure 248 illustrates a PWM in COUNT_UP mode with different CC0 values. The figure also illustrates how a left-aligned and right-aligned PWM can be created using the PWM in COUNT_UP mode.

***Note**: CC0 is changed (to CC0_BUFF, which is not depicted) on a tc event. The switch event is a constant 1.*

A special case of 0 or 100 percent duty cycle is realized using the following setting (for example, left-aligned):

- 0 percent → CC0/1 = 0
- 100 percent → CC0/1 > PERIOD (a PERIOD value of 0xFFFF/0xFFFFFFFF is restricted)

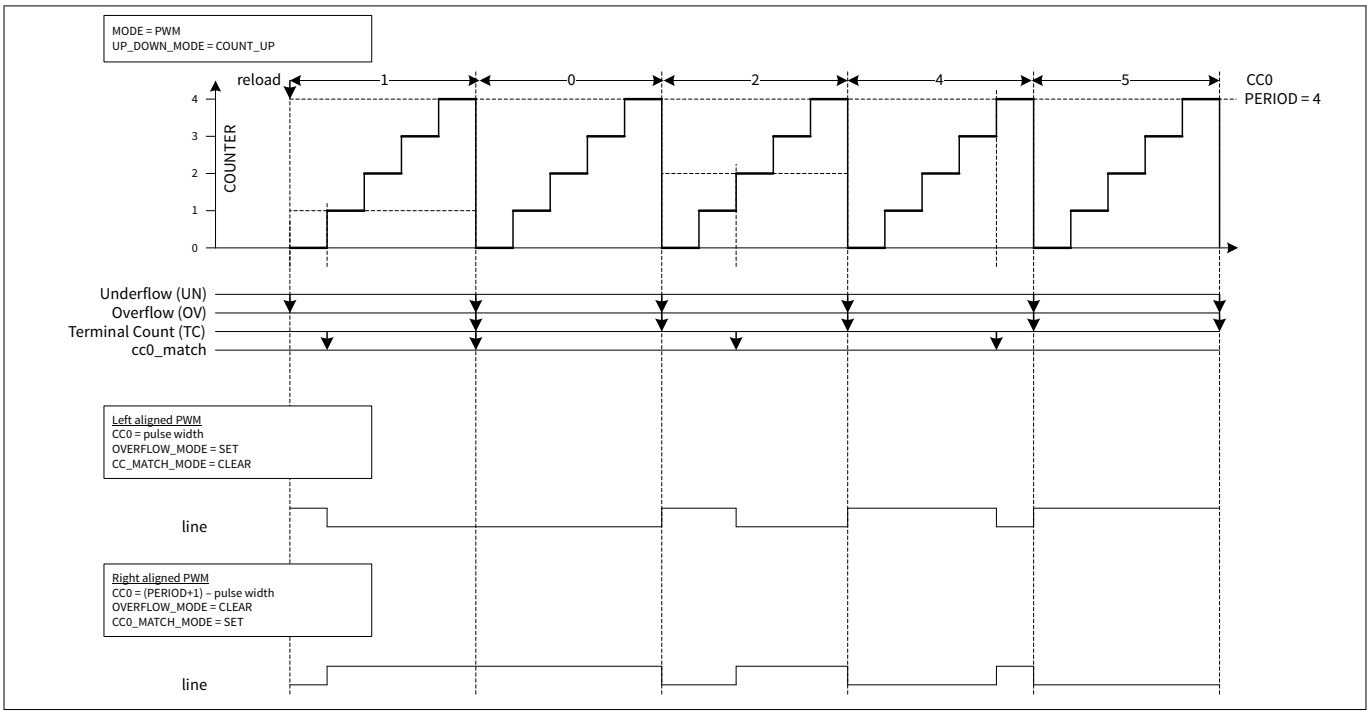


Figure 248 Left- and right-aligned PWM in COUNT_UP mode

Figure 249 illustrates a PWM in COUNT_DOWN mode with different CC0 values. The figure also illustrates how a right-aligned PWM can be created using the PWM in COUNT_DOWN mode.

***Note**: CC0 is changed (to CC0_BUFF, which is not depicted) on a tc event. The switch event is a constant 1.*

A special case of 0 or 100 percent duty cycle is realized using the following setting (for example, right-aligned):

- 0 percent → CC0/1 > PERIOD (a PERIOD value of 0xFFFF/0xFFFFFFFF is restricted)
- 100 percent → CC0/1 = PERIOD

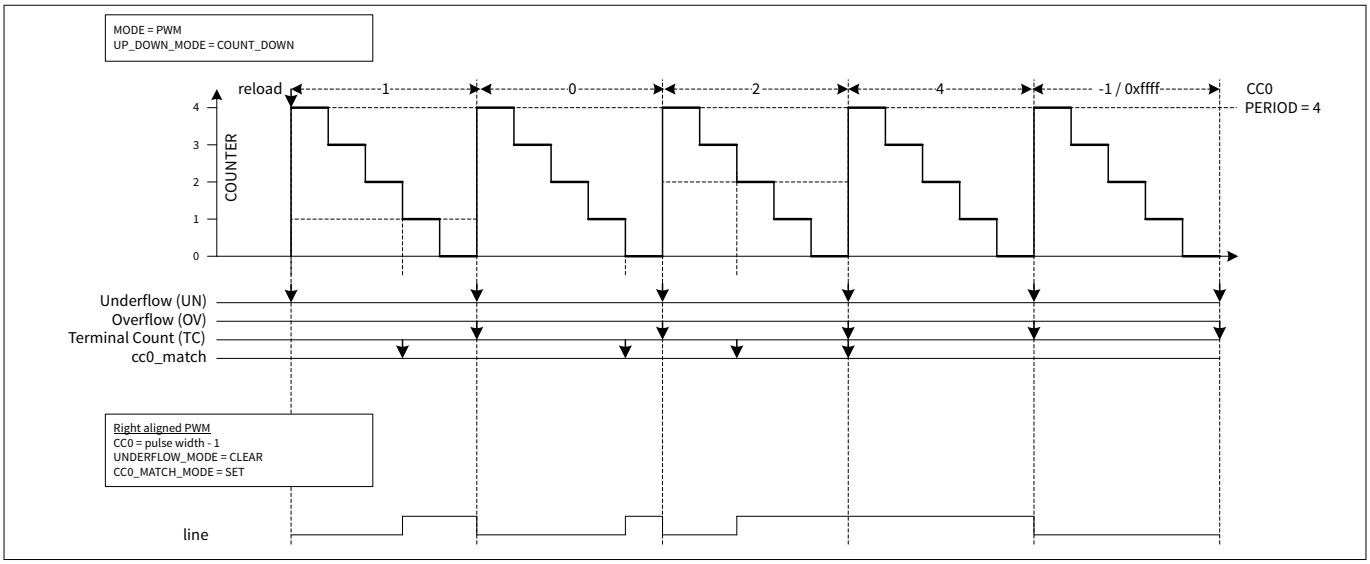


Figure 249 Right-aligned PWM in COUNT_DOWN mode

Figure 250 illustrates a PWM in COUNT_UPDN1 with different CC0 values. The figure also illustrates how a center-aligned PWM can be creating using the PWM in COUNT_UPDN1 mode.

***Note**: The switch event is generated by hardware trigger 1, which is a constant '1' and therefore always active at the TC condition. CC0 is changed (to CC0_BUFF, which is not depicted) on a tc event.*

A special case of 0 or 100 percent duty cycle is realized using the following setting (for example, center-aligned):

- 0 percent → CC0/1 = PERIOD (there is no restriction for PERIOD)
- 100 percent → CC0/1 = 0

cc0_match will generate at the beginning of the count period when the CC0 switches to '0'. In a special case, the cc0_match generates without COUNTER equals CC0 (CC0 changes to 0 and at the same time COUNTER changes to 1 from 0).

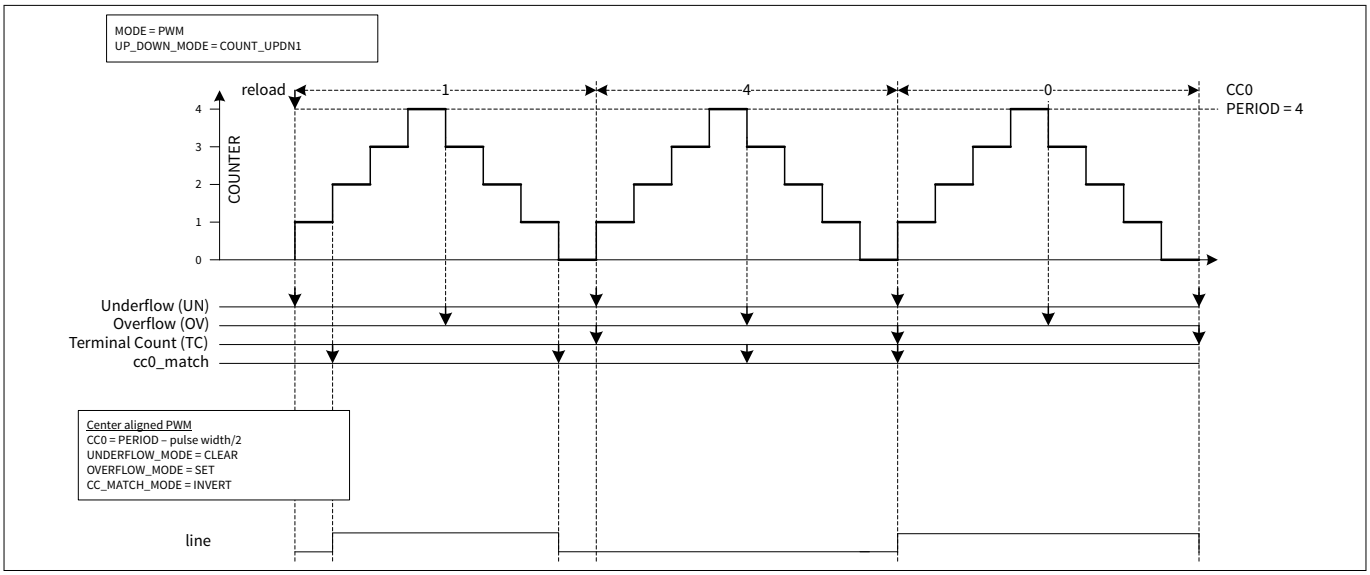


Figure 250 Center-align PWM in UPDN1 mode

Figure 251 shows another corner case that CC0 equals '0' when reload event comes. The actual counter value before the reload event is not used, instead the counter value before the reload event is considered to be '0'. As a result, when the first CC0 value at the reload event is '0', a cc0_match event is generated.

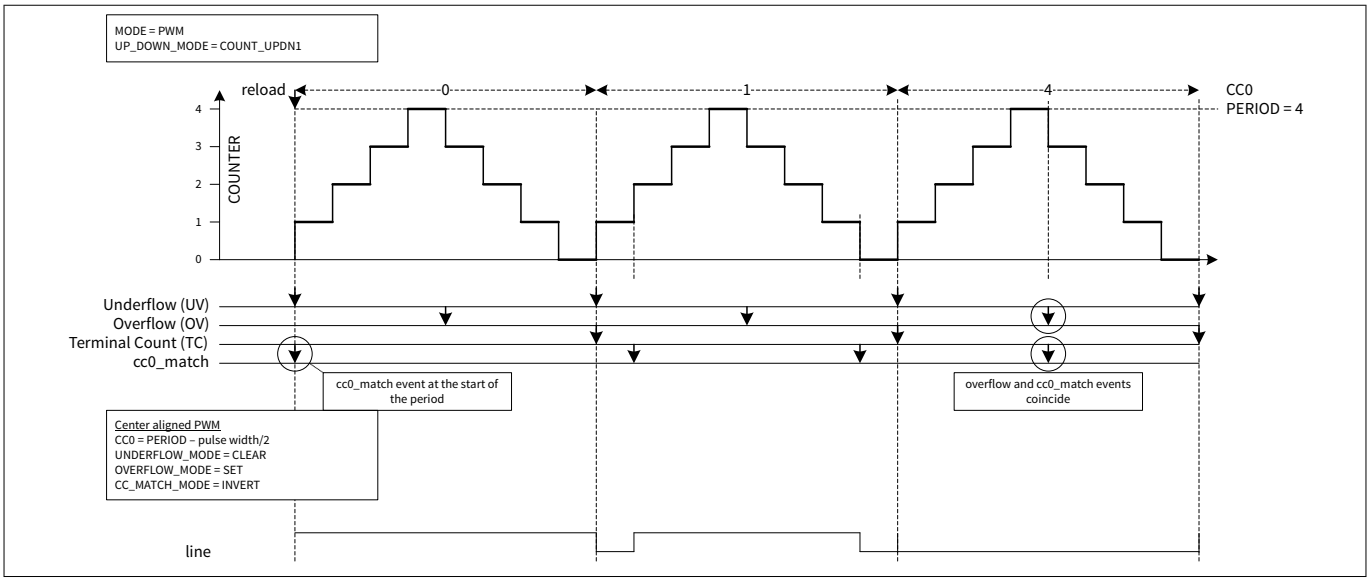


Figure 251 Center-align PWM with CC0 = 0 after reload

Asymmetric PWM

The PWM mode supports the generation of an asymmetric PWM. For an asymmetric PWM, the "line" pulse is not necessarily centered in the middle of the period. This functionality is realized by having a different CC0 value when counting up and when counting down. The CC0 and CC0_BUFF values are exchanged on an overflow event. Note that this restricts the asymmetry of the generated "line" pulse.

The COUNT_UPDN2 mode should use the same period value when counting up and counting down. When PERIOD and PERIOD_BUFF are switched on a tc event (overflow or underflow event), ensure the following:

- Within a PWM period (tc event coincides with an overflow event), the period values are the same (an overflow switch of PERIOD and PERIOD_BUFF should not change the period value; that is, PERIOD_BUFF should be PERIOD)
- Between PWM periods (tc event coincides with an underflow event), the period value can change (an underflow switch of PERIOD and PERIOD_BUFF may change the period value; that is, PERIOD_BUFF may be different from PERIOD).

Figure 252 illustrates how the COUNT_UPDN2 mode is used to generate an asymmetric PWM.

***Note**:*
- *When up counting and the CC0 value at the underflow event is '0', a cc0_match event is generated.*
- *When down counting and the CC0 value at the overflow event is PERIOD, a cc0_match event is generated.*
- *A tc event is generated for both an underflow and overflow event. The tc event is used to exchange the CC0 and CC0_BUFF values.*

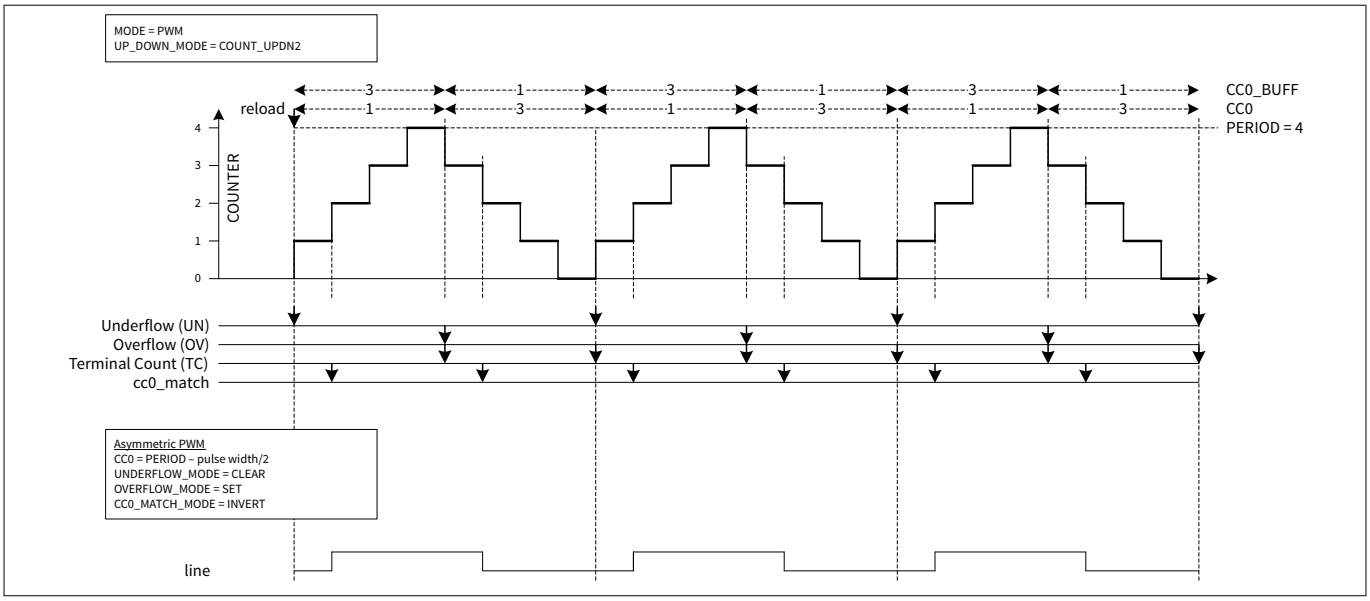


Figure 252 Asymmetric PWM

The previous waveform illustrated functionality when the CC values are neither 0 not PERIOD. Corner case conditions in which the CC values equal 0 or PERIOD are illustrated in the following figures.

Figure 253 illustrates how the COUNT_UPDN2 mode is used to generate an asymmetric PWM.

***Note**:*

- *When up counting and CC0 value at the underflow event is '0', a cc0_match event is generated.*
- *When down counting and CC0 value at the overflow event is PERIOD, a cc0_match event is generated.*
- *A tc event is generated for both an underflow and overflow event. The tc event is used to exchange the CC0 and CC0_BUFF values.*
- *Software updates CC0_BUFF and PERIOD_BUFF in an interrupt handler on the tc event (and overwrites the hardware updated values from the CC0/CC0_BUFF and PERIOD/PERIOD_BUFF exchanges).*

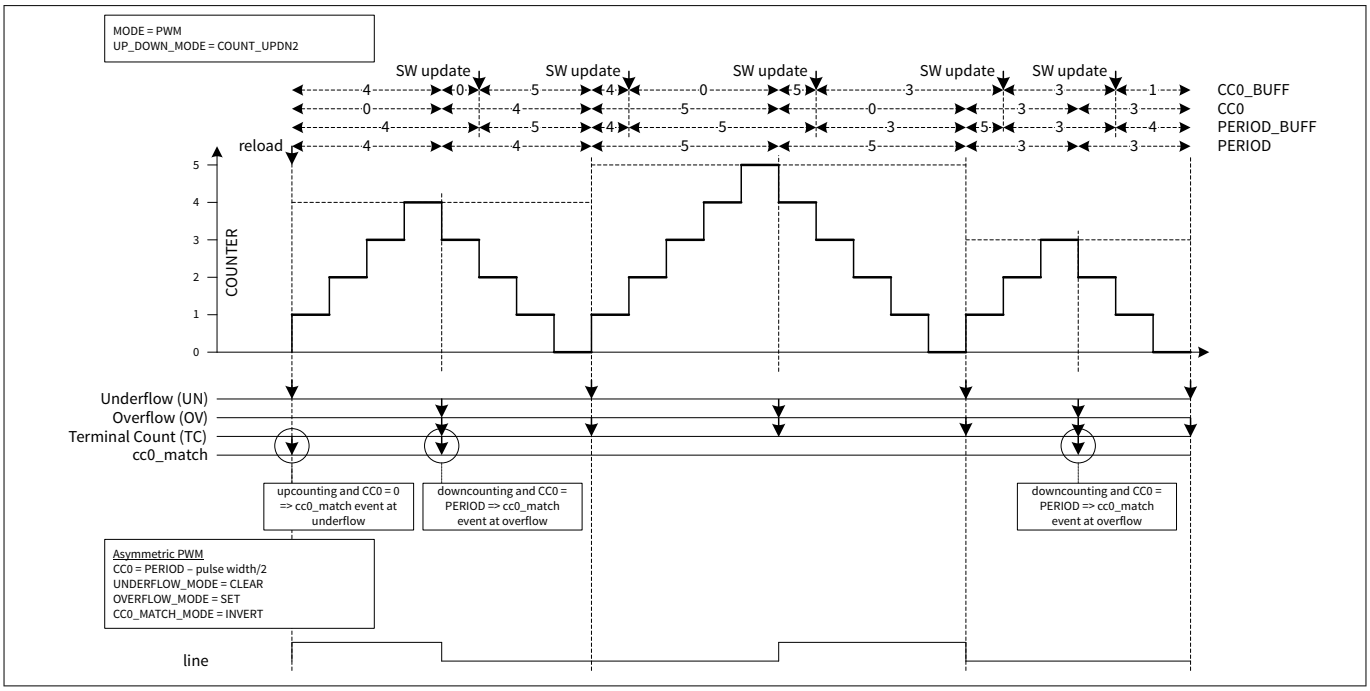


Figure 253 Asymmetric PWM when compare = 0 or period

When the counter group includes also compare function 1 with Registers CC1 and CC1_BUFF, which generate cc1_match event, the compare feature behaves the same as for compare 0 function.

The cc1_match event can also be used to generate the PWM output signals. Using both cc0_match and cc1_match events for PWM output signal generation provides another way to generate an asymmetric PWM as shown in Figure 254.

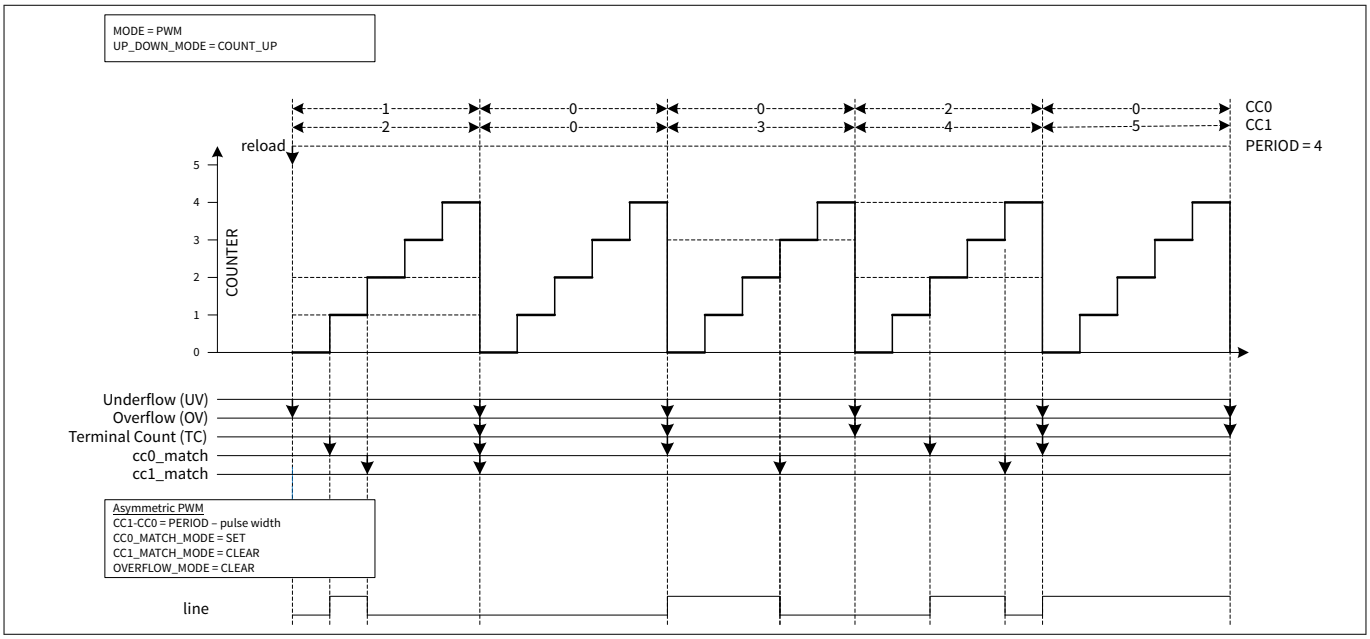


Figure 254 Asymmetric PWM with cc0_match and cc1_match

Such asymmetric PWM generation is more flexible than using only one compare function in the COUNT_UPDN2 mode. However, if another (third) compare function is needed, For example, to trigger an ADC, another synchronously running counter should be used.

For advanced motor control, the generation of compare match 0 and compare match 1 events can be enabled or disabled individually for up and down counting (during COUNT_UPDN1/2 mode). This allows asymmetric PWM generation in COUNT_UPDN1 mode where one compare match event modifies the PWM output only while counting up and the other compare match event modifies the PWM output only while counting down. This is illustrated in the following figure, which shows one of three center-aligned PWM phases for motor control when the duty cycle value is increased from one period to the next (rising part of sign wave modulated onto the PWM signal).

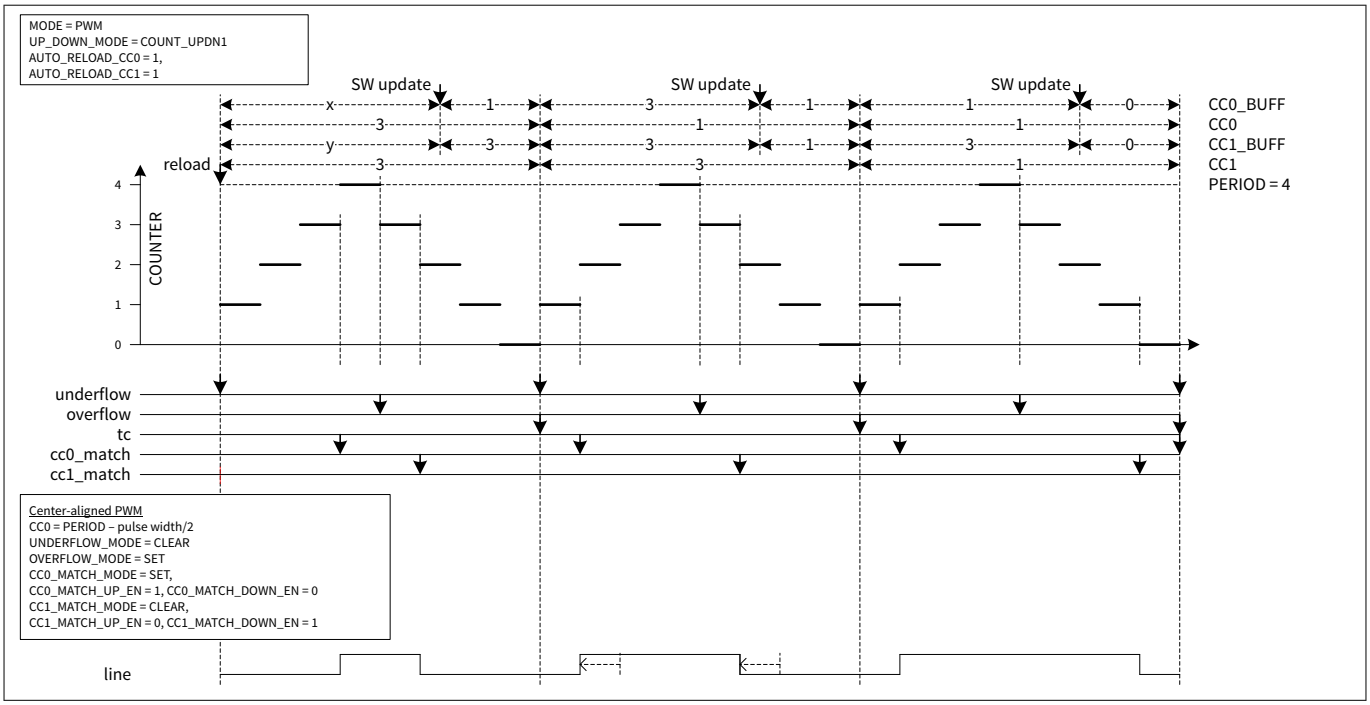


Figure 255 Asymmetric PWM by cc0_match and cc1_match in COUNT_UPDN1 mode

Instead of an always center-aligned PWM, the phase of the PWM signal can be temporarily shifted to allow a single shunt current measurement (current measurement at two triggers with difference calculation in software) for motor control of a permanent-magnet synchronous motor (PMSM) when the current duty cycle values of the three phases do not allow that (too small window where one PWM channel is active and two others are not). Compared to the asymmetric PWM realized with only one compare function in the COUNT_UPDN2 mode this solution uses two independent buffered compare values and generates less CPU load (less interrupts needed). This means all updates can be done for example, in the 'ADC done' interrupt service routine calculating the new duty cycle values and introducing a temporary phase shift for single shunt current measurement.

The required settings for typical, asymmetric PWM output modes are:

- Asymmetric with CC0:
  - Write the ‘3’ to the UP_DOWN_MODE [17:16] field in the CTRL Register to set the counter direction to COUNT_UPDN2 mode.
  - Write ‘0’ (SET) to the OVERFLOW_MODE [3:2] field in the TR_PWM_CTRL Register to set the LINE_OUT signal to '1' when the COUNTER reaches PERIOD value.
  - Write ‘1’ (CLEAR) to the UNDERFLOW_MODE [5:4] field in the TR_PWM_CTRL Register to set the LINE_OUT signal to '0' when the COUNTER reaches '0'.
  - Write ‘2’ (INVERT) to the CC0_MATCH_MODE [1:0] field in the TR_PWM_CTRL Register to invert the LINE_OUT signal when the COUNTER equals CC0 value.
- Asymmetric with CC0 and CC1 (only for counter groups with a second compare function):
  - Write ‘0’ to the UP_DOWN_MODE [17:16] field in the CTRL Register to set the counter direction to COUNT_UP mode.
  - Write ‘0’ (SET) to the CC0_MATCH_MODE [1:0] field in the TR_PWM_CTRL Register to set the LINE_OUT signal to '1' when the COUNTER equals CC0 value.
  - Write ‘1’ (CLEAR) to the CC1_MATCH_MODE [7:6] field in the TR_PWM_CTRL Register to set the LINE_OUT signal to '0' when the COUNTER equals CC1 value.
- Center-align asymmetric with CC0 and CC1 (only for counter groups with a second compare function):
  - Write ‘2’ to the UP_DOWN_MODE [17:16] field in the CTRL Register to set the counter direction to COUNT_UPDN1 mode.

- Write ‘0’ (SET) to the OVERFLOW_MODE [3:2] field in the TR_PWM_CTRL Register to set the LINE_OUT signal to '1' when the COUNTER reaches PERIOD value.
- Write ‘1’ (CLEAR) to the UNDERFLOW_MODE [5:4] field in the TR_PWM_CTRL Register to set the LINE_OUT signal to '0' when the COUNTER reaches '0'.
- Write ‘0’ (SET) to the CC0_MATCH_MODE [1:0] field in the TR_PWM_CTRL Register to set the LINE_OUT signal to '1' when the COUNTER equals CC0 value.
- Write ‘1’ (CLEAR) to the CC1_MATCH_MODE [7:6] field in the TR_PWM_CTRL Register to set the LINE_OUT signal to '0' when the COUNTER equals CC1 value.

Kill mode

PWM mode has different stop/kill modes. The mode is specified by PWM_IMM_KILL, PWM_STOP_ON_KILL, and PWM_SYNC_KILL.

- PWM_IMM_KILL is '1'. The PWM output signals DT_LINE_OUT and DT_LINE_COMPL_OUT are immediately suppressed when a kill event is detected.
- PWM_IMM_KILL is '0'. The PWM output signals DT_LINE_OUT and DT_LINE_COMPL_OUT are suppressed synchronously with the next count clock after a kill event is detected.

Figure 256 and Figure 257 illustrate both configurations.

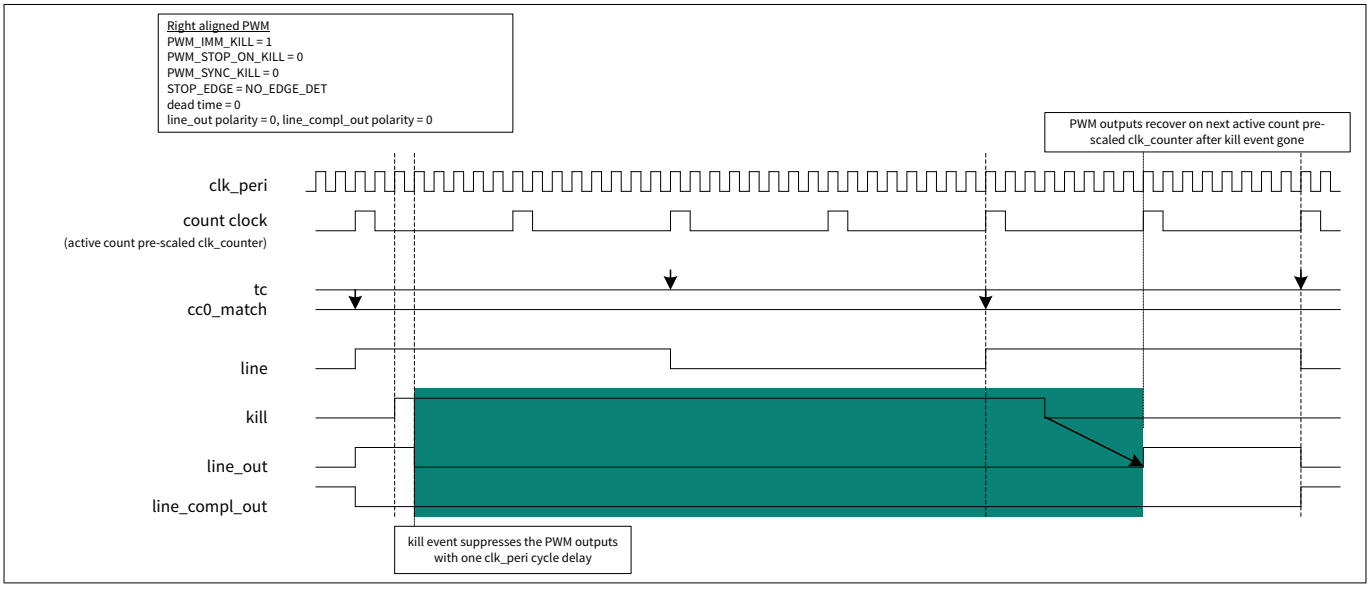


Figure 256 Kill suppresses line output immediately (PWM_IMM_KILL =1)

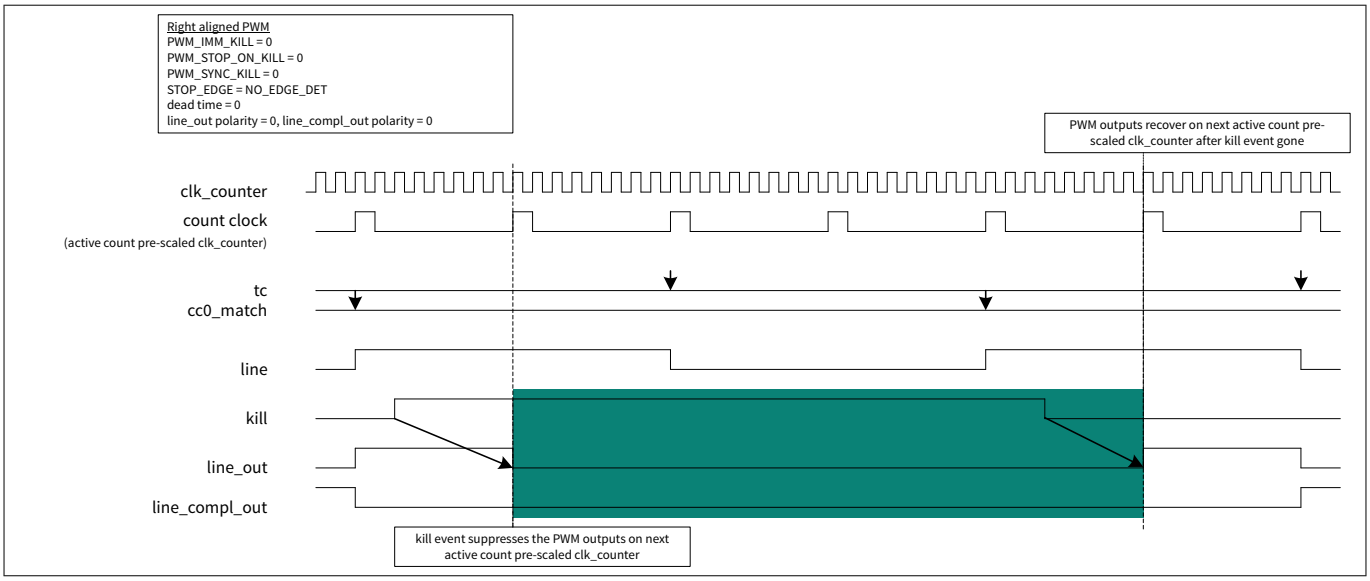


Figure 257 Kill suppresses line output by count clock (PWM_IMM_KILL = 0)

The PWM_STOP_ON_KILL and PWM_SYNC_KILL modes specifies the functionality of kill input. The following three modes are supported:

- PWM_STOP_ON_KILL is '1' (PWM_SYNC_KILL is don't care). This mode stops the counter on a stop/kill event.
- PWM_STOP_ON_KILL is '0' and PWM_SYNC_KILL is '0'. This mode keeps the counter running, but suppresses the PWM output signals synchronously with the next count clock (active count prescaled PCLK_TCPWM[x]_CLOCKS[y]) and continues to do so for the duration of the stop/kill event.
- PWM_STOP_ON_KILL is '0' and PWM_SYNC_KILL is '1'. This mode keeps the counter running, but suppresses the PWM output signals synchronously with the next count clock (active count prescaled PCLK_TCPWM[x]_CLOCKS[y]) and continues to do so until the next tc event without a stop/kill event.

Figure 258, Figure 259, and Figure 260 illustrate these three modes.

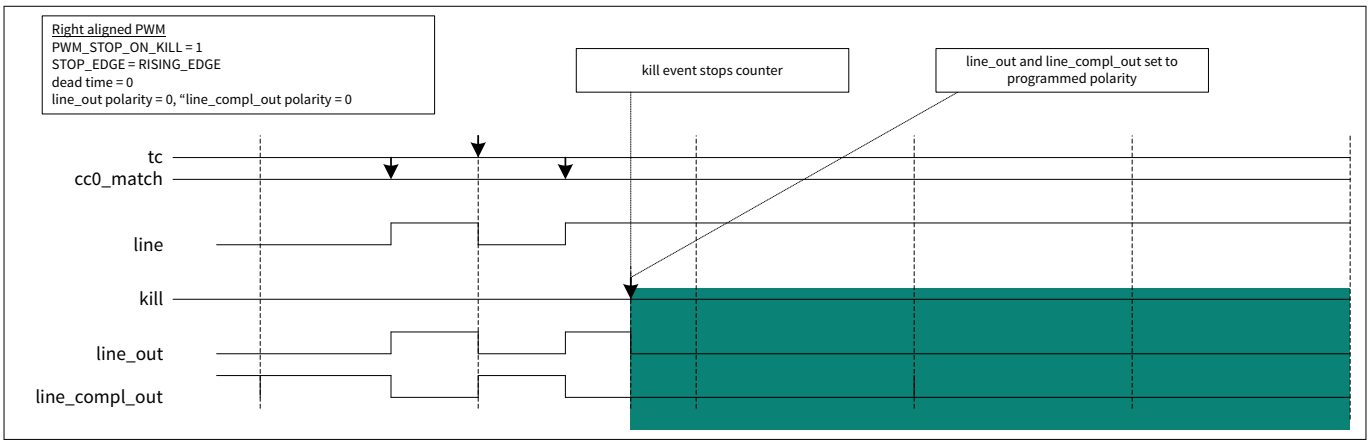


Figure 258 PWM stop on kill

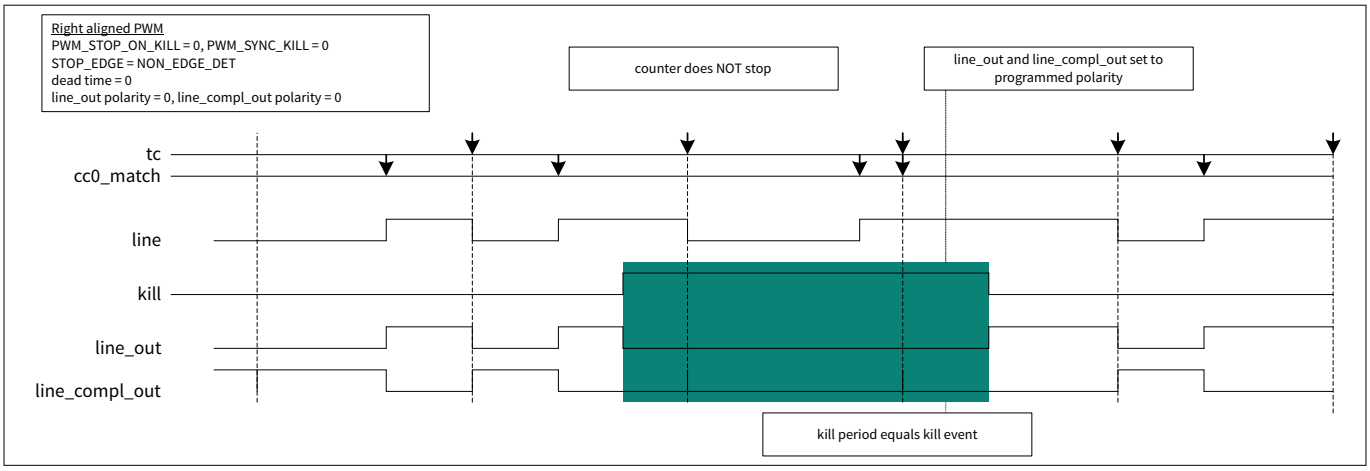


Figure 259 PWM async kill

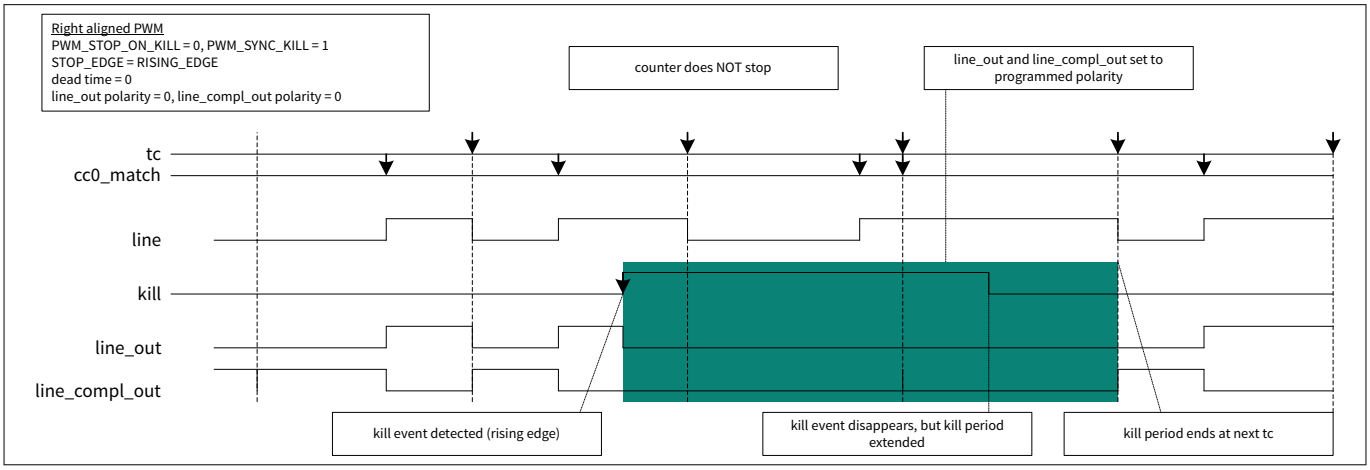


Figure 260 PWM sync kill

Different from asynchronous kill, synchronous kill will stop suppressing line output on TC event after kill is gone.

For counter groups that support capture1 event, a second kill input function is available, similar to a stop event. Both events are OR combined and share the same kill mode settings. Having two stop/kill events for a PWM allows selecting a common trigger for one stop/kill event from a PERI trigger multiplexer (allowing synchronous stop/kill operation of multiple PWMs) while selecting a dedicated ADC out-of-range trigger for the other stop/kill event (for example, allowing real-time hardware stop of a PWM when current of a PWM driven signal is out of range).

25.3.4.2 Configuring Counter for PWM Mode for Stepper Motor Control (SMC)

This section describes how to control a simple two-phase stepper motor. The individual poles of the stepper motor can be controlled by using several TCPWM counters. The counters must be driven in different modes.

Figure 261 illustrates an example of how the coils of a stepper motor can be controlled to generate electromagnetic north and south poles. Coil 1 is controlled by TCPWM counter 0 and coil 2 is controlled by TCPWM counter 1. Both TCPWMs are set into PWM mode. Each TCPWM consist of a pair of complementary output signals, which can also be driven separately in different modes (LINE_OUT and LINE_COMPL_OUT). These 16-bit counters are related to dedicated counter groups that support SMC functionality. See the respective device datasheet to see which TCPWM SMC counter groups are available (for example, 12 TCPWM SMC counters can drive six independent two-phase stepper motors).

For synchronization purposes, a third TCPWM counter 2 in timer mode triggers the other two TCPWM counters as a common input trigger signal. This TCPWM counter can be a counter from a different TCPWM counter group

(such as 16-bit or 32-bit TCPWM counter with no SMC functionality). It can also be used to initiate a DMA transfer with a synchronized timing to update the related buffer registers (see step 2 in Figure 263). For example, the first trigger output signal tr_out0 can be used as input trigger for the two SMC counters. Trigger output signal tr_out1 can be used to trigger a DMA transfer.

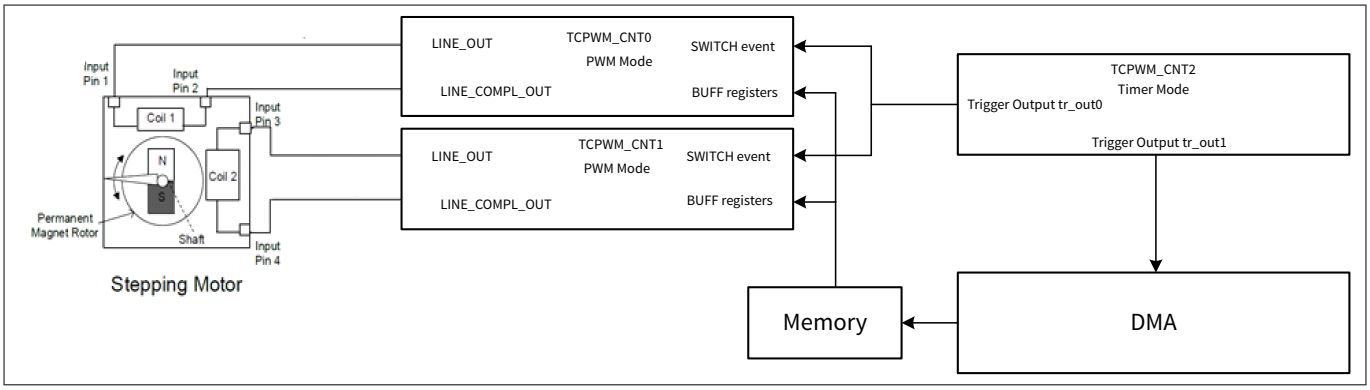


Figure 261 Simple Two-phase stepper motor control

Note that a SWITCH event is related to a capture0 event. See Table 217 for more details.

The steps to configure the two 16-bit TCPWM counters for SMC operation and one 16/32-bit general-purpose TCPWM counter in timer mode and the affected register bits are as follows.

- PWM mode counter
  - Disable the counter by writing '0' to the ENABLE bit of the CTRL register.
  - Select PWM mode by writing '100' to the MODE[26:24] field of the CTRL register.
  - Set clock prescaling by writing to the DT_LINE_OUT_L[7:0] field of the DT register.
  - Set the required 16-bit period in the PERIOD register.
  - Set the 16-bit compare value in the CC0 register and buffer compare value in the CC0_BUFF register to switch values.
  - Set the counter line sources by writing to the OUT_SEL[2:0] and COMPL_OUT_SEL[6:4] fields of the LINE_SEL and LINE_SEL_BUFF registers.
  - Set the TR_IN_SEL0 register to select the trigger sources that causes the SWITCH and RELOAD events. For all TCPWM counters related to one SMC channel, the trigger source must be the same.
  - Set the TR_IN_EDGE_SEL register to select the edge of the trigger that causes the SWITCH event.
  - Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of the CTRL register to configure left-aligned PWM.
  - Set the PWM_IMM_KILL, PWM_STOP_ON_KILL, and PWM_SYNC_KILL fields of the CTRL register as required.
  - LINE_OUT and LINE_COMPL_OUT can be controlled by the TR_PWM_CTRL register to set, reset, or invert upon cc0_match, cc1_match, overflow, and underflow conditions.
  - Enable the counter by writing '1' to ENABLED bit of the CTRL register. For synchronization purpose, another TCPWM counter needs to be used as a start trigger.
- Timer mode counter
  - Disable the counter by writing '0' to the ENABLE bit of the CTRL register.
  - Select Timer mode by writing '000' to the MODE[26:24] field of the CTRL register.
  - Set the required 16- or 32-bit period in the PERIOD register.
  - Set the 16- or 32-bit compare value in the CC0 register and the buffer compare value in the CC0_BUFF register.
  - Set AUTO_RELOAD_CC0 field of the CTRL register, if required to switch values at every CC condition.

- Set clock prescaling by writing to the DT_LINE_OUT_L[7:0] field of the DT register.
- Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of the CTRL register.
- 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 CTRL register.
- Set the TR_IN_SEL0 or TR_IN_SEL1 register to select the trigger that causes the event (Reload, Start, Stop, Capture0/1, and Count).
- Set the TR_IN_EDGE_SEL register to select the edge of the trigger that causes the event (Reload, Start, Stop, Capture0/1, and Count).
- Set TR_OUT_SEL register for both trigger outputs:
  - Tr_out0: CC0_MATCH event to be used as a SWITCH event in PWM counters to update internal double-buffers
  - Tr_out1: CC1_MATCH event to be used to trigger DMA for updating BUFF registers in PWM counters. New BUFF register values must be already written before the SWITCH event is updating internal double buffers
- If required, set the interrupt upon TC or CC0_MATCH or CC1_MATCH condition.
- Enable the counter by writing '1' to ENABLED bit of the CTRL register. A start trigger must be provided through firmware (START bit in the TR_CMD register) to start the counter if the hardware start signal is not enabled.

In PWM mode the dedicated TCPWM SMC counter groups can be used for Stepper Motor Control (SMC) including micro stepping. Therefore, the two PWM output signals LINE_OUT and LINE_COMPL_OUT (which are usually a pair of complementary PWM signals during normal PWM operation) can be set to the following options using an output select register (LINE_SEL register, OUT_SEL, and COMPL_OUT_SEL fields):

- Constant low ('0')
- Constant high ('1')
- PWM signal “line”
- Inverted PWM signal “line”
- 'Z' (high-Z)

The 16-bit TCPWM counters supporting SMC are intended to be connected to GPIO_SMC cells in the IOSS, which support a high-current SMC I/O to drive stepper motors directly for pointer instruments. Figure 262 illustrates the generation of one SMC channel for a two-phase stepper motor control. It includes a pair of SMC counters each of which controls one coil (PWMx_M_y/PWMx_M_y_N and PWMx_M_z/PWMx_M_z_N)[4] with bridge drivers. Each counter can control two outputs by different functions that are controlled by the line select multiplexer. These are dependent on LINE_SEL register settings. The OUT_SEL and COMPL_OUT_SEL bit fields define which signal is routed to the SMC pins. For full-step motor control, const '1' and const '0' can be selected to switch the related output pins to the required polarity. For micro-stepping motor control, PWM and inverted PWM can be routed to the corresponding SMC pin. In addition, the polarity of each SMC pin can be controlled separately; the kill function is also supported as described in PWM outputs. Each multiplexer input control signal has its own buffer register LINE_SEL_BUFF, which allows to update the polarity of each coil control signal on the fly by a switch event. This is used for preloading into internal buffer and to actively load it into the corresponding LINE_SEL register bit fields using a tc event (such as overflow event) to guarantee a glitch free polarity change of the coil controls. By setting the line select multiplexer to 'Z' (high impedance), the corresponding output can be set to not driven and to allow evaluating the back-EMF (BEMF) signal via ADC, which is required for software implementation of zero-point detection (ZPD). *See AN226036 SMC-ZPD Implementation in TRAVEO™ T2G family.*

[4] PWMx_M_y: 16-bit PWM with SMC support, x: TCPWM block (refer to the device datasheet to see how many TCPWM blocks are available; x is not valid for devices with a single TCPWM block), y and z: TCPWM counter number.

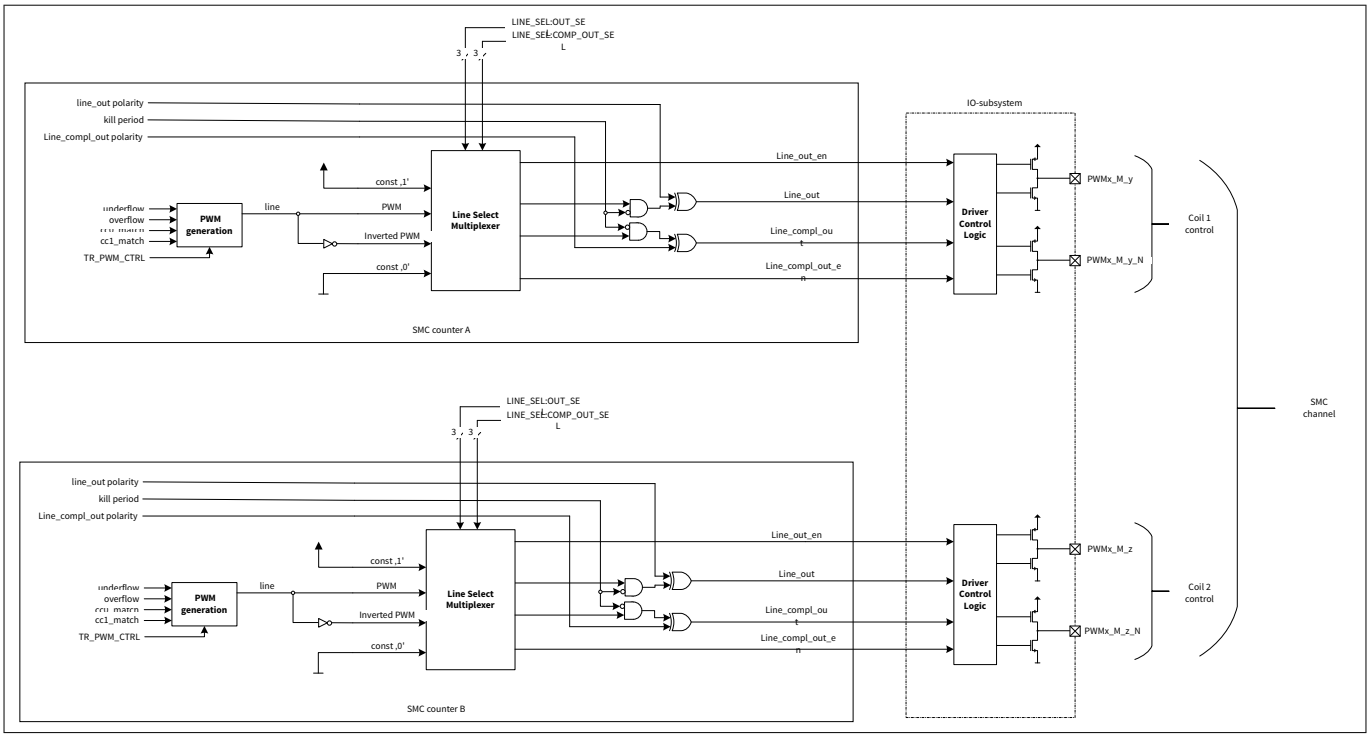


Figure 262 **SMC channel generation (includes control of a coil pair and the corresponding line select multiplexing for each SMC counter)**

Table 222 **Line select multiplexer settings**

LINE_SEL_OUT_SEL/LINE_SEL_COMPL_OUT_SEL	Signal Routed to SMC Output Pin
0	LOW
1	HIGH
2	PWM
3	Inverted PWM
4	'Z' (high impedance)

For full stepping, the TCPWM counter must be set to PWM (or PWM_PR) mode. However, a PWM generation is not needed – the compare function resources (CC0/CC1) are free to be used for other purposes (such as trigger generation).

A separate TCPWM counter is used for each coil of a stepper motor. These counters run synchronously (sharing the same reload/start/stop/count events and the same period). To achieve a synchronous update of the output signals across multiple TCPWM counters the same double buffering and switching method is used as for updating CC0/CC1 or PERIOD registers. Therefore, a LINE_SEL_BUFF register is used, which is exchanged with the LINE_SEL register on a terminal count (tc) event with an actively pending switch event (when specified by AUTO_RELOAD_LINE_SEL bit in the CTRL register).

The following use cases show examples of driving two coils of a stepper motor with two TCPWM counters. Figure 263 (use case A) illustrates three full steps (180°, 270°, 0° (= 360°) electrical angles), Figure 264 (use case B) shows the stepper motor control by three micro steps using a PWM (~30°, ~60°, ~120° electrical angles). These are illustrations that show one step per PWM counter PERIOD. In reality, the steps for micro stepping are significantly slower compared to the PWM period; therefore, the PWM duty cycles (CC0) and/or output select registers (LINE_SEL) are stable over many PWM periods before they are changed by a switch event.

The time base for the steps can be realized using an additional TPCWM counter in timer mode (running on a slower counter clock or with a higher PWM period or counting overflow events of the PWM counter). This

counter can generate interrupts, which trigger the CPU to update PWM duty cycles and/or output select registers, and to generate the SWITCH event using TR_CMD register in the MXPERI block.

The periodical sequence is as follows and is valid for full-stepping as well as for micro-stepping motor control:

- Initialize TCPWM registers CC0, LINE_SEL_OUT_SEL, and LINE_SEL_COMPL_OUT_SEL while the counters are not running and start the counter by a reload trigger event.
- Write new PWM duty cycle values in the CC0_BUFF register and the LINE_SEL_BUFF settings.

  ***Note**: Set a timing that updates all buffer registers before a new switch event occurs.*

- Input switch trigger event to update internal buffers (double-buffering). The new setting is not yet effective.
- Overflow event copies the data from internal buffers to CC0, LINE_SEL_OUT_SEL, and LINE_SEL_COMPL_OUT_SEL registers.

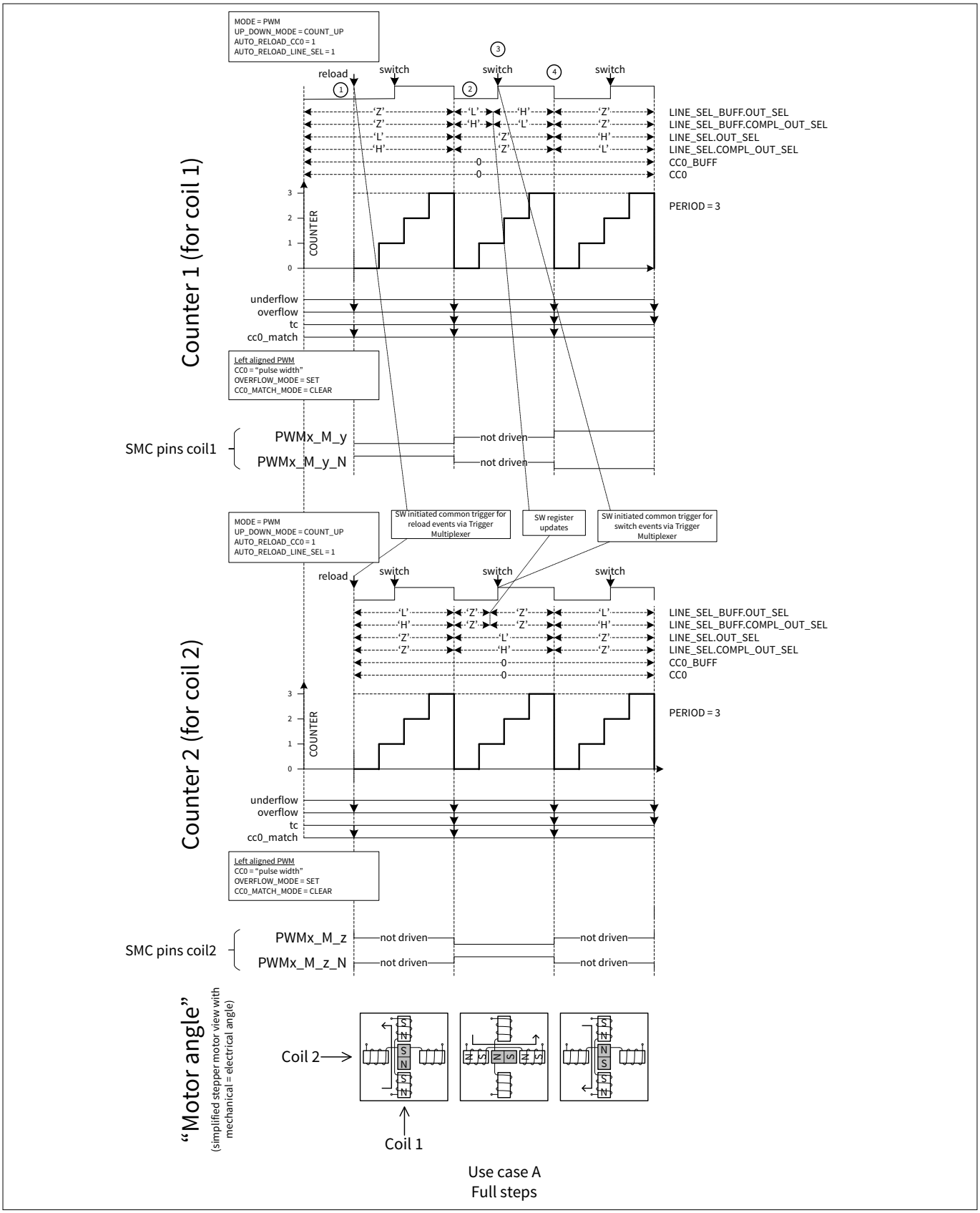


Figure 263 SMC use case A: full step control

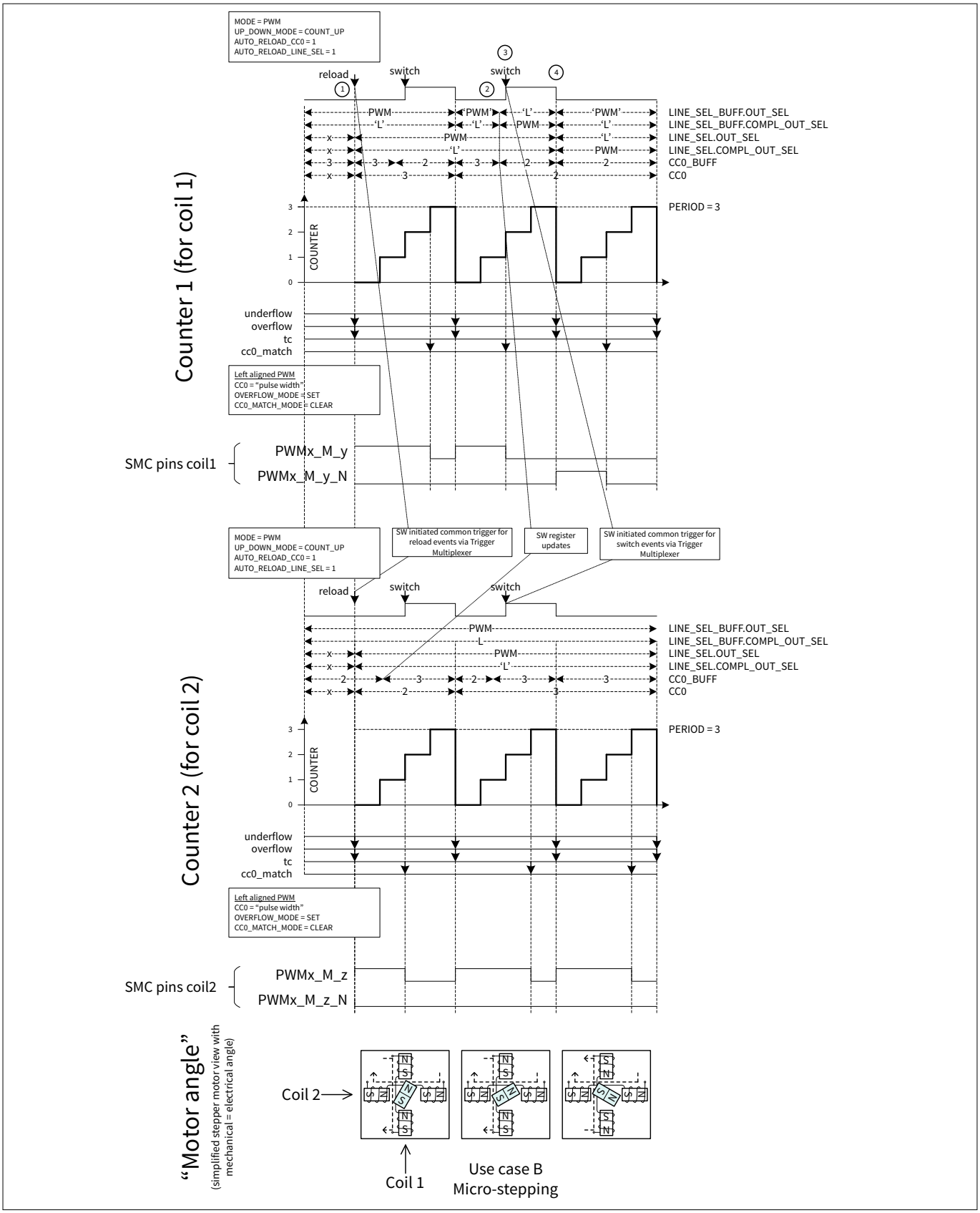


Figure 264 SMC use case B: micro-stepping

25.3.4.3 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.

- Disable the counter by writing '0' to the ENABLE bit of the CTRL Register.
- Select PWM mode by writing '100' to the MODE[26:24] field of the CTRL Register.
- Set clock prescaling by writing to the DT_LINE_OUT_L[7:0] field of the DT Register.
- Set the required 16-bit period in the PERIOD Register and the buffer period value in the PERIOD_BUFF Register to switch values, if required.
- Set the 16-bit compare value in the CC0/1 Register and buffer compare value in the CC0/1_BUFF Register to switch values, if required.
- Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of the CTRL Register to configure left-aligned, right-aligned, or center-aligned PWM.
- Set the PWM_IMM_KILL, PWM_STOP_ON_KILL, and PWM_SYNC_KILL fields of the CTRL Register as required.
- Set the TR_IN_SEL0 or TR_IN_SEL1 Register to select the trigger that causes the event (reload, start, stop, capture0/1, and count).
- Set the TR_IN_EDGE_SEL Register to select the edge of the trigger that causes the event (reload, start, stop, capture0/1, and count).
- LINE_OUT and LINE_COMPL_OUT can be controlled by the TR_PWM_CTRL Register to set, reset, or invert upon cc0_match, cc1_match, overflow, and underflow conditions.
- If required, set the interrupt upon TC or CC0/1 condition.
- Enable the counter by writing '1' to ENABLED bit of the CTRL Register. A start trigger must be provided through firmware (START bit in the TR_CMD Register) to start the counter if the hardware start signal is not enabled.

25.3.5 Pulse width modulation with dead time mode

The PWM-DT functionality is the same as PWM functionality, except for the following differences:

- PWM_DT supports dead time insertion; PWM does not support dead time insertion.
- PWM_DT does not support clock prescaling; PWM supports clock prescaling.

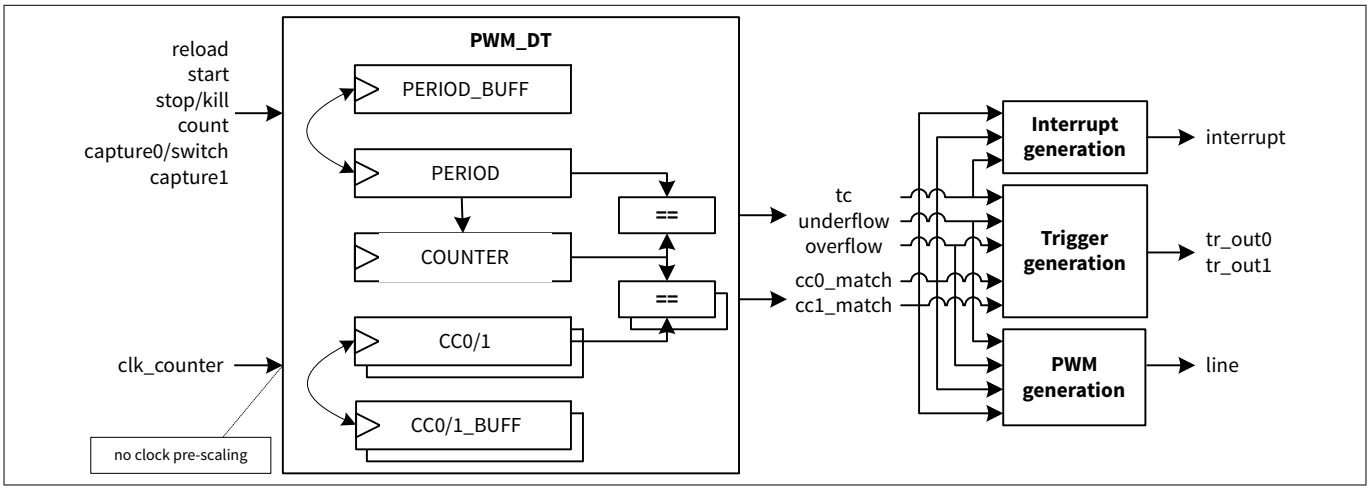


Figure 265 PWM with dead time functionality

Dead time insertion is a step that operates on a preliminary PWM output signal line, as illustrated in Figure 250.

The dead time insertion for two PWM complementary output lines ranges from 0 to 255 (8 bit) or from 0 to 65535 (16 bit, only for counter groups supporting advanced motor control) counter clock cycles. The setup can

be done in DT_LINE_OUT_L [7:0] field (low byte) and in DT_LINE_OUT_L [15:8] field in the DT Register. For the Advanced Motor Control counter, DT_LINE_OUT[15:0] is for LINE_OUT, DT_LINE_COMPL_OUT[15:0] is for LINE_COMPL_OUT.

Figure 266 illustrates dead time insertion for different dead times and different output signal polarity settings.

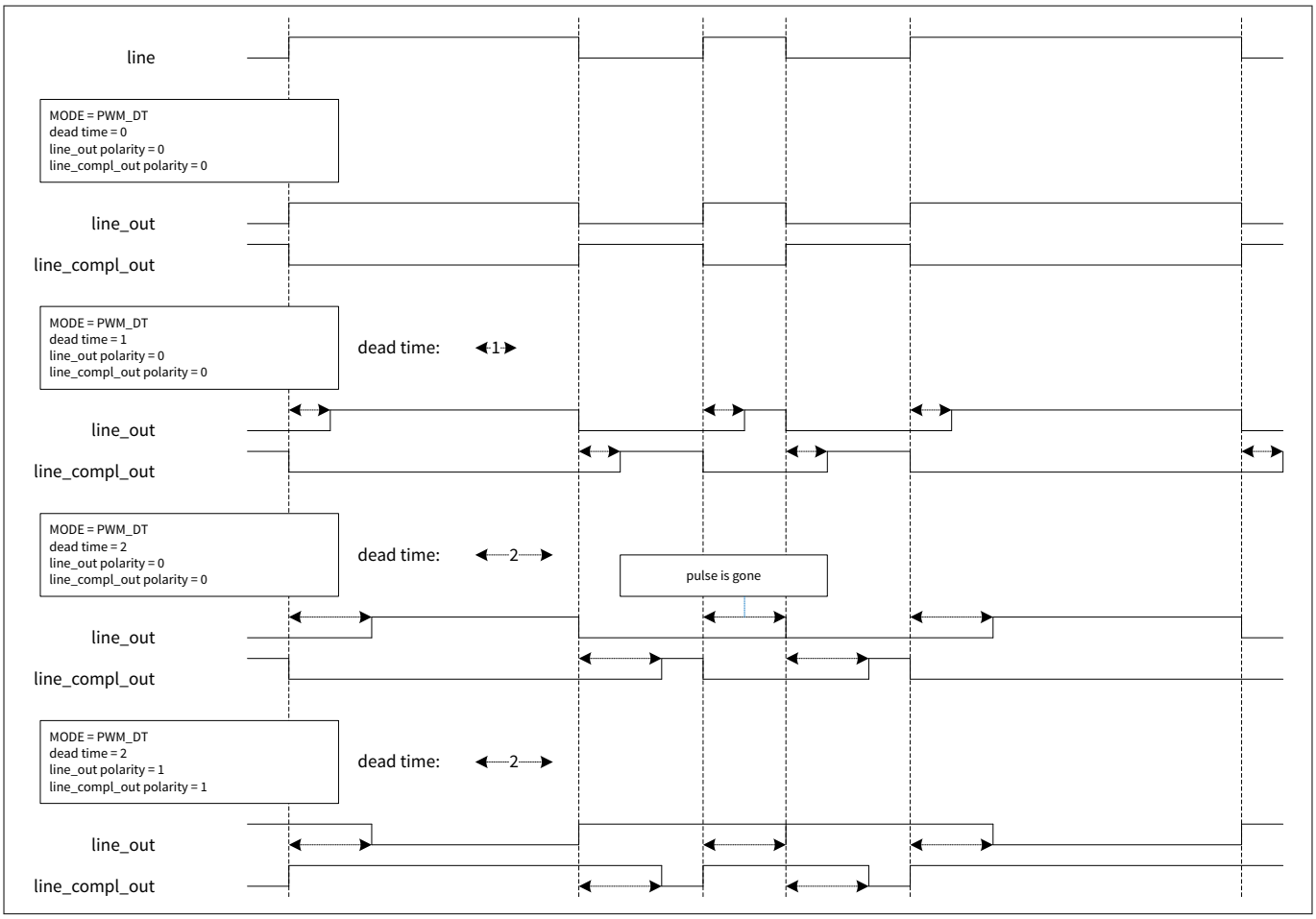


Figure 266 Dead-time timing

Figure 267 illustrates how the polarity settings and stop/kill functionality combined control the PWM output signals LINE_OUT and LINE_COMPL_OUT.

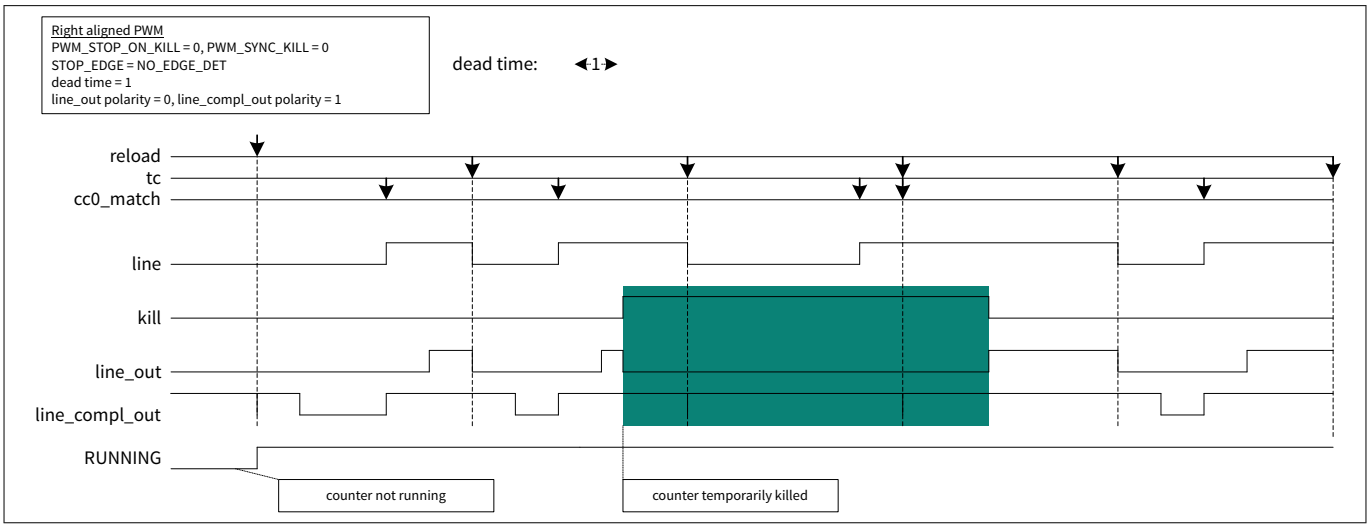


Figure 267 Dead time and kill

25.3.5.1 Configuring counter for PWM with dead time mode

The steps to configure the counter for PWM with dead time mode of operation and the affected register bits are as follows:

1. Disable the counter by writing '0' to the ENABLE bit of the CTRL Register.
2. Select PWM with dead time mode by writing '101' to the MODE[26:24] field of the CTRL Register.
3. Set the required dead time by writing to the DT_LINE_OUT_L [7:0] and DT_LINE_OUT_H [15:8] fields of the DT Register.
4. Set the required 16-bit period in the PERIOD Register and the buffer period value in the PERIOD_BUFF Register to switch values, if required.
5. Set the 16-bit compare value in the CC0/1 Register and buffer compare value in the CC0/1_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 CTRL Register to configure left-aligned, right-aligned, or center-aligned PWM.
7. Set the PWM_IMM_KILL, PWM_STOP_ON_KILL, and PWM_SYNC_KILL fields of the CTRL Register as required.
8. Set the TR_IN_SEL0 or TR_IN_SEL1 Register to select the trigger that causes the event (reload, start, stop, capture0/1, and count).
9. Set the TR_IN_EDGE_SEL Register to select the edge of the trigger that causes the event (reload, start, stop, capture0/1, and count).
10. LINE_OUT and LINE_COMPL_OUT can be controlled by the TR_PWM_CTRL Register to set, reset, or invert upon cc0_match, cc1_match, overflow, and underflow conditions.
11. If required, set the interrupt upon TC or CC0/1 condition.
12. Enable the counter by writing '1' to ENABLED bit of the CTRL Register. A start trigger must be provided through firmware (START bit in the TR_CMD Register) to start the counter if the hardware start signal is not enabled.

25.3.6 Pulse width modulation pseudo-random mode (PWM PR)

The PWM PR functionality changes the counter value using the Linear Feedback Shift Register (LFSR). This results in a pseudo random number sequence. A signal similar to a PWM signal is created by comparing the counter value COUNTER with the CC0/1 Register. The generated signal has different frequency/noise characteristics than a regular PWM signal.

Table 223 Input events of PWM_PR

Generated events	Usage
Reload	Same behavior as start event. Can only be used when the counter is not running.
Start	Starts the counter. The counter is not initialized by hardware. The current counter value is used. Behavior is independent on UP_DOWN_MODE. Note that when the counter is running, the start event has no effect. Can be used when the counter is running or not running.
stop/kill	Stops the counter. Different stop/kill modes exist.
count	Not used

(table continues...)

Table 223 (continued) Input events of PWM_PR

Generated events	Usage
Capture0	This event acts as a switch event. When this event is active, the CC0/CC0_BUFF, CC1/CC1_BUFF, PERIOD/PERIOD_BUFF, and LINE_SEL/LINE_SEL_BUFF Registers are exchanged on a tc event (when specified by AUTO_RELOAD_CC0, AUTO_RELOAD_PERIOD, and AUTO_RELOAD_LINE_SEL bits in the CTRL Register). A switch event requires rising, falling, or rising/falling edge event detection mode. Pass-through mode is not supported, unless the selected event is a constant '0' or '1'. When a switch event is detected and the counter is running, the event is kept pending until the next tc event. When a switch event is detected and the counter is not running, the event is cleared by hardware.
Capture1	This event acts as a second stop/kill event. It has the same function as the stop0/kill0 event. Both events are OR combined.

***Note**: Event detection is on the peripheral clock CLK_PERI.*

Table 224 Basic features of PWM_PR

Supported features	Description
Clock prescaling	Prescales the PCLK_TCPWM[x]_CLOCKS[y].
One shot	Counter is stopped by hardware, after a single period of the counter (counter value equals period value PERIOD).
Auto reload CC	CC0/1 and CC0/1_BUFF are exchanged on a switch event AND tc event (when specified by AUTO_RELOAD_CC bit in CTRL Register).
Auto reload LINE_SEL	LINE_SEL and LINE_SEL_BUFF are exchanged on a switch event and tc event (when specified by AUTO_RELOAD_LINE_SEL bit in the CTRL Register).
Auto reload PERIOD	PERIOD and PERIOD_BUFF are exchanged on a switch event and tc event (when specified by AUTO_RELOAD_PERIOD bit in CTRL Register).
Kill modes	Specified by PWM_SYNC_KILL, PWM_STOP_ON_KILL, and PWM_IMM_KILL.

***Note**: The count event is not used. As a result, the PWM_PR functionality operates on the prescaled counter clock (PCLK_TCPWM[x]_CLOCKS[y]), rather than on an active count prescaled counter clock.*

Table 225 Trigger outputs of PWM_PR

Trigger output	Description
cc0_match (CC)	Counter changes from a state in which COUNTER equals ‘CC0’
cc1_match (CC)	Counter changes from a state in which COUNTER equals ‘CC1’
Underflow (UN)	Not used
Overflow (OV)	Not used
TC	Counter changes from a state in which COUNTER equals ‘PERIOD’

Table 226 PWM_PR PWM outputs

PWM outputs	Description
LINE_OUT	PWM line output
LINE_COMPL_OUT	Complementary PWM line output

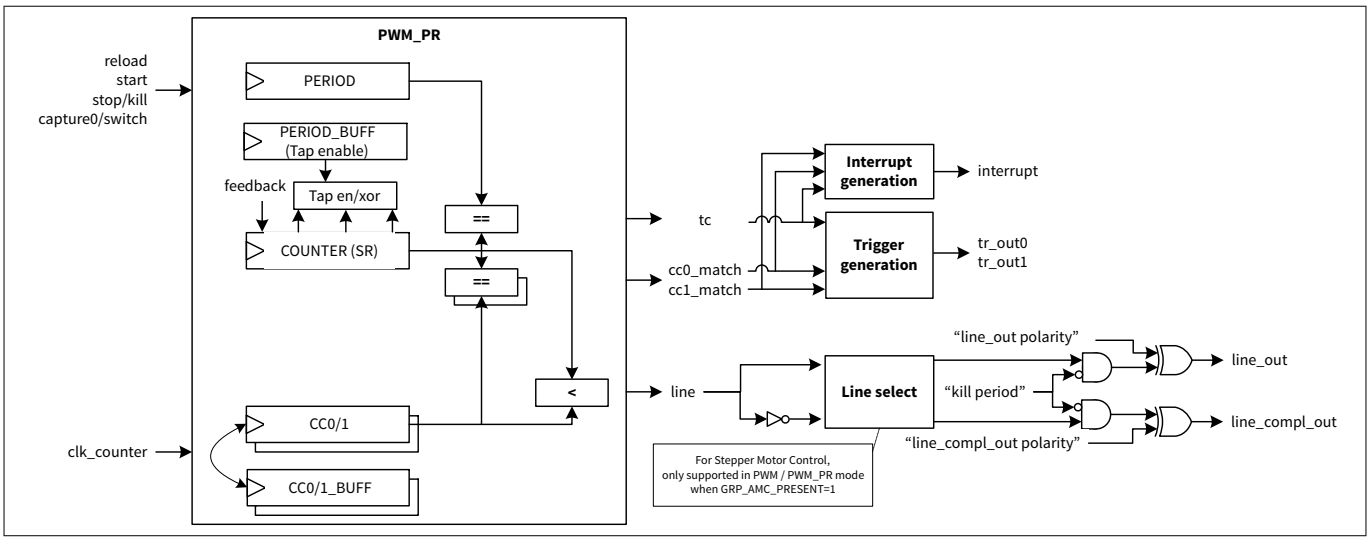


Figure 268 PWM_PR functionality

The PWM_PR functionality is described as follows:

- The counter value COUNTER is initialized by software (to a value different from 0).
- Programmable LFSR length

  The COUNTER is changed based on an LFSR polynomial (http://en.wikipedia.org/wiki/Linear_feedback_shift_register) specified by taps in the PERIOD_BUFF (overloaded) Register. Different period lengths are possible by different programmed polynomials.

  Examples:

  Maximum length 16-bit LFSR: x^16 + x^14 + x^13 + x^11 + 1

  With counter groups including 16-bit counters:

  - temp = COUNTER[5] ^ COUNTER[3] ^ COUNTER[2] ^ COUNTER[0];
  - COUNTER = (temp << 15) | (COUNTER >> 1)

  Maximum length 8-bit LFSR: x^8 + x^6 + x^5 + x^4 + 1

  With counter groups including 16-bit counters, realized in 8 MSbs of 16-bit LFSR:

  - temp = COUNTER[12] ^ COUNTER[11] ^ COUNTER[10] ^ COUNTER[8];
  - COUNTER = (temp << 15) | (COUNTER >> 1)

  Maximum length 32bit LFSR: x^32 + x^30 + x^26 + x^25 + 1

  With counter groups including 32-bit counters:

  - temp = COUNTER[7] ^ COUNTER[6] ^ COUNTER[2] ^ COUNTER[0];
  - COUNTER = (temp << 31) | (COUNTER >> 1)

This will result in a pseudo random number sequence for COUNTER. For example, when COUNTER is initialized to 0xace1 and a 16-bit LFSR with taps 16, 14, 13, and 11 is used, the number sequence is: 0xace1, 0x5670, 0xab38, 0x559c, 0x2ace, 0x1567, 0x8ab3, …, 0x59c3. This sequence will repeat after 65535 counter clock cycles.

The following tables show examples for maximum length LFSRs (from http://courses.cse.tamu.edu/csce680/walker/lfsr_table.pdf) and their equivalent bit positions in the MSbs of the 16-bit COUNTER Register (for

GRP_CNT_WIDTH = 16). This allows possible pseudo random sequences with a period of 2^n-1 with n in [2, 16]. The right column shows example values for the PERIOD Register to generate a tc event when starting with an initialized COUNTER value of 0xace1 (taking the “unused” LSbs into account, which result from right-shifting of the “used” MSbs).

Table 227 Polynomial of maximum length LFSR

n	n-bit LFSR taps	Equivalent bit positions in 16-bit COUNTER Register	TAP value to be programmed to PERIOD_BUFF Register	Period of sequence	Example for PERIOD Register
4	4,3	12,13	0x3000	15	0x591e
5	5,3	11,13	0x2800	31	0x5d8f
6	6,5	10,11	0x0c00	63	0x59bb
7	7,6	9,10	0x0600	127	0x5b24
8	8,6,5,4	8,10,11,12	0x1d00	255	0x5997
9	9,5	7,11	0x0880	511	0x593f
10	10,7	6,9	0x0240	1023	0x59eb
11	11,9	5,7	0x00a0	2047	0x59dc
12	12,11,8,6	4,5,8,10	0x0530	4095	0x59cb
13	13,12,10,9	3,4,6,7	0x00d8	8191	0x59ca
14	14,14,11,9	2,3,5,7	0x00ac	16383	0x59c2
15	15,14	1,2	0x0006	32767	0x59c3
16	16,14,13,11	0,2,3,5	0x002d	65535	0x59c3

The programmable taps allow LFSRs other than maximum cycle LFSRs in Table 227, which can result in periods other than 2^n-1. The following table shows some examples.

Table 228 Polynomial of non-maximum length LFSR

n-bit LFSR taps	Bit positions in 16-bit counter Register	Period of sequence	n-bit LFSR taps	Bit positions in 16-bit counter Register	Period of sequence
16,15	0,1	255	16,15,11	0,1,5	4340
16,14	0,2	126	16,15,10	0,1,6	24573
16,13	0,3	8191	16,15,9	0,1,7	32766
16,12	0,4	60	16,15,8	0,1,8	4681
16,11	0,5	16383	16,15,7	0,1,9	504
16,10	0,6	434	16,15,6	0,1,10	10235
16,9	0,7	63457	16,15,5	0,1,11	3906
16,8	0,8	24	16,15,4	0,1,12	7161
16,15,14	0,1,2	32767	16,15,3	0,1,13	3276
16,15,13	0,1,3	11811	16,15,2	0,1,14	32767
16,15,12	0,1,4	63	16,15,1	0,1,15	30

However, it is not recommended to use such non-maximum cycle LFSRs to generate a pseudo-random PWM signal, even if they result in the same cycle length as shorter maximum cycle LFSRs (for example, 255 cycles for taps 16, 15). This because the values occurring in such sequences are not equally distributed over the possible value space, which results in much bigger errors compared with the desired PWM duty cycle accumulated over a full pseudo random number sequence. For example, the 8-bit LFSR with taps 8, 6, 5, 4 (realized in 8 MSbs of 16-bit LFSR) and the 16-bit LFSR with the taps 16, 15 result both in a period of 255 cycles, but a CC0 value of 0x4000 (for a desired 50 percent “accumulated duty cycle”) results in an accumulated duty cycle of:

- 49.8% for the 8-bit LFSR with taps 8, 6, 5, 4 (realized in 8 MSbs of 16-bit LFSR).
  - 0.39% error
- 46.7% for the 16-bit LFSR with the taps 16, 15
  - 6.66% error
- Asymmetric with CC0
  - Write ‘3' to the UP_DOWN_MODE [17:16] field in the CTRL Register to set the counter direction to COUNT_UPDN2 mode.
- When COUNTER equals CC0 (CC1), a cc0_match (cc1_match) event is generated.
- When COUNTER equals PERIOD, a tc event is generated. Note that the LFSR produces a deterministic number sequence (given a specific counter initialization value). Therefore, it is possible to calculate the COUNTER value after a certain number of LFSR iterations n. This calculated COUNTER value can be used as PERIOD value, and the tc event will be generated after precisely n counter clocks.
- On a tc event, the CC0/CC0_BUFF and CC1/CC1_BUFF can be conditionally exchanged under control of the capture0/switch event and the AUTO_RELOAD_CC0 field of the CTRL Register (see PWM mode functionalities).

***Note**:* *To generate a tc event, PERIOD must be set to a value which the LFSR (Register COUNTER) reaches. When realizing a shorter maximum length LFSR (n < GRP_CNT_WIDTH) within the n MSbs of a GRP_CNT_WIDTH wide LFSR, the “unused” LSbs need to be set to a value which results from right-shifting of the “used” MSbs.*

- The output line reflects: COUNTER[14:0] < CC0[15:0]. Note that only the lower 15 bits of COUNTER are used. As a result, for CC0 greater or equal to 0x8000, “line” is always 1. The line polarity can be inverted (as specified by QUAD_ENCODING_MODE[0] of the CTRL Register). For counter groups including 32-bit counters the output line reflects: COUNTER[30:0] < CC0[31:0].
- During PWM_PR operation:
  - When COUNTER equals CC0 (CC01), a cc0_match (cc1_match) event is generated.
  - When COUNTER equals PERIOD, a tc event is generated.
  - On a tc event, the CC0/CC0_BUFF, CC1/CC1_BUFF and PERIOD/PERIOD_BUFF can be conditionally ex-changed under control of the capture/switch event and the AUTO_RELOAD_CC bit and AUTO_RELOAD_PERIOD bit in the CTRL Register (see PWM mode functionalities).
  - The output line reflects: COUNTER[14:0] < CC0[15:0]. Note that only the lower 15 bits of COUNTER are used for comparison, while the COUNTER itself can run up to 16- or 32-bit values. As a result, for CC greater or equal to 0x8000, “line” is always 1. The line polarity can be inverted (as specified by QUAD_ENCODING_MODE[0] of the CTRL Register). For counter groups including 32-bit counters the output line reflects: COUNTER[30:0] < CC0[31:0].

As mentioned, different stop/kill modes exist. The mode is specified by PWM_STOP_ON_KILL (PWM_SYNC_KILL should be '0' - asynchronous kill mode). The memory map describes the modes and the desired settings for the stop/kill event. The following two modes are supported:

- PWM_STOP_ON_KILL is '1'. This mode stops the counter on a stop/kill event.
- PWM_STOP_ON_KILL is '0'. This mode keeps the counter running, but suppresses the PWM output signals immediately and continues to do so for the duration of the stop/kill event.

Note that the LFSR produces a deterministic number sequence (given a specific counter initialization value). Therefore, it is possible to calculate the COUNTER value after a certain number of LFSR iterations, n. This calculated COUNTER value can be used as PERIOD value, and the tc event will be generated after precisely n counter clocks.

Figure 269 illustrates PWM_PR functionality.

***Note**:*

- *The shaded areas represent the counter region in which the line value is '1', for a CC0 value of 0x4000. There are two areas, because only the lower 15 bits of the counter value are used.*
- *When CC0 is set to 0x4000, roughly one-half of the counter clocks will result in a line value of '1' or in other words a 50 percent PWM duty cycle accumulated over a full pseudo random number sequence.*
- *When a shorter LFSR is realized using programmable taps (for example, an 8-bit LFSR is realized in 8 MSbs of the 16-bit COUNTER Register) the compare is still done on the whole 16-bit COUNTER Register. That means a CC0 set to 0x4000 still results into roughly half of the counter clocks with a "line" value of '1'.*

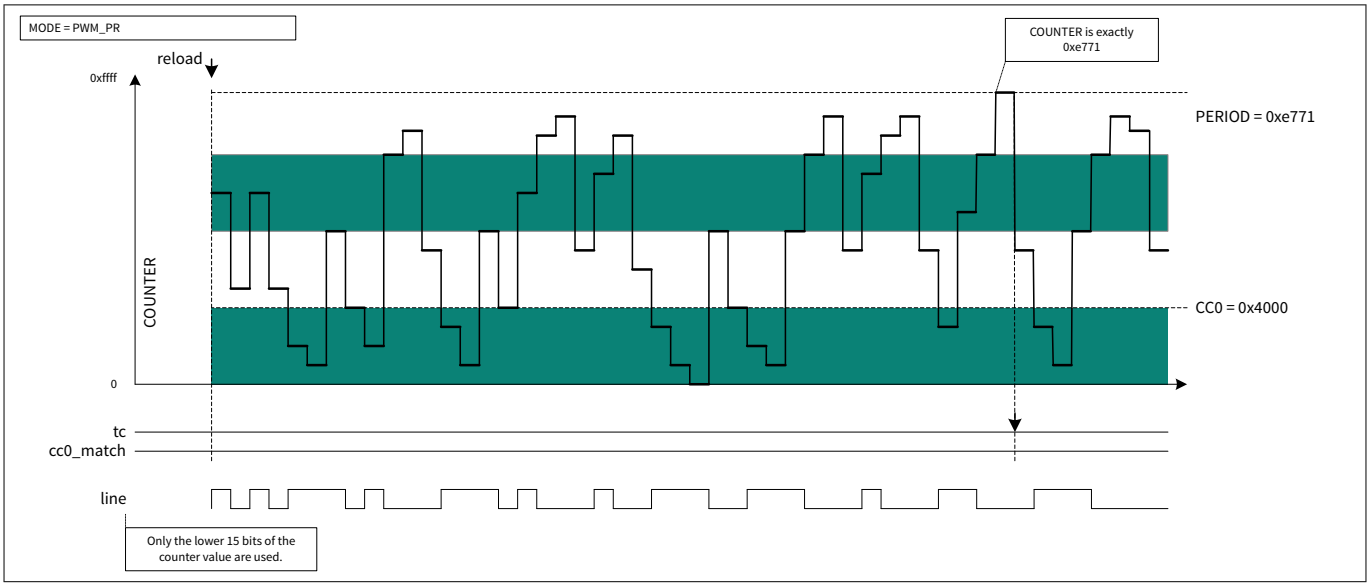


Figure 269 PWM_PR output

25.3.6.1 Configuring counter for pseudo-random PWM mode

The steps to configure the counter for pseudo-random PWM mode of operation and the affected register bits are as follows.

1. Disable the counter by writing '0' to the ENABLE bit of the CTRL Register.
2. Select pseudo-random PWM mode by writing '110' to the MODE[26:24] field of the CTRL Register.
3. Set the PERIOD Register for tc event generation and the LFSR length (16-bit or 32-bit) in the PERIOD_BUFF Register to define the LFSR polynomial.
4. Set the 16-bit compare value in the CC0/1 Register and the buffer compare value in the CC0/1_BUFF Register to switch values.

5. Set the PWM_IMM_KILL, PWM_STOP_ON_KILL, and PWM_SYNC_KILL fields of the CTRL Register as required.
6. Set the TR_IN_SEL0 or TR_IN_SEL1 Register to select the trigger that causes the event (reload, start, stop, capture0/1, and count).
7. Set the TR_IN_EDGE_SEL Register to select the edge of the trigger that causes the event (reload, start, stop, capture0/1, and count).
8. LINE_OUT and LINE_COMPL_OUT can be controlled by the TR_PWM_CTRL Register to set, reset, or invert upon cc0_match, cc1_match, overflow, and underflow conditions.
9. If required, set the interrupt upon TC or CC0/1 condition.
10. Enable the counter by writing '1' to the ENABLED bit of the CTRL Register. A start trigger must be provided through firmware (START bit in the TR_CMD Register) to start the counter if the hardware start signal is not enabled.

25.3.7 Shift Register (SR)

Shift Register functionality shifts the counter value in the right direction. The capture0 input is used to generate the MSB of the next counter value. The line output signal is driven from a programmable tab of the Shift Register (COUNTER).

This implements a signal delay function from the capture0 input to the line output, which can be used for functions such as detecting frequency shift keying (FSK) signals. It further allows parallel-in to serial-out data conversion (by shifting-out a preloaded counter value) as well as serial-in to parallel-out data conversion including compare match functionality (another synchronous TCPWM counter in timer mode to be used as time base for software).

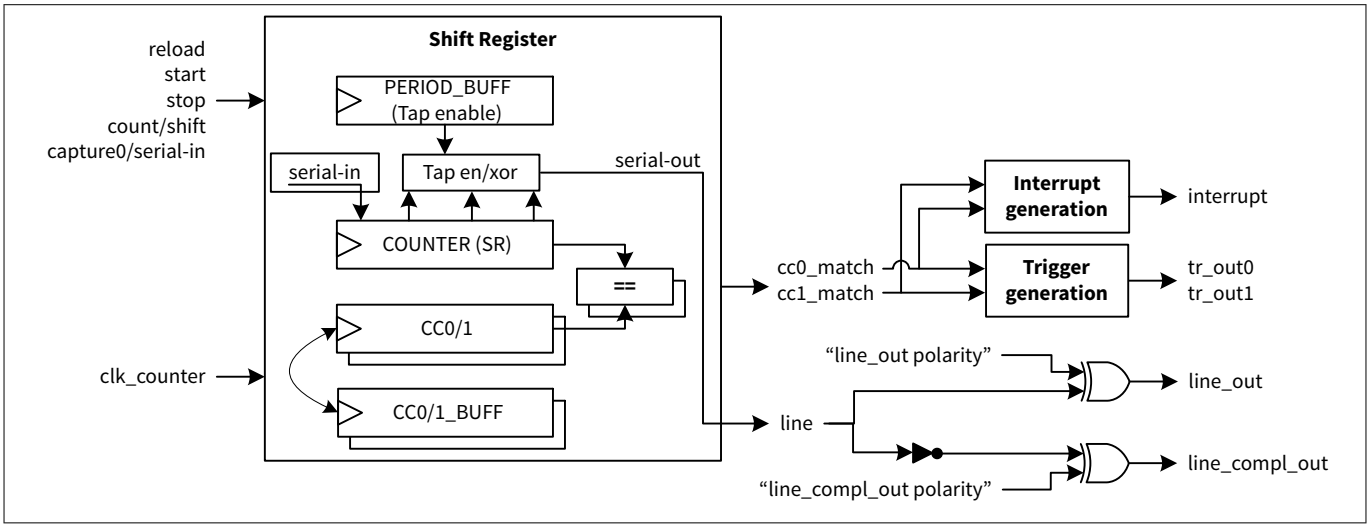


Figure 270 SR function diagram

25.3.7.1 SR mode functionality overview

Table 229 Input events of SR

External events	Usage
Reload	Sets the counter value to '0' and starts the counter shift operation. Can only be used when the counter is not running.

(table continues...)

Table 229 (continued) Input events of SR

External events	Usage
Start	Starts the counter shift operation. The counter is not initialized by hardware. The current counter value is used. Note that when the counter is running, the start event has no effect. Can be used when the counter is running or not running.
Stop	Stops the counter
Count/shift	Shifts the counter in the right direction
Capture0/serial-in	This event input is used as serial input to the MSB of the counter.
Capture1	Stops the counter

***Note**: Event detection is on the peripheral clock, CLK_PERI.*

Count event works to generate active count prescaled counter clock same as other function modes. This is how the shift works. Shifting the counter value is controlled by the count event and the counter clock, PCLK_TCPWM[x]_CLOCKS[y]. A constant '1' as well as synchronized input trigger edges can be used as count events.

Table 230 Basic features of SR

Supported features	Description
Clock prescaling	Prescales the PCLK_TCPWM[x]_CLOCKS[y].
Auto reload CC	CC0 (CC1) and CC0_BUFF (CC1_BUFF) are exchanged on a cc0_match (cc1_match) event (when specified by AUTO_RELOAD_CC0/1 bit in CTRL Register).

Table 231 Internal events of SR

Internal events	Description
cc0_match	Counter changes to a state in which COUNTER equals ‘CC0’
cc1_match	Counter changes to a state in which COUNTER equals ‘CC1’
Underflow	Not used
Overflow	Not used
TC	Not used

Table 232 Line output of SR

Supported features	Description
LINE_OUT	PWM line output. In Shift Register mode it is generated from an XOR combination of all enabled counter taps (bit position) defined by PERIOD_BUFF. For a shift register function only one tap should be used; that is, a one-hot value must be written to PERIOD_BUFF. If multiple bits in PERIOD_BUFF are set then the taps are XOR combined.
LINE_COMPL_OUT	Complementary PWM line output

25.3.7.2 Features of SR mode

Clock prescaling

Same function as in TIMER mode

One shot mode

One-shot mode is not supported

Input event

- A hardware- or software-generated reload event sets the counter value COUNTER to 0. Alternatively, the counter value COUNTER is initialized by software.
- A reload or start event starts the Shift Register operation.
- A stop event will stop the shift register operation, with no shifting even if the count event is active.
- COUNTER shift in the right direction at active count counter clock.
- Capture0 event is the serial input of MSB of COUNTER.

COUNTER shift

The counter value COUNTER is shifted in the right direction and shifts - in the serial input (capture0 event).

- 16-bit counter groups:
  - COUNTER = (serial-in << 15) | (COUNTER >> 1)
- 32-bit counter groups
  - COUNTER = (serial-in << 31) | (COUNTER >> 1)

This means that depending on counter bit length the COUNTER value is right-shifted by 1 and the capture event value is set in the MSB position.

LINE_OUT output

The output line is generated from a programmable COUNTER tap to generate a shifted version of the serial input (capture0 event). For a shift register function, only one tap should be selected via PERIOD_BUFF Register; that is, a one-hot value must be written into PERIOD_BUFF. For a delay of n cycles (from capture0 event to line output), the PERIOD_BUFF bit should be set to '1', and other bits should be set to '0'. If multiple bits are set in PERIOD_BUFF then the selected taps are XOR combined.

Figure 271 illustrates the Shift Register functionality.

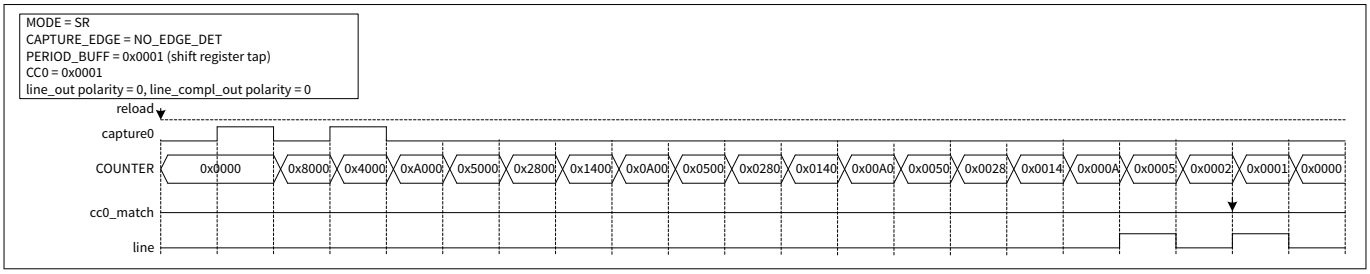


Figure 271 SR shift with PERIOD_BUFF = 0x0001

CC0 and CC0_BUFF auto reload

On a cc0_match (cc1_match) event, the CC0/CC0_BUFF (CC1/CC1_BUFF) can be conditionally exchanged under control of the AUTO_RELOAD_CC0 field in the CTRL Register, no switch event is required. Figure 272 illustrates the behavior of CC0 and CC0_BUFF auto reload.

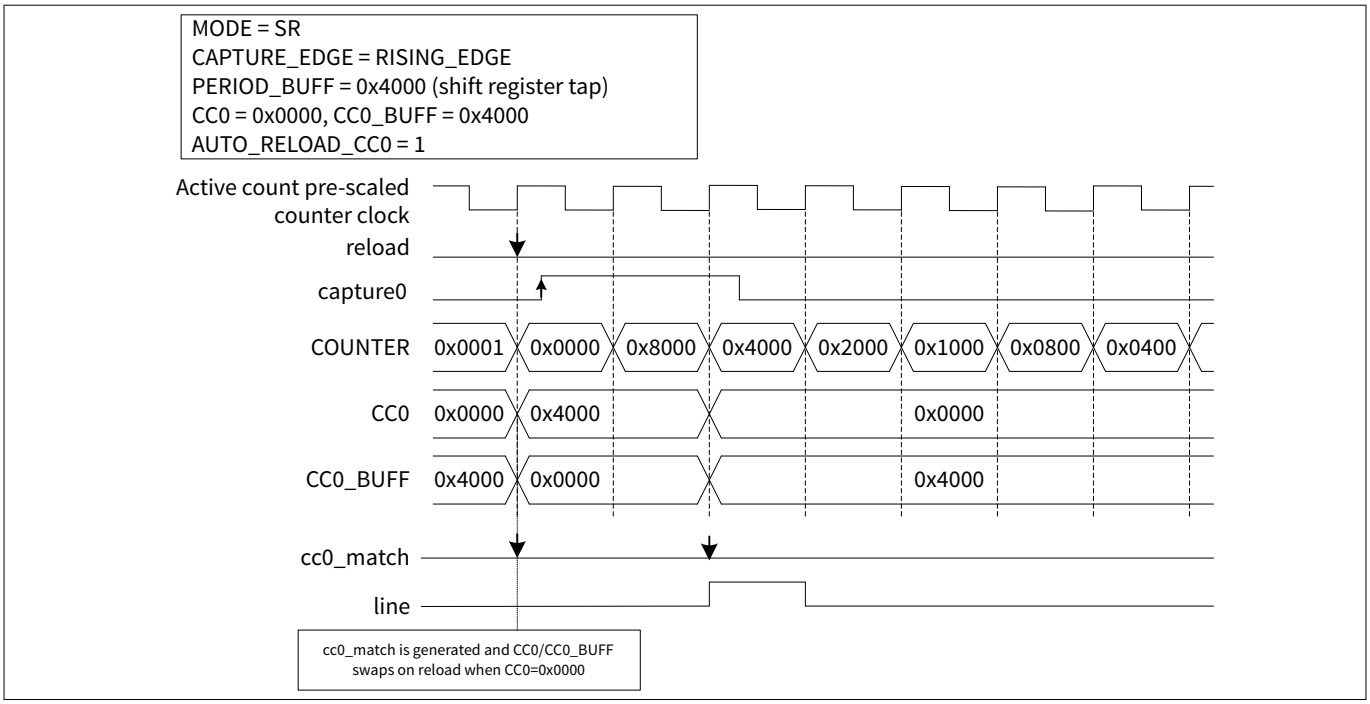


Figure 272 SR shift when PERIOD_BUFF = 0x4000

25.4 Recovery

TCPWM can be recovered with any active reset event, such as:

- Power-on reset (POR)
- External reset (XRES_L)
- Watchdog timer reset (MCWDT and WDT)
- Brownout detection reset
- Over-voltage and over-current detection reset

There is no unexpected state in which the TCPWM can enter.

25.5 Initialize

The initial state of TCPWM pins is Hi-Z. Some registers are reset on an active reset; some of the MMIO Registers are retained in deep sleep. None of the registers are retained through hibernate or other low-power modes. An active reset will reset the pin state back to Hi-Z.

25.6 Pin status

When TCPWM is unused, the status for TCPWM pins will be Hi-Z. To disable TCPWM, make sure the ENABLED and PWM_DISABLE_MODE bits in the CTRL Register are set to ‘0’.

25.7 Registers

Table 233 List of TCPWM Registers

Register	Name	Description
TCPWMx_GRPy_CNTz	Prefix of dedicated counter z Register in counter group y for TCPWM instance x	More details are available in the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual
TCPWMx_GRPy_CNTz_CTRL	Counter Control Register	Selects the counter mode and debug mode, and enables the counter
TCPWMx_GRPy_CNTz_STATUS	Counter Status Register	Reads the direction of counting and dead time duration, and indicates the actual level of trigger input and trigger outputs signals; checks if the counter is running
TCPWMx_GRPy_CNTz_COUNTER	Counter Count Register	Contains the 16- or 32-bit counter value
TCPWMx_GRPy_CNTz_CC0	Counter Compare/Capture 0 Register	Captures the counter value 0 or compares the value with counter value 0
TCPWMx_GRPy_CNTz_CC0_BUFF	Counter Buffered Compare/Capture 0 Register	Buffer Register for Counter CC0 Register
TCPWMx_GRPy_CNTz_CC1	Counter Compare/Capture 1 Register	Captures the counter value 1 or compares the value with counter value 1
TCPWMx_GRPy_CNTz_CC1_BUFF	Counter Buffered Compare/Capture 1 Register	Buffer Register for Counter CC1 Register
TCPWMx_GRPy_CNTz_PERIOD	Counter Period Register	Contains upper value of the counter
TCPWMx_GRPy_CNTz_PERIOD_BUFF	Counter Buffered period Register	Buffer Register for Counter Period Register
TCPWMx_GRPy_CNTz_LINE_SEL	Counter Line Selection Register	Selects the source for the LINE_OUT and LINE_COMPL_OUT output signals
TCPWMx_GRPy_CNTz_LINE_SEL_BUFF	Counter Buffered Line Selection Register	Buffer Register for the LINE_SEL Register
TCPWMx_GRPy_CNTz_DT	Counter PWM Dead Time Register	Configuration of PWM dead time affecting LINE_OUT and LINE_COMPL_OUT signals
TCPWMx_GRPy_CNTz_DT	Counter Trigger Command Register	Enables software-controlled operation for this counter. It includes the software trigger for CAPTURE0, CAPTURE1, RELOAD, START, and STOP

(table continues...)

Table 233 (continued) List of TCPWM Registers

Register	Name	Description
TCPWMx_GRPy_CNTz_TR_IN_SEL0	Counter Input Trigger Selection Register 0	Selects triggers for specific counter events: CAPTURE0, COUNT, RELOAD, or STOP event
TCPWMx_GRPy_CNTz_TR_IN_SEL1	Counter Input Trigger Selection register 1	Selects triggers for specific counter events: CAPTURE1 or START event
TCPWMx_GRPy_CNTz_TR_IN_EDGE_SEL	Counter Input Trigger Edge Selection Register	Determines edge detection for specific counter triggers. Events will only take effect on enabled counters
TCPWMx_GRPy_CNTz_TR_PWM_CTRL	Counter Trigger PWM Control Register	Controls counter LINE_OUT, DT_LINE_OUT, and DT_LINE_COMPL_OUT output signals
TCPWMx_GRPy_CNTz_TR_OUT_SEL	Counter Output Trigger Selection Register	Selects internal events for output trigger generation
TCPWMx_GRPy_CNTz_INTR	Interrupt Request Register	Sets the register bit when TC or CC0/1 condition is detected
TCPWMx_GRPy_CNTz_INTR_SET	Interrupt Set Request Register	Sets the corresponding bits in Interrupt Request Register
TCPWMx_GRPy_CNTz_INTR_MASK	Interrupt Mask Register	Mask for Interrupt Request Register
TCPWMx_GRPy_CNTz_INTR_MASKED	Interrupt Masked Request Register	Bitwise AND of Interrupt Request and Mask Registers

***Note**: In TCPWMx_GRPy_CNTz, 'x' signifies TCPWM instance number, 'y' is the group number and 'z' is the counter in the respective TCWPM group.*

Note that overwriting the same value on each register has different effects and they are explained in the register map by the software access attributes. TCPWM Registers have the following access restrictions:

- All Status Registers are not software-writable.
- TR_CMD is set in software and cleared in hardware.
- INTR is cleared in software and set in hardware (by writing ‘1’ to INTR_SET).
- Read INTR_SET will return the value of INTR.
- Other registers are normal and can be overwritten with the same value.

26 Cryptography block

The cryptography block (Crypto) provides hardware implementation and acceleration of cryptographic functions. Implementation in hardware takes less time and energy than the equivalent firmware implementation. In addition, the block provides true random number generation functionality in silicon, which is not available in firmware. It supports symmetric key encryption and decryption, hashing, message authentication, random number generation (pseudo and true), cyclic redundancy checking, and utility functions such as enable/disable, interrupt settings, and flags.

26.1 Features overview

The cryptography function block of XMC7000 supports the following features:

- Advanced encryption standard (AES) functionality according to FIPS 197:

  The AES component can be used to encrypt/decrypt data blocks of 128-bit length and supports programmable key length (128/192/256-bit key).
- CHACHA20 functionality according to RFC 7539:

  CHACHA20 is a stream cipher, which produces output consisting of 512-bit random-looking bits. These random-bits can be XORed with plain-text to produce cipher-text.
- Triple data encryption standard (TDES):

  The TDES component can be used to encrypt/decrypt data blocks of 64-bit length using a 64-bit key.
- Secure hash algorithm (SHA) functionality according to FIPS 180-4/FIPS-202:

  This component can be used to produce a fixed-length hash (also called “message digest”) of up to 512 bits from a variable-length input data (called “message”). SHA1, SHA2, SHA3 hashes are supported.
- Cyclic redundancy check (CRC) functionality:

  This component performs a cyclic redundancy check with a programmable polynomial of up to 32-bits.
- String (STR) functionality:

  This component can be used to efficiently copy, set, and compare memory data.
- Pseudo random number generator (PR):

  This component generates pseudo random numbers in a fixed range. This generator is based on three Linear Feedback Shift Registers (LFSRs).
- True random number generator (TR):

  This component generates true random numbers of up to 32 bits using ring oscillators.
- Vector unit (VU):

  This component acts as a coprocessor to offload asymmetric key operations, such as RivestShamir-Adleman(RSA) and Elliptic Curve (ECC), from the main processor.
- AHB master-interface:

  This allows to fetch operands directly from the system memory.
- Device key functionality:

  The device key has its usage restricted to specific functionality; they cannot be accessed by the software that implements that functionality. Two independent device keys are supported.

26.2 System diagram

The cryptography block provides the cryptography functionality on XMC7000. The complete cryptography implementation is done in conjunction with third-party software. In a secure system implementation, the cryptography block can be accessed only by the secure master (CM0+). For other masters to avail any cryptography services, they need to request CM0+ via system calls using IPC. For details, see the Inter-processor communication.

26.3 Block diagram

This section explains the major components within the cryptography block.

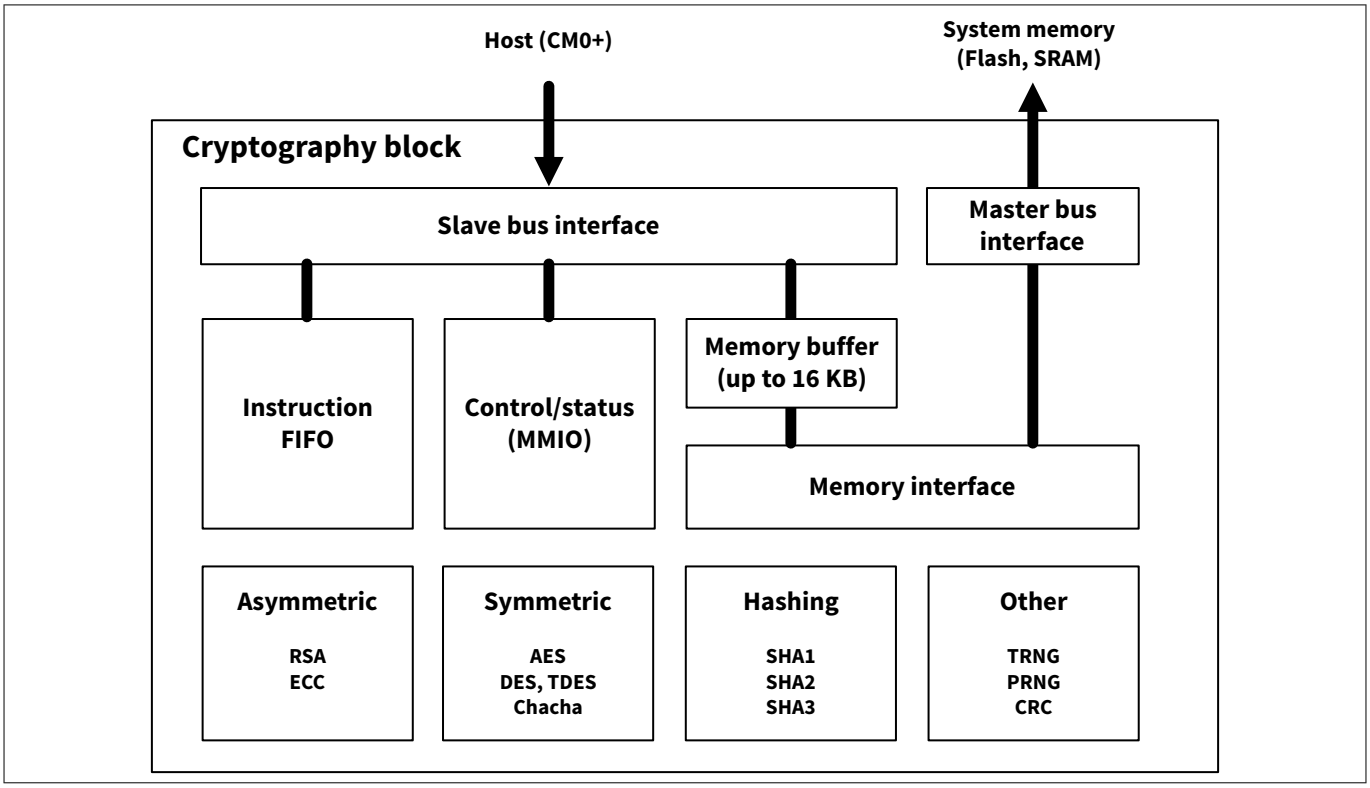


Figure 273 High-level block diagram

The cryptography block provides cryptographic functionality:

- DES, triple DES, AES, and Chacha20 symmetric key ciphers.
- SHA1, SHA2, and SHA3 hashes.
- Pseudo and true random number generators.
- Vector unit for asymmetric key cryptography.
- CRC functionality.

The cryptography block is connected to the AHB-Lite bus infrastructure through a slave bus interface and a master bus interface. The block has the following interfaces:

- An AHB-Lite slave interface connects the cryptography block to the AHB-Lite infrastructure. This interface supports 8/16/32-bit AHB-Lite transfers. MMIO Registers accesses are 32-bit accesses only (8/16-bit accesses to MMIO Registers results in an AHB-Lite bus error). Memory buffer accesses can be 8/16/32-bit accesses.
- An AHB-Lite master interface connects the cryptography block to the AHB-Lite infrastructure. This interface supports 8/16/32-bit AHB-Lite transfers. The interface enables the Crypto block to access operation operand data from system memory (for example, flash or SRAM).
- A single interrupt signal is used to indicate the completion of an operation.
- A clock and reset signal interface connects to the system resources subsystem (SRSS). The cryptography block operates a gated version of “clk” and uses both active and deep sleep reset signals.

26.4 Function description

The basic functions of the cryptography block are described here.

26.4.1 Operating mode

The cryptography block operates only in active/sleep/LPActive/LPsleep power modes. In deep sleep mode, the block retains only the contents of its Retention Registers with optional retention of internal SRAM contents.

26.4.2 Memory map and register definitions

The memory map and register definitions for the cryptography block are located in the product register map.

26.4.3 Instruction set

Most operations in the cryptography block are initiated through an instruction by CM0+ via IPC.

27 Event generator (EVTGEN)

The event generator (EVTGEN) in XMC7000 implements event generation for interrupts and triggers in active mode and only interrupts in deep sleep mode. The active functionality interrupt is connected to the CPU interrupt controller. Active trigger events can be used to trigger a specific device functionality mode (for example, execution of an interrupt handler, a SAR ADC conversion, and so on) in active power mode. The deep sleep functionality interrupts can be used to wake up the CPU from the deep sleep power mode. The event generator includes a single counter and a maximum of 16 comparator structures for each active and deep sleep mode. EVTGEN reduces CPU involvement and thus overall power consumption and jitters.

This chapter explains the features, implementation, and operational modes of the event generator block

27.1 Features

- CPU-free triggers for device functions
- Reduces CPU involvement in triggering device functions, thus reduces overall power consumption and CPU bandwidth
- 16 comparators for each deep sleep and active mode to generate interrupts and triggers
- 32-bit counter, one each for deep sleep and active mode for comparators
- Individual configurable thresholds for each comparator
- Deep sleep and active mode clock sources for counters
- Jitter-free initiation of specific device functionality
- One deep sleep and one active mode interrupt for CPU
- Supported in active, sleep, LPActive, LPSleep, and deep sleep power modes

27.2 Block diagram

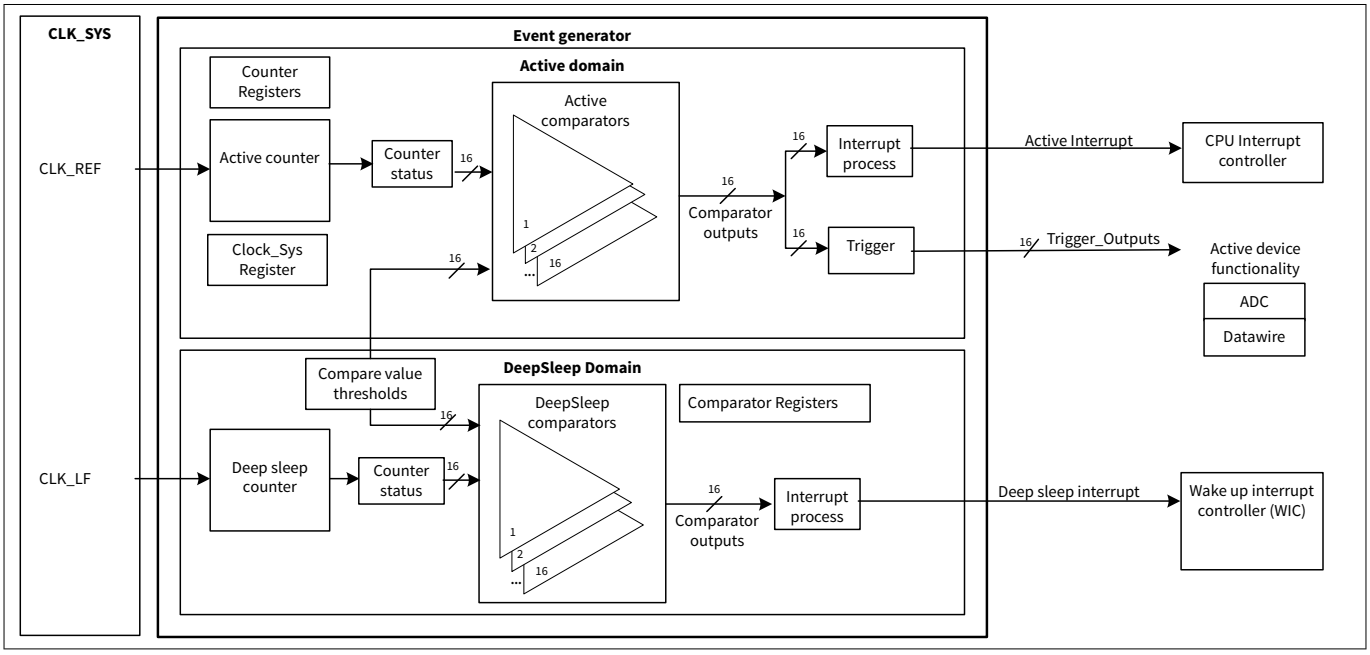


Figure 274 EVTGEN block diagram

The EVTGEN consists of two blocks: Active and deep sleep mode blocks. There are 16 comparator structures and one 32-bit counter for each of the modes. The EVTGEN block has these interfaces:

- Bus interface – Connects the block to the CPU subsystem.
- Trigger interface – Provides one trigger signal from each active mode comparators (EVTGEN_TR_OUTy).

- System interface – Consists of control signals such as clock and reset from the system resources subsystem.
- Interrupts – Provides one interrupt signal from active and deep sleep mode blocks, based on the comparator outputs.

This EVTGEN block can be configured by writing to the EVTGEN Registers. See Register list for more information on all registers required for this block.

27.2.1 Enabling and disabling EVTGEN block

The EVTGEN block can be enabled by setting the Enable bit of the EVTGENx_CTL Register. All non-retention (not retained in sleep mode) Registers (Command and Status Registers) are reset to their default value when this is disabled. All retention (retained in sleep mode) registers retain their value when this is enabled.

27.2.2 Counters

There is one 32-bit counter for each of the active and the deep sleep modes. These counters keep track of time; the time measured is referenced with respect to the CLK_REF clock.

27.2.2.1 Clock and prescaling

The counter working is based on the following two clocks from f_{EVTGEN}.

- **CLK_REF:** Time is measured with respect to a divided version of this clock – CLK_REF_DIV. The divider value is provided by EVTGENx_REF_CLOCK_CTL.INT. The CLK_REF_DIV clock is assumed to have a higher frequency and a higher precision than CLK_LF. The clock is available only in active power mode. Typically, CLK_REF is connected to a high-precision SRSS clock source (for example, a PLL).
- **CLK_LF:** This is a low-frequency clock (typically around16 kHz to 32 kHz). The clock is assumed to have a lower precision than CLK_REF. It is available in both Active and Deep Sleep power modes. Typically, CLK_LF is connected to an 32-kHz internal CLK_ILO (low-frequency clock).

Comparator components are used to compare time with a programmed value and generate control signals when the counter exceeds the programmed value. The clock CLK_REF_DIV provides fine resolution (high frequency) and high precision. This clock is not available in deep sleep power mode. Deep sleep control signals are generate based only on CLK_LF.

- Clock CLK_REF used to generate active control signals.
- Clock CLK_LF is used to generate deep sleep control signals.

The EVTGEN block has the following clocking conditions. f_{EVTGEN} is the clock frequency of the EVTGEN block.

1. $f_{EVTGEN} > f_{CLK_REF} \geq f_{CLK_REF_DIV}$
2. $f_{CLK_REF_DIV} \geq 4 \times f_{CLK_LF}$

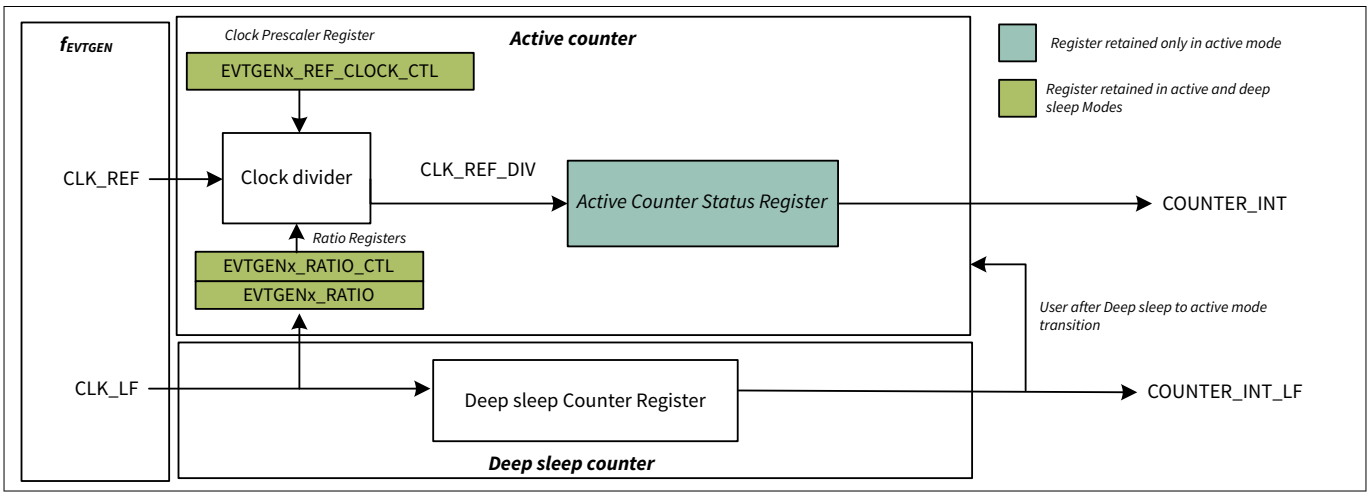


Figure 275 Counter block diagram

27.2.2.2 Ratio

All the count registers and comparator count thresholds are expressed with respect to CLK_REF_DIV domain.

The number of CLK_REF_DIV cycles per CLK_LF cycle can be controlled in either software or hardware.

$$Ratio = \frac{CLK_REF_DIV}{CLK_LF} \tag{9}$$

27.2.2.3 Software control

The software control is provided through EVTGENx_RATIO. This register contains a ratio value expressing the relative frequency of CLK_LF with respect to CLK_REF_DIV. Specifically, this registers contains the average number of CLK_REF_DIV cycles per CLK_LF cycle. The RATIO value has a 16-bit integer component (EVTGENx_RATIO.INT) and an 8-bit fractional component (EVTGENx_RATIO.FRAC). This register is retained in deep sleep mode.

27.2.2.4 Hardware control

Hardware control auto-calibrates and makes the New Ratio value available through the EVTGENx_RATIO Register. Auto-calibration continuously measures the ratio and combines this new measurement with the current RATIO value to calculate the new RATIO value. This calculation is based on a weighted average operation. The weights of the new measurement and the current RATIO value are controlled through EVTGENx_RATIO_CTL Register.

The weighted average operation addresses jitter in the low-frequency clock CLK_LF. The weights of the weighted average calculation try to trade off clock CLK_LF jitter sensitivity and speed of adaptability to a new clock CLK_LF frequency. Note that gradual changes to CLK_LF may occur to differences in operating conditions (such as temperature).

Auto calibration is active functionality logic; that is, the RATIO value is not updated in deep sleep power mode. However, the RATIO value is retained in deep sleep mode.

Hardware control is enabled through DYNAMIC bit in EVTGENx_RATIO_CTL Register. The weight for average calculation is the 3-bit value, which can be set using DYNAMIC_Mode bits in EVTGENx_RATIO_CTL.

The EVTGENx_RATIO Register fields INT16 and FRAC8 are only valid when the VALID bit in EVTGENx_RATIO_CTL is one. This register is retained in deep sleep mode.

The RATIO value is required in the EVTGENx_RATIO.INT16 and EVTGENx_RATIO.FRAC8 register fields. Either:

- Hardware establishes the RATIO value. This process takes a maximum of two CLK_LF cycles.
- Software provides the RATIO value in the EVTGENx_RATIO.INT16 and EVTGENx_RATIO.FRAC8 register fields. This process is immediate.

The RATIO value is used to initialize the deep sleep counter. This process takes one CLK_LF cycle.

27.2.3 Counter status

The active counter functionality is available only in active power mode. This is an UP counter and auto reloads itself. The active counter is not retained in deep sleep power mode. The status of active counter can be read through EVTGENx_COUNTER Register. This register is not retained in deep sleep mode.

The active COUNTER Register value is only valid when the EVTGENx_COUNTER_STATUS has valid bit set.

- After a deep sleep to active power mode transition, the active counter is not immediately valid. On the first CLK_LF clock after the power mode transition, the deep sleep counter value is used to initialize the active counter.
- On every CLK_REF_DIV clock, the active counter is incremented by '1'.

The deep sleep counter functionality is available in both the active and deep sleep power modes.

On every CLK_LF, the RATIO value is added to the deep sleep counter status. On every CLK_LF cycle, the status of the deep sleep counter will be the same as that of the active Counter. The status of the deep sleep counter is not accessible in active or deep sleep mode.

Figure 276 illustrates an example where CLK_REF_DIV is five times as fast as CLK_LF (RATIO = 5).

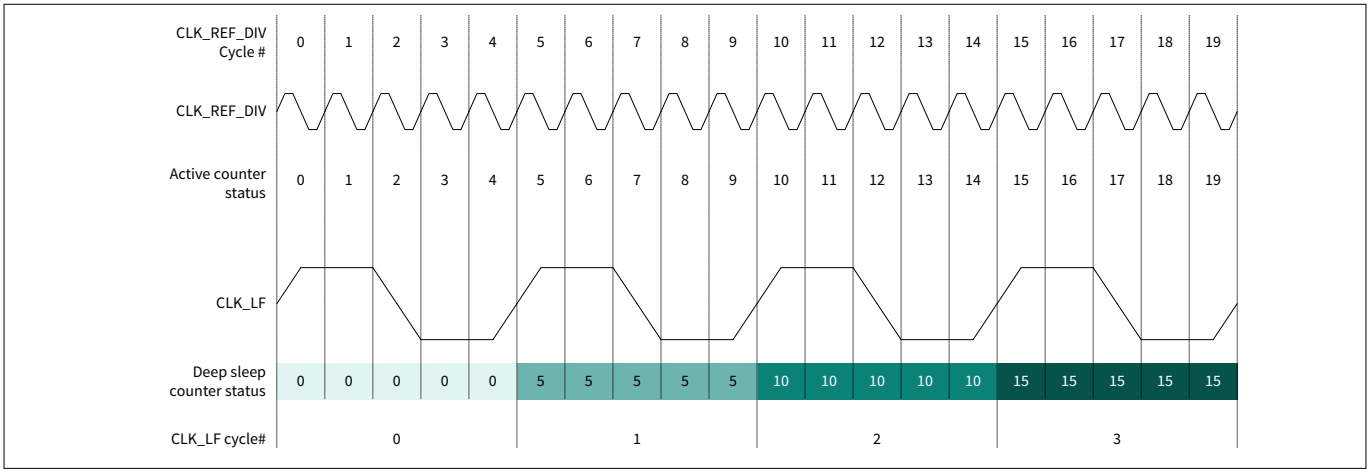


Figure 276 Active and Deep sleep counter status with ratio = 5

After a wakeup from the Deep sleep power mode, the active counter is re-initialized. The deep sleep counter will initialize the active counter and EVTGENx_COUNTER_STATUS.VALID is set to '1'. This process will take at most two CLK_LF cycles.

If the RATIO value (the average number of CLK_REF_DIV cycles per CLK_LF cycle) is not established, the deep sleep counter is not valid.

If the RATIO value is established:

- On the first CLK_LF clock, the current RATIO value is used to initialize the deep sleep counter.
- On every other CLK_LF clock (Deep sleep counter is initialized), the Deep Sleep counter update occurs based on the current power mode.

27.2.4 Deep Sleep Counter update

As long as the module is in the Active power mode, the Deep Sleep counter copies the value of the Active counter on every posedge of "CLK_LF". This ensures that as long as the module is in the Active power mode, both the Active and Deep Sleep counters are in sync.

For this purpose, the Active power domain maintains a "shadow copy" of the Active counter (counter_int_copy), which updates on every CLK_LF tick. In Active power mode,this register (counter_int_copy) is used by the Deep Sleep counter to update its value.

To compensate this, the Deep Sleep counter updates as:

counter_int_lf_next = counter_int_copy +EVTGENx.RATIO.INT16

However, when the system enters Deep Sleep power mode, the Deep Sleep counter will increment based on the "RATIO" register value as follows:

{counter_int_lf_next,counter_frac_lf_next} = {counter_int_lf,counter_frac_lf} + {EVTGENx.RATIO.INT16,EVTGENx.RATIO.FRAC8}.

In order to detect whether we have entered Deep Sleep power mode or not, the module maintains a shadow copy of the "counter_int_copy" register in the Deep Sleep power domain (counter_int_copy_lf). On every posedge of "CLK_LF", if the value of the "counter_int_copy" is different from the "counter_int_copy_lf", then this indicates "Active" power mode. In this case, we update :

counter_int_copy_lf_next = counter_int_copy

However, if these counters are the same on a given posedge of "CLK_LF", then this indicates that the module has entered Deep Sleep power mode. In this case, the counter_int_copy_lf does not update.

27.2.5 Comparator structures

The EVTGEN block supports sixteen comparator structures. Each comparator structure has one comparator in active mode and one comparator in deep sleep mode.

One set of active and deep sleep mode comparators (i.e., one comparator structure) have one individual control Register. Each comparator structure compares the active COUNTER_INT and deep sleep COUNTER_INT_LF with an active and deep sleep compare value respectively.

The active functionality is enabled through enable EVTGENx_COMP_STRUCTy_COMP_CTL COMP0_EN bit in the Comparator Control Register. Similarly, EVTGENx_COMP_STRUCTy_COMP_CTL.COMP1_EN bit in the comparator control register enables/disables deep sleep comparator. There is one of this register for every pair consisting of an active and a deep sleep comparator. This register is retained in deep sleep mode.

Trigger method for the active comparator can be selected through TR_OUT_EDGE.

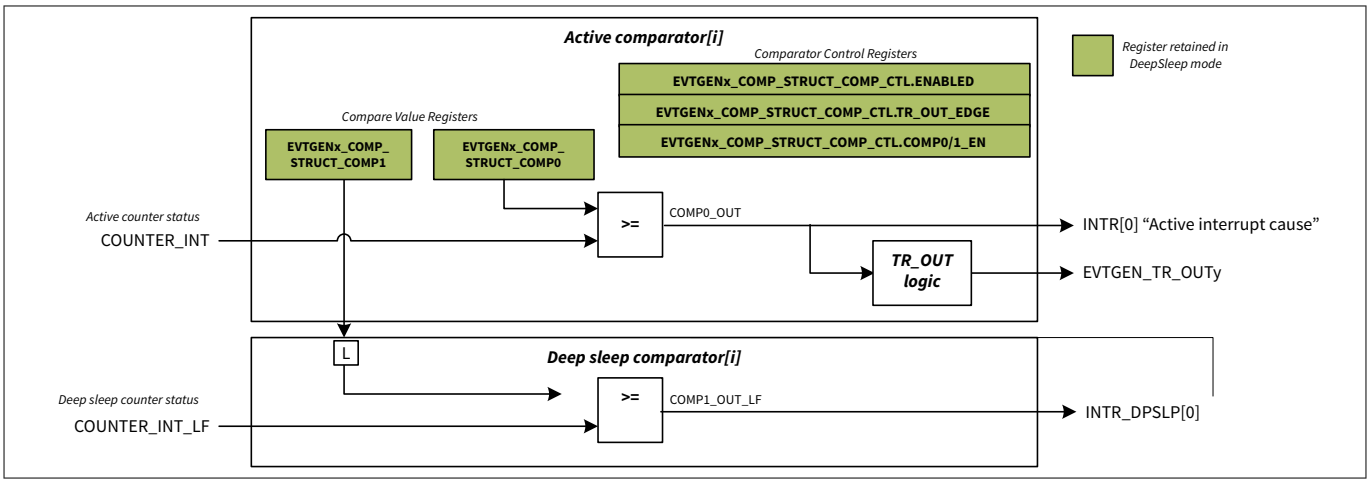


Figure 277 Comparator structure

The active counter EVTGENx_COUNTER is compared to the EVTGENx_COMP_STRUCTy_COMP0 Register value.

- The active comparator COMP0_OUT output is activated when the active counter becomes greater than or equal to the active compare value.
- The active comparator COMP0_OUT output is deactivated, when the comparator is disabled (EVTGENx_COMP_STRUCTy_COMP_CTL.COMP0_EN).

The deep sleep compare functionality is available in both Active and deep sleep power modes. The functionality is enabled through EVTGENx_COMP_STRUCTy_COMP_CTL.COMP1_EN. The deep sleep counter status is compared to the EVTGENx_COMP_STRUCTy_COMP1 Register value.

- The deep sleep comparator COMP1_OUT_LF output is activated, when the deep sleep counter becomes greater than or equal to the deep sleep compare value.
- The deep sleep comparator COMP1_OUT_LF output is deactivated, when the comparator is disabled (EVTGENx_COMP_STRUCTy_COMP_CTL.COMP1_EN).

EVTGENx_COMP_STRUCTy_COMP0 and COMP Registers hold the compare values for the active and deep sleep comparators respectively. There is one register for every one pair consisting of one active and one deep sleep comparator. These registers are retained in deep sleep mode.

When the active counter is initialized (EVTGENx_COUNTER_STATUS.VALID is '1'), COUNTER and future EVTGENx_COMP_STRUCT_COMP0 and EVTGENx_COMP_STRUCT_COMP1 comparator values can be read and programmed. These future comparator values should have a minimum delay with respect to the active counter

value COUNTER. This is to ensure that the counter value has not passed these future values before the comparators are enabled.

The value written to EVTGENx_COMP_STRUCT_COMP0 or EVTGENx_COMP_STRUCT_COMP1 should be at least four CLK_LF cycles ahead of the current COUNTER value. A comparator structure ‘y’ produces a EVTGEN_TR_OUTy trigger and interrupt cause signals.

The active functionality EVTGEN_TR_OUTy trigger is available only in the active power mode.

- The EVTGEN_TR_OUTy trigger is activated, when the active comparator status COMP0_OUT[i] is activated.
- The EVTGEN_TR_OUTy trigger is deactivated, when the comparator is disabled (EVTGENx_COMP_STRUCTy_COMP_CTL.COMP0_EN).

The trigger EVTGEN_TR_OUTy can be used to trigger specific device functions such as execution of an interrupt handler, a SAR ADC conversion.

The status an active comparator can be read from corresponding bit in EVTGENx_COMP0_STATUS Register. The status an deep sleep comparator can be read from corresponding bit in EVTGENx_COMP1_STATUS Register. These register are retained in deep sleep mode.

The active interrupt cause signal is available only in the active power mode.

- The cause is activated, when the active comparator is activated.
- The cause is activated by software through EVTGENx_INTR_SET.COMP0[i].

The deep sleep interrupt cause signal is available in active and deep sleep power modes.

- The cause is activated, when the deep sleep comparator is activated.
- The cause is activated by software through EVTGENx_INTR_DPSLP.COMP1[i].

Typically, the active and deep sleep comparators are used as follows:

- The deep sleep comparator value (EVTGENx_COMP_STRUCT_COMP1) is set to a time X on CLK_LF. The intent is to wake up the device (SRSS wakeup signal) and to ensure that the device is in active power mode at time X+wakeup time.
- The active comparator value (EVTGENx_COMP_STRUCT_COMP0) is set to a time X+wakeup time. The intent is to use the associated output trigger EVTGEN_TR_OUTy to initiate a specific device functionality in active power mode.
- On completion of the specific device functionality, the functions interrupt signal is activated. The CPU interrupt handler may process the result of the specific functionality. The CPU may also setup the event generator and may turn the device to deep sleep power mode through a WFI instruction (resulting in activation of the SRSS deep sleep signal).

27.2.6 Interrupts

The EVTGEN block uses two interrupts:

- **An active interrupt signal:** Each active comparator has a dedicated interrupt cause field. This interrupt notifies the CPU, when an output trigger is activated.
- **A deep sleep INT_DPSLP signal:** Each deep sleep comparator has a dedicated interrupt cause field. This interrupt is connected to CPUs' WICs and allows for wakeup from Deep sleep power mode. The active and deep sleep interrupts are available in the system interrupt sources.

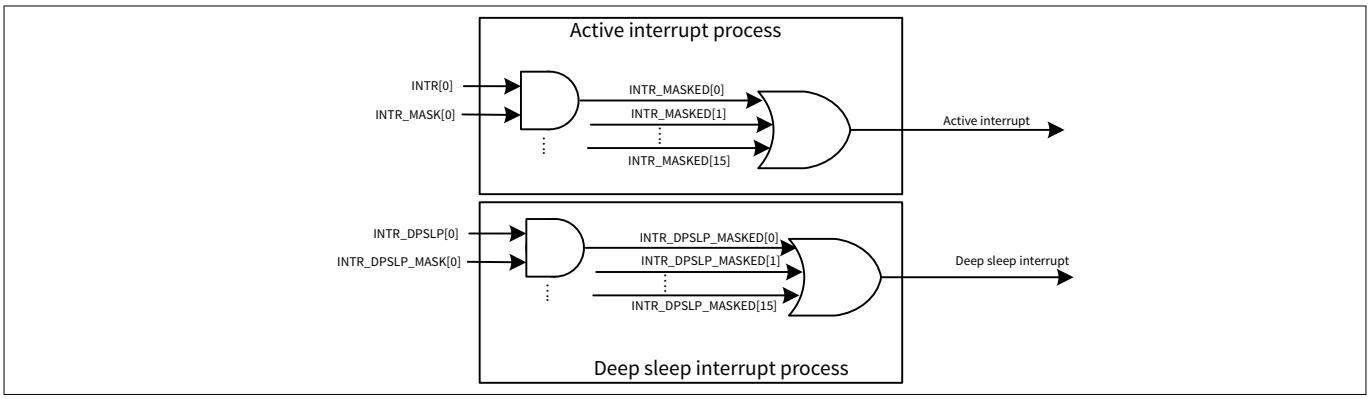


Figure 278 Interrupt process

The active interrupt cause field register bit is set EVTGENx_INTR when a corresponding comparator 0 (COMP0_OUT) event is generated. Each bit in this corresponds to one comparator. Writing one to this register will clear it. The EVTGENx_INTR register is not retained in deep sleep mode.

The deep sleep interrupt cause field register bit is set EVTGENx_INTR_DPSLP when a corresponding comparator 1 (COMP1_OUT_LF) event is generated. Each bit in this corresponds to one comparator. The EVTGENx_INTR_DPSLP Register is retained in deep sleep mode.

The EVTGENx_INTR_SET Register when read, reflects the EVTGENx_INTR Register. For debug purposes, software can write a '1' to activate a specific interrupt cause (this allows for debug of the software ISR, without relying on hardware to activate the interrupt cause). Each bit in this corresponds to one comparator. The EVTGENx_INTR_SET Register is not retained in deep sleep mode.

EVTGENx_INTR_DPSLP_SET Register when read reflects the EVTGENx_INTR_DPSLP Register. For debug purposes, software can write a '1' to activate a specific interrupt cause (this allows for debug of the software ISR, without relying on hardware to activate the interrupt cause). Each bit in this corresponds to one comparator. The EVTGENx_INTR_DPSLP_SET Register is retained in deep sleep mode.

The mask for corresponding bit field of the active comparator can be set using EVTGENx_INTR_MASK Register. The EVTGENx_INTR_MASK Register is retained in deep sleep mode.

The mask for corresponding bit field of the deep sleep comparator can be set using EVTGENx_INTR_DPSLP_MASK Register. The EVTGENx_INTR_MASK Register is retained in deep sleep mode.

EVTGENx_INTR_MASKED Register reflect the logical AND of corresponding EVTGENx_INTR with EVTGENx_INTR_MASK fields. The EVTGENx_INTR_MASKED Register is not retained in deep sleep mode.

EVTGENx_INTR_DPSLP_MASKED Register reflects logical AND of corresponding EVTGENx_INTR_DPSLP with EVTGENx_INTR_DPSLP_MASK fields. The EVTGENx_INTR_DPSLP_MASKED Register is retained in deep sleep mode.

Logical OR operation is applied to all the bit field of EVTGENx_INTR_MASKED Register to generate the active interrupt signal that is connected to the CPU interrupt controller

Logical OR operation is applied to all the bit field of EVTGENx_INTR_DPSLP_MASKED Register to generate the deep sleep interrupt signal that is connected to the CPUs' wakeup interrupt controllers (WICs). When enabled in WIC, this signal activation will wake up the CPU from deep sleep power mode to active power mode.

27.2.7 Usage Guidelines

EVTGEN requires the following clock sources for its operation:

- CLK_REF
- CLK_LF

Before enabling the module, ensure that all clock sources are available. The following sequence of steps should be followed for programming the module:

- Program register EVTGENx.REF_CLOCK_CTL to set the value of "CLK_REF_DIV".
- If SW controls the RATIO value, set the value of "EVTGENx.RATIO" register to the desired value.
- Program register EVTGENx.RATIO_CTL to set the value of the following fields:

```
If (SW control of RATIO)
{
EVTGENx.RATIO_CTL.DYNAMIC=0;
EVTGENx.RATIO_CTL.VALID=1;
}
Else
{
EVTGENx.RATIO_CTL.DYNAMIC=1;
EVTGENx.RATIO_CTL.DYNAMIC_MODE = <desired_value>;
EVTGENx.RATIO_CTL.VALID = 0; //HW is supposed to set this
}
```

- Enable the Interrupt for DeepSleep and Active comparators (EVTGENx.INTR_MASK, EVTGENx.INTR_DPSLP_MASK)
- Enable the module (EVTGENx.CTL.ENABLED)
- Wait for EVTGENx.COUNTER_STATUS.VALID to go to '1' (we can also use an interrupt EVTGENx.INTR.COUNTER)
- If you are using DYNAMIC_MODE ≥ 1, give sufficient DELAY in software for RATIO to stabilize (around 10-20 CLK_LFCLOCK cycles).
- Read the value of the COUNTER (EVTGENx.COUNTER).
- Program the counter values EVTGENx.COMP_STRUCTy.COMP0/1.
- Enable the comparators EVTGENx.COMP_CTL. Go to the Deep Sleep power mode.

27.2.8 Deep sleep interrupt accuracy analysis

In deep sleep power mode, the deep sleep counter value used to wake up from deep sleep is not highly accurate.

For hardware-based "RATIO" (EVTGENx_RATIO) calculation, i.e., if EVTGENx_RATIO_CTL.DYNAMIC=1, the hardware will lock the value of RATIO (CLK_REF_DIV/CLK_LF) during active power mode. On entering deep sleep power mode, the EVTGEN block will rely on this value of "RATIO" to calculate the "wake-up" time. Any error in estimation of "RATIO" will result in wake-up time inaccuracy.

For hardware-based "RATIO" calculation, the maximum ERROR in estimation of "RATIO" is 1 CLK_REF_DIV.

$$Max\ Error\ in\ Ratio = 1\ CLK_REF_DIV \tag{10}$$

Thus, in deep sleep mode for every "tick" of CLK_LF, a maximum error of 1 CLK_REF_DIV is accumulated.

The total error in wake-up time can be estimated as follows:

Assume, W is the expected wake-up time, then number of CLK_LF ticks in this time is:

$$clk_(lf_(ticks_wakeup)) = W/(Tclk_lf) = W/(RATIO * Tclk_ref_div) \quad (11)$$

Overall error in wakeup time = (clk_lf_ticks_wakeup) * * t CLK_REF_DIV

Error estimate = W/RATIO

This formula is only an approximation considering the maximum error case. It provides one important conclusion:

27.2.8.1 Higher the RATIO, lower the error

Hence, it is very important to keep a large value of RATIO to improve the accuracy in deep sleep wake-up time. One recommendation is to keep this RATIO ≥ 100. This ensures an approximate accuracy of 99% or an error of 1%.

27.2.9 Use case

The following waveform illustrates a single wakeup/deep sleep sequence (the wakeup and deep sleep signals initiate SRSS power mode transitions, the EVTGEN_TR_OUTy trigger initiates active functionality).

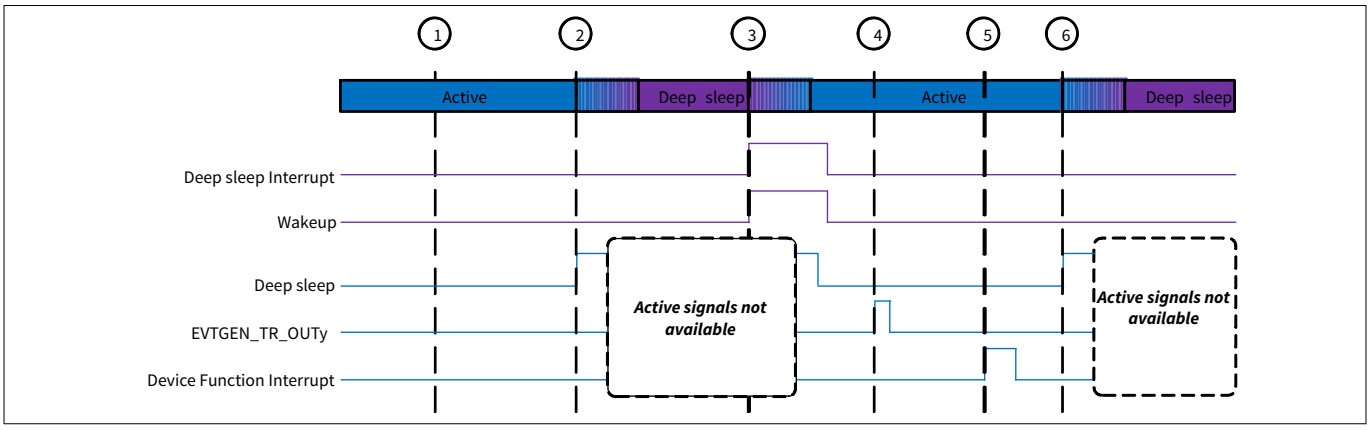


Figure 279 Use case waveform

The waveforms are explained as follows.

1. The CPU reads the active counter and sets the deep sleep comparator to wake up the device at time 3 and the active comparator to generate a trigger at time 4.
2. The CPU enters deep sleep power mode. For example, a WFI instruction with the deep sleep bit field set to '1'. When the CPU is in deep sleep power mode, its deep sleep signal is activated '1'.
3. The event generator activates its deep sleep interrupt and the WIC activates its SRSS wakeup request.
4. The event generator activates the trigger and device-specific functionality is initiated. The CPU may set up the event generator block before the transition to deep sleep power mode.
5. The device-specific functionality completes as indicated by the active interrupt. The CPU clears the function's interrupt causes. The CPU may process the results of the device-specific functionality.
6. The CPU enters deep sleep power mode.

27.2.10 Register list

Register	Name	Description
EVTGENx_CTL	Event Generator Control Register	This is the EVTGEN Module Enable/Disable Register.
EVTGENx_REF_CLOCK_CTL	Event Generator Clock divider Register	This register selects the reference clock divider.
EVTGENx_RATIO	Event Generator Ratio Register	This register selects the integer and fractional component of the ratio value. It contains a ratio value expressing the relative frequency of the deep sleep clock with respect to the active clock.
EVTGENx_RATIO_CTL	Event Generator Ratio Control Register	This register controls the RATIO new value, SW or HW control, and its validity.
EVTGENx_COUNTER	Event Generator Counter Register	This is the active EVTGEN counter.
EVTGENx_COUNTER_STATUS	Event Generator Counter Status Register	This register indicates whether the active EVTGEN counter is valid or invalid.
EVTGENx_COMP_STRUCTy_COMP_CTL	Event Generator Counter Control Register	This register enables/disables active/deep sleep EVTGEN comparators. It specifies the output trigger type, and enables/disables the comparator structure.
EVTGENx_COMP_STRUCTy_COMP0	Event Generator Active Comparator Compare Value Register	This is the unsigned 32-bit active comparator value.
EVTGENx_COMP_STRUCTy_COMP1	Event Generator Deep sleep Comparator Compare Value Register	This is the unsigned 32-bit deep sleep comparator value.
EVTGENx_COMP0_STATUS	Event Generator Active Comparator Status Register	This register gives the active comparator output status.
EVTGENx_COMP1_STATUS	Event Generator Deep sleep Comparator Status Register	This register gives the deep sleep comparator output status.
EVTGENx_INTR	Event Generator Interrupt Register	This interrupt cause field is activated (HW sets the field to '1') when a comparator 0 event is generated (active counter is greater or equal to COMP0.INT[31:0]).
EVTGENx_INTR_DPSLP	Event Generator Deep sleep Interrupt Register	This interrupt cause field is activated (HW sets the field to '1') when a comparator 1 event is generated (Deep sleep counter is greater or equal to COMP1.INT[31:0]).
EVTGENx_INTR_SET	Event Generator Active Interrupt Set Register	When read, this register reflects the INTR Register. For debug purposes, SW can write a '1' to activate a specific interrupt cause (this allows debugging of the SW ISR, without relying on HW to activate the interrupt cause).

Register	Name	Description
EVTGENx_INTR_DPSLP_SET	Event Generator Deep sleep Interrupt Set Register	SW writes a '1' to this field to set the corresponding field in the INTR Register.
EVTGENx_INTR_MASK	Event Generator Active Interrupt Mask Register	Mask bit for the corresponding field in the EVTGENx_INTR Register.
EVTGENx_INTR_DPSLP_MASK	Event Generator Deep sleep Interrupt Mask Register	Mask bit for the corresponding field in the EVTGENx_INTR_DPSLP Register.
EVTGENx_INTR_MASKED	Event Generator Active Interrupt Masked Register	Logical AND of corresponding EVTGENx_INTR and EVTGENx_INTR_MASK fields.
EVTGENx_INTR_DPSLP_MASKED	Event Generator Deep sleep Interrupt Masked Register	Logical AND of corresponding EVTGENx_INTR_DPSLP and EVTGENx_INTR_DPSLP_MASK fields.

***Note**: 'x' signifies the EVTGEN instance, and 'y' signifies the comparator structure number.*

28 Trigger multiplexer

Every peripheral in the XMC7000 device is interconnected using trigger signals. Trigger signals are means by which peripherals inform the occurrence of an event or transit to a state. These triggers are used to effect or initiate an action in other peripherals. They help the user to route triggers from a source peripheral to a destination.

Triggers are produced by a peripheral and consumed by another. Unlike interrupts, triggers are used to synchronize between peripherals, rather than between a peripheral and the Arm® CPU Core.

28.1 Features

Triggers are functional in the active power mode.

- Ability to connect any trigger signal from one peripheral to another,
- Provides up to 16 multiplexer-based trigger groups and up to 16 one-to-one trigger groups
- Supports a software trigger, which can trigger any signal in the block
- Ability to configure a trigger multiplexer with trigger manipulation features in hardware such as inversion and edge/level detection

28.2 Description

Triggers are used to synchronize the functionality of peripherals in hardware (as opposed to software-based synchronization). Peripheral DMA uses triggers to initiate the transfer of a data element from one address location to another. For example, an “ADC conversion done” event may initiate the transfer of an ADC sample from an ADC module to an SRAM memory location.

Triggers are digital signals generated by peripheral blocks to denote a state such as FIFO level, or an event such as completion of an action. These trigger signals typically serve as an initiator of other actions in other peripheral blocks. An example is an ADC peripheral block sampling four channels. After the conversion is complete, a trigger signal will be generated, which in turn triggers a DMA channel that transfers the ADC data to a memory buffer. This example is shown in Figure 280.

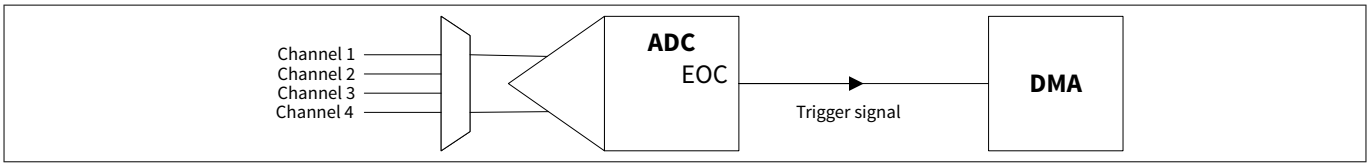


Figure 280 Trigger signal example

A XMC7000 device has multiple peripheral bocks; each of these blocks can be connected to other blocks through trigger signals, based on the system implementation. To support this, the device 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 any trigger signal emanating out of any peripheral block in the device and route it to any other peripheral to initiate or affect an operation at the destination peripheral block.

Triggers come in two types:

- High active, level-sensitive triggers: This type typically reflects a peripheral state. For example, tr_fifo_empty indicates that a FIFO is empty. The trigger remains '1' as long as the FIFO is empty. This type requires an action by the consumer of the trigger for the producer to deactivate the trigger. For example, tr_fifo_empty is deactivated by writing a data element to the associated FIFO. This trigger type can be produced on any clock.
- Rising edge, pulse triggers: This type typically reflects the occurrence of an event. For example, tr_adc_done indicates that a SAR ADC has converted a sample. This trigger type is produced on the

peripheral system interface clock: the trigger remains '1' for a two-cycle pulse on the peripheral system interface clock, and returns to '0' by itself.

If the consumer of the trigger cannot immediately react to the trigger, it needs to be able to remember that the trigger occurred. If the consumer of the trigger is confronted with multiple triggers in short succession, it may need to remember multiple triggers or decide to miss triggers.

Triggers should be treated as asynchronous signals between producer and consumer peripheral: the consumer peripheral should synchronize input triggers. In addition, for pulse triggers, the consumer peripheral may perform rising edge detection and memorize occurrence of the trigger. Treating triggers as asynchronous signals, eases timing closure (similar to DSI signals, triggers may travel a large distance).

At a high-level abstraction, a trigger is a wire connection between a producer and a consumer peripheral. However, at a more detailed level, several processing steps are distinguished. From a platform perspective, it is important these processing steps are implemented consistently. The following text elaborates on these processing steps for input triggers: trigger multiplexing, synchronization, edge detection, and storage. It also elaborates the processing step for output triggers.

28.3 Trigger multiplexing

The trigger component multiplexes trigger signals. The trigger input signals are typically peripheral output signals. The trigger output signals are typically peripheral input signals. Examples of trigger input signals are:

- TCPWM output signals; for example, counter reaches a pre-programmed limit.
- ADC output signals; for example, an ADC conversion has completed.
- I/O input signals.
- P-/M-DMA controller output signals that indicate the completion of a transfer.

Examples of trigger output signals are:

- TCPWM input signals; for example, “start a counter”.
- ADC input signals; for example, “start an ADC conversion”.
- I/O output signals.
- P-/M-DMA controller input signals to start a transfer.

In general, a trigger input signal indicates the completion of a peripheral action or a peripheral event. A trigger output signal initiates a peripheral action. Apart from this, there is aprovision for one-to-one triggers.

The decision for standard multiplexer components (as part of peripheral groups), rather than multiplexer components integrated as part of peripherals, has the following rationale:

- A standard multiplexer component enforces a consistent user interface.
- Any additional functionality provided by the multiplexer components, such as software control over output trigger signal activation, benefits all components.

The trigger module provides multiplexing functionality. It may be required to perform any of the following functions in a peripheral that uses the trigger output signals:

- Synchronization of the trigger signal to the peripheral clock domain
- Edge detection on the trigger signal
- Storing/remembering the trigger signal

A trigger component consists of multiple trigger groups. A trigger group can be of two types:

- A one-to-one-based connectivity group. This group type connects a peripheral input trigger to one specific peripheral output trigger.
- A multiplexer-based connectivity group. This type connects a peripheral input trigger to multiple peripheral output triggers. The selection is under software control: PERI_TR_GRx_TR_CTLy.TR_SEL.

The trigger component can provide up to 16 multiplexer-based trigger groups and up to 16 one-to-one trigger groups.

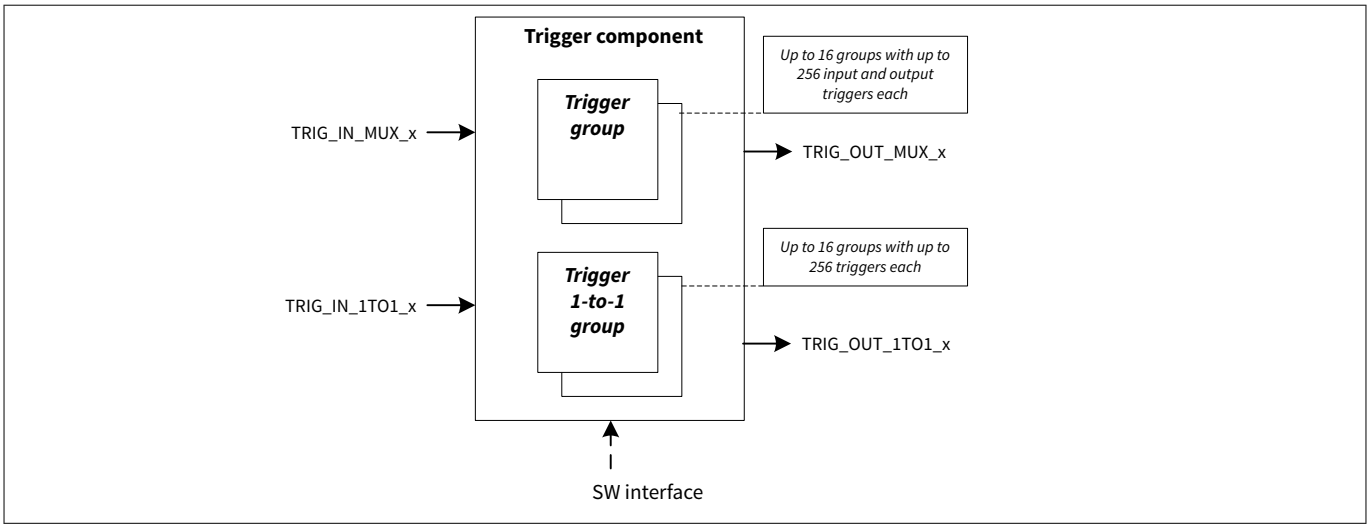


Figure 281 Trigger configuration parameters

Each group is associated with the trigger inputs of a specific peripheral. Figure 281 gives an overview.

Peripheral output triggers are connected to trigger component input triggers TRIG_IN_MUX_x and TRIG_IN_1TO1_x. Peripheral input triggers are connected to trigger component output triggers TRIG_OUT_MUX_x and TRIG_OUT_1TO1_x. It is important to distinguish the functionality that is provided by the trigger component in PERI and the functionality that is provided by the peripheral. The trigger component provides the following functionality (on TRIG_IN_MUX_x and TRIG_IN_1TO1_x):

- For a multiplexer-based connectivity group: an input trigger TRIG_IN_MUX_x is selected for each output trigger TRIG_OUT_MUX_x. Note that all output triggers in a group i share the same input triggers. For a one-to-one based group, an input trigger TRIG_IN_1TO1_x is connected to output trigger TRIG_OUT_1TO1_x.
- Software control is provided for trigger activation. This control allows for activation of a specific trigger. For level-sensitive triggers, the trigger activation is completely under software control. For edge-sensitive triggers, the trigger activation is two high/'1' cycles on CLK_PERI.
- Trigger propagation can be blocked in debug mode (typically when a CPU is halted). This allows it to isolate the trigger consumer peripheral from getting input triggers. The debug mode is indicated by the level trigger input x_DEBUG_FREEZE_TR_IN[5)], which is connected to a CPUSS CTI trigger output, CTI_TR_OUTx.
- Hardware edge-detection is provided to allow pulse triggers that transfer to the output trigger clock domain. This hardware performs asynchronous edge detection to support input triggers that operate on a higher clock frequency than the output trigger clock domain. This functionality is intended for pulse triggers (level triggers typically bypass the edge detection functionality).
- Hardware trigger signal level inversion.
- Most trigger multiplexers have all output signals connected to a common peripheral. The manipulation logic is tied to the clock for that peripheral. However, the debug multiplexer has outputs in many clock domains. This is valid because most destinations are level-sensitive signals, I/Os, or some other debug destinations that might not need any clock manipulations.

A peripheral may provide the following functionality:

- Synchronization of the output triggers TRIG_OUT_MUX_x and TRIG_OUT_1TO1_x.
- Edge detection of the synchronized output triggers.
- Storing/remembering the trigger signal.

5 x: TCPWM, PASS, PERI, SRSS_WDT, SRSS_MCWDT, etc.

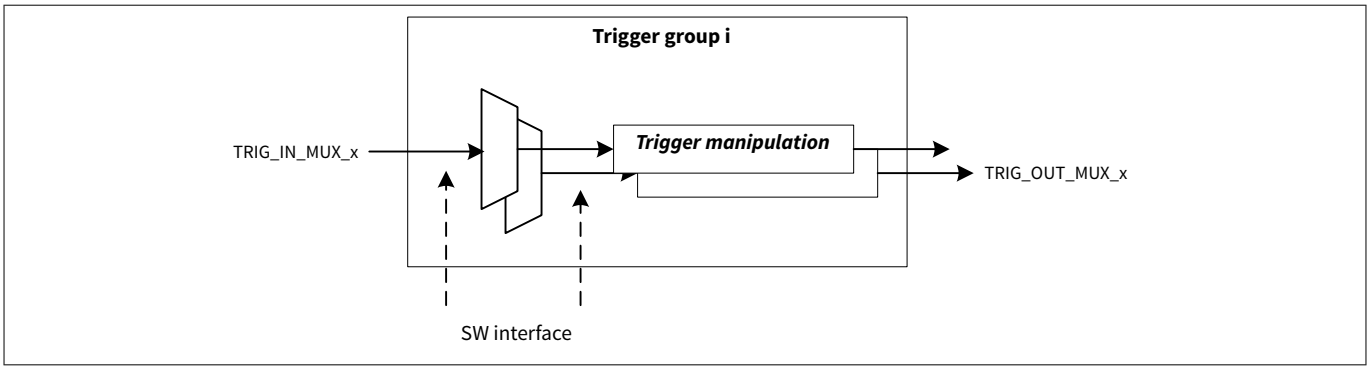


Figure 282 Trigger group

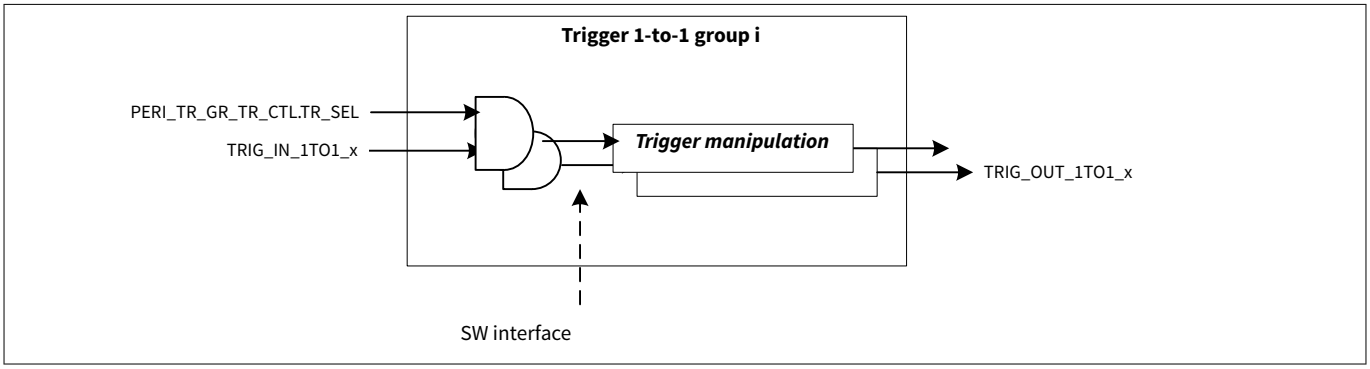


Figure 283 Trigger one-to-one group

Note that a one-to-one group has AND-gate functionality to disable an input trigger.

28.4 Trigger functionality

The following figure gives an overview of a multiplexer group.

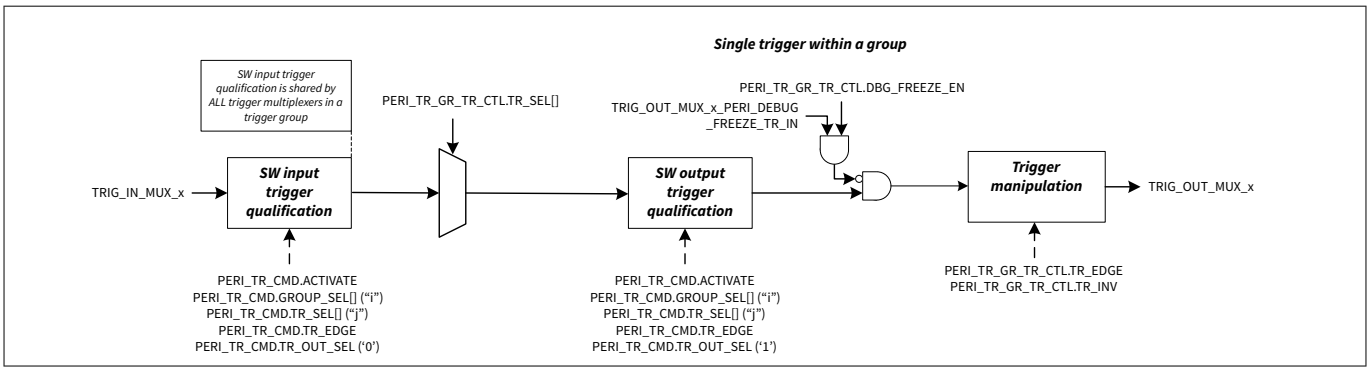


Figure 284 Multiplexer trigger group

Group trigger configuration:

PERI_TR_GR[Group Number].PERI_TR_GR_TR_CTL[Trigger Number].TR_SEL = Input trigger to be connected to
PERI_TR_GR[Group Number].PERI_TR_GR_TR_CTL[Trigger Number].TR_INV = Invert the trigger or not
PERI_TR_GR[Group Number].PERI_TR_GR_TR_CTL[Trigger Number].TR_EDGE = Edge or level-sensitive trigger

Activating the group trigger via software:

PERI_TR_CMD.TR_SEL = Input of Trigger number
PERI_TR_CMD.GROUP_SEL = Trigger number Group_Nr
PERI_TR_CMD.TR_EDGE = Edge or level-sensitive trigger
PERI_TR_CMD.OUT_SEL = 1 for output trigger and 0 for input trigger
PERI_TR_CMD.ACTIVATE = 1 signifies configured trigger activation

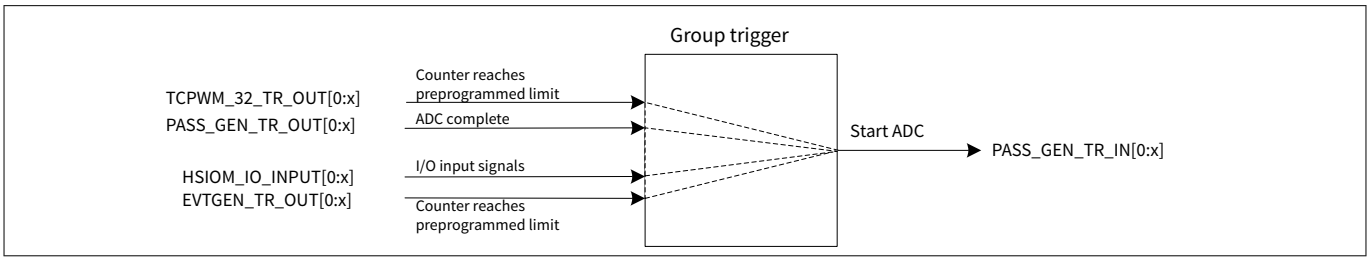


Figure 285 Example of group trigger

The following figure gives an overview of a one-to-one group.

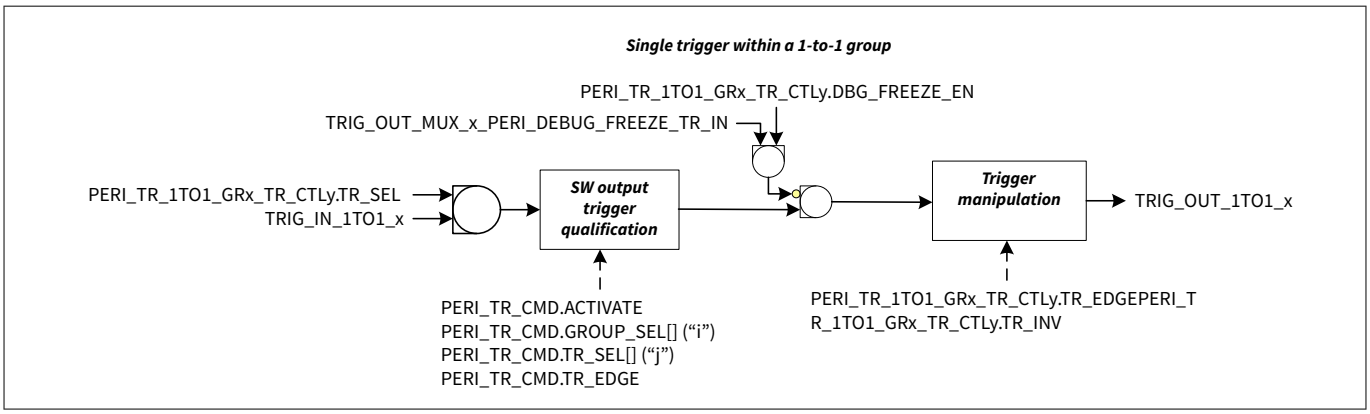


Figure 286 One-to-one trigger group

1TO1 trigger configuration:

PERI_TR_1TO1_GR[Group Number].PERI_TR_1TO1_GR_TR_CTL[Trigger Number].TR_SEL = True (input trigger) or false (constant signal level '0')

PERI_TR_1TO1_GR[Group Number].PERI_TR_1TO1_GR_TR_CTL[Trigger Number].TR_INV = Invert the output trigger or not

PERI_TR_1TO1_GR[Group Number].PERI_TR_1TO1_GR_TR_CTL[Trigger Number].TR_EDGE = Edge or level-sensitive trigger

A trigger group consists of multiple trigger multiplexers with associated trigger manipulation logic. All trigger multiplexers in a trigger group share the same number of input triggers.

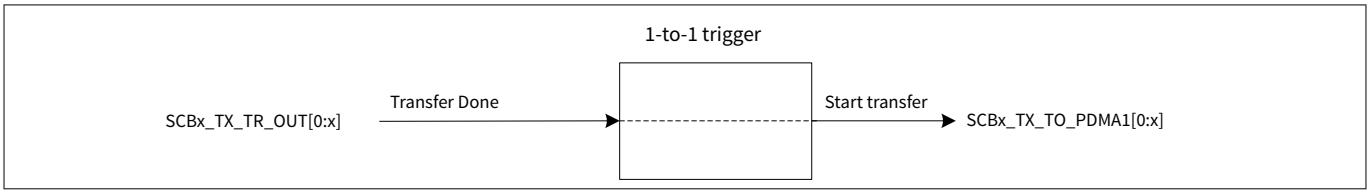


Figure 287 Example of 1-to-1 trigger

As mentioned, the trigger manipulation logic provides asynchronous edge detection logic. The trigger manipulation is detailed in the following figure.

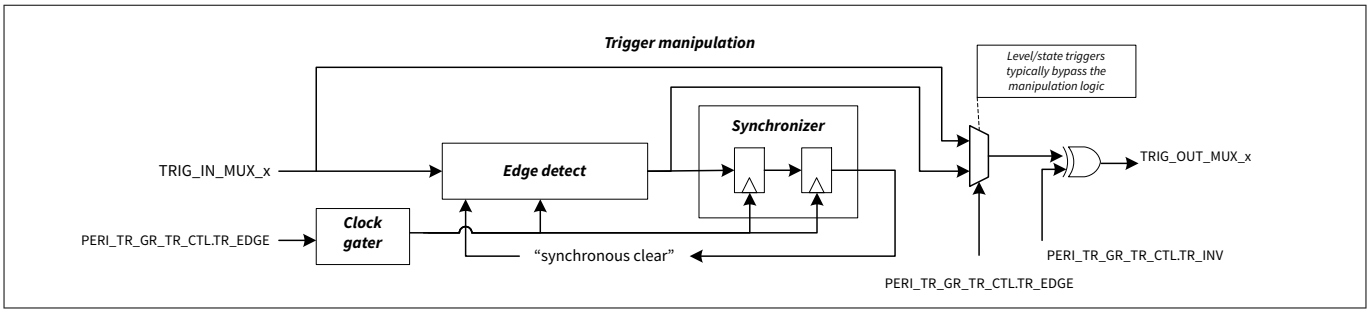


Figure 288 Trigger manipulation

In addition, there are I/Os (TRIG_IN[0:x]) that can be used to generate triggers. These inputs can be used to trigger TCPWM, SAR ADC, PERI, and CPU-CTI. For device-specific trigger mux assignments; refer to the device datasheet.

28.5 Registers

Table 234 List of Registers

Symbol	Name	Description
PERI_TR_CMD	Trigger Command Register	This register provides software control over trigger activation. This is useful for software-initiated triggers (such as P-/M-DMA transfers) or for debugging purposes. The control enables software activation of one specific input trigger or output trigger of the trigger multiplexer structure.
PERI_TR_GRx_TR_CTLy	Trigger Group Control Register	Controls group trigger actions and connects a peripheral input trigger to multiple peripheral output triggers
PERI_TR_1TO1_GRx_TR_CTLy	Trigger 1-to-1 Group Control Register	Controls the one-to-one trigger actions and connects a peripheral input trigger to a specific peripheral output trigger.

***Note**: x - indicates the trigger group number, y - indicates the trigger number*

29 Ethernet MAC

29.1 Overview

The Ethernet media access controller (MAC) module in the device implements a 10/100/1000 Mbps Ethernet MAC compatible with the IEEE 802.3 standard, supporting MII, RMII and RGMII PHY interfaces to support several industrial applications.[6)]

29.1.1 Supported features

- Both full store and forward mode and partial store and forward mode for full-duplex operation
- 10 Mbps, 100 Mbps, or 1 Gbps operation
- MII, RMII and RGMII PHY interface modes
- OPEN alliance specified RGMII V2.2
- 1536 bytes of maximum frame length
- Three transmit and receive priority queues
- IEEE Std 802.1Qav – Forwarding and queuing enhancements for time-sensitive streams
- IEEE Std 802.1AS – Timing and synchronization for time-sensitive application in bridged LANs
- IEEE Std 1588 – Precision time protocol
- IEEE Std 802.1Qbb – Priority based flow control
- 16 of each screening type 1 and type 2 Registers
- IEEE 802.3 pause frame and MAC PFC priority based pause frame support
- Receive and transmit IP, TCP, and UDP checksum offload
- Automatic pad and CRC generation on transmitted frames
- MDIO interface for PHY management
- Strict priority, DWRR, or enhanced transmission selection (ETS – 802.1Qaz) on transmit queues
- Support for 802.3az EEE
- AHB (32-bit address width, 32-bit data width) or AXI (32-bit address width, 64-bit data width) DMA master interface. See the device- specific datasheet for more information

6 Please check device specific datasheet to confirm which of these interfaces are supported in the device.

29.2 Block diagram

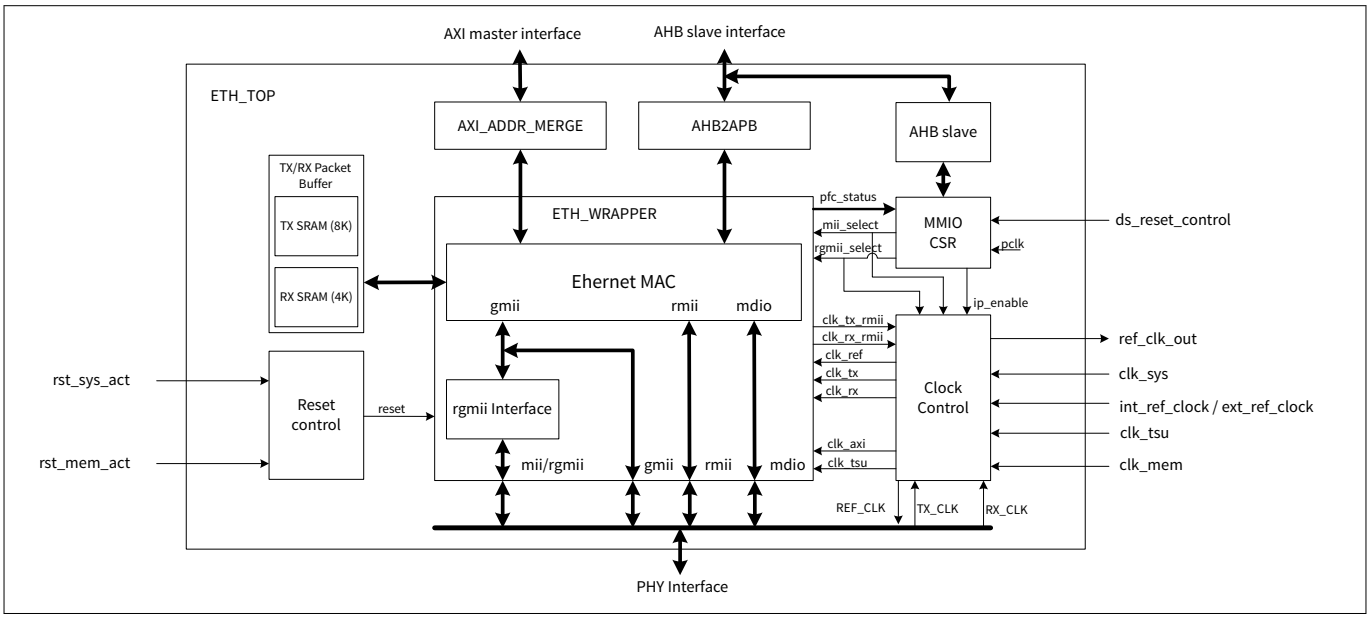


Figure 289 Ethernet MAC block diagram

29.3 Ethernet MAC operation

29.3.1 DMA interface

Ethernet MAC accesses data from other available system memory through DMA interface and stores fetched data in local dedicated TX/RX packet buffer. DMA is attached to the Ethernet MAC external FIFO interface to provide a scatter gather type capability for packet data storage. Configured for packet buffering mode, DMA uses dual port memory to store fetched data. This configuration makes application to use either of these mentioned operation modes to store and forward data.

- Full Store and forward mode
- Partial store and forward mode

In full store and forward mode, a packet will automatically be replayed directly from the packet buffer memory rather than having to re-fetch from system memory through AXI. If transmission fails, received erroneous packets are automatically dropped before they are sent to system memory, thus reducing AXI activity.

In partial store and forward mode, the transmitter will only forward the packet to the MAC when there is enough frame data stored in the packet buffer. Similarly, in case of receive operation, the receiver will only begin to forward the packet to the external AHB or AXI slave when enough frame data are stored in the local packet buffer.

The following features have also become available due to this approach:

- Transmit TCP/IP checksum offload
- Priority queuing
- RX packet flush when there is lack of resource
- Burst padding at end of packet and end of buffer to maximize AXI efficiency
- TX/RX timestamp capture to buffer descriptor entry

29.3.1.1 AXI interface

***Note**: This section is applicable for devices with AXI-base DMA access; refer to the device-specific datasheet to check if the device supports this feature.*

AXI master interface attached to the Ethernet MAC provides separate data channels and common address channels for read and write operations. With stated channels, interface supports two outstanding transactions on both the Read and Write channels.

EMAC requires to store configuration parameters for each transmit and receive frame through descriptors.

TX and RX descriptor reads are issued up-front and stored in a local buffer to feed the underlying DMA when required. This optimizes performance and avoids the need for the underlying DMA to pause while new descriptor fetches are sent to the system bus.

TX and RX descriptor writes issued by the underlying DMA are buffered locally to avoid holding up the underlying DMA when the system delays the completion of descriptor writes. Note that a descriptor write transaction is not considered complete until the write response (BRESP) associated with that transaction has arrived.

The maximum burst lengths the DMA can use are programmable. Single accesses and bursts with up to four beats can be selected. With 64-bit data path and a burst length setting of 4, 32 bytes transfers can be made with a single request. The burst length is controlled via the *ETHx_dma_config* Register.

29.3.1.2 Partial store and forward using packet buffer DMA

This feature is enabled via the TX and RX Partial Store and Forward Programmable Registers. When the transmit partial store and forward mode is activated, the transmitter will only begin to forward the packet to the MAC when there is enough packet data stored in the packet buffer. Similarly, when the receive partial store and forward mode is activated, the receiver will only begin to forward the packet to the external AHB or AXI slave when enough packet data is stored in the packet buffer.

The amount of packet data required to activate the forwarding process is programmable via watermark registers, which are located at the same address as the partial store and forward enable bits. Note that the minimum operational value for the TX partial store and forward watermark is 20. There is no operational limit for the RX partial store and forward watermark. To reduce the bandwidth requirements of the receive buffer manager the receive buffer size can be increased above its default value of 128 bytes by writing to the DMA Configuration Register.

Enabling partial store and forward is a useful means to reduce latency and increase Ethernet throughput.

***Note**: Partial store and forward is not supported for an AHB configuration with more than one priority queue. Partial store and forward mode is, however, supported for AXI configurations with priority queues.*

29.3.1.3 Full store and forward mode using packet buffer DMA

In full store and forward mode, MAC only starts transmission when the complete transmit frame is written into the local TX buffer. It will be flushed from the local buffer only after MAC completes the transmission and TX BD is updated with the status fields.

In receive process, DMA starts forwarding data to configured memory address only after the entire frame is received and does not contain any error. Received frame will be flushed from local packet buffer only after frame is copied and RX BD is updated with the status.

When the EMAC DMA is configured in the full store and forward mode, a receive over run condition occurs when the receive packet buffer memory is full, or because an AXI error occurred. In partial store and forward mode, a receive overrun condition occurs when either the AXI bus was not granted quickly enough, or because an AXI

error occurred, or because a new frame is detected by the receive block when the status update or write back for the previous frame is not yet finished. For a receive over run condition, the receive over run interrupt is asserted and the buffer being written is recovered. The next frame that is received whose address is recognized reuses the buffer.

The benefits of full store and forward mode are:

- Discard packets that are received with errors before they are partially written out of DMA thus saving AXI bandwidth and driver processing overhead
- Retry failed transmit frames from the packet buffer itself, thus saving AXI bus bandwidth
- Implement transmit IP/TCP/UDP checksum offload
- Allows multi-buffer frames

29.3.1.4 DMA transaction

EMAC DMA uses separate transmit and receive lists of buffer descriptors, with each descriptor describing a buffer area in system memory. This allows Ethernet packets to be broken up and scattered around the system memory.

The DMA controller performs six types of operation on the AMBA bus. In order of priority these are:

- Receive buffer manager write/read
- Transmit buffer manager write/read
- Receive data DMA write
- Transmit data DMA read

All read operations are routed to the AXI read channel and all write operations to the AXI write channel. Both read and write channels may operate simultaneously. Arbitration logic is used when multiple requests are active on the same channel. For example: when transmit DMA requests a transmit data read at the same time receive DMA requests a receive descriptor read; in these case, the receive DMA is granted the bus before the transmit DMA. However, majority of requests are either receive data writes or transmit data reads both of which can operate in parallel and can execute simultaneously.

Transfer size is set to 64-bit words by default in the *ETHx_network_config* Register and burst length can be programmed in the range from single access up to 16 accesses per burst using the *ETHx_dma_config* Register. It is recommended to set burst length maximum to 4 to have quicker arbitration for all masters accessing the bus.

29.3.1.5 Receive buffers

Received frames, optionally including FCS, are written to receive buffers located in system memory. The receive buffer depth (rx_buf_size[7:0]) is programmable in the range of 64 bytes to 16 KB in the DMA Configuration Register, with the default being 1536 bytes. If received frames are being routed to different priority queues via Screening Registers, it is possible to program different receive buffer depths for each queue. For queue 0, the receive buffer depth is programmed through the DMA Configuration Register (offset 0x10). For the other queues, they are programmed through specific queue configuration registers (starting from offset 0x4a0). Default is 128 bytes.

Start address for each receive buffer is stored in system memory in a list of receive buffer descriptors at an address location described by the receive buffer queue pointer. The base address of the receive buffer queue pointer (also referred as list of buffer descriptors) must be configured by software using the receive buffer queue base address registers (*ETHx_receive_q_ptr*, *ETHx_receive_q1_ptr*, *ETHx_receive_q2_ptr*).

Each buffer descriptor can be either of two or four words, depending on configured buffer descriptor (BD) mode (*ETHx_dma_config[28]*), whereas word is defined as 32 bits. The first two words (word 0 and word 1) are used in both BD modes.

In extended buffer descriptor mode (*ETHx_dma_config[28]* = 1), two BD words (word 2 and word 3) are added for timestamp capture if timestamp capture mode is enabled (*ETHx_rx_bd_control*[5:4] > 0_h). Therefore, BD modes will be either of two or four words size and each BD will have the same size. To summarize,

- Each BD must be of 64 bits when descriptor time capture mode is disabled
- Each BD must be of 128 bits when descriptor time capture mode is enabled

Following description details about word 0 and word 1 of each BD. Word 0 contains the start location of the receive buffer and word 1 contains the receive status. If the length of a received frame exceeds the DMA buffer length, the status word (word 1) in relevant BD is written with zeros except for the start of frame bit, which is always set for the first BD in a frame. Bit zero of the address field is set to '1' to show the buffer has been used. The receive buffer manager then reads the location of the next receive buffer and fills that with the next part of the received frame data. Receive buffers are filled until the frame is complete, and the final buffer descriptor status word contains the complete frame status. See Table 235 for details of the receive buffer descriptor list

When using receive descriptor timestamp capture (*ETHx_dma_config[28]* = 1), bit 2 of Word 0 is used to indicate a valid timestamp is captured in the BD. The use of bit 2 for this purpose also necessitates the data buffer being located on a 64-bit address boundary (EMAC only supports 32-bit address).

Each receive buffer start location is a word address. The start of the first buffer in a frame can be offset by up to three bytes depending on the value written to bits 15 and 14 of the Network Configuration Register (*ETHx_receive_buffer_offset[1:0]*) and bit 2 of word 0.

Table 235 Receive buffer byte offset configuration

Receive buffer offset configuration bit 2 of word 0	ETHx_receive_buffer_offset[1]	ETHx_receive_buffer_offset[0]	Number of bytes offset
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3

If the start location of the buffer is offset the available length of the first buffer is reduced by the corresponding number of bytes.

Table 236 Word 0 and word 1 description of each BD

Bit	Function
Word 0	
31:3	Address [31:3] of beginning of buffer
2	Address [2] of beginning of buffer or In extended buffer descriptor mode, indicates a valid timestamp in the BD entry.
1	Wrap - marks last descriptor in receive buffer descriptor list.
0	Ownership - needs to be '0' for the EMAC to write data to the receive buffer. EMAC sets this to '1' after it has successfully written a frame to memory. Software must clear this bit before the buffer can be used again.
Word 1	
31	Global all ones broadcast address detected
30	Multicast hash match

(table continues...)

Table 236 (continued) Word 0 and word 1 description of each BD

Bit	Function
29	Unicast hash match
28	External address match
27	Unused
26:25	Specific Address Register match. Encoded as follows: 00-Specific Address 1 Register match (lowest priority) 01-Specific Address 2 Register match 10-Specific Address 3 Register match 11-Specific Address 4 Register match (highest priority) If more than one specific address is matched only one of them is indicated with priority 4 down to 1.
24	This bit has a different meaning depending on whether RX checksum offloading is enabled (ETHx_network_config[24] = 1). With RX checksum offloading disabled: Type ID Register match found, bit 22 and bit 23 indicate which Type ID Register causes the match. With RX checksum offloading enabled: 0-The frame was not SNAP encoded and/or had a VLAN tag with the CFI bit set. 1-The frame was SNAP encoded and had either no VLAN tag or a VLAN tag with the CFI bit not set.
23:22	This bit has a different meaning depending on whether RX checksum offloading is enabled. With RX checksum offloading disabled: Type ID Register match. Encoded as follows: 00-Type ID Match 1 Register 01-Type ID Match 2 Register 10-Type ID Match 3 Register 11-Type ID Match 4 Register If more than one type ID is matched only one of them is indicated with priority 4 down to 1. With RX checksum offloading enabled: 00-Neither the IP header checksum nor the TCP/UDP checksum was checked. 01-The IP header checksum was checked and was correct. Neither the TCP nor UCP checksum was checked. 10-Both the IP header and TCP checksum were checked and were correct. 11-Both the IP header and UDP checksum were checked and were correct.
21	VLAN tag detected - Type ID of 8100_h. For packets incorporating the stacked VLAN processing feature, this bit will be set if the second VLAN tag has a Type ID of 8100_h.
20	Priority tag detected - Type ID of 8100_h and null VLAN identifier. For packets incorporating the stacked VLAN processing feature, this bit will be set if the second VLAN tag has a Type ID of 8100_h and a null VLAN identifier.

(table continues...)

Table 236 (continued) Word 0 and word 1 description of each BD

Bit	Function
19:17	VLAN priority - only valid if bit 21 is set. 000-Priority 0 (lowest) BK Background 001-Priority 1 BE best effort 010-Priority 2 EE excellent effort 011-Priority 3 CA critical applications 100-Priority 4 VI video, <100 ms latency and jitter 101-Priority 5 VO voice, <10 ms latency and jitter 110-Priority 6 IC internetwork Control 111-Priority 7 (highest) NC network control
16	Canonical format indicator (CFI) bit - only valid if bit 21 is set.
15	End of frame - when set the buffer contains the end of a frame. If end of frame is not set, then the only valid status bit is start of frame (bit 14).
14	Start of frame - when set the buffer contains the start of a frame. If both bits 15 and 14 are set, the buffer contains a whole frame.
13	This bit has a different meaning depending on whether jumbo frames and ignore FCS mode are enabled (network_configuration[3], network_configuration[26]). If neither mode is enabled this bit will be 0. With jumbo frame mode enabled: Additional bit for length of frame (bit 13), that is concatenated with bits [12:0] With ignore FCS mode enabled and jumbo frames disabled: This indicates per frame FCS status as follows: 0 -Frame had good FCS 1 -Frame had bad FCS, but was copied to memory as ignore FCS is enabled.
12:0	These bits represent the length of the received frame, which may or may not include FCS depending on whether FCS discard mode is enabled (network_configuration[17] = 1). With FCS discard mode disabled: Least significant 12 bits for length of frame including FCS. If jumbo frames are enabled, these 12 bits are concatenated with bit 13 of the descriptor. With FCS discard mode enabled: Least significant 12 bits for length of frame excluding FCS. If jumbo frames are enabled, these 12 bits are concatenated with bit 13 of the descriptor.

When descriptor timestamp capture mode is enabled, the following table identifies the added descriptor words.

Table 237 Word 3 and word 4 description for receive BD

Bit	Function
Word 2	
31:30	Timestamp seconds [1:0][1)]
29:0	Timestamp nanoseconds [29:0][1)]
Word 3	
31:10	Unused
9:0	Timestamp seconds [11:2][1)]

1) The timestamp mode is controlled using the RX BD Control Register (ETHx_rx_bd_control). The timestamp bits are written back to the last buffer descriptor of a frame only.

To receive frames, the receive buffer descriptors must be initialized by writing an appropriate address to bits [31:2] (or [31:3] for timestamp capture mode) in the word 0 of each BD. Bit 0 must be written as '0'. Bit 1 is the wrap bit and indicates the last entry in the buffer descriptor list.

The start location of the receive buffer descriptor list must be written with the receive buffer queue base address before reception is enabled (*ETHx_network_control[2]*=1). When reception is enabled, any writes to the receive buffer queue base address register are ignored.

***Note**: Writing receive buffer queue base address register may require three AXI clock cycles to take effect. Therefore, reception cannot be enabled until three AXI clock cycles after receive buffer queue base address register is updated. This restriction need to be taken care by firmware.*

The receive buffer queue pointer increments by two or four words after each buffer is used. It re-initializes to the receive buffer queue base address if any descriptor has its wrap bit set. When receive buffer queue base address register is read, it returns the current pointer position in the descriptor list, though this is only valid and stable when receive is disabled.

If the filter block indicates that a frame should be copied to memory, the receive data DMA operation starts writing data into the receive buffer. As receive buffers are used, the receive buffer manager sets bit 0 of the first word of the descriptor to '1' indicating the buffer has been used.

Software should search through the "Used" bits in the buffer descriptors to find out how many frames are received, checking the Start of Frame and End of Frame bits.

If full store and forward mode is configured, only good received frames are written to the receive buffers, so no fragments will exist in the local packet buffers due to MAC receiver errors. However, there is still possibility of fragments due to EMAC DMA errors; for example, used bit read on the second buffer of a multi-buffer frame.

If bit 0 in Word 0 of the receive BD is already set when the receive buffer manager reads the BD, then the buffer has already been used and cannot be used again until software has processed the frame and clear bit 0. In this case, the "buffer not available" bit in the receive Status Register is set and an interrupt triggered. The receive resource errors statistics register is also incremented.

When EMAC DMA is configured in the full store and forward mode, it can be selected to indicate whether received frames should be automatically discarded when no buffer resource is available; that is, "Used" bit is set for all receive BDs. This feature is selected via bit 24 of the *ETHx_dma_config* Register. By default, the received frames are not automatically discarded. If this feature is off, then received packets will remain stored in the RX packet buffer memory until system memory resource becomes available again. This may lead to packet buffer overflow if packets continue to be received and still bit 0 ("Used" bit) of the receive BD remains

set. Note that after a “Used” bit is read, the receive buffer manager will re-read the receive BD every time a new packet is received.

When the Ethernet MAC DMA is configured for packet buffer mode, the upper bits of the data buffer address stored in bits [31:2] in the first word of each list entry can be dynamically altered in real-time without physically changing the system memory holding the list entry. This feature is useful if the destination must be selected based on CPU usage or other flow control hardware. It is achieved using a mux structure whereby it can be defined whether the upper four bits of the 32-bit data-buffer AXI address should come from the descriptor list entry or from a Programmable Register. See the receive DMA Data Buffer Address Mask Register for further details. Note that any changes to this register will be ignored while the Ethernet MAC DMA is processing a receive packet. It will only affect the next full packet to be written to system memory.

29.3.1.6 Transmit buffers

Frames to be transmitted can be stored in one or more transmit buffers. Transmit frames can be between 1 and 1536 bytes long. Note that zero length buffers are allowed and the maximum number of buffers permitted for each transmit frame is 128.

The start addresses of each transmit buffer is stored in system memory in a list of transmit buffer descriptors located at the transmit buffer queue pointer. The base addresses of transmit BD list must be configured by software using the Transmit Buffer Queue Base Address Registers (*ETHx_transmit_q_ptr*, *ETHx_transmit_q1_ptr*, *ETHx_transmit_q2_ptr*).

Each buffer descriptor can be either of two or four words, depending on the configured BD mode, whereas word is defined as 32 bits. The first two words (Word 0 and Word 1) are used in both BD modes.

In extended buffer descriptor mode (*ETHx_dma_config[29]* = 1), two BD words (word 2 and word 3) are added for timestamp capture if timestamp capture mode is enabled (*ETHx_tx_bd_control[5:4]* > 0_h). Therefore, transmit BDs will be either of two or four words size and each BD will have the same size. To summarize,

- Each transmit BD must be of 64 bits when descriptor time capture mode is disabled
- Each transmit BD must be of 128 bits when descriptor time capture mode is enabled

The following description details Word 0 and Word 1 of TX BD. Word 0 of the each transmit BD is the start address of the transmit buffer and the word 1 consist of transmit control and status bits. For the packet buffer DMA, the start location for each transmit buffer is a byte address, the bottom bits of the address being used to offset the start of the data from the data-word boundary (that is, bits 2, 1, and 0 are used to offset the address for 64-bit data paths).

Frames can be transmitted with or without automatic CRC generation. If it is configured to generate CRC automatically, pad bytes will also be automatically generated to take frames to a minimum length of 64 bytes. If it is not configured to generate CRC automatically (as defined in word 1 of the transmit buffer descriptor), the frame is assumed to be at least 64 bytes long and pad bytes are not generated.

Table 238 Word 0 and word 1 description of transmit buffer descriptors

Bit	Function
Word 0	
31:0	Byte address of buffer
Word 1	
31	Used – must be 0 for the EMAC to read data to the transmit buffer. The EMAC sets this to ‘1’ for the first buffer of a frame after it is successfully transmitted. Software must clear this bit before the buffer can be used again.
30	Wrap – marks last descriptor in transmit buffer descriptor list. This can be set for any buffer within the frame.

(table continues...)

Table 238 (continued) Word 0 and word 1 description of transmit buffer descriptors

Bit	Function
29	Retry limit exceeded, transmit error detected
28	Unused
27	Transmit frame corruption due to AXI error – set if an error occurs whilst midway through reading through reading transmit frame from the AXI, including RRESP/BRESPor HRESP errors and buffers exhausted mid frame (if the buffers run out during transmission of a frame then transmission stops, FCS shall be bad and TX_ER asserted). Also set if single frame is too large for the transmit packet buffer memory size.
26	Transmit error detected
25:24	Reserved
23	For extended buffer descriptor mode, this bit indicates a timestamp is captured in the BD. Otherwise it is unused.
22:20	Transmit IP/TCP/UDP checksum generation offload errors: 000 – No error 001 – The packet was identified as a VLAN type, but the header was not fully complete, or had an error in it. 010 – The packet was identified as a SNAP type, but the header was not fully complete, or had an error in it. 011 – The packet was not of an IP type, or the IP packet was invalidly short, or the IP was not of type IPv4/IPv6 100 – The packet was not identified as VLAN, SNAP, or IP. 101 – Unsupported packet fragmentation occurred. For IPv4 packets, the IP checksum was generated and inserted. 110 – Packet type detected was not TCP or UDP, TCP/UDP checksum was therefore not generated. For IPv4 packets, the IP checksum was generated and inserted. 111 – A premature end of packet was detected and the TCP/UDP checksum could not be generated.
19:17	Reserved. must be set to 3'b000
16	No CRC to be appended by MAC. When set this implies that the data in the buffers already contains a valid CRC and hence no CRC or padding is to be appended to the current frame by the MAC. This control bit must be set for the first buffer in a frame and will be ignored for the subsequent buffers of a frame. Note that this bit must be '0' when using the transmit IP/TCP/UDP checksum generation offload, otherwise checksum generation and substitution will not occur. Note this bit must also be '0' when TX partial store and forward mode is active.
15	Last buffer; when '1', this bit will indicate the last buffer in the current frame is reached.
14	Reserved
13:0	Length of buffer

When descriptor timestamp capture mode is enabled, the following table identifies the added descriptor words.

Table 239 Word 3 and word 4 of TX BD when timestamp capture mode enabled

Bit	Function
Word 2	
31:30	Timestamp seconds [1:0][1)]
29:0	Timestamp nanoseconds [29:0][1)]
Word 3	
31:10	Unused
9:0	Timestamp seconds [11:2][1)]

1) The timestamp mode is controlled using the TX BD Register (*ETHx_tx_bd_control*). After transmission the timestamp bits are written back only to the first buffer descriptor.

To transmit frames, the buffer descriptors must be initialized by writing the start address of the buffers to bits [31:0] in the first word (Word 0) of each descriptor.

Word 1 of the transmit buffer descriptor must be initialized with control information that indicates the length of the frame, whether the MAC is required to append CRC and whether the buffer is the last buffer in the frame.

After transmission, the status bits of Word 1 of the first BD for a frame are updated by EMAC along with the “Used” bit. “Used” bit is written to ‘1’, after the frame is transmitted. Bits [29:20] indicate various transmit error conditions. Bit 23 indicates a valid timestamp is captured in the BD. Bit 30 is the “Wrap” bit, which can be set for any buffer within a frame. If no wrap bit is encountered the queue pointer continues to increment to fetch next BD.

The Transmit Buffer Queue Base Address Register can only be updated whilst transmission is disabled or halted; otherwise any attempted write will be ignored.

- When transmission is halted the transmit buffer queue pointer will maintain its value. Therefore, when transmission is restarted the next descriptor read from the queue will be from the frame immediately after the last successfully transmitted frame.
- When transmit is disabled (*ETHx_network_control[3]* = 0), the transmit buffer queue pointer resets to point to the address indicated by the Transmit Buffer Queue Base Address Register.

Note that disabling receive does not have the same effect on the receive buffer queue pointer.

After the transmit queue is initialized, transmit is activated by writing to the transmit start bit (*ETHx_network_control[9]*). Transmit is halted when a buffer descriptor with its “Used” bit set is read or a transmit error occurs, or by writing to the transmit halt bit of the Network Control Register (*ETHx_network_control[10]*). Transmission is suspended if a pause frame is received while the pause enable bit is set in the Network Control Register (*ETHx_network_control[13]*). Rewriting the start bit while transmission is active is allowed. *transmit_go* bit (*ETHx_transmit_status[3]*) reset under following conditions:

- Transmit is disabled.
- A buffer descriptor with its ownership bit set is read.
- Bit 10, *tx_halt_clk*, of the Network Control Register is written.
- There is a transmit error such as too many retries or a transmit under run.

When *transmit_go* is cleared, DMA will stop to fetch new packet from system memory and EMAC will not complete transmission until packet buffer is empty.

To set transmit_go write to bit 9, tx_start_clk, of the ETHx_network_control. Transmit halt does not take effect until any ongoing transmit finishes.

If the transmit BD list is incorrectly set up, for example a “used” bit set is read mid-way through a multi buffer frame, transmission will stop. If cut-through is in operation and the MAC has actually started transmitting the

frame that has its used bit set, the MAC treats it as a transmit error, and asserts tx_er truncates the frame and corrupts the FCS.

29.3.1.7 DMA burst

When performing data transfers, the burst length used can be programmed using bits [4:0] of the DMA Configuration Register. Either single accesses (burst length = 1) or incrementing bursts of up to 4 can be used.

When there is sufficient space and enough data to be transferred, the burst of programmed length will be used. If there is not enough data or space available, for example when at the end of a packet or buffer, burst lengths of less than the programmed burst length value will be issued. Single accesses will be used when a 4 Kb boundary for AXI or a 1 Kb boundary for AHB is crossed by the burst to not violate the AXI/AHB specification.

The EMAC DMA can also be configured to pad the remaining bursts at the end of a buffer to the programmed burst length value available via bits 26 and 25 of the ETHx_dma_config Register. Bit 26 will control the transactions for TX and bit 25 for RX. For RX, the data to burst is padded with “0”s up to the burst boundary defined by burst length. For TX, the extra data that is read is ignored by the DMA. This feature is included for performance reasons when AXI slaves that are being accessed by EMAC perform better when accessed using fixed length bursts.

The EMAC DMA will not terminate fixed length bursts early if receive/transmit operation is disabled by writing to *ETHx_network_control* register bit 2/3.

29.3.1.8 DMA packet buffer

The packet buffer DMA mode allows multiple packets to be buffered in both transmit and receive directions and allows the DMA to withstand variable levels of access latencies on the AXI fabric. Using packet buffers, AXI bandwidth has been used most efficiently in the device.

Figure 290 illustrates the structure of the EMAC data paths.

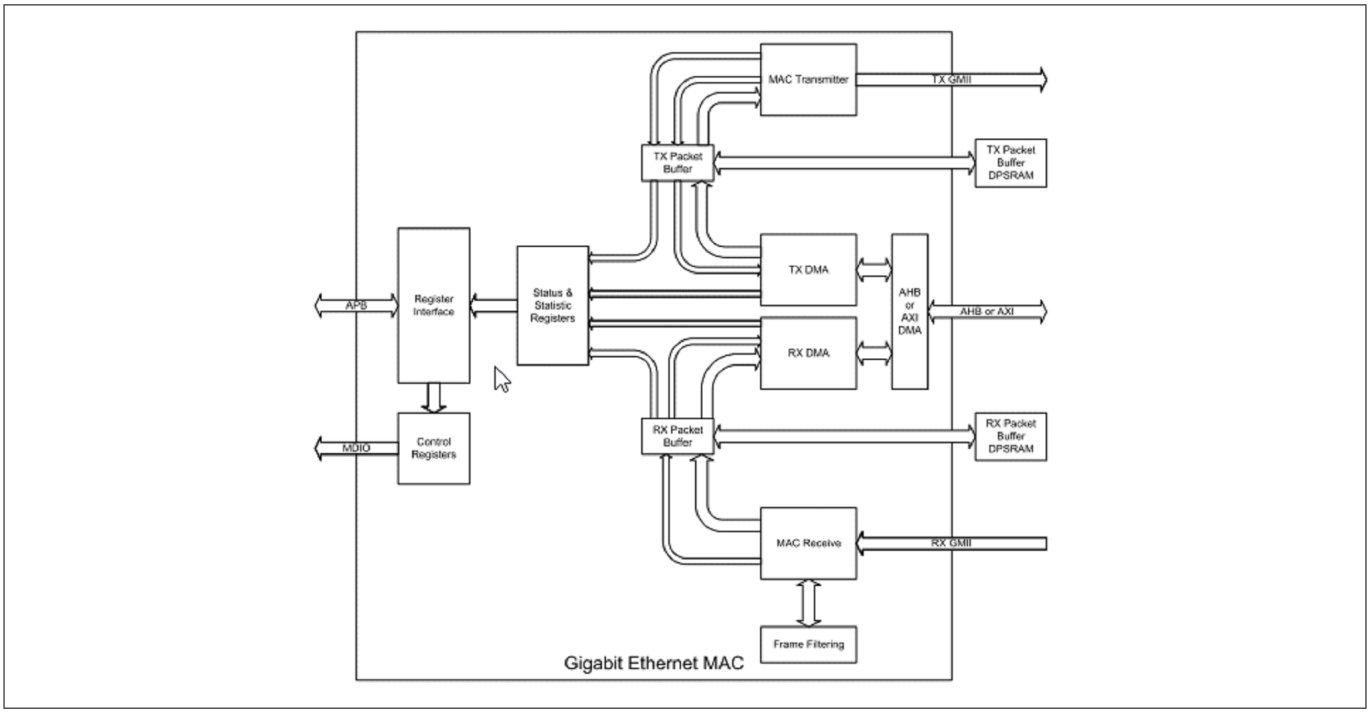


Figure 290 EMAC data path structure

In the transmit direction, the DMA will continue to fetch packet data up to a limit of 256 packets, or until the TX packet buffer memory is full. In the receive direction, if the RX packet buffer memory becomes full, then an overflow will occur. An overflow will also occur if the limit of 256 packets is reached.

Transmit packet buffer

The transmit packet buffer (TX packet buffer) will continue attempting to fetch frame data from the system memory until the TX packet buffer memory is full or up to a limit of 256 packets, at which point it will attempt to maintain its full level.

To accommodate the status and statistics associated with each frame, two words per packet are reserved at the end of the packet data. If the packet was bad and requires to be dropped, the status and statistics are the only information held on that packet. Storing the status in the packet buffer is required to decouple the TX DMA interface of the buffer from the MAC transmitter interface, to update the MAC status/statistics and to generate interrupts in the order in which the packets that they represent were fetched from the system memory.

If any errors occur on the AXI whilst reading the transmit frame, then fetching of packet data from memory is halted. The MAC transmitter will continue to fetch packet data, thereby emptying the TX packet buffer memory and allowing any good non-erroneous frames to be transmitted successfully. When these are fully transmitted, the status/statistics for the erroneous frame will be updated and software will be informed via an interrupt that an AXI error occurred. This way, the error is reported in the correct packet order.

The TX packet buffer will only attempt to read more frame data from the system memory when space is available in the TX packet buffer memory. If space is not available, it must wait until a packet fetched by the MAC transmitter completes transmission and is subsequently removed from the TX packet buffer memory. Note that if full store and forward mode is active and if a single frame is fetched that is too large for the TX packet buffer memory, the frame is flushed, and the TX DMA halted with an error status. This is because a complete frame must be written into the TX packet buffer memory before transmission can begin (If frame is split into multiple buffers, DMA will measure the buffer length against the available TX packet buffer room to decide if it will keep fetching).

In full store and forward mode, after the complete transmit frame is written into the TX packet buffer memory, a trigger is sent across to the MAC transmitter, which will then begin reading the frame from the TX packet buffer memory. Because the whole frame is present and stable in the TX packet buffer memory, an underflow of the MAC transmitter is not possible. The frame is kept in the TX packet buffer memory until notification is received from the MAC transmitter that the frame data has either been successfully transmitted or can no longer be re-transmitted. When this notification is received the frame is flushed from TX packet buffer memory to make room for a new frame to be fetched from system memory. The frame is removed from the packet buffer on the fly after the frame is successfully transmitted.

In partial store and forward mode, a trigger is sent across to the MAC transmitter as soon as sufficient packet data is available in the internal buffer, which will then begin fetching the frame from the packet buffer memory. Later, if MAC transmitter is fetching data from the packet buffer faster than the DMA can fill it, an underflow of the transmitter will occur. In this case, the transmission is terminated early, and the packet buffer is flushed. Transmission can only be restarted by writing to the transmit start bit.

Receive packet buffer

The receive packet buffer (RX packet buffer) stores frames from the MAC receiver along with their status and statistics. Frames with errors are flushed from the RX packet buffer memory, while good frames are pushed onto the AXI master interface.

When programmed in full store and forward mode, if the frame has an error the frame data is immediately flushed from the RX packet buffer memory allowing subsequent frames to use the freed-up space. The status and statistics for bad frames are still used to update the EMAC Registers.

To accommodate the status and statistics associated with each frame, up to two words (one being for descriptor timestamp capture when enabled) per packet are reserved at the end of the packet data. If the packet was bad and requires to be dropped, the status and statistics are the only information held on that packet.

The RX packet buffer will also indicate a full condition such that an overflow condition can be detected. If this occurs, subsequent packets will be dropped, and an RX overflow interrupt is raised.

For full store and forward, the RX DMA will only begin packet fetches when the status and statistics for a frame are available. If the frame has a bad status due to a frame error, the status and statistics are passed onto the EMAC Registers. If the frame has a good status, the information is used to read the frame from the RX packet buffer memory and burst onto the AXI master interface using the DMA buffer management protocol.

If partial store and forward mode is active, the DMA will begin fetching the packet data before the status is available. As soon as the status becomes available, the DMA will fetch this information before continuing to fetch the remainder of the frame. When the last frame data has been transferred to the buffer in system memory, the status and statistics are updated to the EMAC Registers.

29.3.1.9 Priority queuing in EMAC DMA

Ethernet MAC supports three transmit and receive priority queues. Each queue has an independent list of buffer descriptors pointing to separate data streams. By default, each queue is active. Queues can be disabled by setting bit 0 of the transmit or receive buffer queue base address register. Note that at least one queue must always remain enabled and only the top indexed queues may ever be disabled. For example, if only two queues are being used, the user would disable queue 2 by setting bit 0 of the transmit or receive buffer queue base address register.

In the transmit direction, higher priority queues are always serviced before lower priority queues. This strict priority scheme requires the user to ensure that high-priority traffic is constrained such that lower priority traffic will have the required bandwidth. The DMA will determine the next queue to service by initiating a sequence of buffer descriptor reads interrogating the ownership bits of each.

The buffer descriptor corresponding to the highest priority queue is read first. If the ownership bit of this descriptor is set, then the DMA will progress to reading the second highest priority queue's descriptor. If ownership bit of the second highest priority queue is also set, then the DMA will read the third highest priority queue's descriptor. If all the descriptors return an ownership bit set, then a resource error occurs, an interrupt is generated, and transmission is automatically halted.

Transmission can only be restarted by setting the START bit in the ETHx_network_control Register. The DMA will identify the highest available queue to transmit from when the START bit in the ETHx_network_control Register is written to and the TX is in a halted state, or when the last word of any packet has been fetched from external memory. The transmit DMA will maximize the effectiveness of priority queuing by ensuring that high priority traffic be transmitted as early as possible after being fetched from external memory.

For each queue, there is an associated transmit buffer queue base address register. For the lowest priority queue (or the only queue when a single queue is selected), the transmit buffer queue base address is located at offset address of 0x1C. For all other queues, the transmit buffer queue base address registers are located at sequential addresses starting at offset 0x440.

In the receive direction each data packet is written to the internal packet buffer in the order that it is received. For each queue, there is an independent set of receive buffers. Therefore, a separate receive buffer queue base address register for each queue is allocated. For the lowest priority queue, the receive buffer queue base address register is located at address offset of 0x18. For all other queues, these registers are located at sequential addresses starting at the address offset of 0x480. Every received packet will pass through a programmable screening algorithm, which will allocate that frame to a specific queue. The user interface to the screener is via two banks of Programmable Registers – ETHx_screener_type_1 and ETHx_screener_type_2.

Screener Type 1 Registers allow routing the received frames based on IP and UDP fields extracted from the received frames. Specifically, these fields are DS (differentiated services field of IPv4 frames), TC (traffic class field of IPv6 frames), and/or the UDP destination port. These fields are compared against the values stored in each of the Screener Type 1 Match Registers. If the result of this comparison is positive, then the received frame is routed to the priority queue specified in that Screener Type 1 Register. XMC7000 supports 16 Screener Type 1 Registers.

Screener Type 2 Match Registers operate independent of Screener Type 1 Registers and offer additional match capabilities, extending the capabilities into vendor specific protocols. The type 2 screening allows configuring a screen, which is a combination of all or any of the following comparisons:

- An enabled VLAN priority. A VLAN priority match will be performed if the VLAN priority enable is set. The extracted priority field in the VLAN header is compared against three bits within the Screener Type 2 Register.

- An enabled ethertype. The ethertype field inside the Screener Type 2 Register maps to one of eight ethertype match registers. The extracted ethertype is compared against the ethertype register designated by this ethertype field.
- An enabled field compare A
- An enabled field compare B
- An enabled field compare C

Compare A, B, and C fields of the Screener Type 2 Match Register are pointers to a pool of up to 32 Compare Registers. When enabled, the compare is true if the data at the OFFSET into the frame ANDed with the MASK value (if the mask is enabled) is equal to the COMPARE value. Either a 16-bit comparison or a 32-bit comparison is done. This selection is made via a control bit in the associated compare word1. If a 16-bit comparison is selected, then a 16-bit mask is also available to the user to select which bits should be compared. If the 32-bit compare option is selected, then no mask is available. The byte at the OFFSET number of bytes from the index start is compared through bits 7:0 of the configured VALUE. The byte at the OFFSET number of bytes + 1 from the index start is compared through bits 15:8 of the configured VALUE, and so on. The OFFSET can be configured to be from 0 to 127 bytes from either the start of the frame, the byte following the ethertype field, the byte following the end of the IP header (IPv4 or IPv6), or the byte following the end of the TCP/UDP header. Note that the logic to decode the IP header or the TCP/UDP header is reused from the TCP/UDP/IP checksum offload logic and has the same restrictions on use (the main limitation is that IP fragmentation is not supported). See Checksum offload for IP, TCP, and UDP for further details. The compare register field points to a single pool of 32 compare registers. Compare A, B, and C use a common set of Compare Registers.

Note compare A, B, and C together allow matching an arbitrary 48 bits of data; therefore, they can be used to match a MAC address.

All enabled comparisons are ANDed together to form the overall type 2 screener match. XMC7000 supports 16 Screener Type 2 Registers.

Each Screener Register is programmable. Although it is not recommended, it is possible that more than one Screener Register can be programmed to match against a single frame. If this happens, consider the following:

- If a received frame matches multiple screeners of the same type, then the frame will route to the queue mapped by the screener located at the lowest numeric APB address. For example, if screener type 2 #0 and screener type 2 #1 both match, then the frame will route to the queue identified in bits [3:0] of the screener type 2 #0 Register.
- If a received frame matches a type 2 screener and a type 1 screener, then the type 1 screener will take precedence.

When a screener is matched, the received frame will be routed to a queue defined inside bits 3:0 of the Screener Register. Unmatched frames are routed to queue 0.

The interrupt outputs from the Ethernet MAC match the number of supported priority queues. Only Ethernet MAC DMA-related events are reported using the individual interrupt outputs, because the Ethernet MAC can relate these events to specific queues. All other events generated within the Ethernet MAC are reported in the interrupt associated with the lowest priority queue (queue 0). For the lowest priority queue, the Interrupt Status Register is located at offset address 0x024. For all other priority queues, this register is located at sequential offset addresses starting at 0x400.

29.3.2 Transmit scheduling algorithm

When multiple priority queues are selected, the transmit scheduler is automatically included in the design and is responsible for selecting the next queue to be serviced from the attached DMA. There are four scheduling algorithms available to the user; with some exceptions detailed further below, the user can select one of the four modes for each queue.

29.3.2.1 802.1Qav support - credit based shaping

A credit-based shaping algorithm is available on the two highest priority active queues and is defined in 802.1Qav: forwarding and queuing enhancements for time-sensitive streams. Traffic shaping is enabled via the ETHx_CBS_control Register (0x4bc) or the ETHx_tx_sched_ctrl Register (offset 0x580). These two registers are aliased, so updating one register will automatically update the other. Note that it is permitted to enable CBS only on the second-highest-priority queue and not on the highest, in which case the highest-priority queue would always take precedence.

Enabling CBS on a queue will enable a counter, which stores the amount of transmit 'credit', measured in bytes, that a queue has. A queue may only transmit if it has non-negative credit.

If a queue has data to send but is held off from doing as another queue is transmitting, then credit will accumulate in the credit counter at the rate defined in the IdleSlope Register for that queue. IdleSlope is the rate of change of credit when waiting to transmit and must be less than the value of the portTransmitRate. When this queue is transmitting, the credit counter is decremented at the rate of sendSlope, which is defined as the portTransmitRate - IdleSlope. A queue can accumulate negative credit when transmitting, which will hold off any other transfers from that queue until credit returns to a non-negative value. No transfers are halted when a queue's credit becomes negative; it will accumulate negative credit until the transfer completes.

To ensure that the CBS scheduling is completely accurate, a single transmit buffer should be used per Ethernet frame (rather than multi-buffer transmit frames).

29.3.2.2 Fixed priority

Any of the active queues can be selected as fixed priority; this is the default mode of operation for all queues. The queue index is used as the priority, where a higher index will have a higher priority than a lower index. The scheduler will always attempt to transmit from fixed-priority queues with the highest priority. This means that a fixed-priority queue with a high queue index will always take precedence over a priority queue with a lower index.

29.3.2.3 Deficit weighted round robin (DWRR)

Any of the active queues can be selected as DWRR. If DWRR is required, then at least two of the active queues should be selected as DWRR. It should not be used in conjunction with ETS, as both algorithms operating together would make little practical sense. A DWRR-enabled queue has lower priority than a CBS-enabled queue or a fixed-priority queue with a higher index.

The DWRR algorithm works by scanning all non-empty queues in sequence. Each queue is allocated a 'deficit counter' and an 8-bit weighting (or quantum) value. The value of the deficit counter is the maximum number of bytes that can be sent at the current time. If the deficit counter of the scanned queue is greater than the length of the packet waiting for transmission, then the packet will be transmitted and the value of the deficit counter is decremented by the packet size. If it is not greater, the scheduler will skip to the next DWRR-enabled queue. If there is insufficient credit to transmit, the queue is simply skipped. If the queue is empty, the value of the deficit counter is reset to '0'. If all queues have insufficient credit, at each tx_clk cycle, every queue's deficit counter is incremented by its quantum value until a queue's deficit counter obtains sufficient credit to transmit its first queued frame. The higher the quantum value chosen, the quicker the deficit counter will reach the required value. If all DWRR queues have the same weighting, then all queues will be granted the same overall bandwidth. The weighting value is stored in four Programmable Registers starting at offset 0x590. See the register descriptions for further details.

Note that if fixed-priority queues are to be used in conjunction with DWRR, the fixed-priority queues must be at a higher index value than the DWRR queues. A consequence of this is that the enabled DWRR queues will form a contiguous set of queues starting from queue 0.

If CBS is also used in conjunction with DWRR, the DWRR queues will share the remaining bandwidth after the CBS allocation is deducted.

***Note**: Transmit cut-thru (partial store and forward mode) should not be enabled if the transmit scheduler is used.*

29.3.2.4 Enhanced transmission selection (ETS)

The ETS algorithm is defined in 802.1Qaz: enhanced transmission selection for bandwidth sharing between traffic, and allows traffic on specific queues to be bandwidth limited. Any of the active queues can be selected as ETS. If ETS is required, then at least two of the active queues should be selected as ETS. It should not be used in conjunction with DWRR as both algorithms operating together would make little practical sense. An ETS-enabled queue has a lower priority than a CBS-enabled queue or a fixed-priority queue with a higher index.

For each ETS-enabled queue, the user should program the bandwidth requirement for each queue as a percentage of the total bandwidth (an 8-bit register is used and the sum of values programmed should not exceed decimal 100). This will be the maximum bandwidth to be granted to that queue. The actual scheduling algorithm operates in a round-robin style from lowest indexed queues up to the highest indexed queue in sequence. The bandwidth allocation percentage is stored in four programmable registers starting at offset 0x590 – these are the same registers used for DWRR. See the register descriptions for further details.

If CBS is also used in conjunction with ETS, the sum of the ETS queue percentages should equal the remaining bandwidth after the CBS allocation is deducted.

***Note**: Transmit cut-thru (partial store and forward mode) should not be enabled if the transmit scheduler is used.*

29.3.3 MAC transmitter

The MAC transmitter operates in full duplex and transmits frames in accordance with the Ethernet IEEE 802.3 standard.

A small input buffer receives data through the DMA, which, depending on the data_bus_width control bits in the *ETHx_network_config* Register, will extract data in 128-bit form. All subsequent processing before the final output is performed in bytes.

Transmit data can be output using one of four PHY interface – MII, RMII or RGMII.

Frame assembly starts by adding preamble and the start frame delimiter (SFD). Data is taken from the TX packet buffer a word at a time. If necessary, padding is added to take the frame length to 60 bytes. CRC is calculated using an order 32-bit polynomial. This is inverted and appended to the end of the frame taking the frame length to a minimum of 64 bytes. If the “No CRC” bit (bit 16) is set in the second word (Word 1) of the last buffer descriptor of a transmit frame neither pad nor CRC are appended.

In full duplex mode, frames are transmitted immediately. Back-to-back frames are transmitted 96-bit times apart to guarantee the inter packet gap.

In all modes of operation, if the TX DMA under runs, a bad CRC is automatically appended using the same mechanism as jam insertion and the TX_ER signal is asserted. For a properly configured system this should never happen, as the complete frame is buffered in TX Packet Buffer Memory.

29.3.4 MAC receiver

The MAC Receiver block checks for valid preamble, FCS, alignment, and length, presents received frames to the DMA and, stores the frame destination address for use by the address checking block.

Ethernet frames are normally stored in the receive buffer in the AXI memory complete with the FCS. Setting the fcs_remove bit in the Network Configuration Register (*ETHx_network_config[17]*) causes frames to be stored without their corresponding FCS. The reported frame length field is reduced by four bytes to reflect this operation.

The MAC receive block signals to the register block to increment the alignment, CRC (FCS), short frame, long frame, jabber, or receive symbol errors when any of these exception conditions occur.

If bit 26 is set in the *ETHx_network_config* Register, CRC errors will be ignored, and CRC erroneous frames will not be discarded, though the *ETHx_fcs_errors* Statistics Register will still be incremented. Additionally, if configured to use the DMA and not enabled for jumbo frames mode, then bit 13 of the receive buffer descriptor word 1 will be updated to indicate the FCS validity for the particular frame. This is useful for applications such as EtherCAT whereby individual frames with FCS errors must be identified.

Received frames can be checked for length field error by setting the length_field_error_frame_discard bit of the *ETHx_network_config* Register (bit 16). When this bit is “1”, the receiver compares a frame’s measured length with the length field (bytes 13 and 14) extracted from the frame. The frame is discarded if the measured length is shorter. This checking procedure is for received frames between 64 bytes and 1518 bytes in length.

Each discarded frame is counted in the 10-bit length field *ETHx_fcs_errors* Statistics Register. Frames where the length field value is greater than or equal to 0600_h (1536) will not be checked.

29.3.5 Checksum offload for IP, TCP, and UDP

Ethernet IP can be programmed to perform IP, TCP, and UDP checksum offloading in both receive and transmit directions, which is enabled by setting bit 24 in the ETHx_network_config Register for receive and bit 11 in the *ETHx_dma_config* Register for transmit.

IPv4 packets contain a 16-bit checksum field, which is the 16-bit 1's complement of the 1's complement sum of all 16-bit words in the header. TCP and UDP packets contain a 16-bit checksum field, which is the 16-bit 1's complement of the 1's complement sum of all 16-bit words in the header, the data, and a conceptual IP pseudo header.

To calculate these checksums in software requires processing each byte of the packet. For TCP and UDP this can use a large amount of processing power. Offloading the checksum calculation to hardware can result in significant performance improvements.

For IP, TCP, and UDP checksum offload to be useful, the operating system containing the protocol stack must be aware that this offload is available so that it can make use of the fact that the hardware can either generate or verify the checksum.

29.3.5.1 Receive checksum offload

When receive checksum offloading is enabled in the Ethernet IP, the IPv4 header checksum is checked as per RFC791, where the packet meets the following criteria:

- If present, the VLAN header must be four octets long and the CFI bit must not be ‘1’. (for receive, one stacked VLAN is supported)
- Encapsulation must be RFC 894 Ethernet type encoding or RFC 1042 SNAP Encoding or PPPoE encoding
- IPv4 packet
- IP header is of valid length
- IP options are supported

The Ethernet IP also checks the TCP checksum as per RFC 793, or UDP checksum as per RFC 768, if the following criteria are met:

- IPv4 or IPv6 packet
- IP options and all IPv6 extension headers (hop-by-hop, routing, and destination) are supported (except fragmentation headers)
- Good IP header checksum (if IPv4)
- IP fragmentation is not supported. (If a packet is fragmented, then the checksum will not be checked)
- TCP or UDP packet

When an IP, TCP, or UDP frame is received, the receive buffer descriptor gives an indication if the Ethernet IP was able to verify the checksums. There is also an indication if the frame had SNAP encapsulation. These indication bits will replace the type ID match indication bits when the receive checksum offload is enabled. For details of these indication bits, refer to the Table 235.

If any of the checksums are incorrectly verified by the Ethernet IP, the packet is discarded, and the appropriate statistics counter is incremented.

29.3.5.2 Transmit checksum offload

The transmitter checksum offload is only available if Ethernet IP is configured to use the DMA in packet buffer mode, and full store and forward mode is enabled. This is because the complete frame to be transmitted must be read into the TX packet buffer memory before the checksum can be calculated and written back into the headers at the beginning of the frame.

Transmitter checksum offload is enabled by setting bit 11 in the ETHx_dma_config Register. When enabled, it will monitor the frame as it is written into the TX packet buffer memory to automatically detect the protocol of the frame. Protocol support is identical to the receiver checksum offload.

For transmit checksum generation and substitution to occur, the protocol of the frame must be recognized, and the frame must be provided without the FCS field, by making sure that bit 16 of the transmit descriptor Word 1 is clear (VLAN tagged frames will be recognized but stacked VLAN tagged frames will not be recognized). If the frame data already had the FCS field, this would be corrupted by the substitution of the new checksum fields.

If these conditions are met, the transmit checksum offload engine will calculate the IP, TCP, and UDP checksums as appropriate. When the full packet is completely written into TX packet buffer memory, the checksums will be valid and the relevant memory locations will be updated for the new checksum fields as per standard IP/TCP and UDP packet structures.

If the transmitter checksum engine is prevented from generating the relevant checksums, bits [22:20] of the transmit buffer descriptor entry will be updated to identify the reason for the error. Note that the frame will still be transmitted but without the checksum substitution; this is because, the reason the substitution did not occur is typically because the protocol was not recognized.

29.3.6 Jumbo frame support

The jumbo frames enable bit in the ETHx_network_config Register (bit 3) allows EMAC to receive jumbo frames up to a software configurable number of bytes in size. This operation is not part of IEEE Std 802.3 specification and is by default disabled. When jumbo frames are enabled, frames received with a frame size greater than the configured value are discarded.

Oversize frame received Register (*ETHx_excessive_rx_length*) will count the number of very long frames received.

The jumbo frames maximum length can be controlled using the jumbo-frame maximum length Register (*ETHx_jumbo_max_length*). In EMAC, the maximum length of jumbo frame is 1536 bytes.

The jumbo frames maximum length can be controlled using the jumbo-frame maximum length register (*ETHx_jumbo_max_length*). Its default value is 1536 bytes.

- If jumbo frame is enabled (*ETHx_network_config[3]*):
  - The *ETHx_jumbo_max_length* register has default value 1536; the user does not need to set it for a 1536 bytes transfer.
  - The user can modify the *ETHx_jumbo_max_length* register, and the maximum length of the frame received is determined by this register.
  - The value of ETHx_receive_1536_byte_frames does not matter.
- If jumbo_frame is disabled:
  - If ETHx_network_config.receive_1536_byte_frames is set, maximum length will be 1518.
  - If ETHx_network_config.receive_1536_byte_frames is not set, maximum length will be 1500.
  - The value of ETHx_jumbo_max_length does not matter.

In EMAC, the maximum length of jumbo frame is 1536 bytes. The TX and RX buffer size does not allow longer frames to be transmitted or received in full store and forward mode.

29.3.7 MAC filtering block

The filter block determines which frames should be written to the EMAC DMA.

Whether a frame is passed depends on what is enabled in the *ETHx_network_config* register, contents of the specific address (*ETHx_spec_addi_top, ETHx_spec_addi_bottom*), type ID match (*ETHx_spec_typei*), and hash (*ETHx_hash_top, ETHx_hash_bottom*) registers and the frame's destination address and type fields.

Ethernet frames are transmitted one byte at a time, least significant bit first. The first six bytes of an Ethernet frame make up the destination address. The first bit of the destination address, which is the LSB of the first byte of the frame, is the group or individual bit. This is '1' for multicast addresses and '0' for unicast ones. The "all ones" address is the broadcast address and a special case of multicast.

The EMAC supports recognition of specific source or destination addresses. The number of specific source or destination address filters is four. Each specific address filter consists of two registers, specific address bottom i register and specific address top i register. Specific address bottom i register stores the first four bytes of the compare source or destination address. Specific address top i register contains the last two bytes of this address, a control bit to select between source or destination address filtering, and a 6-bit byte mask field to allow masking certain bytes during the comparison. The first filter (filter 1) is slightly different to all other filters in that there is no byte mask. Instead address comparison against individual bits of specific address 1 register can be masked using the unique specific address mask 1 register. The addresses stored in all filters can be specific (unicast), group (multicast), local, or universal.

The destination or source address of received frames is compared against the data stored in the specific address registers after they are activated. The addresses are deactivated at reset or when their corresponding specific address bottom i register is written. They are activated when the corresponding specific address top i register is written. If a receive frame address matches an active address, the frame is written to the DMA memory if used.

Frames may be filtered using the type ID field for matching. Four type ID match registers exist and each can be enabled for matching by writing a "1" to the MSB (bit 31) of the respective register. When a frame is received, the matching is implemented as an OR function of the various types of match.

The content of each type ID register (when enabled) is compared against the length/type ID of the frame being received (for example, bytes 13 and 14 in non-VLAN and non-SNAP encapsulated frames) and copied to system memory if a match is found. The encoded type ID match bits (Word 0, bit 22, and bit 23) in the receive buffer descriptor status are set indicating which type ID match register generated the match, if the receive checksum offload is disabled.

The reset state of the type ID match registers is "0", hence each is initially disabled.

The following example illustrates the use of the address and type ID match registers for a MAC address of 21:43:65:87:A9:CB:

Preamble	55h
SFD	D5h
DA (Octet 0 - LSB)	21h
DA (Octet 1)	43h
DA (Octet 2)	65h
DA (Octet 3)	87h
DA (Octet 4)	A9h
DA (Octet 5 - MSB)	CBh
SA (Octet 0 - LSB)	00[1)]
SA (Octet 1)	00[1)]
SA (Octet 2)	00[1)]
SA (Octet 3)	00[1)]
SA (Octet 4)	00[1)]
SA (Octet 5 - MSB)	00[1)]
Type ID (MSB)	43h
Type ID (LSB)	21h

1) Contains the address of the transmitting device.

The sequence above shows the beginning of an Ethernet frame. Byte order of transmission is from top to bottom as shown. For a successful match to specific address 1, the following address match registers must be set up:

ETHx_spec_add1_bottom (0x088) *87654321h*

ETHx_spec_add1_top (0x08C) *0000CBA9h*

And for a successful match to the type ID, the specific address 1 register must be set up:

ETHx_spec_type1 (0x0A8) *80004321h*

29.3.7.1 Broadcast address

Frames with the broadcast address of FFFFFFFFFFFFh are stored to memory only if the no_broadcast bit in the *ETHx_network_config* register is set to '0'.

29.3.7.2 Hash addressing

The 64-bit hash register (*ETHx_hash_top/ETHx_hash_bottom*) takes up two locations in the memory map. The least significant bits are stored in the *ETHx_hash_bottom* register and the most significant bits in the *ETHx_hash_top register*. The unicast hash enable and the multicast hash enable bits in the *ETHx_network_config* register enable the reception of hash matched frames. The destination address is reduced to a 6-bit index into the 64-bit Hash register using the following hash function (*{ETHx_hash_top,ETHx_hash_bottom}[2hash_index[5:0]]*).

The hash function is an XOR of every sixth bit of the destination address.

hash_index[05] = da[05] ^ da[11] ^ da[17] ^ da[23] ^ da[29] ^ da[35] ^ da[41] ^ da[47]
hash_index[04] = da[04] ^ da[10] ^ da[16] ^ da[22] ^ da[28] ^ da[34] ^ da[40] ^ da[46]
hash_index[03] = da[03] ^ da[09] ^ da[15] ^ da[21] ^ da[27] ^ da[33] ^ da[39] ^ da[45]
hash_index[02] = da[02] ^ da[08] ^ da[14] ^ da[20] ^ da[26] ^ da[32] ^ da[38] ^ da[44]
hash_index[01] = da[01] ^ da[07] ^ da[13] ^ da[19] ^ da[25] ^ da[31] ^ da[37] ^ da[43]
hash_index[00] = da[00] ^ da[06] ^ da[12] ^ da[18] ^ da[24] ^ da[30] ^ da[36] ^ da[42]

- da[00] represents the least significant bit of the first byte received, that is, the multicast/unicast indicator, and da[47] represents the most significant bit of the last byte received.
- If the hash index points to a bit that is set in the hash register, then the frame will be matched according to whether the frame is multicast or unicast.
- A multicast match will be signaled if the multicast hash enable bit is set, da[00] is '1' and the hash index points to a bit set in the hash register.
- A unicast match will be signaled if the unicast hash enable bit is set, da[00] is '1' and the hash index points to a bit set in the hash register.
- To receive all multicast frames, the hash register must be set with all '1' and the multicast hash enable bit must be set to '1' in the *ETHx_network_config register*.

29.3.7.3 Copy all frames

If the copy all frames bit is set in the *ETHx_network_config* register then all frames (except those that are too long, too short, have FCS errors, or have RX_ER asserted during reception) will be copied to memory. Frames with FCS errors will be copied if bit 26 is set in the *ETHx_network_config* register.

29.3.7.4 Disable copy of pause frames

Pause frames can be prevented from being written to memory by setting the disable copying of pause frames control bit 23 in the *ETHx_network_config* register. When set, pause frames are not copied to system memory regardless of the copy all frames bit, whether a hash match is found, a type ID match is identified, or if a destination address match is found.

29.3.7.5 VLAN support

An Ethernet encoded 802.1Q VLAN tag appears as follows:

Tag protocol identifier (TPID) 16 bits	Tag control information (TCI) 16 bits
8100h	First 3 bits priority, then CFI bit, last 12 bits VID

The VLAN tag is inserted at the thirteenth byte of the frame adding an extra four bytes to the frame. To support these extra four bytes, the EMAC can accept frame lengths up to 1536 bytes by setting bit 8 in the *ETHx_network_config* register. If the VID (VLAN identifier) is null (000h) this indicates a priority-tagged frame.

The following bits in the receive buffer descriptor status word give information about VLAN tagged frames:

- Bit 21 set if receive frame is VLAN tagged (type ID of 8100h).
- Bit 20 set if receive frame is priority tagged (type ID of 8100h and null VID). If bit 20 is set bit 21 will also be set.
- Bit 19, 18, and 17 set to priority if bit 21 is set.
- Bit 16 set to CFI if bit 21 is set.

EMAC can be configured to reject all frames except VLAN-tagged frames by setting the discard non-VLAN frames bit in the *ETHx_network_config* register.

29.3.8 IEEE 1588 and IEEE 802.1AS support

IEEE Std 1588 is a standard for precision time synchronization in local area networks. It works with the exchange of special precision time protocol (PTP) frames. The PTP messages can be transported over IEEE Std 802.3/Ethernet, over internet protocol version 4 (IPv4) or over internet protocol version 6 (IPv6) as described in the annex of IEEE Std 1588-2008.

Most IEEE Std 1588 functionality can be implemented in software but for greatest accuracy, hardware assist is required to detect when PTP event messages pass the MII interface (clock timestamp point).

Synchronization between master and slave clocks is a two-stage process.

First, the offset between the master and slave clocks is corrected by the master sending a sync frame to the slave with a follow-up frame containing the exact time the sync frame was sent. Hardware assist modules at the master and slave side detect exactly when the sync frame was sent by the master and received by the slave. The slave then corrects its clock to match the master clock.

Second, the transmission delay between the master and slave is corrected. The slave sends a delay request frame to the master, which sends a delay response frame in reply. Hardware assist modules at the master and slave side detect exactly when the delay request frame was sent by the slave and received by the master. The slave will now have enough information to adjust its clock to account for delay. For example, if the slave assumes zero delay the actual delay will be half the difference between transmit and receive time of the delay request frame (if transmit and receive times are equal) because the slave clock will be lagging the master clock by the delay time already.

For hardware assist, it is necessary to timestamp when sync and Delay_Req messages are sent and received. The timestamp is taken when the message timestamp point passes the clock timestamp point. For Ethernet, the message timestamp point is the SFD and the clock timestamp point is the MII interface. (The IEEE Std 1588 specification refers to Sync and Delay_Req messages as event messages because these require time stamping. Follow up, delay response, and management messages do not require time stamping and are referred to as general messages.)

IEEE Std 1588 version 2 (IEEE Std 1588-2008) defines two additional PTP event messages. These are the peer delay request (Pdelay_Req) and peer delay response (Pdelay_Resp) messages. These messages are used to calculate the delay on a link. Nodes at both ends of a link send both types of frames (regardless of whether they contain a master or slave clock). The Pdelay_Resp massage contains the time at which a Pdelay_Req was received and is itself an event message. The time at which a Pdelay_Resp message is received is returned in a Pdelay_Resp_Follow_Up message.

IEEE Std 1588 version 2 introduces two kinds of transparent clocks, peer-to-peer (P2P) and end-to-end (E2E). Transparent clocks measure the transit time of event messages through a bridge and amend a correction field within the message to allow for the transit time. P2P transparent clocks additionally correct the delay in the receive path of the link using the information gathered from the peer delay frames. With P2P transparent clocks Delay_Req messages are not used to measure link delay. This simplifies the protocol and makes larger systems more stable.

The Ethernet MAC recognizes ten different encapsulations for PTP event messages:

- IEEE Std 1588 version 1 (UDP/IPv4 multicast)
- IEEE Std 1588 version 1 (UDP/IPv4 multicast with VLAN)
- IEEE Std 1588 version 2 (UDP/IPv4 multicast)
- IEEE Std 1588 version 2 (UDP/IPv4 multicast with VLAN)
- IEEE Std 1588 version 2 (UDP/IPv4 unicast)
- IEEE Std 1588 version 2 (UDP/IPv4 unicast with VLAN)
- IEEE Std 1588 version 2 (UDP/IPv6 multicast)
- IEEE Std 1588 version 2 (UDP/IPv6 multicast with VLAN)
- IEEE Std 1588 version 2 (Ethernet multicast)
- IEEE Std 1588 version 2 (Ethernet multicast with VLAN)

***Note**: IEEE Std 1588 version 1 (IEEE Std 1588-2002)*

Unicast PTP frame recognition is enabled via bit 20 of the *ETHx_network_config register*. The unicast addresses themselves are programmable via the PTP Unicast IP destination address (*ETHx_rx_ptp_unicast*/ *ETHx_tx_ptp_unicast*) register. The first holds the RX unicast IP destination address and the other, the TX unicast destination address. The PTP Unicast IP destination address register should only be changed when the unicast PTP frame recognition is disabled.

Example of a sync frame in the IEEE Std 1588 version 1 format:

Preamble/SFD	55555555555555D5h
DA (Octets 0 - 5)	–
SA (Octets 6 - 11)	–
Type (Octets 12 - 13	0800h
IP stuff (Octets 14 - 22)	–
UDP (Octet 23)	11h
IP stuff (Octets 24 - 29)	–
IP DA (Octets 30 - 32)	E00001h
IP DA (Octet 33)	81h or 82h or 83h or 84h
source IP port (Octets 34 - 35)	–
dest IP port (Octets 36 - 37)	013Fh
other stuff (Octets 38 - 42)	–
version PTP (Octet 43)	01h
other stuff (Octets 44 - 73)	–
control (Octet 74)	00h
other stuff (Octets 75 - 168)	–

Example of a Delay_Req frame in the IEEE Std 1588 version 1 format:

Preamble/SFD	55555555555555D5h
DA (Octets 0 - 5)	–
SA (Octets 6 - 11)	–
Type (Octets 12 - 13)	0800h
IP stuff (Octets 14 - 22)	–
UDP (Octet 23)	11h
IP stuff (Octets 24 - 29)	–
IP DA (Octets 30 - 32)	E00001h
IP DA (Octet 33)	81h or 82h or 83h or 84h
source IP port (Octets 34 - 35)	–
dest IP port (Octets 36 - 37)	013Fh
other stuff (Octets 38 - 42)	–
version PTP (Octet 43)	01h

other stuff (Octets 44 - 73)	–
control (Octet 74)	01h
other stuff (Octets 75 - 168)	–

For IEEE Std 1588 version 1 messages sync and Delay_Req frames are indicated by the EMAC if the frames type field indicates TCP/IP, UDP protocol is indicated, the destination IP address is 224.0.1.129/130/131/132, the destination UDP port is 319, and the control field is correct. The control field is 00h for sync frames and 01h for Delay_Req frames.

For IEEE Std 1588 version 2 messages, the type of frame is determined by the message type field in the first byte of the PTP frame. Whether a frame is version 1 or version 2 can be determined by the version PTP field in the second byte of version 1 and version 2 PTP frames.

In version 2 messages, sync frames have a message type value of 0h, Delay_Req frames have 1h, Pdelay_Req frames have 2h, and Pdelay_Resp frames have 3h.

Example of a sync frame in the IEEE Std 1588 version 2 (UDP/IPv4) format:

Preamble/SFD	55555555555555D5h
DA (Octets 0 - 5)	–
SA (Octets 6 - 11)	–
Type (Octets 12 - 13)	0800h
IP stuff (Octets 14 - 22)	–
UDP (Octet 23)	11h
IP stuff (Octets 24 - 29)	–
IP DA (Octets 30 - 33)	E0000181h
source IP port (Octets 34 - 35)	–
dest IP port (Octets 36 - 37)	013Fh
other stuff (Octets 38 - 41)	–
message type (Octet 42)	00h
version PTP (Octet 43)	02h

Example of a Pdelay_Req frame in the IEEE Std 1588 version 2 (UDP/IPv4) format:

Preamble/SFD	55555555555555D5h
DA (Octets 0 - 5)	–
SA (Octets 6 - 11)	–
Type (Octets 12 - 13)	0800h
IP stuff (Octets 14 - 22)	–
UDP (Octet 23)	11h
IP stuff (Octets 24 - 29)	–
IP DA (Octets 30 - 33)	E000006Bh
source IP port (Octets 34 - 35)	–
dest IP port (Octets 36 - 37)	013Fh
other stuff (Octets 38 - 41)	–

message type (Octet 42)	02h
version PTP (Octet 43)	02h

Example of a sync frame in the IEEE Std 1588 version 2 (UDP/IPv6) format:

Preamble/SFD	55555555555555D5h
DA (Octets 0 - 5)	–
SA (Octets 6 - 11)	–
Type (Octets 12 - 13)	86DDh
IP stuff (Octets 14 - 19)	–
UDP (Octet 20)	11h
IP stuff (Octets 21 - 37)	–
IP DA (Octets 38 - 53)	FF0X000000000181h
source IP port (Octets 54 - 55)	–
dest IP port (Octets 56 - 57)	013Fh
other stuff (Octets 58 - 61)	–
message type (Octet 62)	00h
other stuff (Octets 63 - 93)	–
version PTP (Octet 94)	02h

Example of a Pdelay_Req frame in the IEEE Std 1588 version 2 (UDP/IPv6) format:

Preamble/SFD	55555555555555D5h
DA (Octets 0 - 5)	–
SA (Octets 6 - 11)	–
Type (Octets 12 - 13)	86DDh
IP stuff (Octets 14 - 19)	–
UDP (Octet 20)	11h
IP stuff (Octets 21 - 37)	–
IP DA (Octets 38 - 53)	FF0200000000006Bh
source IP port (Octets 54 - 55)	–
dest IP port (Octets 56 - 57)	013Fh
other stuff (Octets 58 - 61)	–
message type (Octet 62)	03h
other stuff (Octets 63 - 93)	–
version PTP (Octet 94)	02h

Example of a sync frame in the IEEE Std 1588 version 2 (Ethernet multicast) format. For the multicast address 011B19000000h sync and Delay_Req frames are recognized depending on the message type field, 00h for sync, and 01h for Delay_Req:

Preamble/SFD	55555555555555D5h
DA (Octets 0 - 5)	011B19000000h
SA (Octets 6 - 11)	–
Type (Octets 12 - 13)	88F7h
message type (Octet 14)	00h
version PTP (Octet 15)	02h

Example of a Pdelay_Req frame in the IEEE Std 1588 version 2 (Ethernet multicast) format is given here. These need a special multicast address so they can get through ports blocked by the spanning tree protocol. For the multicast address 0180C200000Eh Sync, Pdelay_Req, and Pdelay_Resp frames are recognized depending on the message type field, 00h for Sync, 02h for Pdelay_Req, and 03h for Pdelay_Resp:

Preamble/SFD	55555555555555D5h
DA (Octets 0 - 5)	0180C200000Eh
SA (Octets 6 - 11)	–
Type (Octets 12 - 13)	88F7h
message type (Octet 14)	02h
version PTP (Octet 15)	02h

The EMAC contains a timestamp unit (TSU), which consists of a timer and registers to capture the time at which PTP event frames cross the message timestamp point. The registers are accessible through the EMAC's AHB slave interface. An interrupt is issued when a capture register is updated.

29.3.8.1 Support for time stamping and timestamp accuracy

The MAC has the responsibility of sampling the TSU timer value when the TX or RX SOF event of the frame passes the MII boundary. This event is an existing signal synchronous to MAC TX/RX clock domains. The MAC uses the sampled timestamp to insert the timestamp into transmitted PTP Sync frames (if one step sync feature is enabled), to pass to the Ethernet MAC's register block to capture the timestamp (TS) in registers, or to pass to the Ethernet MAC DMA to insert into TX or RX buffer descriptors. For each of these, the SOF event, which is captured in the TX and RX clock domains respectively, is synchronized to the TSU clock domain and the resulting signal is used to sample the TSU count value. This value will be kept stable for an entire frame, or specifically at least 64 TX/RX clock cycles, because the minimum frame size in Ethernet is 64 bytes and worst case is a transfer rate of one byte per cycle. It is used as the source for all the components within the Ethernet MAC that require the timestamp value. The SOF event must pass a clock boundary and there may be a degree of inaccuracy in the captured timestamp. The level of inaccuracy depends on the frequency of the TSU clock (clk_tsu). There will be no more than one clock cycle of inaccuracy.

In the best case, the SOF event (which is in the TX/RX clock domain) just meets the setup time of the TSU clock domain at the input to the first synchronization flip-flop. The captured TS is N+2, but it really should be N+1. In the worst case, the captured TS is also N+2, but it really should be N.

29.3.8.2 Single step time stamping

Support of one step clock for TX sync frames can be enabled by setting bit 24 in the *ETHx_network_control* register. In this mode, the timestamp field within the IEEE Std 1588 version 2 sync frame, is replaced by the TSU

timestamp value at the time the sync frame SOF passes the MII interface. To use single step time stamping, the sampled timestamp must be stable before the point at which EMAC requires to insert the timestamp. This can be guaranteed by enforcing a rule that TSU clock (clk_tsu) is greater than one-eighth the frequency of TX clock (TX_CLK) or RX clock (RX_CLK).

29.3.8.3 Timestamp capture registers

Four 80-bit timestamp status registers capture the time at which PTP event frames are transmitted and received.

- ETHx_tsu_ptp_rx_msb_sec, ETHx_tsu_ptp_rx_sec, ETHx_tsu_ptp_rx_nsec
- ETHx_tsu_ptp_tx_msb_sec, ETHx_tsu_ptp_tx_sec, ETHx_tsu_ptp_tx_nsec
- ETHx_tsu_peer_rx_msb_sec, ETHx_tsu_peer_rx_sec, ETHx_tsu_peer_rx_nsec
- ETHx_tsu_peer_tx_msb_sec, ETHx_tsu_peer_tx_sec, ETHx_tsu_peer_tx_nsec

An interrupt is issued when these registers are updated.

29.3.8.4 Timestamp capture in DMA descriptors

The TX and RX timestamp can optionally be captured in an extended buffer descriptor when configured using bits [29:28] in the *ETHx_dma_config* register. The timestamp can be captured for a number of frame types (PTP event, PTP general, all frames, or none as defined in the ETHx_tx_bd_control/ETHx_rx_bd_control registers) and a bit within buffer descriptor word 0/1 is used to indicate that the timestamp is present.

29.3.8.5 Controlling timestamp unit

The timer is implemented as a 102-bit register with the upper 48 bits counting seconds, the next 30 bits counting nanoseconds, and the lowest 24 bits counting sub-nanoseconds. The lower 54 bits roll over when they have counted to one second. An interrupt is generated when the seconds increment. The timer value can be read, written, and adjusted through the AHB slave interface. The timer is clocked with TSU clock (clk_tsu).

The two operation modes that control the way the timer varies over time are:

- Increment mode – Increments the timer by a fixed value every TSU clock.
- Alternative increment mode – Increments the timer by a fixed value for a fixed number of clocks, followed by an alternative increment value for a single clock. This is the legacy mode and is not recommended for use.

The timer count value can be compared to a programmable comparison value. For the comparison, the 48 bits of the seconds value and the upper 22 bits of the nanoseconds value are used. A signal is output from the core to indicate when the TSU timer count value is equal to the comparison value stored in the TSU timer comparison value registers (0x0DC, 0x0E0, and 0x0E4). An interrupt can be issued when the timer count value and the comparison value in the IEEE 1588 timer comparison value registers (*ETHx_tsu_msb_sec_cmp, ETHx_tsu_sec_cmp and ETHx_tsu_nsec_cmp*) are equal. The interrupt can be enabled with bit 29 in the interrupt enable (*ETHx_int_enable*) register.

EMAC is designed to recognize sync frames with both IEEE Std 1588 and IEEE Std 802.1AS addresses, and can support their frame recognition simultaneously.

29.3.8.6 Increment mode

The amount by which the timer increments each clock cycle is controlled by the timer increment (ETHx_tsu_timer_incr) register. Bits [7:0] are the default increment value in nanoseconds and an additional 16 bits of sub-nanosecond resolution are available using the timer increment *ETHx_sub_nsec* register (ETHx_tsu_timer_incr_sub_nsec). If the rest of the timer increment register is written with ‘0’ the timer increments by the value in bits [7:0] and the value in timer increment ETHx_sub_nsec register, each clock cycle.

29.3.8.7 Alternate increment mode

Bits [15:8] of the timer increment register are the alternative increment value in nanoseconds and bits [23:16] are the number of increments after which the alternative increment value is used. If bits [23:16] are 00h, then the alternative increment value will never be used.

29.3.9 MAC 802.3 pause frame support

The EMAC supports both hardware-controlled pause of the transmitter upon reception of a pause frame and hardware-generated pause frame transmission.

29.3.9.1 IEEE Std 802.3 pause frame reception

Bit 13 of the *ETHx_network_config* register is the pause enable control for reception. If this bit is set, transmission will pause when a non-zero pause quantum frame is received.

If a valid pause frame is received, then the receive pause quantum (ETHx_pause_time) register is updated with the new frame's pause time regardless of whether a previous pause frame is active. An interrupt (either bit 12 or 13 of the *ETHx_int_status* register) is triggered when a pause frame is received, but only if the interrupt is enabled. Pause frames received with non-zero quanta are indicated through the interrupt bit 12 of the interrupt status (*ETHx_int_status*) register. Pause frames received with zero quanta are indicated on bit 13 of *ETHx_int_status*.

After the receive pause quantum (*ETHx_pause_time*) is loaded and the frame currently being transmitted is sent, no new frames are transmitted until the pause time reaches zero. The loading of a new pause time, and hence pausing of transmission, occurs because the EMAC is operating in full duplex mode. A valid pause frame is defined as having a destination address that matches either the address stored in specific address 1 register or the reserved address of 0180C2000001h. It must also have the MAC control frame type ID of 8808h and the pause opcode of 0001h.

Pause frames that have FCS or other errors will be treated as invalid and will be discarded. IEEE Std 802.3 pause frames that are received after priority-based flow control (PFC) is negotiated will also be discarded. Valid pause frames received will increment the pause frames received statistics register.

The receive pause quantum register decrements every 512-bit times when transmission stops. For test purposes, the retry test bit can be set (bit 12 in *ETHx_network_config*), which causes the receive pause quantum register to decrement every TX_CLK cycle after transmission has stopped.

The interrupt (bit 13 in the *ETHx_int_status* register) is asserted whenever the receive pause quantum register decrements to zero and it is enabled. This interrupt is also set when a zero quantum pause frame is received.

29.3.9.2 IEEE Std 802.3 pause frame transmission

Automatic transmission of pause frames is supported through the transmit pause frame bits of the *ETHx_network_control* register. If either bit 11 or bit 12 of *ETHx_network_control* is set to '1', an IEEE Std 802.3 pause frame will be transmitted, provided the MAC transmitter is enabled (bit 3) in the *ETHx_network_control* register.

Pause frame transmission will happen immediately if transmit is inactive or if transmit is active between the current frame and the next frame due to be transmitted.

Transmitted pause frames comprise of the following:

- A destination address of 0180C2000001h
- A source address taken from specific address 1 register
- A type ID of 8808h (MAC control frame)
- A pause opcode of 0001h
- A pause quantum register

- Fill of 00h to take the frame to minimum frame length
- Valid FCS

The pause quantum used in the generated frame will depend on the trigger source for the frame as follows:

- If bit 11 is written with '1', the pause quantum will be taken from the transmit pause quantum (*ETHx_tx_pause_quantum*) register. This register resets to a value of 0xFFFF giving maximum pause quantum as the initial value.
- If bit 12 is written with '1', the pause quantum will be zero.

After transmission, a pause frame transmitted interrupt will be generated (bit 14 of *ETHx_int_status*) and the only statistics register that will be incremented will be the pause frames transmitted (*ETHx_pause_frame_txed*) register.

Pause frames can also be transmitted by the MAC using normal frame transmission methods.

29.3.10 MAC PFC priority-based pause frame support

EMAC supports PFC priority-based pause transmission and reception. Before PFC pause frames can be received, bit 16 of the network control register must be set to '1'.

29.3.10.1 PFC pause frame reception

The ability to receive and decode priority-based pause frames is enabled by setting bit 16 of the network control (*ETHx_network_control*) register. When this bit is set, EMAC will match either classic IEEE Std 802.3 pause frames or PFC priority-based pause frames. After a priority-based pause frame is received and matched, the EMAC will only match priority-based pause frames (this is IEEE Std 802.1Qbb requirement, known as PFC negotiation). When a priority-based pause is negotiated, any received IEEE Std 802.3x format pause frames will not be acted upon.

If a valid priority-based pause frame is received, then EMAC will decode the frame and determine which, if any, of the eight priorities require to be paused. Up to eight pause time registers are then updated with the eight pause times extracted from the frame regardless of whether a previous pause operation is active. An interrupt (ETHx_int_status.pause_frame_with_non_zero_pause_quantum_received) is triggered when a non-zero PFC pause frame is received, but only if the interrupt is enabled (bit 12 of the interrupt mask register). Pause frames received with non-zero quantum are indicated through the interrupt bit 12 of the interrupt status register. PFC pause frames received with zero quantum cannot trigger an interrupt; that is, bit 13 is never set for PFC pause frames. The loading of a new pause time occurs because the EMAC is operating in full duplex mode. A valid pause frame is defined as having a destination address that matches either the address stored in specific address 1 register or if it matches the reserved address of 0180C2000001h. It must also have the MAC control frame type ID of 8808h and the pause opcode 0101h.

Pause frames that have FCS or other errors will be treated as invalid and will be discarded. Valid pause frames received will increment the pause frames received statistic register.

The receive pause quantum (*ETHx_pause_time*) register decrement every 512 bit times immediately, following the PFC frame reception. For test purposes, the retry test bit can be set (bit 12 in the *ETHx_network_config* register), which causes the receive pause quantum (*ETHx_pause_time*) register to decrement every RX_CLK cycle when transmission has stopped.

29.3.10.2 PFC pause frame transmission

Automatic transmission of pause frames is supported through the transmit priority-based pause frame bit of the network control (*ETHx_network_control*) register. If *ETHx_network_control.transmit_pfc_priority_based_pause_frame* is set to '1', a PFC pause frame will be transmitted, provided the MAC transmitter is enabled (bit 3) in *ETHx_network_control*. When bit 17 of

ETHx_network_control is set to '1', the fields of the priority base pause frame will be built using the values stored in the transmit PFC pause (*ETHx_tx_pfc_pause*) register.

Pause frame transmission will happen immediately if transmit is inactive or if transmit is active between the current frame and the next frame due to be transmitted.

Transmitted pause frames comprise of the following:

- A destination address of 0180C2000001h
- A source address taken from specific address 1 register
- A type ID of 8808h (MAC control frame)
- A pause opcode of 0101h
- A priority enable vector taken from the transmit PFC pause (*ETHx_tx_pfc_pause*) register
- Eight pause quanta in four registers (*ETHx_tx_pause_quantum, ETHx_tx_pause_quantum1, ETHx_tx_pause_quantum2, ETHx_tx_pause_quantum3*)
- Fill of 00h to take the frame to minimum frame length
- Valid FCS

The pause quantum registers used in the generated frame will depend on the trigger source for the frame as follows:

- If bit 17 of the network control (*ETHx_network_control*) register is set to '1', the priority enable vector of the priority-based pause frame will be equal to the value stored in the transmit PFC pause (*ETHx_tx_pfc_pause*) register bits [7:0]. For each entry equal to '0' in *ETHx_tx_pfc_pause* [15:8], the pause quantum field of the pause frame associated with that entry will be taken from the transmit pause quantum (*ETHx_tx_pause_quantum*) register. For each entry equal to '1' in *ETHx_tx_pfc_pause* [15:8], the pause quantum associated with that entry will be '0'. The *ETHx_tx_pause_quantum* register resets to a value of FFFFh giving maximum pause quantum as the initial value.
- The pause quantum registers are classified as static and these registers should be updated only when no PFC frame is transmitted.
- To use the eight priority pause quanta stored in the four *ETHx_tx_pause_quantum* registers, set bit 24 to '1' in the *ETHx_network_control* register.

After transmission, a pause frame transmit interrupt will be issued (bit 14 of the ETHx_int_status register) and the only statistics register that will be incremented will be the pause frames transmitted (ETHx_pause_frame_txed) register.

PFC pause frames can also be transmitted by the MAC using normal frame transmission methods.

29.3.11 Energy efficient Ethernet support

IEEE Std 802.3az adds support for energy efficiency to Ethernet (EEE). These are the key features of IEEE Std 802.3az:

- Allows a system's transmit path to enter a low-power mode if there is nothing to transmit.
- Allows a PHY to detect whether its link partner's transmit path is in low-power mode, therefore allowing the system's receive path to enter low-power mode.
- Link remains up during low-power mode and no frames are dropped.
- Asymmetric, one direction can be in low-power mode while the other is transmitting normally.
- Low-power idle (LPI) signaling is used to control entry and exit to and from low-power modes.
- LPI signaling can only take place if both sides have indicated support for it through auto-negotiation.

IEEE Std 802.3az operation:

- Low-power control is done at MII (reconciliation sublayer).

- As an architectural convenience in writing the 802.3az, it is assumed that transmission is deferred by asserting carrier sense; in practice it will not be done this way. This system will know when it has nothing to transmit and only enter low-power mode when it is not transmitting.
- LPI should not be requested unless the link has been up for at least one second.
- A PHY on seeing LPI requested on MII will send the sleep signal before going quiet. After going quiet, it will periodically emit refresh signals.
- The sleep, quiet, and refresh periods are defined in Table 78-2 of the IEEE Std 802.3az. For 1000BASE-X, the sleep period is 20 microseconds, the quiet period is 2.5 milliseconds and the refresh period is 20 microseconds. For 100BASE-TX, the sleep period (Ts) is 100 microseconds, the quiet period is (Tqt)/(Tqr) are 20/24 milliseconds, and the refresh period (Tr) is 100 microseconds. 10BASE-T is not supported.
- 1000BASE-X/100BASE-TX is required to go quiet after sleep is signaled.
- LPI mode ends by transmitting normal idle for the wake time. This has a default time, which can be adjusted in software using the link layer discovery protocol (LLDP) described in clause 79 of IEEE Std 802.3az.
- LPI is indicated at the receive side when sleep and refresh signaling is detected.

29.3.11.1 LPI operation in EMAC

Auto-negotiation:

- Indicate EEE capability using auto-negotiation.

For the transmit path:

- If the link has been up for 1 second and nothing is being transmitted, write to the LPI bit (bit 19) in the network control register.
- Wake up by clearing the LPI bit in the network control register.

For the receive path:

- Wait for an interrupt to indicate that LPI is received.
- Wait for an interrupt to indicate that regular idle is received and then re-enable the receive path.

29.3.12 MDIO interface

MDIO is a single bi-directional tristate signal between the EMAC and PHY.

PHY maintenance register (*ETHx_phy_management*) is implemented as a shift register. Writing to the register starts a shift operation, which is signaled as complete when bit two is set in the network status register (about 2000 pclk cycles later when bits [18:16] are set to 010 in the ETHx_network_config register). An interrupt is generated as this bit is set.

During this time, the MSb of the register is output on the mdio_out pin and the LSb updated from the mdio_in pin with each MDC cycle. This causes the transmission of a PHY management frame on MDIO.

Reading during the shift operation will return the current contents of the shift register. At the end of the management operation the bits will have shifted back to their original locations. For a read operation, the data bits will be updated with data read from the PHY. It is important to write the correct values to the register to ensure a valid PHY management frame is produced.

MDC should not toggle faster than 2.5 MHz (minimum period of 400 ns), as defined by the IEEE 802.3 standard. MDC is generated by dividing clk_sys. Three bits in the network configuration register determine by how much pclk should be divided to produce MDC.

29.3.13 Interrupts

Several interrupt conditions can be enabled in EMAC. The interrupt outputs from the Ethernet MAC match the number of supported priority queues. Only EMAC DMA related events are reported using the individual

interrupt outputs, as the Ethernet MAC can relate these events to specific queues. All other events generated within the Ethernet MAC are reported in the interrupt associated with the lowest priority queue (queue 0). For the lowest priority queue, the interrupt status (ETHx_int_status) register is located at offset address 0x024. For all other priority queues, this register is located at sequential offset addresses starting at 0x400.

At reset all interrupts are disabled. To enable an interrupt, write to interrupt enable register with the pertinent interrupt bit set to 1. To disable an interrupt, write to interrupt disable register with the pertinent interrupt bit set to 1. To check whether an interrupt is enabled or disabled, read interrupt mask register: if the bit is set to '1', the interrupt is disabled.

29.3.14 Media independent interfaces

29.3.14.1 MII

EMAC connects to PHY using the MII interface (ETHx_TXD, ETHx_TX_ER, ETHx_TX_CLK, ETHx_RX_ER, ETHx_RX_CLK).

Configuration and status information is exchanged between the EMAC and PHY using the management interface (ETHx_MDC and ETHx_MDIO).

Table 240 Signals used in MII/RGMII

Signal name	Description	Direction
ETHx_TXD	4 Transmit data lines for MII	MAC to PHY
ETHx_TX_ER	Transmit error	MAC to PHY
ETHx_TX_CLK	Transmit clock	PHY to MAC for MII [10/100 Mbps]
ETHx_RXD	4 Receive data lines for MII	PHY to MAC
ETHx_RX_ER	Receive error	PHY to MAC
ETHx_RX_CLK	Receive clock	PHY to MAC

29.3.14.2 RGMII

In RGMII mode, the core receives a data nibble on ETHx_RXD[3:0] on both the edges of the ETHx_RX_CLK along with a single-bit control signal. The lower nibble of the data byte is received on the rising edge of ETHx_RX_CLK, and the upper nibble is received on the falling edge. The RGMII module receives a control bit on ETHx_RX_CTL on both edges of ETHx_RX_CLK. This control bit contains the data valid signal ETHx_RX_DV and encoded receiver error information of ETHx_RX_ER. The RGMII also uses the data and control signals to extract the in-band PHY status for carrier sense, collision, speed, link, and duplex.

Similarly, RGMII transmits the lower nibble of the data byte on ETHx_TXD[3:0] on the rising edge of ETHx_TX_CLK and the upper nibble is transmitted on the following falling edge. Control signals ETHx_TX_EN and ETHx_TX_ER are multiplexed on ETHx_TX_CTL.

The RGMII standard specifies a source synchronous clock. It relies on the clock having a longer path delay than the data so that the data is re-sampled using the same edge of the clock on which it was generated. Therefore, delay on clock signals are essential, which can be achieved either through trace lines or through delaying clocks from the source. EMAC supports RGMII specifications from OPEN alliance. During the transmission process MAC operates in delay-on-destination (DoD) mode; hence required delay must be accomplished by the PHY or through trace lines. For receive operation, MAC operates in delay-on-source (DoS) mode, in which delay must be achieved either by the PHY or through trace lines.

Configuration and status information is exchanged between the EMAC and PHY using the management interface (ETHx_MDC and ETHx_MDIO).

Table 241 Signals used in RGMII

Signal name	Description	Direction
ETHx_TXD	4 Transmit data lines	MAC to PHY
ETHx_TX_CTL	Transmit enable	MAC to PHY
ETHx_TX_CLK	Transmit clock	MAC to PHY
ETHx_RXD	4 Receive data lines	PHY to MAC
ETHx_RX_CTL	Receive data valid and error signal	PHY to MAC
ETHx_RX_CLK	Receive clock	PHY to MAC
ETHx_REF_CLK	High-speed reference clock	Input to MAC

29.3.14.3 RMII

As the name suggests, reduced media independent interface (RMII) communicates data to PHY through lower number of pins compared to MII. Transmit and received data lines are reduced from four to two, and clock is doubled. The clock source for RMII can be selected from external high-precision clock through clock line ETHx_REF_CLK; required clock for transmit and receive operations will be generated internally from this reference clock.

The configuration and status information is exchanged between the EMAC and PHY using the management interface (ETHx_MDC and ETHx_MDIO).

Table 242 Signals used in RMII

Signal name	Description	Direction
ETHx_TXD	2 Transmit data lines	MAC to PHY
ETHx_TX_CTL	Transmit enable	MAC to PHY
ETHx_RXD	2 Receive data lines	PHY to MAC
ETHx_RX_ER	Receive error	PHY to MAC
ETHx_RX_CTL	Receive data valid	PHY to MAC
ETHx_REF_CLK	External 50 MHz reference clock	Input to MAC

As described, EMAC supports four different PHY interfaces – MII, RMII and RGMII. Table 243 shows the required settings to select any of the interfaces.

Table 243 PHY interface selection

ETHx_CTL.ETH_MODE	ETHx_network_config[0] (speed)	ETHx_network_config[10] (gigabit_mode_enable)	PHY mode
2'd0	0	0	MII - 10 Mbps
2'd0	1	0	MII – 100 Mbps
2'd1	0	1	Reserved
2'd2	0	0	RGMII – 10 Mbps (4 bits/cycle)
2'd2	1	0	RGMII – 100 Mbps (4 bits/cycle)
2'd2	0	1	RGMII – 1000 Mbps (8 bits/cycle)

(table continues...)

Table 243 (continued) PHY interface selection

ETHx_CTL.ETH_MODE	ETHx_network_config[0] (speed)	ETHx_network_config[10] (gigabit_mode_enable)	PHY mode
2'd3	0	0	RMII – 10 Mbps
2'd3	1	0	RMII – 100 Mbps

***Note**:*
- *ETH_MODE must be configured before configuring network_config register.*
- *In the RGMII interface:*
  - *MAC is in delay-on-destination (DoD) mode for transmission; the delay needs to be accomplished by PHY.*
  - *MAC is in delay-on-source (DoS) mode for receive operation; the delay needs to be accomplished by PHY.*

29.3.14.4 Clock sources for PHY interface

Clock requirements and configurations are different for each interface. Following are the required clocks and source of clocks.

- MII
  - Both TX and RX clocks are supplied from external PHY.
- RMII
  - Both TX and RX clocks can be supplied from either internal reference clock or from external clock source.
  - ETHx_CTL.REFCLK_SRC_SEL must be used to select reference clock source from on-chip system resource or from HSIO.
  - Both TX and RX clocks are supplied from external PHY.
- RGMII
  - RX clock is supplied from PHY.
  - TX clock source can be selected either from internal clock source or from HSIO.
  - ETHx_CTL.REFCLK_SRC_SEL must be used to select clock source for TX functionality.
  - ETHx_CTL.REFCLK_DIV is used to divide reference clock and to generate required clock of 125 MHz.
  - Clock out will be enabled internally when the internal clock source is selected; TX reference clock to PHY can be provided through ETHx_TX_CLK.

***Note**: Use a more precise external clock source than the internal PLL for RGMII transmit operations.*

29.3.14.5 PHY interface

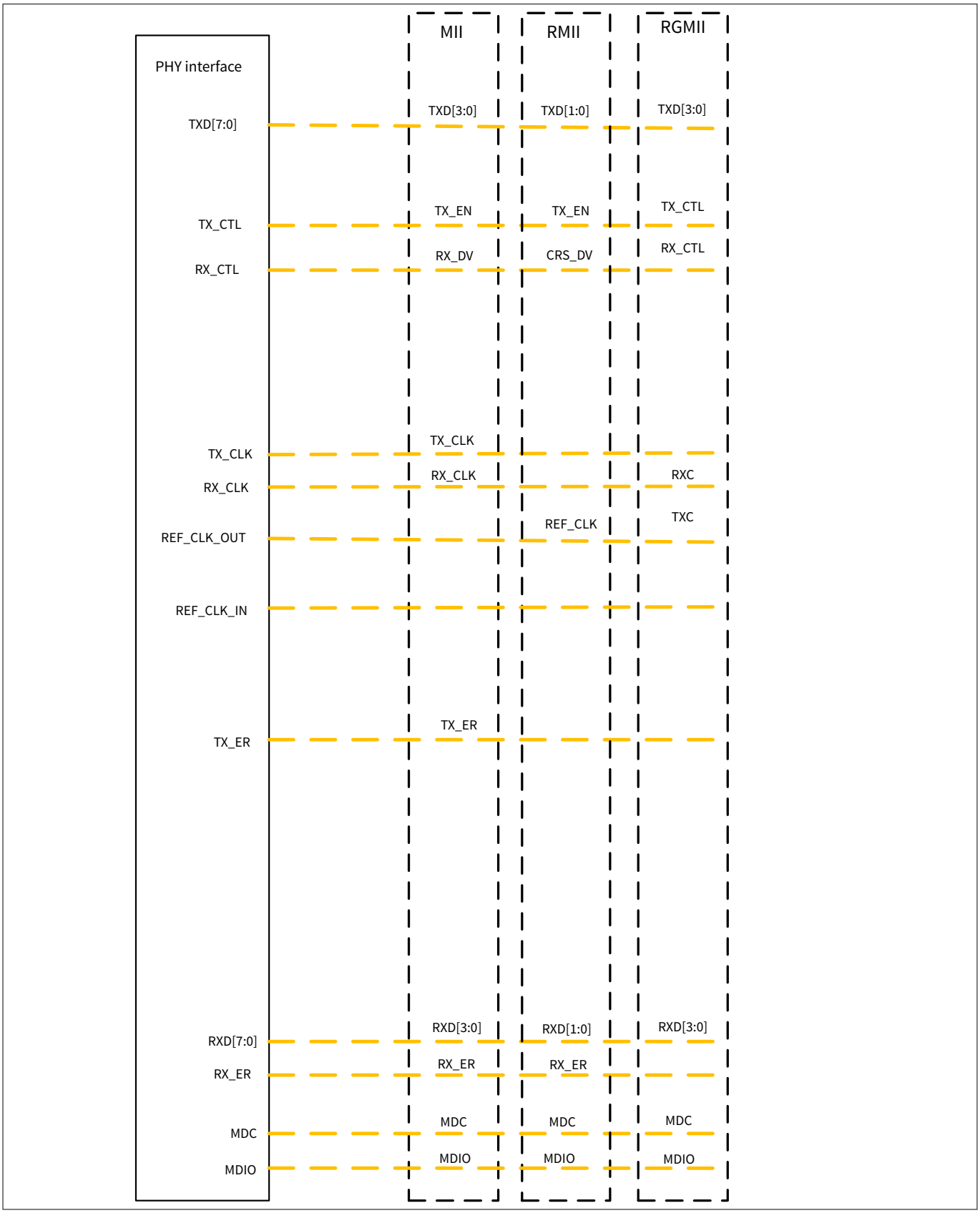


Figure 291 PHY interface

29.3.15 Clocks to EMAC

Other than clocks described in Clock sources for PHY interface, EMAC requires clocks to perform internal operation such as buffer data transfers or TSU operations. To perform those operations the following clocks are used. For more information about configuring mentioned clocks, see Clocking system.

Table 244 Clocks to EMAC

Clock	Function
clk_sys	To generate MDC clock and for AHB operation, clk_sys is derived from CLK_PERI
clk_tsu	TSU clock for TSU timer
clk_mem	Fast clock for AXI operation
clk_slow	Clock for AHB master operations
ref_int_clk	Internal reference clock supplied from PLL

***Notes**:*

1. *See the device-specific datasheet for assigned clocks to different EMAC instances.*

2. *See the device-specific datasheet for the minimum bus frequency requirements under ethernet specifications.*

29.3.15.1 Power modes

Table 245 shows EMAC availability in different device power modes. See the Device power modes for more details.

Table 245 EMAC status during different device power modes

Device power mode	IP status
Active	EMAC is fully operational in active power mode with power on and clocks running.
LPActive	EMAC is fully operational in LPActive power mode with power on and clocks running; clocks can be limited to save some power during LPActive mode.
Sleep	EMAC is fully operational in sleep power mode.
LPSleep	EMAC is fully operational in LPSleep power mode with power on and clocks running; clocks can be limited to save some power during LPSleep mode.
Deep sleep	No clock is provided during deep sleep power mode, hence the logic is not functional. All retention registers will hold the values in deep sleep mode.
Hibernate	Entire EMAC including retention register are not functional during hibernate power mode.

29.4 Registers

Table 246 EMAC registers

Register	Name	Description
ETHx_CTL	Control register	Control register to select type PHY mode and reference clock, and to enable the IP
ETHx_STATUS	Status register	Status register shows status about PFC frames
ETHx_network_control	Network Control register	The network control register contains general MAC control functions for both receiver and transmitter
ETHx_network_config	Network Configuration register	The network configuration register contains functions to set the mode of operation for the gigabit Ethernet MAC
ETHx_network_status	Network Status register	The network status register shows status regarding PHY management interface
ETHx_dma_config	DMA Configuration register	Register to configure DMA transfers for transmit and receive operation
ETHx_transmit_status	Transmit Status register	Register provides the status of a transmit. After it is read, individual bits can be cleared by writing 1 to them. It is not possible to set a bit to 1 by writing to the register.
ETHx_receive_q_ptr	Receive Queue 0 pointer	This register holds the start address of the receive buffer queue (receive buffers descriptor list).
ETHx_transmit_q_ptr	Transmit Queue 0 pointer	This register holds the start address of the transmit buffer queue (transmit buffers descriptor list).
ETHx_receive_status	Receive Status register	Register provides the status of receive operation. After it is read, individual bits may be cleared by writing 1 to them. It is not possible to set a bit to 1 by writing to the register
ETHx_int_status	Interrupt Status register	If not configured for priority queuing, the EMAC generates a single interrupt. This register indicates the source of this interrupt
ETHx_int_enable	Interrupt Enable	At reset all interrupts are disabled. Writing a one to the relevant bit location enables the required interrupt. This register is write only and when read will return zero.
ETHx_int_disable	Interrupt Disable	Register to disable interrupts

(table continues...)

Table 246 (continued) EMAC registers

Register	Name	Description
ETHx_int_mask	Interrupt Mask register	The interrupt mask register is a read only register indicating which interrupts are masked. All bits are set at reset and can be reset individually by writing to the interrupt enable register or set individually by writing to the interrupt disable register
ETHx_phy_management	PHY Management register	This register is implemented as a shift register. Writing to the register starts a shift operation which is signaled as complete when bit-2 is set in the network status register.
ETHx_pause_time	Pause Quantum register	Received pause quantum register
ETHx_tx_pause_quantum	Quantum register	Register to hold quantum to be transmitted
ETHx_pbuf_txcutthru	Packet buffer cut through configuration	Register is used to configure partial store and forward mode for transmit operation
ETHx_pbuf_rxcutthru	Packet buffer cut through configuration	Register to configure partial store and forward mode for receive operation
ETHx_jumbo_max_length	Jumbo length configuration	Jumbo frame size configuration
ETHx_axi_max_pipeline	AXI pipeline configuration	Used to set the maximum amount of outstanding transactions on the AXI bus between AR/R channels and AW/W channels.
ETHx_int_moderation	Interrupt Moderation register	Used to moderate the number of transmit and receive complete interrupts issued. With interrupt moderation enabled receive and transmit interrupts are not generated immediately a frame is transmitted or received. Instead when a receive or transmit event occurs a timer is started, and the interrupt is asserted after it times out.
ETHx_sys_wake_time	–	Used to pause transmission after low-power idle is deasserted
ETHx_hash_bottom	Hash Configuration register	The unicast hash enable and the multicast hash enable bits in the network configuration register enable the reception of hash matched frames. Hash register bottom (31 to 0 bits)
ETHx_hash_top	Hash Top Configuration register	Top 32 bits of hash configuration
ETHx_spec_add1_bottom to ETHx_spec_add4_bottom	Specific address (bottom) configuration	The addresses stored in the specific address registers are deactivated at reset or when their corresponding specific address register bottom is written. They are activated when specific address register top is written

(table continues...)

Table 246 (continued) EMAC registers

Register	Name	Description
ETHx_spec_add1_top to ETHx_spec_add4_top	Specific address (top) configuration	Specific top address
ETHx_spec_type1 to ETHx_spec_type4	Type ID Match register	Type ID match register
ETHx_stretch_ratio	IPG stretch configuration	Inter packet gap stretch register
ETHx_stacked_vlan	Stacked VLAN	Stacked VLAN register
ETHx_tx_pfc_pause	Transmit Pause register	Transmit PFC pause register
ETHx_mask_add1_bottom	Address Mask register	Specific address mask 1 bottom (31 to 0 bits)
ETHx_mask_add1_top	Mask register	Specific address mask 1 top (47 to 32 bits)
ETHx_dma_addr_or_mask	Receive DMA data buffer address mask	Receive DMA data buffer address mask
ETHx_rx_ptp_unicast	PTP RX unicast IP destination address	Unicast IP destination address to be detected during receiving PTP frame process.
ETHx_tx_ptp_unicast	PTP TX unicast IP destination address	Unicast IP destination address to be detected during transmitting PTP frame process
ETHx_tsu_nsec_cmp	TSU timer comparison value nanoseconds	TSU timer comparison value nanoseconds
ETHx_tsu_sec_cmp	TSU timer comparison value seconds	TSU timer comparison value seconds (31 to 0 bits)
ETHx_tsu_msb_sec_cmp	TSU timer comparison value seconds	TSU timer comparison value seconds (47 to 37 bits)
ETHx_tsu_ptp_tx_msb_sec	PTP Event Frame Transmitted (seconds) register	PTP event frame TX seconds. The register is updated with the value that the 1588 timer seconds register held when the SFD of a PTP transmit primary event crosses the MII interface.
ETHx_tsu_ptp_rx_msb_sec	PTP Event Frame Received (seconds) register	PTP event frame RX seconds. The register is updated with the value that the 1588 timer seconds register held when the SFD of a PTP receive primary event crosses the MII interface
ETHx_tsu_peer_tx_msb_sec	PTP Peer Event Frame Transmitted Seconds register (47 to 32 bits)	PTP Peer event frame TX seconds. The register is updated with the value that the 1588 timer seconds register held when the SFD of a PTP transmit peer event crosses the MII interface.
ETHx_tsu_peer_rx_msb_sec	PTP Peer Event Frame Received Seconds register (47 to 32 bits)	PTP Peer event frame RX seconds. The register is updated with the value that the 1588 timer seconds register held when the SFD of a PTP transmit peer event crosses the MII interface.
ETHx_dpram_fill_dbg	DMA packet buffer fill level	Register provides fill level of TX and RX packet buffer

(table continues...)

Table 246 (continued) EMAC registers

Register	Name	Description
ETHx_revision_req	Module identification number	Contains module identification number and revision number
ETHx_octets_tx_bottom	Octets transmitted	Register shows total valid octets transmitted (lower 32 bits)
ETHx_octets_tx_top	Octets transmitted	Register shows total valid octets transmitted (47 to 32 bits)
ETHx_frames_txed_ok	Transmitted frames	Register shows number of frames transmitted without error
ETHx_broadcast_txed	Broadcast frames transmitted	Number of broadcast frames transmitted without error
ETHx_multicast_txed	Multicast frames transmitted	Number of multicast frames transmitted without error
ETHx_pause_frames_txed	Pause frames transmitted	Number of pause frames transmitted without error
ETHx_frames_txed_64	64 byte frames transmitted	Number of 64 byte frames transmitted without error
ETHx_frames_txed_65	65 to 127 byte frames transmitted	Number of 65 to 127 byte frames transmitted without error
ETHx_frames_txed_128	128 to 255 byte frames transmitted	Number of 128 to 255 byte frames transmitted without error
ETHx_frames_txed_256	256 to 511 byte frames transmitted	Number of 256 to 511 byte frames transmitted without error
ETHx_frames_txed_512	512 to 1023 byte frames transmitted	Number of 512 to 1023 byte frames transmitted without error
ETHx_frames_txed_1024	1024 to 1518 byte frames transmitted	Number of 1024 to 1518 byte frames transmitted without error
ETHx_frames_txed_1519	Greater than 1518 byte frames transmitted	Greater than 1518 byte frames transmitted without error
ETHx_tx_underruns	Transmit underrun	10-bit register counting the number of frames not transmitted due to a transmit under run
ETHx_octets_rxed_bottom	Octets received (31 to 0 bits)	Number of octets received
ETHx_octets_rxed_top	Octets received (47 to 32 bits)	Number of octets received
ETHx_frames_rxed_ok	Frames received	Number of frames received without error
ETHx_broadcast_rxed	Broadcast frames received	Number of broadcast frames received without error
ETHx_multicast_rxed	Multicast frames received	Number of multicast frames received without errors

(table continues...)

Table 246 (continued) EMAC registers

Register	Name	Description
ETHx_pause_frames_rxed	Pause frames received	Number of pause frames received without errors
ETHx_frames_rxed_64	64 byte frames received	Number of frames with 64 bytes received without error
ETHx_frames_rxed_65	65 to 127 byte frames received	Number of frames with 65 to 127 bytes received without error
ETHx_frames_rxed_128	128 to 255 byte frames received	Number of frames with 128 to 255 bytes received without error
ETHx_frames_rxed_256	256 to 511 byte frames Received	Number of frames with 256 to 511 bytes received without error
ETHx_frames_rxed_512	512 to 1023 byte frames received	Number of frames with 512 to 1023 bytes received without error
ETHx_frames_rxed_1024	1024 to 1518 byte frames received	Number of frames with 1024 to 1518 bytes received without error
ETHx_frames_rxed_1519	1519 to maximum byte frames received	Number of frames with 1519 or higher bytes received without error
ETHx_undersize_frames	Undersized frames received	10-bit register counting the number of frames received less than 64 bytes in length
ETHx_excessive_rx_length	Oversize frames received	10-bit register counting the number of frames received exceeding 1518 bytes
ETHx_rx_jabbers	Jabbers received	Jabbers frames received - a 10-bit register counting the number of frames received exceeding 1518 bytes
ETHx_fcs_errors	Frame check sequence errors	Frame check sequence errors - a 10-bit register counting frames that are an integral number of bytes, have bad CRC and are between 64 and 1518 bytes in length
ETHx_rx_length_errors	Length field frames errors	Length field frame errors - this 10-bit register counts the number of frames received that have a measured length shorter than that extracted from the length field (bytes 13 and 14).
ETHx_rx_symbol_errors	Receive symbol errors	Receive symbol errors - a 10-bit register counting the number of frames that had rx_er asserted during reception
ETHx_alignment_errors	Alignment errors	10-bit register counts alignment errors in received frames. Symbol error also increments this register

(table continues...)

Table 246 (continued) EMAC registers

Register	Name	Description
ETHx_rx_resource_errors	Receive resource errors	18-bit register counting the number of frames that were successfully received by the MAC but could not be copied to memory because no receive buffer was available.
ETHx_rx_overruns	Receive overrun error counter	10-bit register counts number of frames which were received correctly but were dropped due to receive overrun condition
ETHx_rx_ip_ck_erros	IP header checksum errors	8-bit register counting the number of frames discarded due to an incorrect header checksum
ETHx_rx_tcp_ck_errors	TCP checksum errors	8-bit register counting the number of frames discarded due to an TCP checksum error
ETHx_rx_udp_ck_errors	UDP checksum errors	8-bit register counting the number of frames discarded due to an incorrect UDP checksum
ETHx_auto_flushed_pkts	Receive DMA flushed packets	16-bit register counting the number of frames that are flushed from the receive SRAM based packet buffer under some specific conditions
ETHx_tsu_timer_incr_sub_nsec	Timer Increment register	1588 timer increment register sub nsec
ETHx_tsu_timer_msb_sec	Timer Seconds register	1588 timer seconds register (47 to 32 bits)
ETHx_tsu_strobe_msb_sec	Timer Sync Strobe Seconds register	1588 timer sync strobe seconds register (47 to 32 bits)
ETHx_tsu_strobe_sec	Timer Sync Strobe Seconds register	1588 timer sync strobe seconds register (31 to 0 bits)
ETHx_tsu_strobe_nsec	Timer Sync Strobe Seconds register	1588 timer sync strobe nanoseconds register
ETHx_tsu_timer_sec	Timer Seconds register	1588 timer seconds register (31 to 0 bits)
ETHx_tsu_timer_nsec	Timer Nanoseconds register	1588 timer nanoseconds register
ETHx_tsu_timer_adjust	Timer Adjust register	This register is used to adjust the value of the timer in the TSU. It allows an integral number of nanoseconds to be added or subtracted from the timer in a one-off operation. This register returns all zeros when read.
ETHx_tsu_timer_incr	1588 Timer Increment register	Timer increment register is used to set a count of nanoseconds by which the 1588 timer nanoseconds register will be incremented each clock cycle
ETHx_tsu_ptp_tx_sec	PTP event frame transmitted seconds	The register is updated with the value that the 1588 timer seconds register held when the SFD of a PTP transmit primary event crosses the MII interface

(table continues...)

Table 246 (continued) EMAC registers

Register	Name	Description
ETHx_tsu_ptp_tx_nsec	PTP event frame transmitted nanoseconds	The register is updated with the value that the 1588 timer nanoseconds register held when the SFD of a PTP transmit primary event crosses the MII interface
ETHx_tsu_ptp_rx_sec	PTP event frame received seconds	The register is updated with the value that the 1588 timer seconds register held when the SFD of a PTP receive primary event crosses the MII interface
ETHx_tsu_ptp_rx_nsec	PTP event frame received nanoseconds	The register is updated with the value that the 1588 timer nanoseconds register held when the SFD of a PTP receive primary event crosses the MII interface
ETHx_tsu_peer_tx_Sec	PTP peer event frame transmitted seconds register	The register is updated with the value that the 1588 timer seconds register held when the SFD of a PTP transmit peer event crosses the MII interface
ETHx_tsu_peer_tx_nsec	PTP Peer Event Frame Transmitted Nanoseconds register	The register is updated with the value that the 1588 timer nanoseconds register held when the SFD of a PTP transmit peer event crosses the MII interface
ETHx_tsu_peer_rx_sec	PTPPeer Event Frame Received Seconds register	The register is updated with the value that the 1588 timer seconds register held when the SFD of a PTP receive peer event crosses the MII interface
ETHx_tsu_peer_rx_nsec	PTP Peer Event Frame Received Nanoseconds register	The register is updated with the value that the 1588 timer nanoseconds register held when the SFD of a PTP receive peer event crosses the MII interface
ETHx_tx_pause_quantum1	Transmit Pause Quantum register 1	Transmit pause quantum - written with the pause quantum value for pause frame transmission of priority 2 and 3
ETHx_tx_pause_quantum2	Transmit Pause Quantum register 2	Transmit pause quantum - written with the pause quantum value for pause frame transmission of priority 4 and 6
ETHx_tx_pause_quantum3	Transmit Pause Quantum register 3	Transmit pause quantum - written with the pause quantum value for pause frame transmission of priority 7 and 8
ETHx_rx_lpi	Received LPI transitions	Count of RX LPI transitions. A count of the number of times there is a transition from receiving normal idle to receiving low-power idle

(table continues...)

Table 246 (continued) EMAC registers

Register	Name	Description
ETHx_rx_lpi_time	Received LPI time	Time in LPI. This register increments once every 16 pclk cycles when the LPI indication bit 20 is set in the receive configuration register
ETHx_tx_lpi	Transmit LPI transitions	Count of LPI transmissions. A count of the number of times the enable LPI transmission bit 20 goes from low to high in the transmit control register.
ETHx_tx_lpi_time	Transmit LPI time	Time in LPI. This register increments once every 16 pclk cycles when the LPI indication bit 20 is set in the transmit configuration register
ETHx_int_q1_status	Priority Queue Interrupt Status register	Priority queue 1 interrupt status register
ETHx_int_q2_status	Priority Queue Interrupt Status register	Priority queue 2 interrupt status register
ETHx_transmit_q1_ptr	Transmit Q1 pointer	This register holds the start address of the transmit buffers descriptor list for queue 1
ETHx_transmit_q2_ptr	Transmit Q2 pointer	This register holds the start address of the transmit buffers descriptor list for queue 2
ETHx_receive_q1_ptr	Receive Q1 pointer	This register holds the start address of the receive buffers descriptor list for queue 1
ETHx_receive_q2_ptr	Receive Q2 pointer	This register holds the start address of the receive buffers descriptor list for queue 2
ETHx_dma_rxbuf_size_q1	Receive buffer queue 1 size	DMA receive buffer size in system memory. The value defined by these bits determines the size of buffer to use in main system memory when writing received data
ETHx_dma_rxbuf_size_q2	Receive buffer queue 2 size	DMA receive buffer size in system memory. The value defined by these bits determines the size of buffer to use in main system memory when writing received data
ETHx_cbs_control	Credit Based Shaping Control register	Register is used to enable credit based shaping on high-priority queues
ETHx_cbs_idleslope_q_a	Queue A Idle Slope Configuration register	Register is used to configure idle slope for highest priority queue, which is queue 2
ETHx_cbs_idleslope_q_b	Queue B Idle Slope Configuration register	Register is used to configure idle slope for second highest priority queue, which is queue 1
ETHx_tx_bd_control	TX BD control	Transmit buffer descriptor mode configuration register

(table continues...)

Table 246 (continued) EMAC registers

Register	Name	Description
ETHx_rx_bd_control	RX BD control	Receive buffer descriptor mode configuration register
ETHx_screening_type_1_register_0 to ETHx_screening_type_1_register_15	Screening Type 1 register	Screening type 1 registers are used to allocate up to 16 priority queues to received frames based on certain block or UDP fields of incoming frames
ETHx_screening_type_2_register_0 to ETHx_screening_type_2_register_15	Screening Type 2 register	Screening type 2 registers operate independently of screening type 1 registers and offer additional match capabilities, extending the capabilities into vendor specific protocols
ETHx_tx_sched_ctrl	Transmit Scheduling Control register	This register controls the transmit scheduling algorithm the user can select for each active transmit queue. By default, all queues are initialized to fixed priority, with the top indexed queue having overall priority
ETHx_bw_rate_limit_q0toq3	Bandwidth Allocation register	This register holds the DWRR weighting value or the ETS bandwidth percentage value used by the transmit scheduler for queues 0 to 2
ETHx_int_q1_enable	Interrupt enable for Q1	At reset all interrupts are disabled. Writing a one to the relevant bit location enables the required interrupt. This register is write-only and when read will return zero
ETHx_int_q2_enable	Interrupt enable for queue 2	At reset all interrupts are disabled. Writing a one to the relevant bit location enables the required interrupt. This register is write-only and when read will return zero
ETHx_int_q1_disable	Interrupt disable for queue 1	Writing a one to the relevant bit location disables that interrupt. This register is write-only and when read will return zero.
ETHx_int_q2_disable	Interrupt disable for queue 2	Writing a one to the relevant bit location disables that interrupt. This register is write-only and when read will return zero.
ETHx_int_q1_mask	Interrupt mask register for queue 1	The interrupt mask register is a read-only register indicating which interrupts are masked.
ETHx_int_q2_mask	Interrupt mask register for queue 2	The interrupt mask register is a read-only register indicating which interrupts are masked.
ETHx_screening_type_2_ethertype_reg_0 to ETHx_screening_type_2_ethertype_reg_7	Screener Type 2 Ethertype Compare register	Screening register to compare ethertype

(table continues...)

Table 246 (continued) EMAC registers

Register	Name	Description
ETHx_type2_compare_0_word_0 to ETHx_type_compare_31_word_0	Screener Type 2 Compare register	Type 2 compare register word 0; see the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual for full description
ETHx_type2_compare_0_word_1 to ETHx_type2_compare_31_word_1	Screener Type 2 Compare register	Type 2 compare register word 1; see the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual for full description

30 Serial memory interface

The serial memory interface (SMIF) block is a master that provides a low pin count connection to off-chip (single/dual/quad/octal) SPI devices, such as EEPROM, FRAM, MRAM, or NAND in SDR or DDR mode, and HYPERBUS™ devices such as HYPERFLASH™ (NOR flash) and HYPERRAM™ (PSRAM and pseudo static RAM).

This chapter explains the features, implementation, and operational modes of SMIF.

30.1 Features

- SPI or HYPERBUS™ master functionality only
- HYPERBUS™ protocol
- SPI protocol
  - SPI mode 0 only, with configurable MISO sampling timing
  - Supports single/dual/quad/octal SPI
  - Supports dual-quad SPI mode
  - Supports single data rate (SDR) and dual data rate (DDR) transfers
- Memory device
  - Supports overall device capacity in the range of 64 KB to 4 GB in power of two multiples
  - Supports configurable external device capacities
  - Supports two external memory devices
- Memory mapped I/O (MMIO) operation mode
- XIP mode
  - eXecute-in-place (XIP) operation mode for both read and write accesses
  - XIP mode supports on-the-fly encryption and decryption
  - XIP operation mode via AHB interface for CM0 and AXI interface for CM7 core
  - Supports up to four outstanding transactions
- Memory interface logic
  - Supports stalling of SPI and HYPERBUS™ transfers to address back pressure on FIFOs
  - Supports an asynchronous SPI/HYPERBUS™ transmit and receive interface clock
  - Supports read-write-data-strobe (RWDS)
  - Supports multiple interface receive clocks
  - Supports flexible external SPI memory devices data signal connections
  - Independent SPI interface transmitter clock from PLL/FLL
  - SPI interface logic supports flexible external memory devices data signal connections

30.2 System diagram

The SMIF allows the XMC7000 device to connect to off-chip NOR/NAND flash, SRAM, FRAM, nonvolatile SRAM, or DRAM (using single/dual/quad SPI or HYPERBUS™ interface) memories.

Figure 292 shows how to connect to external serial devices; up to two SPI/HYPERBUS™ devices can be connected.

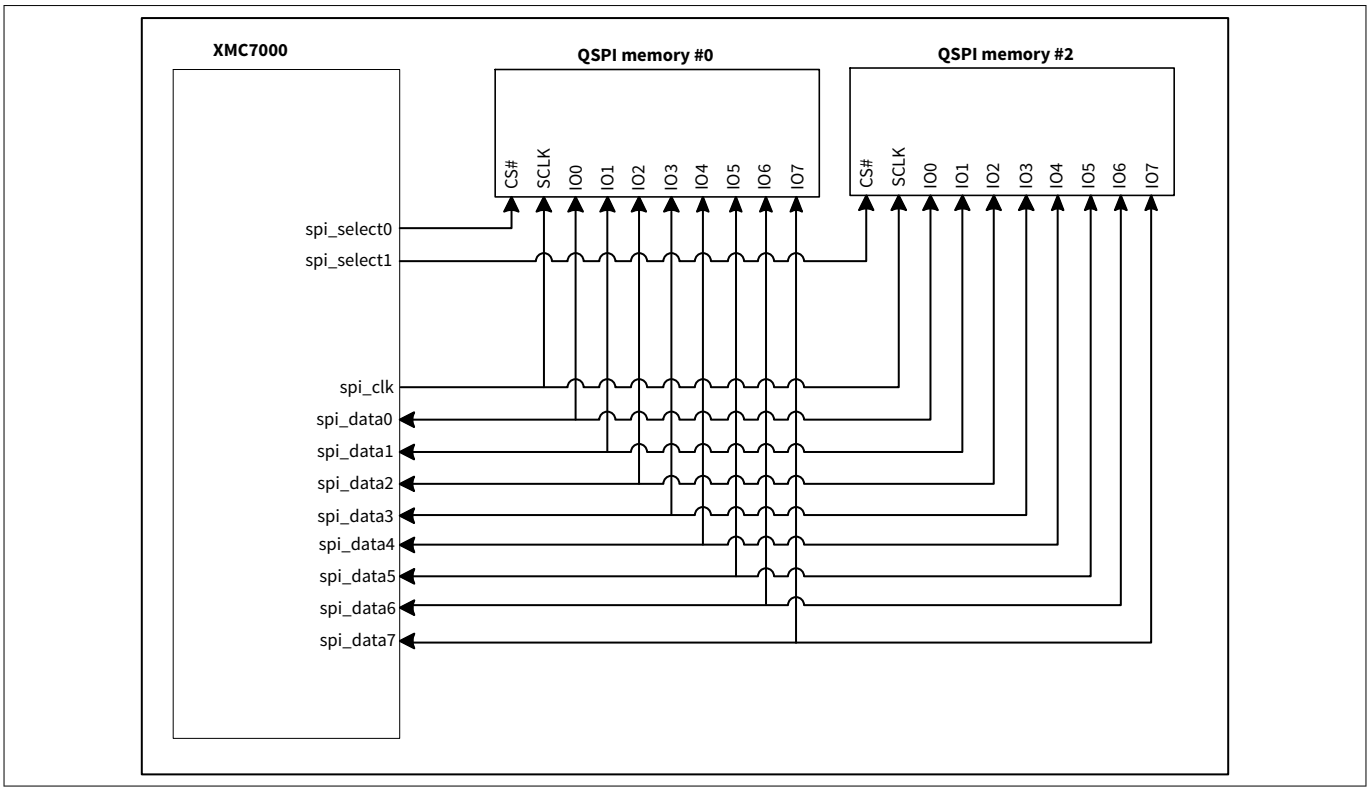


Figure 292 QSPI device example system diagram

30.3 Block diagram

Figure 293 gives a high-level overview of the SMIF. The bottom part of the figure shows the SPI signal connections to the I/O subsystem (IOSS). The top part of the figure shows the interface to the CM7 core (one fast AXI interface and one slow AHB interface) and the AHB-Lite slave interface to the peripheral group. CM7 uses the fast AXI interface.

The memory interface logic supports an asynchronous interface clock (clk_if) from which the interface transmit clock (clk_if_tx) and the interface receive clock (clk_if_rx) are derived.

Note that each XIP AHB-Lite interface has a dedicated cache. Cache coherency is not supported by the hardware. For example, an XIP interface 0 write to an address in the XIP interface 0 cache invalidates the associated cache subsector in the XIP interface 0 cache, but not in the XIP interface 1 cache.

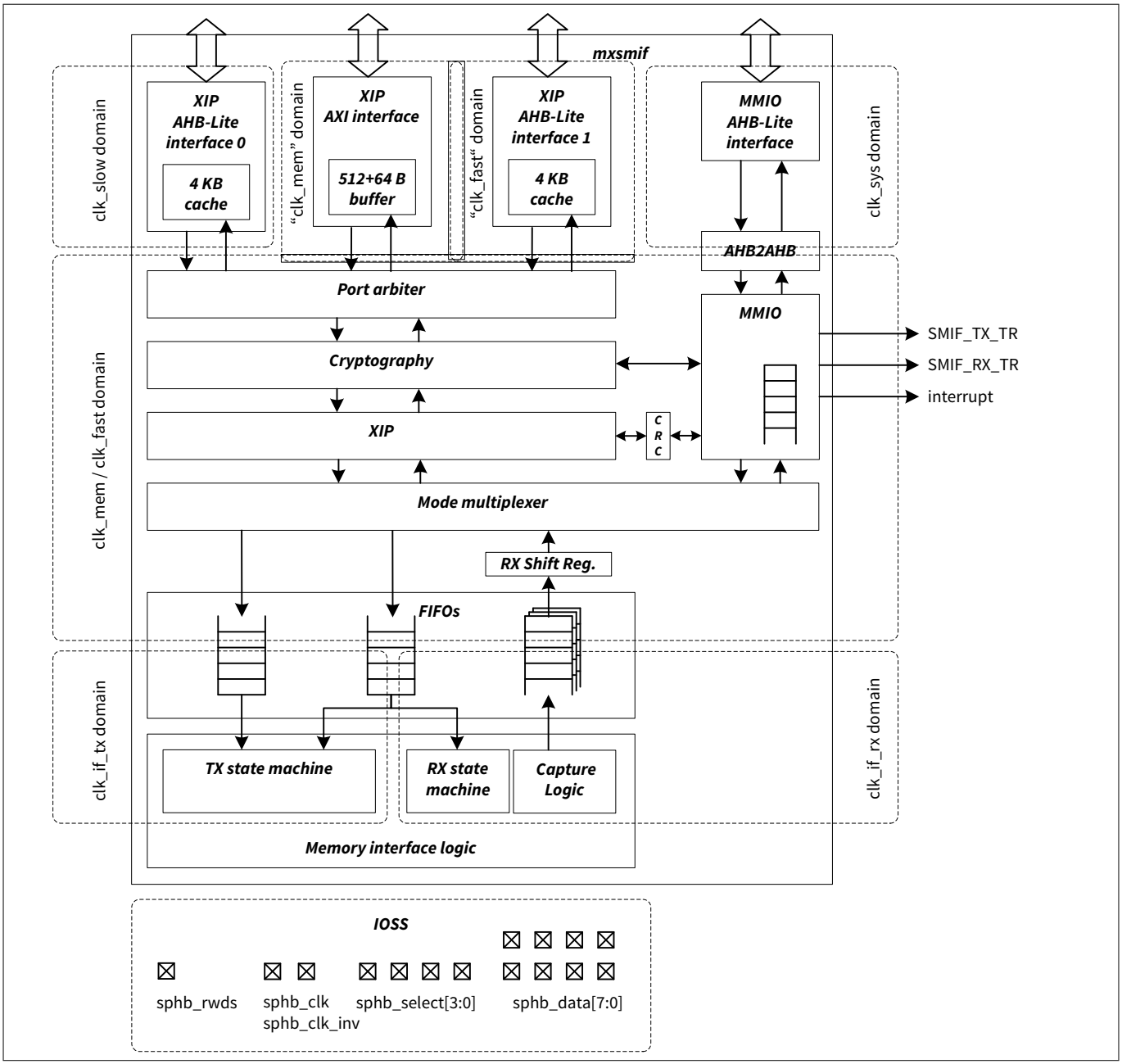


Figure 293 High-level block diagram of mxsmif (SMIF)

***Note**: The inverted clock signal 'sphb_clk_inv' is generated by a special I/O cell and not inside the SMIF.*

30.3.1 Clocks

The SMIF uses four different clock domains:

- **CLK_MEM**: It clocks the CPUSS fast infrastructure and is a divided version of CLK_HF. CLK_MEM is used for the XIP AXI interface in the SMIF configuration CM7 core. It is also used for the shared XIP-related blocks (port arbiter, cryptography, XIP, and memory interface TX and RX FIFOs).
- **CLK_SLOW**: This is the source clock for the Cortex®-M0+. It is a divided version of CLK_MEM. CLK_SLOW is used for the XIP AHB-Lite interface.

- **CLK_SYS**: This clock is used as the interface clock of the MMIO interface. However, the MMIO registers are clocked by CLK_MEM (same clock domain as the blocks where MMIO registers are used or depend on). CLK_SYS is a gated clock, which can be synchronous or asynchronous to CLK_MEM.
- **CLK_IF**: Memory interface clock signals CLK_IF and sphb_clk_in are asynchronous to the signals in the other groups. These clocks are used for the SPI/HYPERBUS™ interface logic. To generate the spihb_clk_out memory clock, a divided by 2 CLK_IF is used.

30.4 Functional description

This section describes the basic functions of SMIF.

30.4.1 Operating modes

The SMIF has the following interfaces:

- An AHB-Lite interface to access the MMIO registers.
- An AHB-Lite and an AXI interface to support XIP.

All interfaces provide access to external memory devices. At any time, either the MMIO AHB-Lite interface or the two XIP interfaces have access to the SPI interface logic and external memory devices. The operation mode is specified by XIP_MODE in the CTL register. The operation mode should not be modified when the SMIF is busy, indicated by the BUSY bit in the STATUS register.

In the MMIO AHB-Lite interface, access is supported through software writes to transmit (TX) FIFOs and software reads from a receive (RX) FIFO. The FIFOs are mapped on MMIO registers. This interface provides the flexibility to implement any SPI device transfer. For example, the SPI device transfers to set up, program, or erase the external memory devices.

In a XIP AHB-Lite interface, access is supported through XIP: AHB-Lite read and write transfers are automatically translated (by the hardware) in SPI device read and write transfers. This interface provides efficient implementation of SPI device read and write transfers, but does not support other types of SPI device transfers. To improve XIP performance, the XIP AHB-Lite interface has a 4-KB cache.

As mentioned, MMIO mode and XIP mode are mutually exclusive. The operation modes share TX and RX FIFOs and SPI interface logic. In MMIO mode, the TX and RX FIFOs are mapped on MMIO registers and under software control. In XIP mode, the TX and RX FIFOs are under hardware control. The SPI interface logic is controlled through the TX and RX FIFOs and is agnostic of the operation mode.

30.4.1.1 MMIO_MODE

The MMIO mode can be activated by writing ‘0’ to the XIP_MODE bit in the CTL register. The software generates SPI or HYPERBUS™ (HB) transfers by accessing the TX FIFOs and RX FIFO. The TX FIFOs are write-accessible and read accesses are done from the RX FIFO. The TX command FIFO has formatted commands (TX, TX_COUNT, RX_COUNT, DUMMY_COUNT) that are described in the memory map.

The software should ensure that it generates correct memory transfers and access the FIFOs correctly. For example, if a memory transfer is generated to read four bytes from a memory device, software should read the four bytes from the RX data FIFO. Similarly, if a memory transfer is generated to write four bytes to a memory device, the software should write the four bytes to the TX command FIFO or TX data FIFO.

Incorrect software behavior can lock up the memory interface. For example, a memory transfer to read 32 bytes from a memory device, without the software reading the RX data FIFO will lock up the memory transfer as the memory interface cannot provide more than eight bytes to the RX data FIFO (the RX data FIFO has eight entries). This would prevent any successive memory transfers from taking place. Note that a locked memory transfer due to TX or RX FIFO states is still compliant to the memory bus protocol (but undesirable): the SPI or HB protocol allows shutting down the interface clock SPIHB_CLK in the middle of a memory transfer.

30.4.1.2 XIP_MODE

If the XIP_MODE bit is set to ‘1’ in the CTL register, the SMIF is in XIP mode. The hardware automatically generates (without software intervention) memory transfers by accessing the TX FIFOs and RX FIFO. The hardware only supports memory read and write transfers.

- Hardware generates memory read transfers for AHB-Lite or AXI read transfers (that is, only for AHB-Lite read transfers that miss in the cache or AXI read transfers).
- Hardware generates memory write transfers for AHB-Lite or AXI write transfers.

This is done in the XIP block, which:

- Translates read or write transfer requests from the AHB-Lite or AXI interfaces to commands in the TX command FIFO
- Sends/receives data to/from the TX/RX data FIFOs

As different memory devices support different types of memory read and write transfer, it is necessary to provide the hardware with device specifics, such that it can perform the automatic translations. To this end, each memory device has a set of MMIO registers that specify its memory read and write transfers. This specification includes:

- Presence and value of the SPI or HB command byte
- Number of address bytes
- Presence and value of the mode byte
- Number of dummy cycles

In addition, the data transfer widths and data transfer mode (SDR or DDR) are specified.

The XIP interface logic produces an AHB-Lite/AXI bus error under the following conditions:

- The SMIF is disabled (ENABLED bit is set to ‘0’ in CTL register).
- The SMIF is not in XIP_MODE (XIP_MODE bit is set to ‘0’ in CTL register).
- The transfer request is not in a memory region.
- The transfer is a write and the identified memory region does not support writes (WR_EN bit is set to '0' in CTL register).
- In XIP mode (XIP_MODE bit is set to '1' in CTL register) and Dual-Quad SPI mode (DIV2 bit is set to '1' in the ADDR_CTL register), the transfer address of a write access is not a multiple of 2.
- In XIP mode (XIP_MODE bit is set to '1' in CTL register) and Dual-Quad SPI mode (DIV2 bit is set to '1' in ADD_CTL register), the transfer size of a write access is not a multiple of 2.
- In XIP mode (XIP_MODE bit is set to '1' in CTL register) and Octal SPI DDR mode or HYPERBUS™ mode, the transfer address of a write access is not a multiple of 2 and memory write byte masking is not supported (RWDS_EN bit is set to '0' in WR_DUMMY_CTL register).
- In XIP mode (XIP_MODE is set to '1' in CTL register) and Octal SPI DDR mode or HYPERBUS™ mode, the transfer size of a write access is not a multiple of 2 and memory write byte masking is not supported (RWDS_EN bit is set to '0' in WR_DUMMY_CTL register).

30.4.1.3 Continuous transfer merging

To improve performance of multiple linear continuous transfers (subsequent transfer starts at address following the final address of the previous transfer), the transfers can be merged to a single transfer at the memory interface. This is especially useful to improve XIP performance over the AXI interface, which splits longer transactions into multiple transfer requests of 16 byte each and allows it to merge these split transfers back to a single transfer at the memory interface. This avoids the overhead of multiple command, address, mode, and especially dummy (latency) cycles. However, not only split AXI transactions, but also sequential AXI or AHB transactions can be merged.

Continuous transfer merging can be done in MMIO and XIP modes.

- MMIO mode: In MMIO mode, this is under full software control. For each TX, TX_COUNT, or RX_COUNT command in the TX command FIFO it can be specified whether it is the last command, That is, if the memory is deselected after the end of that command processing. This way TX, TX_COUNT, or RX_COUNT commands can be executed in a sequence without deselecting the memory.
- XIP mode: In XIP mode, the continuous transfer merging can be enabled or disabled per memory device. If disabled, the XIP block sets the last bit in a TX_COUNT or RX_COUNT causing the memory interface logic to deselect the memory after the transfer is complete. The transfer requests (output from the AHB/AXI interface) and the sequence in the TX command FIFO generated by the XIP block are illustrated in the following figure.

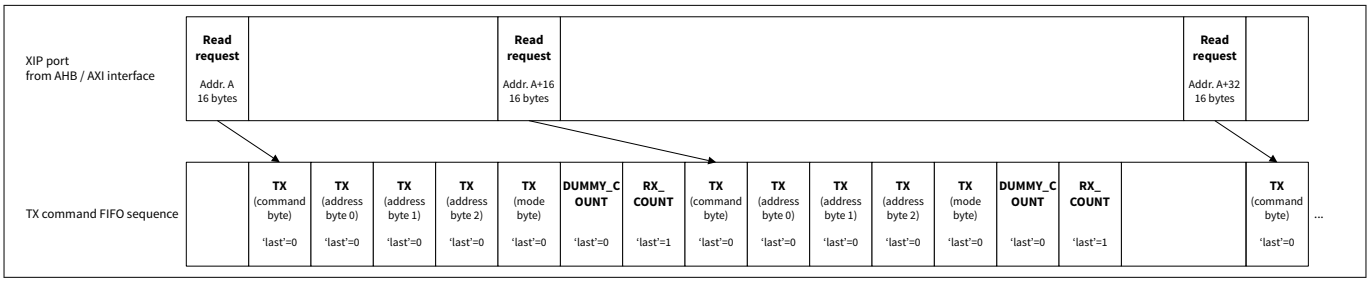


Figure 294 Read sequence with continuous merging disabled

If enabled, the XIP block does not set the last bit in a TX_COUNT or RX_COUNT command for the data phase of a transfer. This causes the memory interface logic to keep the memory device selected (select signals stays active 0 while no clocks are generated).

When a new transfer of the same (read or write) type as the previous is requested from one of the XIP interfaces and its start address is a continuation of the last data phase of the previous transfer, then the XIP interface continues the transfer with a TX_COUNT or RX_COUNT command. This skips the overhead of new command, address, mode, and dummy cycles as illustrated in the following figure.

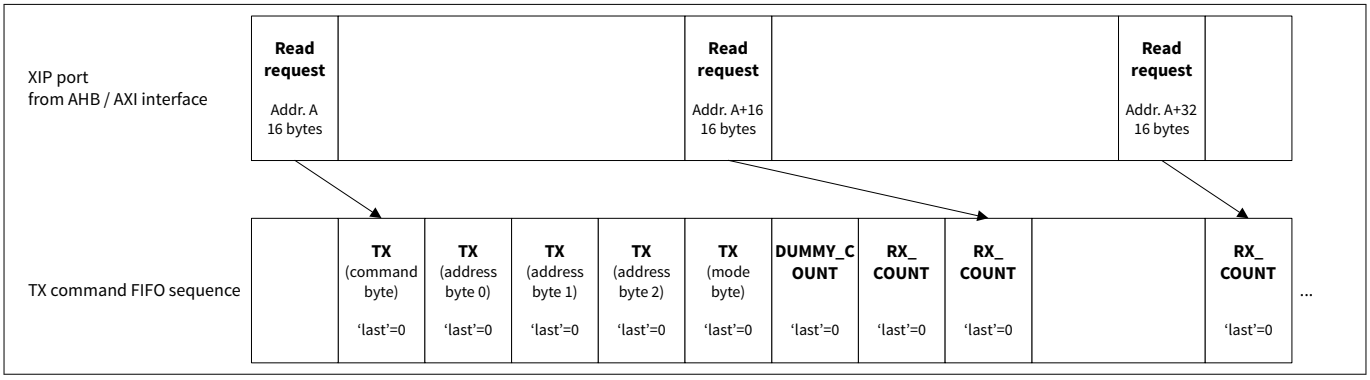


Figure 295 Read sequence with continuous merging enabled - merging always possible

When a new transfer is requested, which is of a different (read or write) type or its start address is not a continuation of the previous one, then the XIP interface generates a DESELECT command and later starts the new transfer again with the required TX commands for command, address, mode, and/or dummy cycles as illustrated for the last transfer in the following figure.

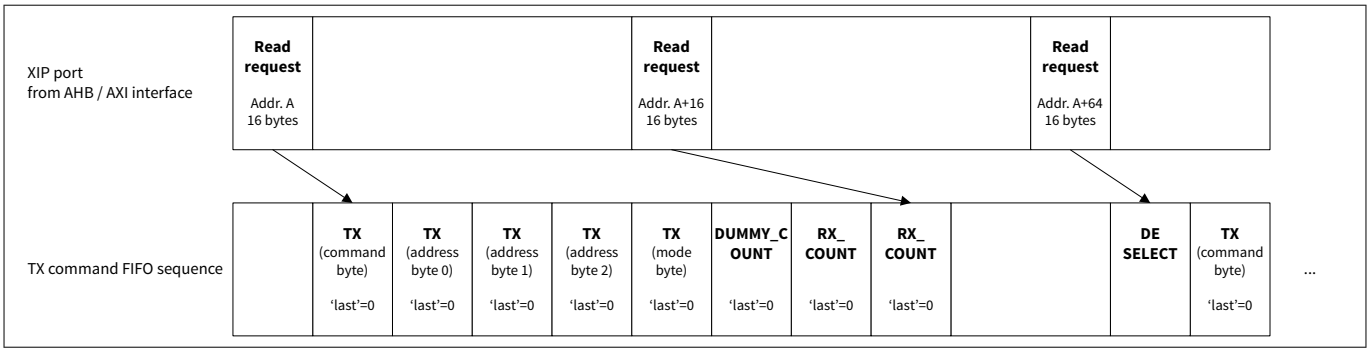


Figure 296 Read sequence with continuous merging enabled - merging possible once

To avoid keeping the memory selected for a long time while not doing any transfer (which may cause a higher power consumption) per memory device, a continuous transfer merge timeout in CLK_mem cycles can be specified. This timeout value can be 1, 16, 256, 4096, or 65536 CLK_mem cycles. If the timeout occurs before the next transfer is requested the memory device is deselected and a later transfer always starts with command, address, mode, and/or dummy cycles. This is illustrated in the following figure.

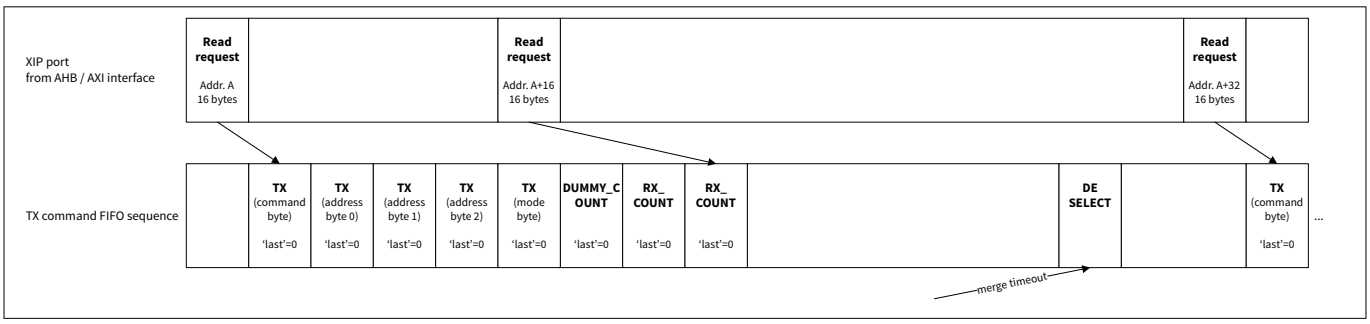


Figure 297 Read sequence with continuous merging enabled - merging always possible, but continuous transfer merge timeout occurs

The XIP block contains a 16-bit counter, which starts counting when the last byte of the previous transfer has been written to the TX data FIFO or read from the RX data FIFO (data FIFOs not shown in Figure 297). Note that due to the asynchronous clock domain transfer of the commands in the TX command FIFO, the actual remaining time the memory is selected can differ by a few cycles from the specified timeout value.

Additional to the continuous merge timeout there is also a total transfer timeout. This is used for RAM devices requiring refresh cycles. The value needs to be derived from the RAMs maximum transaction length time (t_{CMS}) minus the time of transferring 16-byte data block (data granularity of the XIP ports).

The total timeout period counting is done in the XIP block using CLK_mem cycles. It starts when the first command of a new (not merged) transfer is written to the TX command FIFO causing the interface logic to select the memory. After this period the memory device is deselected as shown in the following figure.

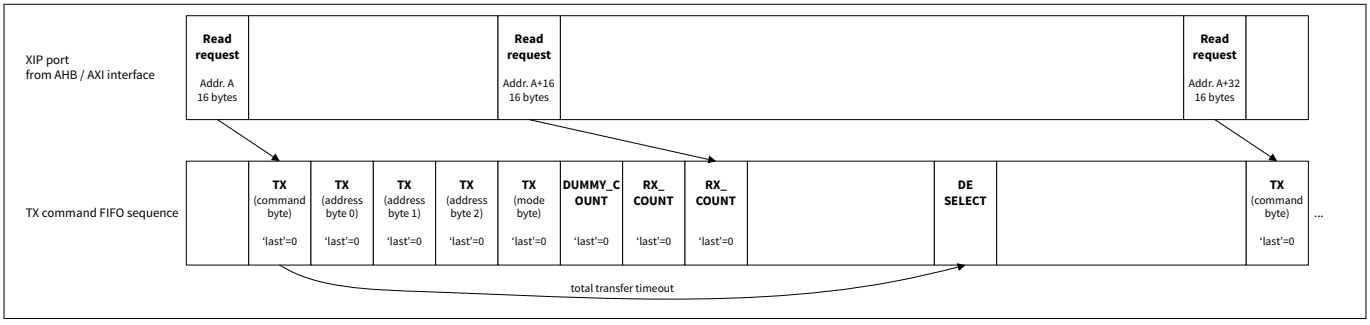


Figure 298 Read sequence with continuous merging enabled - merging always possible, but total transfer merge timeout occurs

30.4.2 Off-chip interfaces

30.4.2.1 Clock polarity and phase

SMIF acts as a master for SPI and HYPERBUS™ applications. SPI requires the definition of clock polarity and phase (while HYPERBUS™ does not). In SPI SDR (single data rate) mode, SMIF supports a single clock polarity and phase configuration:

- Clock polarity (CPOL) is 0: the base value of the clock SPIHB_CLK is “0”.
- Clock phase (CPHA) is 0: driving of data is on the falling edge of the clock SPIHB_CLK; capturing of data is specified by CLOCK_IF_RX_SEL bit in the CTL register.

The above configuration is also known as SPI configuration 0 and is supported by SPI memory devices.

30.4.2.2 Specifying memory devices

SMIF requires that the memory devices are defined. It supports up to two memory devices. Each device is defined by a set of MMIO registers. The MMIO definition includes:

- The device base address and capacity. The MMIO ADDR register specifies the memory device's base address in the XMC7000 address space and the MMIO MASK register specifies the memory device size and capacity. If a memory device is not present or is disabled, the MMIO ADDR and MASK registers specify a memory device with 0 B capacity. Typically, the device address regions in the XMC7000 address space are non-overlapping (except dual-quad configuration), to ensure that the activation of select signals is mutually exclusive.
- The device data signal connections (as described in the next section).
- The definition of a read transfer to support XIP mode.
- The definition of a write transfer to support XIP mode.

Each memory device uses a dedicated device select signal; memory device 0 uses SPIHB_SEL[0], memory device 1 uses SPIHB_SEL[1], and so on. In other words, there is a fixed, one-to-one connection between memory device, MMIO register set, and select signal connection. For example, memory device 0 uses MMIO register set ‘0’ and select signal SPIHB_SEL[0].

In XIP mode, the XIP AHB-Lite bus transfer address is compared with the device region. If the address is within the device region, the device select signal is activated. If a XIP AHB-Lite bus transfer address is within multiple regions (this is possible if the device regions overlap in dual-quad configuration only), all associated device select signals are activated. This overlap enables XIP in dual-quad SPI mode: the command, address, and mode byte can be driven to two quad SPI devices simultaneously.

- Dual-quad SPI

  In XIP mode, dual-quad SPI mode requires the DIV2 field in the ADDR_CTL register of the selected memory devices to be set to '1'. When this field is '1', the transfer address is divided by 2 and the ‘divided by 2’ address is provided to the memory devices. Each memory device contributes a 4-bit nibble for each 8-bit byte. However, both memory devices are quad SPI memories with a byte interface. Therefore, the memory transfer size must be a multiple of 2 and the memory transfer address must be 2-byte aligned (must also be a multiple of 2).

  When the XIP transfer size or address for a read access is not a multiple of 2, then the memory transfer size is extended and/or the memory transfer address is aligned as needed. Then, only the relevant read byte(s) are used and the non-relevant byte(s) are discarded.

  Examples:

  - An XIP read access to address offset ‘0’ with a length of 1 byte is extended to a memory read access to address offset ‘0’ with a length of 2 bytes.

- An XIP read access to address offset ‘1’ with a length of 1 byte is extended to a memory read access to address offset ‘0’ with a length of 2 bytes.
- An XIP read access to address offset ‘1’ with a length of 2 bytes is extended to a memory read access to address offset ‘0’ with a length of 4 bytes.

When the XIP transfer size or address for a write access is not a multiple of 2, then no memory transfer is done and an XIP_ALIGNMENT_ERROR interrupt cause is set.

The XIP_ALIGNMENT_ERROR interrupt cause is set under the following conditions (in XIP mode and when ADDR_CTL.DIV2 is '1'):

- The transfer address of a write access is not a multiple of 2.
- The transfer size of a write access is not a multiple of 2.

- Octal SPI DDR

  In XIP mode, for octal SPI DDR configuration the SIZE3 field in the ADDR_CTL register needs to be set to ‘1’ or ‘3’ causing a 2-byte or 4-byte address generation. The DIV2 field in the ADDR_CTL register of the selected memory device needs to be set to '0' if the memory expects a byte address (typical case), but can be set to '1' if the memory expects a 16-bit word address. However, the octal SPI memory in DDR configuration has a 16- bit word-based protocol; that is, the smallest addressable item in octal SPI DDR configuration is a 2-byte word. Therefore, the memory transfer size must be a multiple of 2 and the memory transfer byte address must be 2- byte aligned (must also be a multiple of 2).

  When the XIP transfer size or address for a read access is not a multiple of 2, then the memory transfer size is extended and/or the memory transfer address is aligned as needed. Then only the relevant read bytes are used and the non-relevant bytes are discarded (examples shown above for dual-quad SPI configuration also apply here).

  When the XIP transfer size or address for a write access is not a multiple of 2, then the behavior depends on the memory write byte masking capability. If the memory supports write byte masking by driving RWDS (DQS) = '1' (specified by setting RWDS_EN bit to '1' in WR_DUMMY_CTL register) then the memory transfer size is extended and/or the memory transfer address is aligned (same as for read accesses) and the RWDS (DQS) signal is driven to '1' for the non-relevant byte(s) to avoid that they get written. If the memory does not support write byte masking (specified by setting RWDS_EN bit to '0' in the WR_DUMMY_CTL register), then no memory transfer is done and an XIP_ALIGNMENT_ERROR interrupt cause is set.

  The XIP_ALIGNMENT_ERROR interrupt cause is set under the following conditions (in XIP mode):

  - The transfer address of a write access is not a multiple of 2 and memory write byte masking is not supported (RWDS_EN bit is set to '0' in the WR_DUMMY_CTL register).
  - The transfer size of a write access is not a multiple of 2 and memory write byte masking is not supported (RWDS_EN bit is set to '0' in the WR_DUMMY_CTL register).

- HYPERBUS™

  In XIP mode, for HYPERBUS™ configuration the SIZE3 field in the ADDR_CTL register needs to be set to ‘7’ causing a 5-byte address generation with HYPERBUS™ protocol (including reserved bits in transaction address fields). The DIV2 field in the ADDR_CTL register of the selected memory device is ignored (does not matter). However, because the HYPERBUS™ is a 16-bit word-based protocol, the XIP byte address is always divided by 2 to generate a HYPERBUS™ word address. The smallest addressable item of a HYPERBUS™ memory is a 2-byte word. Therefore, the memory transfer size must be a multiple of 2 and the memory transfer byte address must be 2-byte aligned (must also be a multiple of 2).

  When the XIP transfer size or address for a read access is not a multiple of 2, then the memory transfer size is extended and/or the memory transfer address is aligned as needed. Then, only the relevant read bytes are used and the non-relevant bytes are discarded (examples shown above for dual-quad SPI configuration also apply here).

  When the XIP transfer size or address for a write access is not a multiple of 2, then the memory transfer size is extended and/or the memory transfer address is aligned (same as for read accesses) and the RWDS signal is driven to '1' for the non-relevant bytes to avoid them from being written. In HYPERBUS™ configuration

RWDS_EN bit in the WR_DUMMY_CTL register must be set to '1' (indicating the byte write masking capability of HYPERBUS™ memories); otherwise, an XIP_ALIGNMENT_ERROR interrupt cause is set for unaligned write accesses (same as for octal SPI DDR configuration).

30.4.2.3 Connecting SPI memory devices

Memory device I/O signals (SCK, CS#, SI/IO0, SO/IO1, IO2, IO3, IO4, IO5, IO6, IO7) are connected to the SMIF I/O signals (SPIHB_CLK, SPIHB_SEL[1:0] and SPIHB_DATA[7:0]). Not all memory devices provide the same number of I/O signals.

Table 247 Memory device I/O signals

Memory device	I/O signals
Single SPI memory	SCK, CS#, SI, SO. This memory device has two data signals (SI and SO).
Dual SPI memory	SCK, CS#, IO0, IO1. This memory device has two data signals (IO0, IO1).
Quad SPI memory	SCK, CS#, IO0, IO1, IO2, IO3. This memory device has four data signals (IO0, IO1, IO2, IO3).
Octal SPI memory	SCK, CS#, IO0, IO1, IO2, IO3, IO4, IO5, IO6, IO7. This memory device has eight data signals (IO0, IO1, IO2, IO3, IO4, IO5, IO6, IO7).

Table 247 illustrates that each memory has a single clock signal SCK, a single (low active) select signal (CS#), and multiple data signals (IO0, IO1, …).

Each memory device has a fixed select signal connection (to SPIHB_SEL[1:0]).

Each memory device has programmable data signal connections (to SPIHB_DATA[7:0]): the MMIO DATA_SEL[1:0] field in CTLi register specifies how device data signals are connected. This information is used by SMIF to drive data on the correct SPIHB_DATA[] outputs and capture data from the correct SPIHB_DATA[] inputs. If multiple device select signals are activated, the same data is driven to all selected devices simultaneously.

Not all data signal connections are legal/supported. Supported connections are dependent on the type of memory device.

Table 248 Data signal connections.

CTL.DATA_SEL[1:0]	Single SPI device	Dual SPI device	Quad SPI device	Octal SPI device
0	SPIHB_DATA[0]=SI SPIHB_DATA[1]=SO	SPIHB_DATA[0]=IO0 SPIHB_DATA[1]=IO1	SPIHB_DATA[0]=IO0 … SPIHB_DATA[3]=IO3	SPIHB_DATA[0]=IO0 … SPIHB_DATA[7]=IO7
1	SPIHB_DATA[2]=SI SPIHB_DATA[3]=SO	SPIHB_DATA[2]=IO0 SPIHB_DATA[3]=IO1	Illegal	Illegal
2	SPIHB_DATA[4]=SI SPIHB_DATA[5]=SO	SPIHB_DATA[4]=IO0 SPIHB_DATA[5]=IO1	SPIHB_DATA[4]=IO0 … SPIHB_DATA[7]=IO3	Illegal
3	SPIHB_DATA[6] =SI SPIHB_DATA[7]=SO	SPIHB_DATA[6]=IO0 SPIHB_DATA[7]=IO1	Illegal	Illegal

Memory devices can:

- Use shared data signal connections.
- Use dedicated data signal connections. This reduces the load on the data lines, which allows for faster signal level changes. This in turn allows for a faster I/O interface.

Note that dual-quad SPI mode requires dedicated data signals to enable read and/or write data transfer from and to two quad SPI devices simultaneously.

Figure 299 illustrates memory device 0, which is a single SPI memory with data signals connections to SPIHB_DATA[1:0].

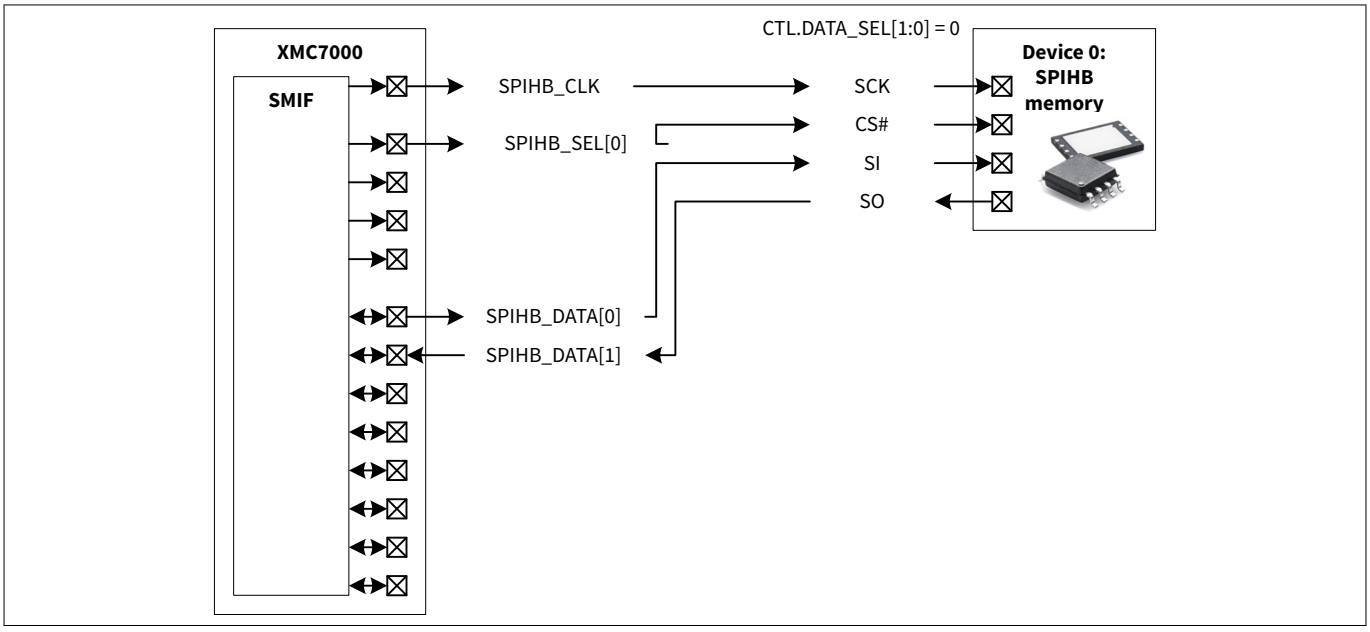


Figure 299 Single SPI device 0 connected to SPIHB_DATA[1:0]

The XMC7000 pin layout for example, may make it desirable to connect a memory device to specific data lines. Figure 300 illustrates memory device 0, which is a single SPI memory with data signals connections to SPIHB_DATA[7:6].

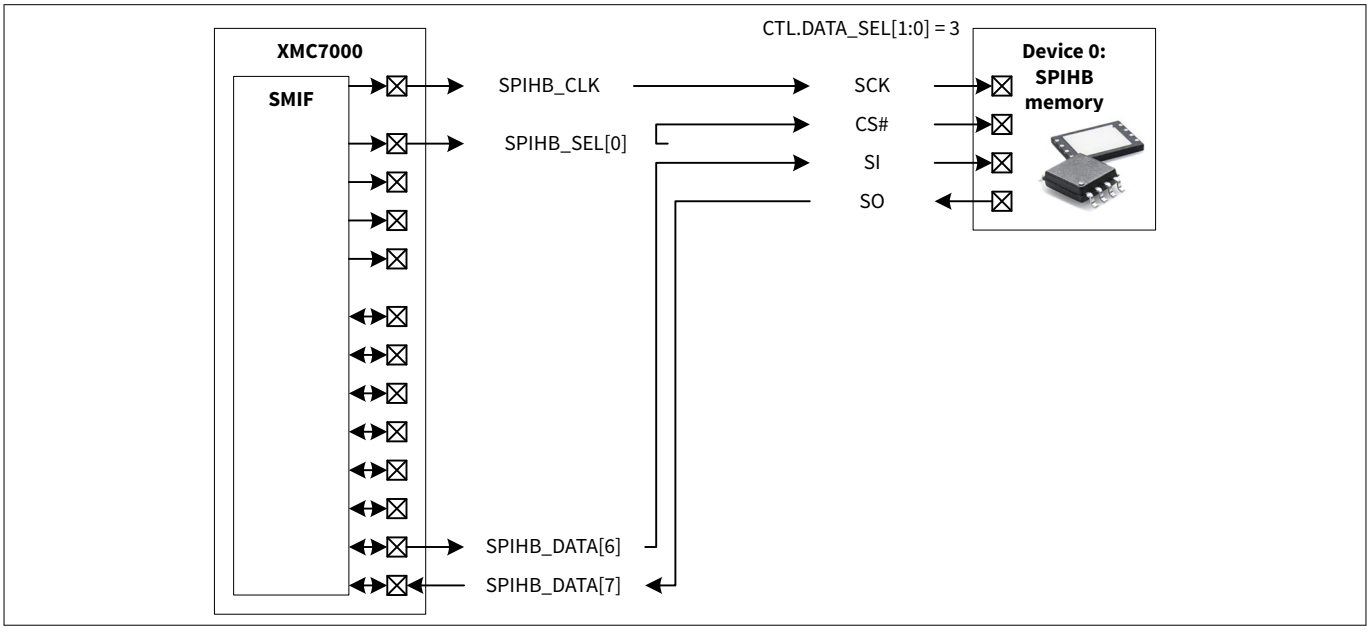


Figure 300 Single SPI device 0 connected to SPIHB_DATA[7:6]

Figure 301 illustrates memory device 1, which is a single SPI memory with data signals connections to SPIHB_DATA[5:4].

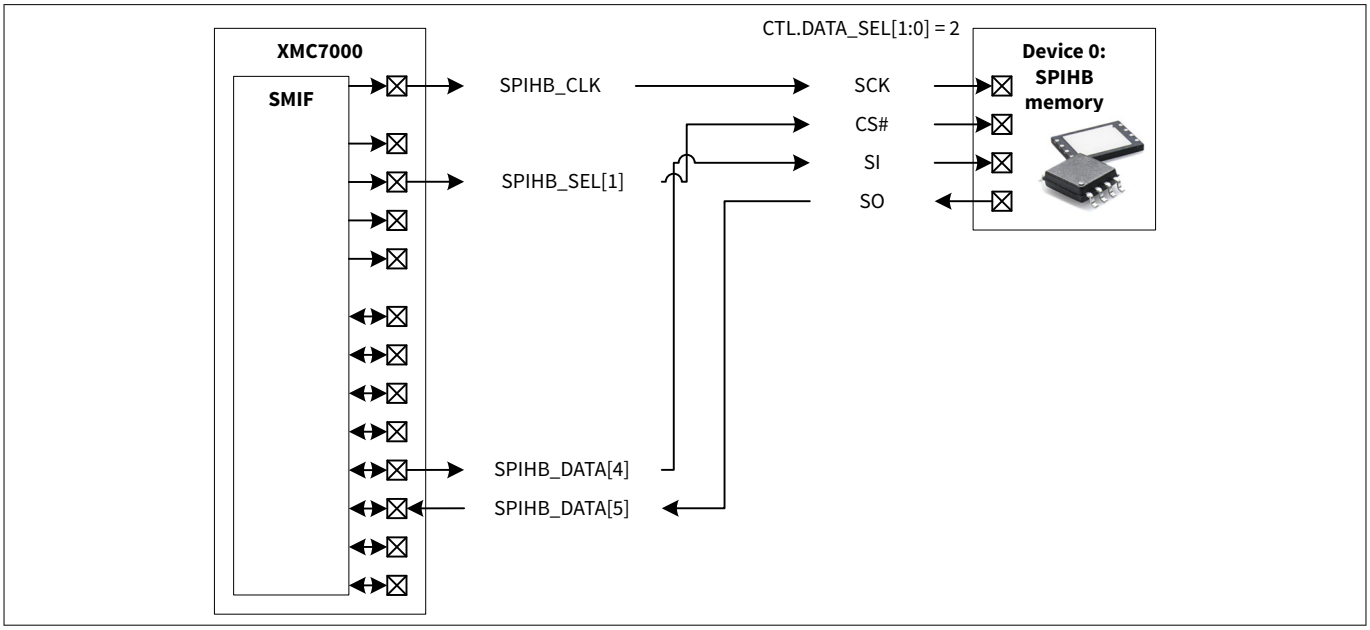


Figure 301 Single SPI device 0 connected to SPIHB_DATA[5:4]

Figure 302 illustrates memory devices 0 and 1, both of which are single SPI memories. Each device uses dedicated data signal connections. The device address regions in the XMC7000 address space must be non-overlapping to ensure that the activation of SPIHB_SEL[0] and SPIHB_SEL[1] are mutually exclusive.

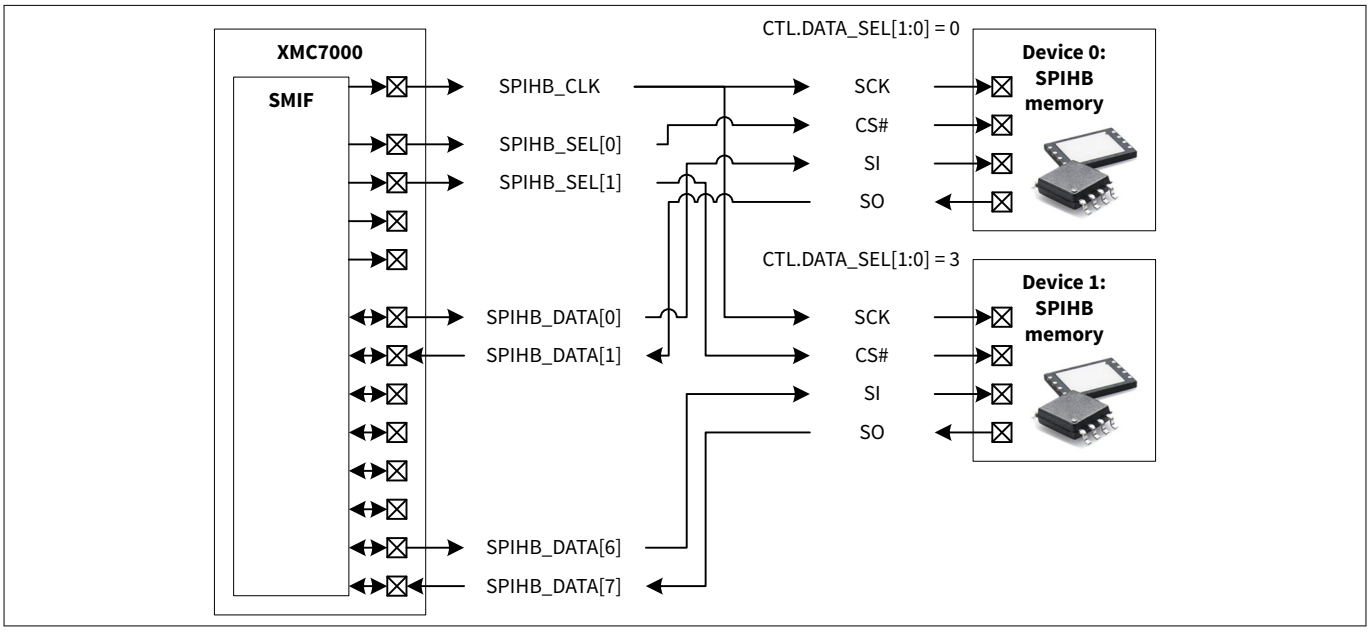


Figure 302 Single SPI device 0 to SPIHB_DATA[1:0], single SPI device 1 connected to SPIHB_DATA[7:6]

Figure 303 illustrates memory devices 0 and 1, both of which are single SPI memories. Both devices use shared data signal connections. The device address regions in the XMC7000 address space must be non-overlapping to ensure that the activation of SPIHB_SEL[0] and SPIHB_SEL[1] are mutually exclusive. Note that this solution increases the load on the data lines, which may result in a slower I/O interface.

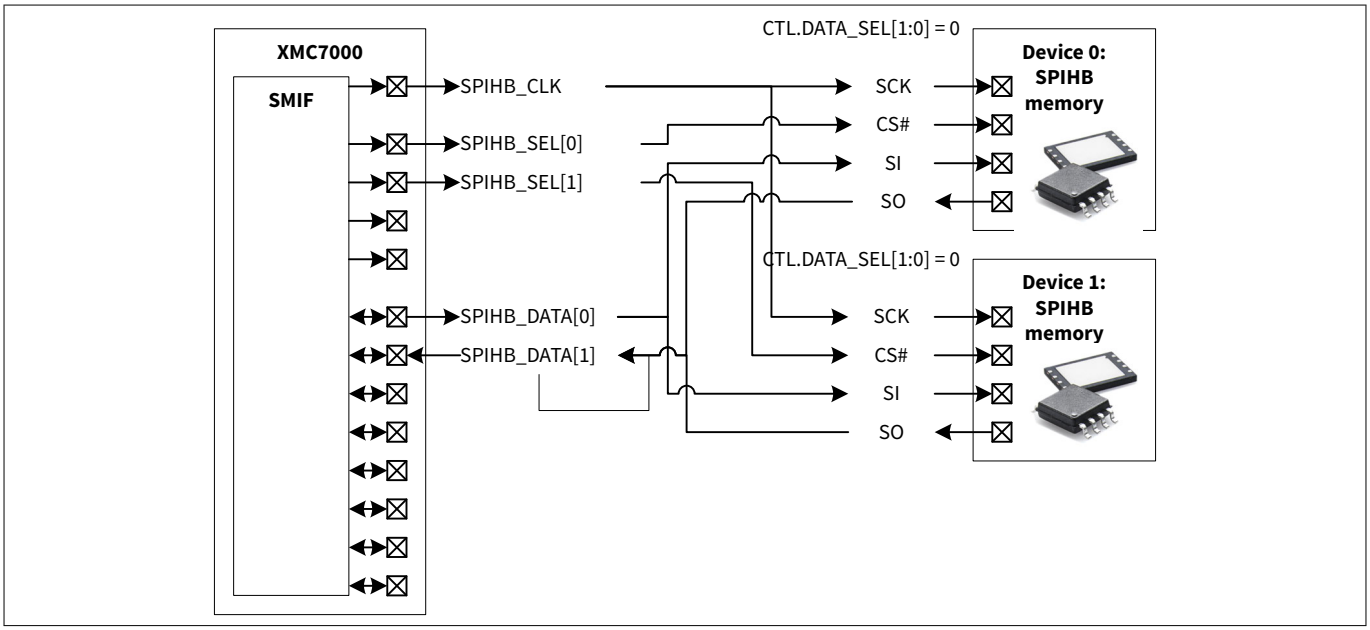


Figure 303 Single SPI device 0 to SPIHB_DATA[1:0], single SPI device 1 connected to SPIHB_DATA[1:0]

Figure 304 illustrates memory device 0, which is a quad SPI memory with data signals connections to SPIHB_DATA[7:4].

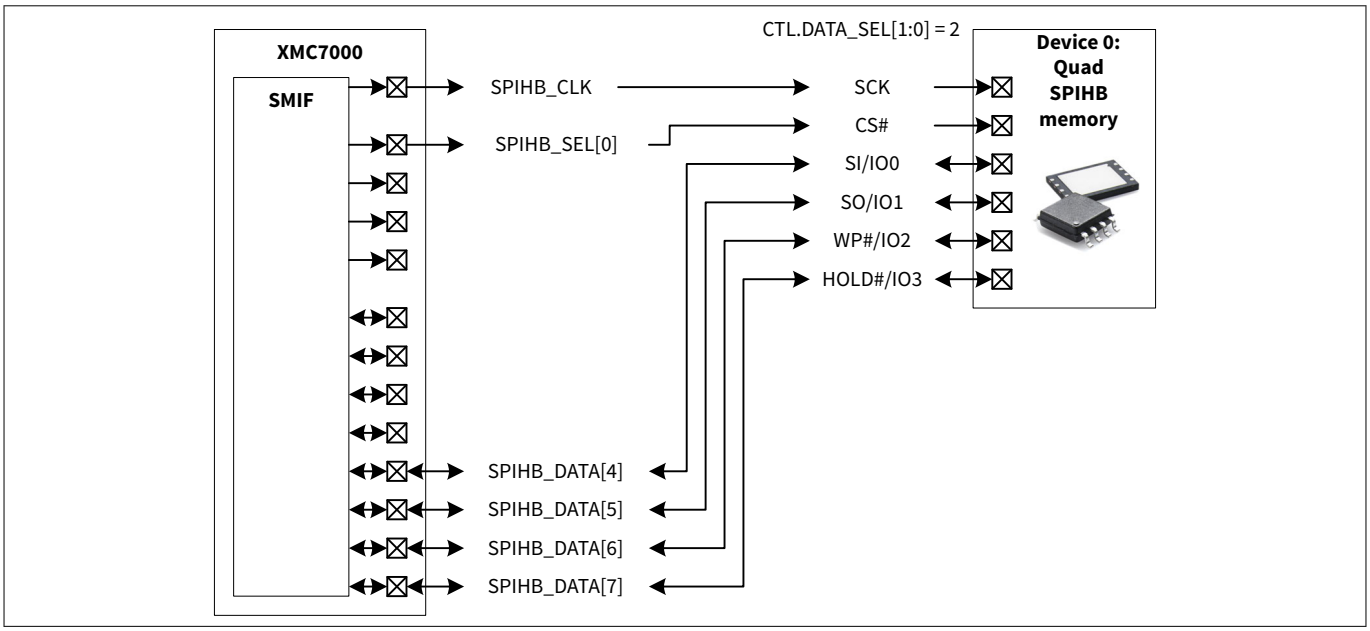


Figure 304 Quad SPI device 0 connected to SPIHB_DATA[7:4]

Figure 305 illustrates memory devices 0 and 1, device 0 is single SPI memory and device 1 is quad SPI memory. Each device uses dedicated data signal connections. The device address regions in the XMC7000 address space must be non-overlapping to ensure that the activation of SPIHB_SEL[0] and SPIHB_SEL[1] are mutually exclusive.

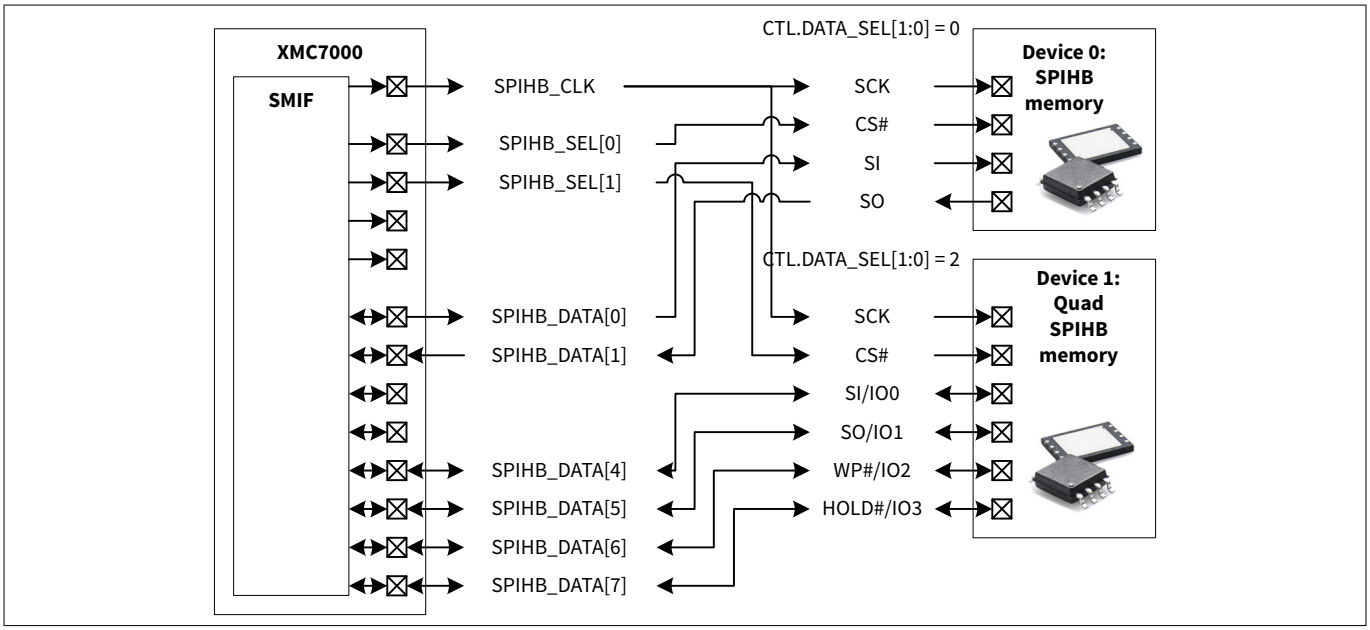


Figure 305 Single SPI device 0 to SPIHB_DATA[1:0], Quad SPI device 1 connected to SPIHB_DATA[7:4]

Figure 306 illustrates memory devices 0 and 1, device 0 is single SPI memory and device 1 is quad SPI memory. Both devices use shared data signal connections. The device address regions in the XMC7000 address space must be non-overlapping to ensure that the activation of SPIHB_SEL[0] and SPIHB_SEL[1] are mutually exclusive.

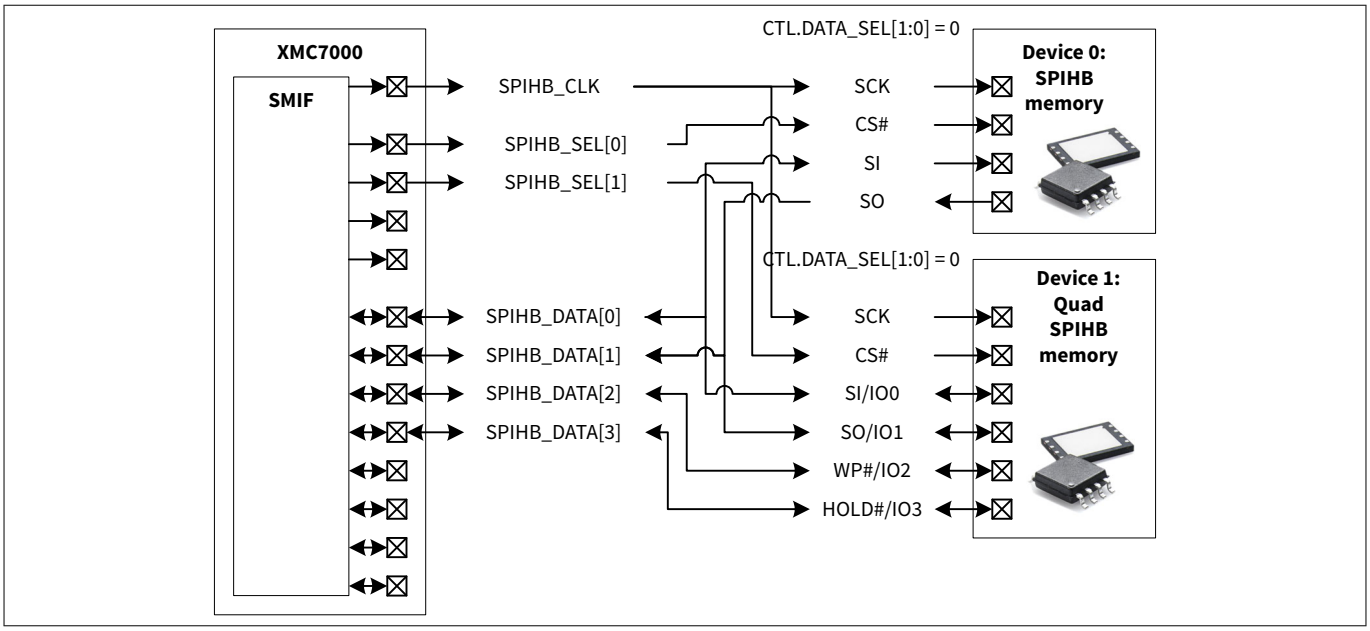


Figure 306 Single SPI device 0 to SPIHB_DATA[1:0], Quad SPI device 1 connected to SPIHB_DATA[3:0]

Figure 307 illustrates memory devices 0 and 1, both of which are quad SPI memories. Each device uses dedicated data signal connections. The device address regions in the XMC7000 address space are the same to ensure that the activation of SPIHB_SEL[0] and SPIHB_SEL[1] are the same (in XIP mode). This is known as a dual-quad configuration: during SPI read and write transfers, each device provides a nibble of a byte.

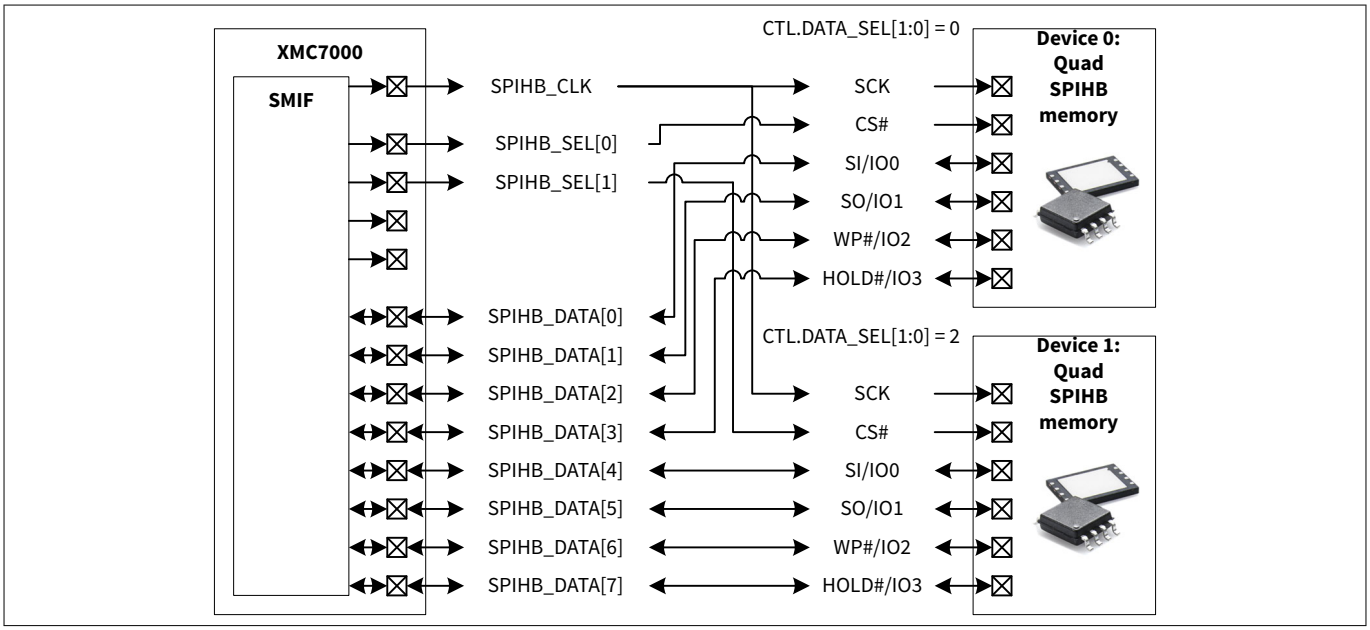


Figure 307 Quad SPI device 0 to SPIHB_DATA[3:0], Quad SPI device 1 connected to SPIHB_DATA[7:4]

Figure 308 illustrates memory device 0, which is an octal SPI memory with data signal connections to SPIHB_DATA[7:0].

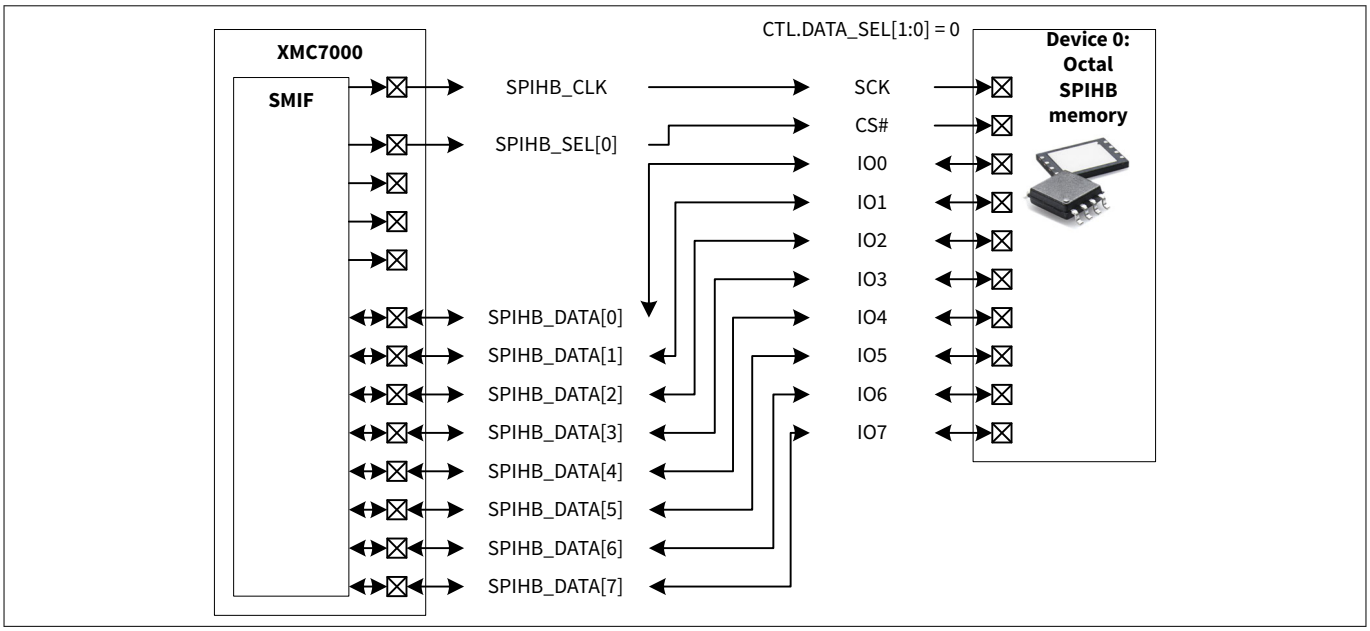


Figure 308 Octal SPI device 0 to SPIHB_DATA[7:0]

30.4.2.4 Connecting HB memory devices

Memory device I/O signals (SCK, SCK#, CS#, RWDS, IO0, SO/IO1, IO2, IO3, IO4, IO5, IO6, IO7) are connected to the SMIF I/O signals (SPIHB_CLK, SPIHB_CLK_inv, SPIHB_SEL[1:0], SPIHB_rwds and SPIHB_DATA[7:0]). Not all memory devices use the SCK# but can be operated single-ended.

Table 249 Memory device I/O signals

Memory device	I/O signals
HYPERBUS™ memory	SCK, CS#, IO0, IO1, IO2, IO3, IO4, IO5, IO6, IO7. This memory device has eight data signals (IO0, IO1, IO2, IO3, IO4, IO5, IO6, IO7).

Table 249 illustrates that each HYPERBUS™ memory has a single clock signal SCK and optional inverted clock signal SCK#, a single (low active) select signal (CS#), a single read-write-data_strobe RWDS, and eight data signals (IO0, IO1, …).

Each memory device has a fixed select signal connection (to SPIHB_SEL[1:0]).

Each memory device has programmable data signal connections (to SPIHB_DATA[7:0]): the MMIO CTL.DATA_SEL[1:0] field specifies how device data signals are connected. This information is used by SMIF to drive data on the correct SPIHB_DATA[] outputs and capture data from the correct SPIHB_DATA[] inputs. Because HYPERBUS™ devices use all eight data signals, the only valid setting for DATA_SEL[1:0] is 0.

Figure 309 illustrates memory device 0, which is a single HYPERBUS™ memory with data signals connections to SPIHB_DATA[7:0].

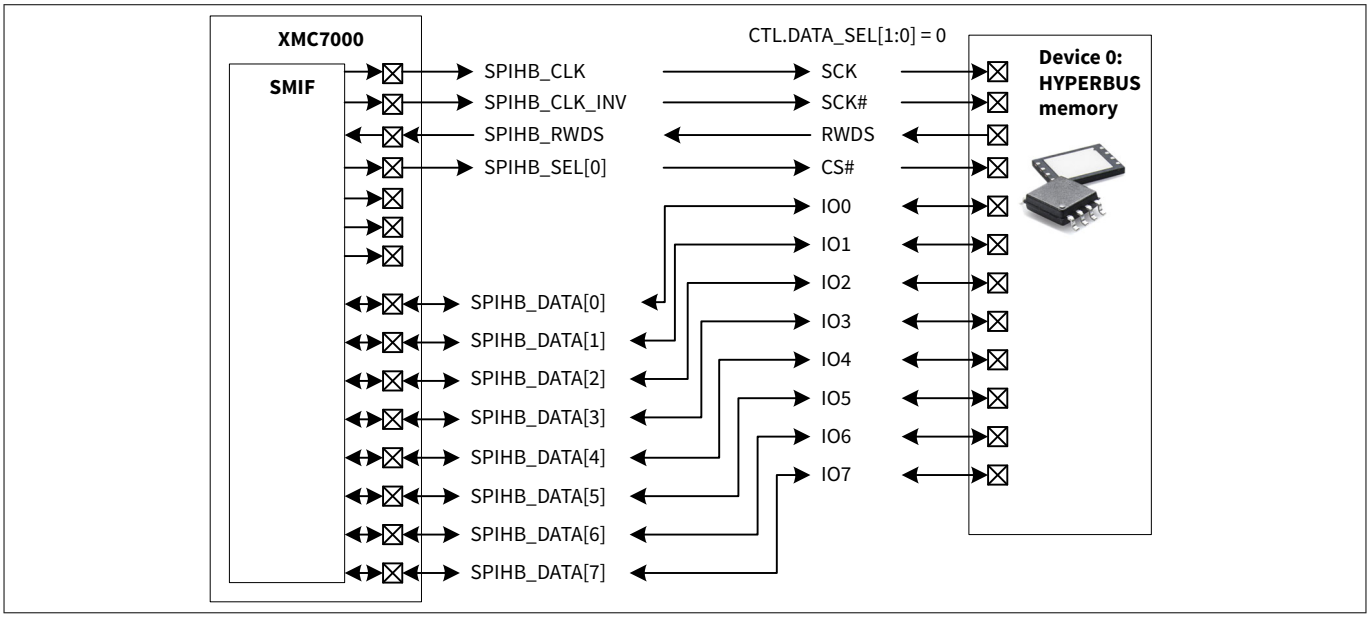


Figure 309 HYPERBUS™ SPI device 0 to SPIHB_DATA[7:0]

30.4.2.5 SPI data transfer

SPI data transfer uses most-significant-byte (MSB) first data transfer. This means that for a byte B, consisting of bits b7, b6, …, b0, bit b7 is transferred first, followed by bit b6, and so on. For dual, quad, dual-quad, and octal SPI transfers, multiple bits are transferred per cycle. For a single SPI device and device data signal connections to SPIHB_DATA[1:0] (SPIHB_SEL is 0), Table 250 summarizes the transfer of a byte B.

Table 250 Single data transfer

Cycle	Data transfer
0	For a write transfer: b7 is transferred on SPIHB_DATA[0] and SI/IO0 For a read transfer: b7 is transferred on SPIHB_DATA[1] and SO/IO1
1	For a write transfer: b6 is transferred on SPIHB_DATA[0] and SI/IO0 For a read transfer: b6 is transferred on SPIHB_DATA[1] and SO/IO1
2	For a write transfer: b5 is transferred on SPIHB_DATA[0] and SI/IO0 For a read transfer: b5 is transferred on SPIHB_DATA[1] and SO/IO1
3	For a write transfer: b4 is transferred on SPIHB_DATA[0] and SI/IO0 For a read transfer: b4 is transferred on SPIHB_DATA[1] and SO/IO1
4	For a write transfer: b3 is transferred on SPIHB_DATA[0] and SI/IO0 For a read transfer: b3 is transferred on SPIHB_DATA[1] and SO/IO1
5	For a write transfer: b2 is transferred on SPIHB_DATA[0] and SI/IO0 For a read transfer: b2 is transferred on SPIHB_DATA[1] and SO/IO1
6	For a write transfer: b1 is transferred on SPIHB_DATA[0] and SI/IO0 For a read transfer: b1 is transferred on SPIHB_DATA[1] and SO/IO1
7	For a write transfer: b0 is transferred on SPIHB_DATA[0] and SI/IO0 For a read transfer: b0 is transferred on SPIHB_DATA[1] and SO/IO1

Note that in single SPI data transfer, the SPIHB_data signals are unidirectional: in the table, SPIHB_DATA[0] is exclusively used for write data connected to the device SI input signal and SPIHB_DATA[1] is exclusively used for read data connected to the device SO output signal.

For a dual SPI device and device data signal connections to SPIHB_DATA[1:0] (SPIHB_SEL is 0), Table 251 summarizes the transfer of a byte B.

Table 251 Dual data transfer

Cycle	Data transfer
0	b7, b6 are transferred on SPIHB_DATA[1:0] and IO1, IO0
1	b5, b4 are transferred on SPIHB_DATA[1:0] and IO1, IO0
2	b3, b2 are transferred on SPIHB_DATA[1:0] and IO1, IO0
3	b1, b0 are transferred on SPIHB_DATA[1:0] and IO1, IO0

For a quad SPI device and device data signal connections to SPIHB_DATA[3:0] (SPIHB_SEL is 0), Table 252 summarizes the transfer of a byte B.

Table 252 Quad data transfer

Cycle	Data transfer
0	b7, b6, b5, b4 are transferred on SPIHB_DATA[3:0] and IO3, IO2, IO1, IO0
1	b3, b2, b1, b0 are transferred on SPIHB_DATA[3:0] and IO3, IO2, IO1, IO0

For a octal SPI device and device data signal connections to SPIHB_DATA[7:0] (SPIHB_SEL is 0), Table 253 summarizes the transfer of a byte B.

Table 253 Octal data transfer

Cycle	Data transfer
0	b7, b6, b5, b4, b3, b2, b1, b0 are transferred on SPIHB_DATA[7:0] and IO7, IO6, IO5, IO4, IO3, IO2, IO1, IO0

In Dual-quad SPI mode, two quad SPI devices are used.

- The first device (the device with the lower device structure index) should have device data signal connections to SPIHB_DATA[3:0] and must be connected to SPIHB_SEL[0].
- The second device (the device with the higher device structure index) should have device data signal connection to SPIHB_DATA[7:4] and must be connected to SPIHB_SEL[0].

The “command” and “data” phases of the SPI transfer use different width data transfers:

- The command, address, and mode byte use quad SPI data transfer.
- The read data and write data use octal data transfer. Each device provides a nibble of each data byte: the first device provides the lower nibble and the second device provides the higher nibble.

Table 254 summarizes the transfer of a read data and write data byte B.

Table 254 Dual-quad SPI mode, octal data transfer

Cycle	Data transfer
0	b7, b6, b5, b4 are transferred on SPIHB_DATA[7:4] and 2nd device IO3, IO2, IO1, IO0. b3, b2, b1, b0 are transferred on SPIHB_DATA[3:0] and 1st device IO3, IO2, IO1, IO0.

30.4.2.6 SPI - putting it all together

Devices 0 and 1 are used to implement dual-quad SPI mode. Both devices are 1MB/8Mb devices; that is, the address requires 3 bytes. Device 0 has device data signal connections to SPIHB_DATA[3:0] and device 1 has device data signal connections to SPIHB_DATA[7:4].

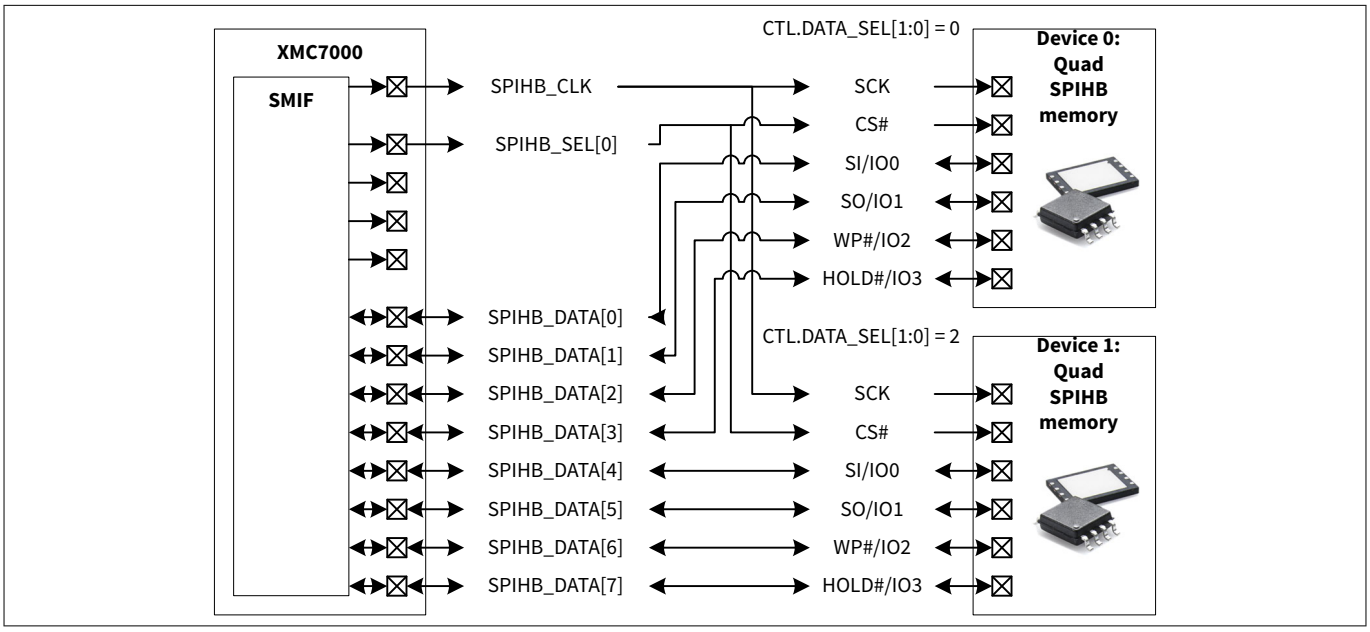


Figure 310 Dual-Quad SPI mode by connecting quad SPI device 0 to SPIHB_DATA[3:0], Quad SPI device 1 connected to SPIHB_DATA[7:4]

General settings

The clock settings are set for SDR timing.

```
MMIO_SMIF_CTL    = (1UL << 31)        // ENABLED
                 | (3 << 12)           // CLOCK_IF_RX_SEL: “spi_clk_in” (feedback clock) for SDR
                      | (0 << 10)           // DDR_CAPTURE_CYCLE, not used
                      | (0 << 9)           // INT_CLOCK_DL_ENABLED, not used
                 | (0 << 8)           // INT_CLOCK_DEL_TAP_ENABLED, not used
                 | (0 << 4)           // CLOCK_IF_TX_SEL: “clk_if_tx_div_inv” for DDR
                 | (0 << 0);           // MMIO_MODE: select MMIO/XIP mode
```

For dual-quad SPI mode, the AHB-Lite bus transfer address is divided by two. Cryptography and write functionality are disabled:

```
DEV0_ADDR        = CPUSS_SMIF_BASE;
DEV0_MASK        = 0xfff00000;        // MASK: 1 MB region
DEV0_CTL         = (1UL << 31)        // ENABLED
                 | (0 << 28)           // TOTAL_TIMEOUT_EN
                 | (0 << 16)           // TOTAL_TIMEOUT
                 | (0 << 15)           // MERGE_EN
                 | (0 << 12)           // MERGE_TIMEOUT
                 | (0 << 8)           // DATA_SEL: spi_data[3:0]
                 | (0 << 4)           // CRYPTO_EN
                 | (0 << 0);           // WR_EN
DEV0_ADDR_CTL    = (1 << 8)            // DIV2: enabled
                 | ((3-1) << 0));    // SIZE: 3 B address

DEV1_ADDR        = CPUSS_SMIF_BASE;
DEV1_MASK        = 0xfff00000;        // MASK: 1 MB region
DEV1_CTL         = (1UL << 31)        // ENABLED
                 | (0 << 28)           // TOTAL_TIMEOUT_EN
                 | (0 << 16)           // TOTAL_TIMEOUT
                 | (0 << 15)           // MERGE_EN
                 | (0 << 12)           // MERGE_TIMEOUT
                 | (2 << 8)           // DATA_SEL: spi_data[7:4]
                 | (0 << 4)           // CRYPTO_EN
                 | (0 << 0);           // WR_EN
DEV1_ADDR_CTL    = (1 << 8)            // DIV2: enabled
                 | ((3-1) << 0));    // SIZE: 3 B address
```

For XIP read transfers, the “0xeb” command/instruction is used (Figure 311 illustrates a two byte transfer from devices 0 and 1 in Dual-Quad SPI mode).

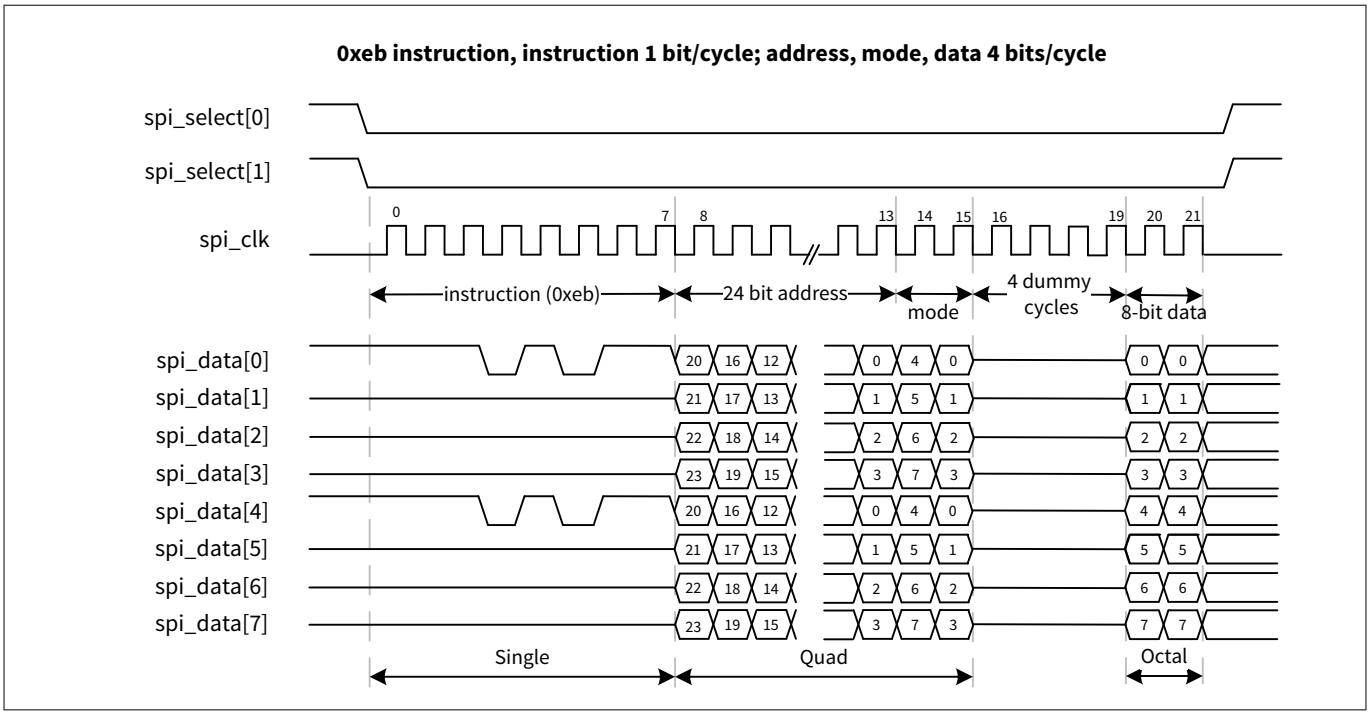


Figure 311 0xeb instruction, instruction 1-bit/cycle; Address, mode, data 4-bits/cycle

The definition of a read transfer is as follows:

```
DEV0_RD_CMD_CTL    = (1UL << 30)    // PRESENT
                   | (0 << 16)        // WIDTH: single data transfer
                      | 0xeb;            // CODE
DEV0_RD_ADDR_CTL    = (2 << 16);    // WIDTH: quad data transfer
DEV0_RD_MODE_CTL    = (1UL << 31)    // PRESENT
                   | (2 << 16)        // WIDTH: quad data transfer
                   | 0x00;            // CODE
DEV0_RD_DUMMY_CTL    = (1UL << 30)    // PRESENT
                   | ((4-1) << 0);    // SIZE: 4 dummy cycles
DEV0_RD_DATA_CTL    = (3 << 16);    // WIDTH: octal data transfer
```

Note that the command uses single data transfer, the address and mode byte use quad data transfer, and the read data byte uses octal data transfer. All transaction fields use SDR (single data rate) mode.

30.4.2.7 SPI- slave select signal during power up

Typically, SPI device datasheets specify that the chip select (CS#, which is SPIHB_SEL[]) must follow Vcc applied to the device. This can be achieved by adding a weak pull-up on slave select pin at board level.

30.4.2.8 HB data transfers

HYPERBUS™ is 2-byte addressed. Therefore, data transfers start with the most-significant byte (MSB) followed by the least-significant byte (LSB) of an addressed 2-byte word.

This means that for 2 bytes B1 and B0, consisting of bits b15, b14, …, b0, bit b15 to b8 is transferred first, followed by bit b7 to bit b0. Table 255 summarizes the transfer of 2 bytes B1 and B0 for a single HYPERBUS™ device.

Table 255 Single data transfer

Half cycle	Data transfer
0	b15 to b8 are transferred on SPIHB_DATA[7:0]
1	b7 to b0 are transferred on SPIHB_DATA[7:0]

- HB variable initial latency

  HYPERRAM™ memory devices have the option of a variable initial latency. If the memory is doing a refresh cycle a double initial latency is signaled using an active RWDS indicator during the command/address cycles. This is supported in the SMIF.

  To enable the HYPERRAM™ variable initial latency mode in MMIO mode bit 16 of the DUMMY_COUNT command in the TX command FIFO needs to be set to '1'.

  ***Note**: The SMIF TX interface must consider that the HYPERBUS™ latency cycle definition includes the last address cycle while the dummy cycles specified by bits 4:0 of the DUMMY_COUNT commands in the TX command FIFO do not include that.*

  Example: The single HYPERRAM™ latency count may be six cycles, so the doubled HYPERRAM™ latency cycles count for a refresh cycle is 12 cycles.

  The dummy cycle count specified in the DUMMY_COUNT command excludes the last address cycle. That is, bits 4:0 of the DUMMY_COUNT command need to be set to '4' (defining five dummy cycles). If the variable initial latency mode is enabled and the RWDS refresh indicator is active, the SMIF TX interface needs to double the latency cycles. It needs to set the dummy cycle count to 11 ((4+2)*2 - 1).

  To enable the HYPERRAM™ variable initial latency mode in XIP mode the PRESENT2 field must be set to '2' in the related RD/WR_DUMMY_CTL registers.

  If enabled, the SMIF XIP block sets bit 16 of the DUMMY_COUNT command in the TX command FIFO to '1'.

- HB page boundary crossing latency

  HYPERFLASH™ memory devices may require page boundary crossing latency cycles.

  In today's HYPERFLASH™ devices they only apply at the first page boundary crossing or more precisely when the second half-page boundary is crossed. However, future HYPERFLASH™ devices may need them for subsequent page boundary crossings. Also, future HYPERFLASH™ devices may have a different number of sub pages per page (other than 2).

  The presence and number of page boundary crossing latency cycles depend on the latency count and the start address of the burst transaction. The following two tables show examples of memories with eight words (16 bytes) per sub page, two sub pages per page, and a latency count of 11, 16, or 20 cycles (depending on the frequency).

First Boundary Crossing During Linear Read (Latency Count = 11 Clocks)

| Target Address | Clock Cycle After CS# Goes Low | | |

The SMIF supports the page boundary crossing latencies.

In MMIO mode this is completely under software control (RX_DATA_COUNT and DUMMY_COUNT commands must be written into the TX command FIFO as required by the memory).

In XIP mode the page boundary crossing behavior is defined by the RD_BOUND_CTL register. This only applies when continuous transfer merging is enabled (otherwise, the longest possible transaction only reads 16 bytes = 8 words and therefore, cannot cross a page boundary.

The following variables can be specified:

- The base latency cycle count (latency cycle count of the memory, which may be different from the dummy cycle count specified in the RD_DUMMY_CTL register due to the last address cycle not included)
- The size of a memory sub page
- The number of sub pages (banks) per page
- The page boundary crossing latency is only generated at the first page boundary crossing (the first time a SUB_PAGE_NRth sub page boundary is crossed). For example, with two sub pages per page when the second sub page boundary is crossed. A page boundary crossing latency can also be generated at the first and subsequent page boundary crossings (every time a SUB_PAGE_NRth sub page boundary is crossed). For example, with two sub pages per page when the second, fourth, sixth, … sub page boundary is crossed.

30.4.2.9 HB - putting it all together

One device is used to implement HB mode. HB devices require 6 bytes for command and address including reserved bits. Device 0 has device data signal connections to SPIHB_DATA[7:0].

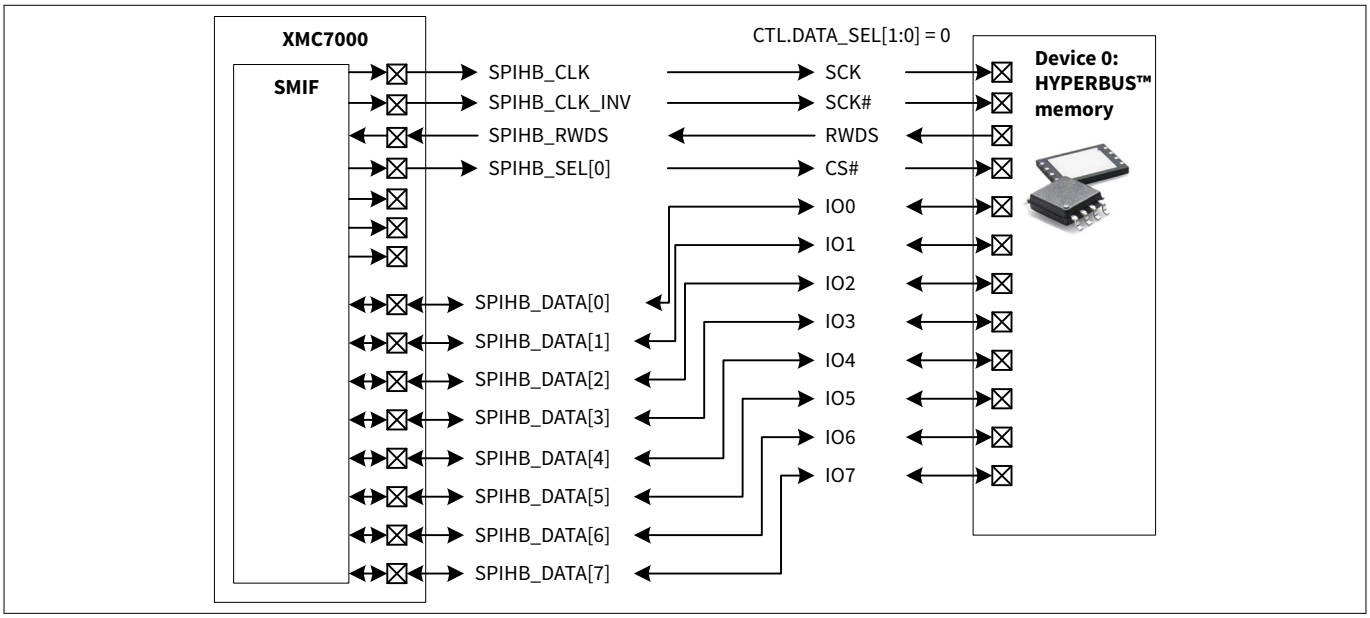


Figure 312 HYPERBUS™ device connection

To avoid an undriven RWDS input signal (used as an RX capture clock) while the memory is not selected, a PULLDOWN needs to be used for the RWDS signal (as '0' is the inactive RWDS state during latency cycles and after last data read before deselection).

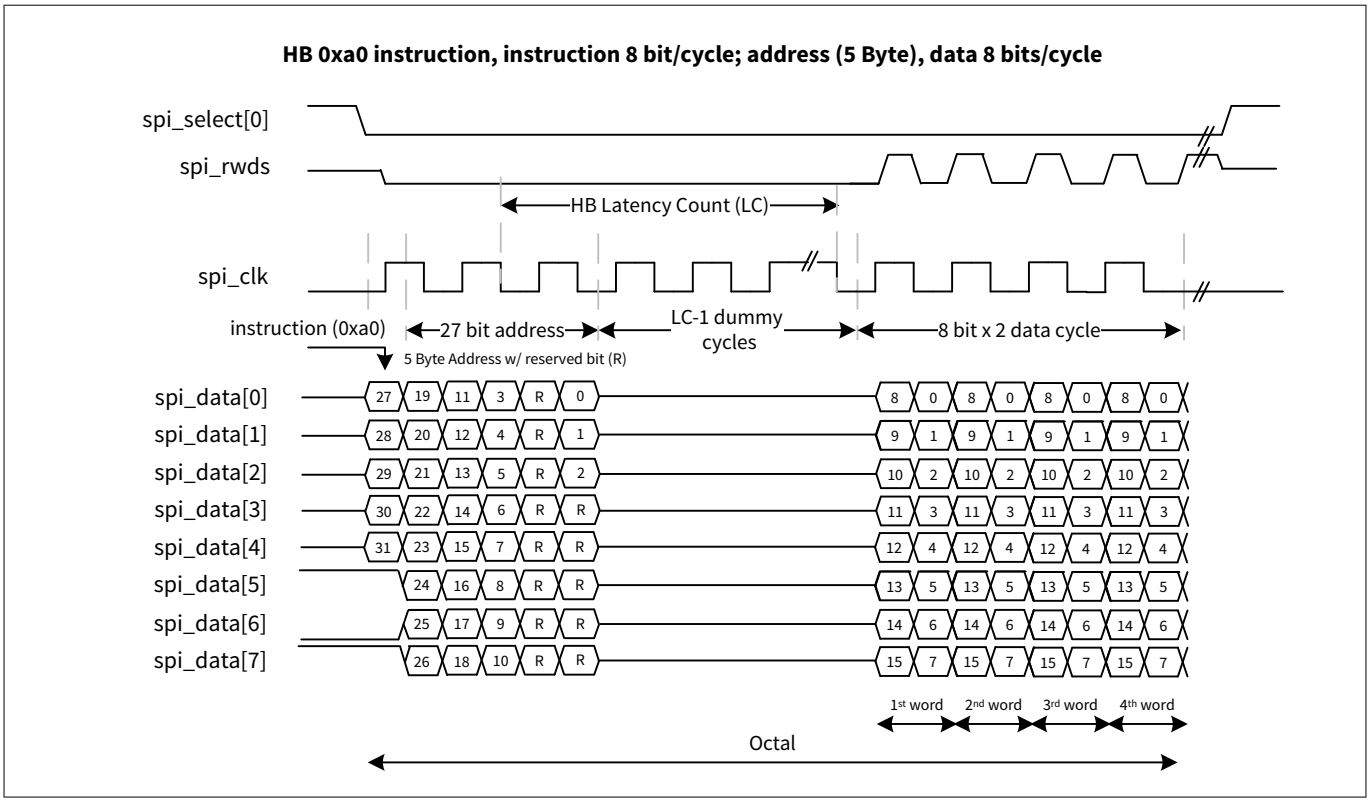


Figure 313 HYPERBUS™ transfer waveform

- General settings
  The clock settings are set for DDR timing with read-write-data-strobe (RWDS).

```
MMIO_SMIF_CTL             = (1UL << 31)    // ENABLED
                          | (6 << 12)       // CLOCK_IF_RX_SEL: “spi_rwds” for DDR with RWDS
                          | (0 << 10)       // DDR_CAPTURE_CYCLE, not used
                          | (0 << 9)       // INT_CLOCK_DL_ENABLED, not used
                          | (0 << 8)       // INT_CLOCK_DEL_TAP_ENABLED, not used
                          | (1 << 4)       // CLOCK_IF_TX_SEL: “clk_if_tx_inv_div” for DDR
                          | (0 << 0);       // MMIO_MODE: select MMIO/XIP mode
MMIO_SMIF_DELAY_TAP_SEL    = (0 << 28)        // BIT7, not used
                          | (0 << 24)       // BIT6, not used
                          | (0 << 20)       // BIT5, not used
                          | (0 << 16)       // BIT4, not used
                          | (0 << 12)       // BIT3, not used
                          | (0 << 8)       // BIT2, not used
                          | (0 << 4)       // BIT1, not used
                          | (8 << 0);       // BIT0, used for RWDS delay (for all bits)
```

Note that the delay tap setting is based on calibration by software

- MMIO mode

  The definition of a read transfer in MMIO mode is as follows (write commands to TX command FIFO via TX_CMD_FIFO_WR):

```
1st command word:    (0 << 24)            // TX command
                   | ((1 << 2) << 20)     // slave 2
                   | (0 << 19)             // NOT the last command of transfer
                   | (1 << 18)             // DDR mode
                   | (3 << 16)             // WIDTH: 8 bits/cycle
                   | 0xa000             // CMD (HB Read/MEM/Lin. opcode), word address[31:19]
2nd command word:    (0 << 24)            // TX command
                   | ((1 << 2) << 20)     // slave 2
                   | (0 << 19)             // NOT the last command of transfer
                   | (1 << 18)             // DDR mode
                   | (3 << 16)             // WIDTH: 8 bits/cycle
                   | 0x4000             // word address[18:3]
3rd command word:    (0 << 24)            // TX command
                   | ((1 << 2) << 20)     // slave 2
                   | (0 << 19)             // NOT the last command of transfer
                   | (1 << 18)             // DDR mode
                   | (3 << 16)             // 8 bits/cycle
                   | 0x0000             // Reserved bits + word address[2:0]
4th command word:    (3 << 24)            // DUMMY_COUNT command
                   | (0 << 17)             // RWDS (write mask) output generation
                   | (0 << 16)             // variable latency based on RWDS refresh indicator
                   | ((15-1) << 0)         // (LC-1) = 15 dummy cycles (16 HB LC)
5th command word:    (2 << 24)            // RX_COUNT command
                   | (3 << 16)             // 8 bits/cycle
                   | ((4-1) << 0)         // 4 Cycles (8 bytes)
```

The data words can be read from RX data FIFO via RX_DATA_MMIO_FIFO_RD2/RX_DATA_MMIO_FIFO_RD4 registers.

- XIP mode

The definition of a read transfer in XIP mode is as follows:

```
DEV0_ADDR            = CPUSS_SMIF_BASE;
DEV0_MASK            = 0xf0000000;        // MASK: 256 MB region
DEV0_CTL            = (1UL << 31)        // ENABLED
                    | (0 << 28)           // TOTAL_TIMEOUT_EN
                    | (0 << 16)           // TOTAL_TIMEOUT
                    | (0 << 15)           // MERGE_EN
                    | (0 << 12)           // MERGE_TIMEOUT
                    | (0 << 8)           // DATA_SEL: spi_data[7:0]
                    | (0 << 4)           // CRYPTO_EN
                    | (0 << 0));        // WR_EN
DEV0_ADDR_CTL        = (0 << 8)           // DIV2: disabled
                    | (7 << 0));        // SIZE: 5 B address w/ HB protocol
DEV0_RD_CMD_CTL        = (1UL << 30)        // PRESENT
                    | (1 << 18)           // DDR mode
                    | (3 << 16)           // WIDTH: 8 bits/cycle (octal data transfer)
                    | 0xa0;               // CODE HB (Read/MEM/Lin. transfer opcode)
DEV0_RD_ADDR_CTL    = (1 << 18)           // DDR mode
                    | (3 << 16));        // WIDTH: 8 bits/cycle (octal data transfer)
DEV0_RD_MODE_CTL    = (0 << 31)           // NOT PRESENT
DEV0_RD_DUMMY_CTL    = (1UL << 30)        // PRESENT
                    | ((15-1) << 0));    // SIZE: (LC-1) = 15 dummy cycles (16 HB LC)
DEV0_RD_DATA_CTL    = (1 << 18)           // DDR mode
                    | (3 << 16));        // WIDTH: 8 bits/cycle (octal data transfer)
```

Note that the RX (or TX) count is loaded directly from the AHB/AXI bus.

30.4.3 AXI interface

The SMIF provides an XIP AXI interface. It is intended to be used by the M7 CPU.

Figure 314 gives an overview of the SMIF AXI interface.

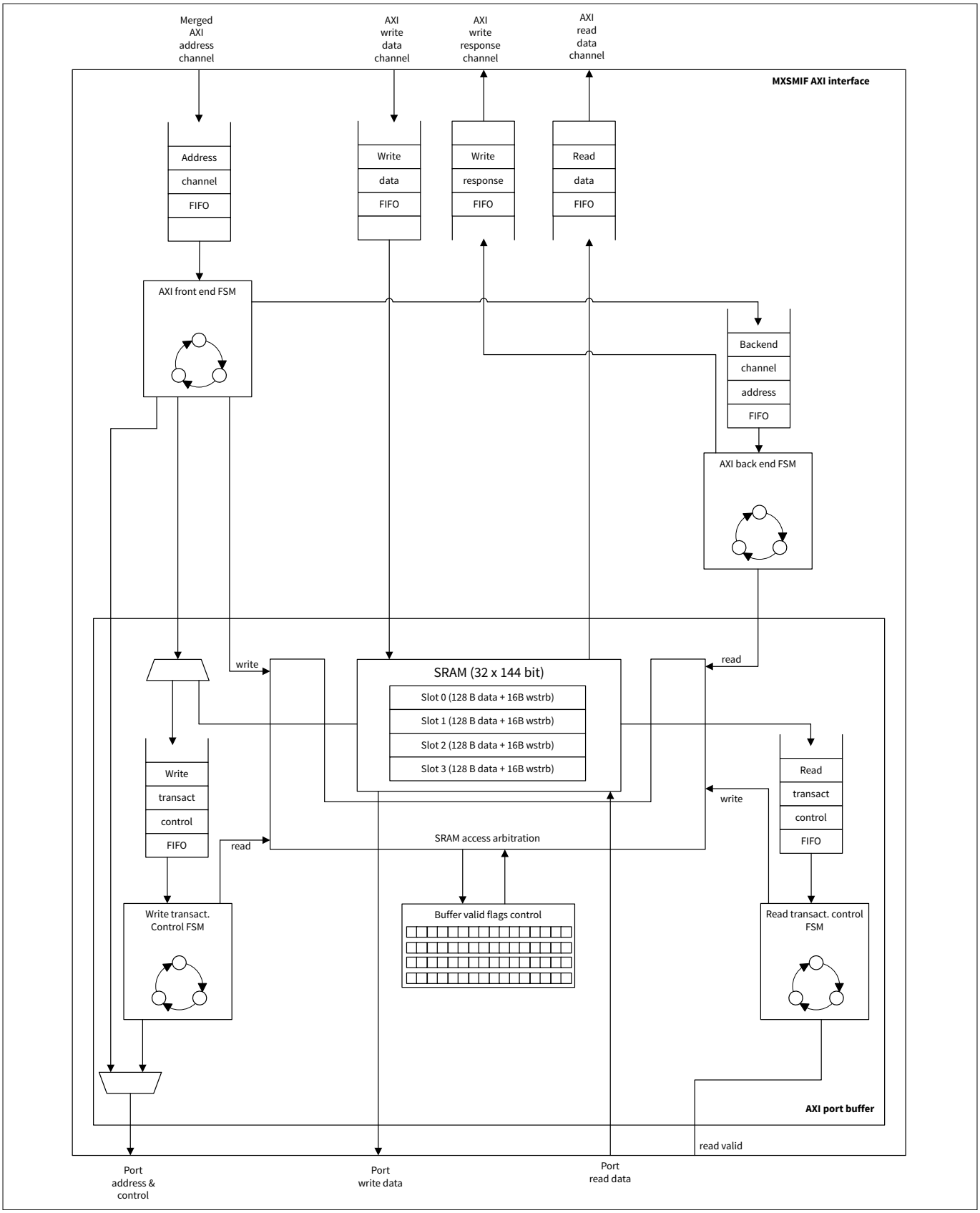


Figure 314 AXI slave interface block

The SMIF AXI interface contains four FSMs.

- AXI front-end FSM, responsible for
  - Allocating the buffer slots
  - Writing write transaction data (from AXI write data channel) into the buffer and forwarding write transaction control information to buffer (write transaction control FSM)
  - Initiating block read transactions to the serial memory interface (including translation of wrapping read bursts to incrementing read bursts)
- AXI back-end FSM, responsible for
  - Reading read transaction data from the buffer and forwarding that to the AXI read data channel (including backward translation of incrementing read bursts to wrapping read bursts)
  - Generating the AXI write data response
- Buffer write transaction control FSM, responsible for
  - Reading write transaction data from the buffer and forwarding them to the serial memory interface
  - Initiating block write transactions to the serial memory interface (including translation of wrapping write bursts to incrementing write bursts)
- Buffer read transaction control FSM, responsible for
  - Writing read transaction data from the serial memory interface to the buffer

The SMIF AXI interface contains an SRAM-based buffer of 576 bytes. The buffer is used to hold outstanding transaction data and (in case of writes accesses) write strobe information. The buffer has four slots of 128 bytes data and 16 bytes write strobes each. Each slot is further divided into blocks of 8 bytes of data and 1 byte strobe, which is the maximum size of one AXI transfer (one beat of an AXI burst). For every block a related valid flag exists. For write transactions the communication between AXI front-end FSM and buffer write transaction control FSM as well as for read transactions the communication between AXI back end FSM and buffer read transaction control FSM is done via these valid flags.

30.4.3.1 Support for multiple outstanding transactions

Multiple outstanding transactions are supported to make optimum use of the bandwidth given by the SMIF and not waste time for SMIF internal logic. This means, the transactions are pipelined and includes the following aspects:

- While one read or write transaction is ongoing at the serial memory interface, the following transaction (when available from the AXI interface) is already prepared.
- If the AXI read data channel is temporarily stalled, the previously read data is stored and the serial memory interface already serves the next read or write transaction.

At least three outstanding transactions should be supported; one read transaction is finished and needs to be temporarily stored (due to stalled AXI read data channel), a second read or write transaction is served by the serial memory interface and a third read or write transaction is currently prepared to be served immediately after the second one.

These three outstanding transactions are rounded up to the next power-of-two number which is ‘4’. Therefore, a buffer that provides storage for four AXI transactions is used.

***Note**: The number of at least three outstanding transactions here represents only the minimum number of outstanding transactions the SMIF XIP pipeline and therefore the AXI buffer needs to support. However, due to the AXI address channel input FIFO and the AXI read/write response channel output FIFOs, the maximum total outstanding transactions (including these waiting AXI channel FIFO entries) can be higher.*

The SMIF AXI interface does not reorder transactions. The transactions are processed in the same order as they occur at the (merged) AXI address channel.

- Bandwidth

  The SMIF can achieve a maximum possible memory interface bandwidth of 400 Mbyte/s when using an 8-bit wide memory interface with double data rate (DDR) at a maximum speed of 200 MHz (HYPERBUS™ or octal SPI with DDR). With the ability of merging continuous transfers and its pipelined operation the SMIF is designed to support this bandwidth and not generate stall cycles. However, to limit the area overhead for FIFOs in the AXI interface and XIP block, the maximum bandwidth is only supported for AXI data streams consisting of transactions of at least 16 bytes. For transactions of only 8 bytes or smaller the streaming performance is not optimized. The assumption is that bandwidth-critical AXI masters will use outstanding transactions with a total transaction size of at least 16 bytes (for example, CM7 cache line fills are 32-byte bursts).After initial memory protocol overhead cycles (command, address, and latency cycles) the full theoretically possible bandwidth of 400 Mbps can be achieved as long as a continuous (linear) data stream is read or written.

  For non-continuous transactions, the bandwidth is significantly lower because the overhead cycles need to be sent repeatedly.

30.4.3.2 Wrapping bursts

HYPERBUS™ memories and many SPI memories support wrapping bursts. Additionally, for SPI memories, a Memory Control register needs to be written whenever there is a change from wrapping to incrementing bursts or vice versa. Also, for SPI and HYPERBUS™ memories, a Memory Control register needs to be written whenever there is a change in the wrapping boundary. This is not possible in XIP mode (and does not fit into the concept of the XIP mode).

30.4.3.3 Fixed bursts

Fixed bursts to a memory address region is uncommon, but they are supported to be AXI compliant.

For read transactions the SMIF performs the read transfer to the memory only once but its AXI interface provides the same data multiple times as requested by the length of the fixed burst. For write transactions the SMIF performs a write transfer to the memory only for the last AXI write transfer.

30.4.3.4 Splitting bursts to smaller blocks

The SMIF has multiple XIP interfaces (at least one AHB and one AXI interface). A priority-based arbitration is done between the interfaces. To not block high-priority transactions (from a latency critical master) by lower priority long transactions (from a high-bandwidth master), interrupting lower priority transactions is preferred.

Longer transactions are therefore split to multiple blocks, which can be potentially interrupted by the arbiter. The block size is 16 bytes. This matches the block size of the cache subsector fetch or pre-fetch in the AHB interface and the block size of an AES encryption/decryption.

30.4.3.5 Write strobes

The AXI protocol allows write byte masking, that is, write transactions with any combination of write strobes. Therefore, any combination of actual bytes to be written is possible within a transaction. The CM7 store buffer uses it to merge multiple store instructions to normal memory as a performance optimization.

Example 1:

- CM7 code:

```
MOV r0, #0x4000
STR r1, [r0, #0xC]        ; Store a word at 0x400C
```

- This can result in the following write transaction
  - address 0x4008, one transfer, size 8 bytes (asize = 3), write strobes (wstrb) set to 0xF0
- Here only the write strobes for the higher 4 bytes are enabled within an 8-byte transfer.

Example 2:

- CM7 code:

```
MOV r0, #0x4000
STRH r1, [r0, #0x18]       ; Store a halfword at 0x4018
STR r2, [r0, #0xC]         ; Store a word at 0x400C
STMIA r0, {r4-r7}          ; Store four words at 0x4000
STRB r3, [r0, #0x1D]       ; Store a byte at 0x401D
```

- These instructions can result in the following write transaction:
  - address 0x4000, four transfers, size 8 bytes (asize = 3), write strobes (wstrb) set to 0xFF, 0xFF, 0x00, and 0x23
- The four store instructions are merged to a single AXI transaction by the CM7 store buffer. The second store instruction is completely skipped (because its target address is overwritten by the third store instruction). The write strobes are not adjacent and even a transfer without any write strobe set is generated.

HYPERBUS™ and some octal SPI RAM devices support byte masking with RWDS or DQS signals. When writing data to an external HYPERBUS™ or octal SPI RAM, this can be used to directly translate any AXI write transaction to a single memory burst.

However, other SPI devices do not support byte masking. To support such RAM devices an AXI write transaction with non-adjacent write strobes should be split into multiple SPI memory bursts.

These RAM devices play almost no role in the market, but for serial RAM devices only HYPERBUS™ and octal SPI RAM devices are relevant. Therefore, AXI write byte masking for SPI RAM devices without byte masking is not supported in SMIF (when RWDS_EN bit is set to '0' in WR_DUMMY_CTL register). This means an error response is generated when the memory device does not support write byte masking and not all of the AXI write strobes are enabled according to the transfer size.

However, such RAM devices can still be used with the CM7 as strongly-ordered or device memory. The SMIF address space is usually located in a memory region with a normal default memory type, but the CM7 allows it to override default memory types. That means, if an SPI RAM device without byte masking is used, the MPU must be configured to override the memory type of that RAM device region from normal to strongly-ordered or device memory. This prevents CM7 to merge multiple store instructions and ensures that all AXI write channel write strobe bits are enabled according to the transfer size. Then the above examples look as follows in case of a strongly-ordered or device memory region:

Example 1:

- CM7 code:

```
MOV r0,    #0x4000
STR r1, [r0, #0xC]        ; Store a word at 0x400C
```

- This results in the following write transaction
  - address 0x400C, one transfer, size 4 bytes (asize = 2), write strobes (wstrb) set to 0xF0
- Here all four write strobes are enabled within a 4-byte transfer.

Example 2:

- CM7 code:

```
MOV r0, #0x4000
STRH r1, [r0, #0x18]    ; Store a halfword at 0x4018
STR r2, [r0, #0xC]      ; Store a word at 0x400C
STMIA r0, {r4-r7}       ; Store four words at 0x4000
STRB r3, [r0, #0x1D]    ; Store a byte at 0x401D
```

- This results in the following five write transactions:
  - address 0x4018, one transfer, size 2 bytes (asize = 1), write strobes (wstrb) set to 0x03
  - address 0x400C, one transfer, size 4 bytes (asize = 2), write strobes (wstrb) set to 0xF0
  - address 0x4000, two transfers, size 4 bytes (asize = 2), write strobes (wstrb) set to 0x0F, 0xF0
  - address 0x4008, two transfers, size 4 bytes (asize = 2), write strobes (wstrb) set to 0x0F, 0xF0
  - address 0x401D, one transfer, size 1 byte (asize = 0), write strobes (wstrb) set to 0x01
- The store instructions are causing individual AXI transactions. No skipping or merging is done. All write strobes are set according to the transfer size.

It is expected that other AXI masters (such as a graphic block) do not use write byte masking.

30.4.4 Triggers

The SMIF has two level-sensitive triggers:

- SMIF_TX_TR_OUT is associated with the TX data FIFO
- SMIF_RX_TR_OUT is associated with the RX data FIFO
- If the SMIF is enabled (ENABLED is set to '1' in the CTL register) and MMIO operation mode is selected (XIP_MODE is set to '0' in the CTL register), the trigger functionality is enabled. The trigger functionality is defined as follows:
- The TRIGGER_LEVEL field in TX_DATA_FIFO_CTL register specifies a number of FIFO entries. The SMIF_TX_TR_OUT trigger is active when the number of used TX data FIFO entries is smaller than or equal to the specified number; that is, USED4 in TX_DATA_FIFO_STATUS register ≤ TRIGGER_LEVEL.
- The TRIGGER_LEVEL field in RX_DATA_FIFO_CTL register specifies a number of FIFO entries. The SMIF_RX_TR_OUT trigger is Active when the number of used RX data FIFO entries is greater than the specified number; that is, USED4 in RX_DATA_FIFO_STATUS register > TRIGGER_LEVEL.

30.4.5 Interrupts

The SMIF has a single interrupt output. This interrupt has three interrupt causes:

- TR_TX_REQ in INTR register. This interrupt cause is activated in MMIO mode, when the SMIF_TX_TR_OUT trigger is activated.
- TR_RX_REQ in INTR register. This interrupt cause is activated in MMIO mode, when the SMIF_RX_TR_OUT trigger is activated.
- XIP_ALIGNMENT_ERROR in INTR register. This interrupt cause is activated in XIP mode, when dual-quad SPI mode or octal SPI DDR/HYPERBUS™ mode without memory write byte masking is selected and the AHB-Lite/AXI bus address is not a multiple of 2 or the requested transfer size is not a multiple of 2. This interrupt

cause identifies erroneous behavior in dual-quad SPI mode (the selected device DIV2 field is set to '1' in the ADDR_CTL register), octal SPI DDR mode, or HYPERBUS™ mode.

This interrupt cause is activated in XIP mode when the selected memory device does not support write byte masking (RWDS_EN is set to '0' in WR_DUMMY_CTL register) and an AXI transfer occurs with not all of the AXI write strobes (wstrb) enabled according to the transfer size (assize).

- TX_CMD_FIFO_OVERFLOW in INTR register. This interrupt cause is activated in MMIO mode, on an AHB-Lite write transfer to the TX command FIFO (TX_CMD_FIFO_WR) with not enough free entries available.
- TX_DATA_FIFO_OVERFLOW in INTR register. This interrupt cause is activated in MMIO mode, on an AHB-Lite write transfer to the TX data FIFO (TX_DATA_FIFO_WR1, TX_DATA_FIFO_WR2, TX_DATA_FIFO_WR4) with not enough free entries available.
- RX_DATA_FIFO_OVERFLOW in INTR register. This interrupt cause is activated in MMIO mode, on an AHB-Lite read transfer from the RX data FIFO (RX_DATA_FIFO_RD1, RX_DATA_FIFO_RD2, RX_DATA_FIFO_RD4) with not enough entries available.

30.4.6 Monitor signals

The SMIF has five monitor signals. These signals reflect active SPI transfers:

- The monitor_smif_SPIHB_SEL[i] (i = 0, 1) signal is active '1' during SPI transfers to memory device i (SPIHB_select_out[i] is '0'). In other words, the monitor signals are the logical inverse of the SPI select signals.
- The monitor_smif_SPIHB_select_any signal is the logical OR of the monitor_smif_SPIHB_SEL[] signals; that is, it is active/'1' when any of the monitor_smif_SPIHB_SEL[] signals is active/'1'.

The monitor signals are driven by dedicated flip-flops that are driven by the negative edge of the transmitter clock, CLK_if_tx_inv.

The monitor signals are connected to an energy profiler, which accumulates the duration of active SPI transfers. The energy profiler and its associated software use the SPI transfer activity as a proxy for the amount of energy consumed by the SMIF.

30.5 Supply rails and power domains

30.5.1 Power modes

Active, sleep, deep sleep, and hibernate are the different power modes defined in SRSS. The following table describes the status of SMIF during different power modes.

Table 256 SRSS power modes

System power mode	Description
ACTIVE/SLEEP	Active and sleep are standard Arm®-defined power modes, supported by the Arm® CPUs and ISA. Active: In this mode, CPUs will execute code, and all the logic and memories are powered ON. Sleep: In this mode, CPUs will not execute code and the clock is stopped. All the logic and memories are powered ON. It is identical to active power mode from the peripheral point of view.

(table continues...)

Table 256 (continued) SRSS power modes

System power mode	Description
LPACTIVE/LPSLEEP	LPActive/LPSleep are similar to active/sleep, except the current is limited and clocks may run at a lower frequency (and some functions are not available or limited.) Transitions between active and LPActive (or sleep and LPSleep) are initiated by SRSS PWR_CONTROL.LOW_POWER control bit.
DEEP SLEEP	Deep sleep is a lower power mode where high-frequency clocks are disabled. Active power domain is powered off (vccact power supply is OFF). CPUs and most MMIO state is retained (through retention flops). System SRAM retains its data. Flash and ROM memories are powered OFF.
HIBERNATE	Hibernate is an even lower power mode than Deep Sleep, but on wakeup the CPUs (and all peripherals) go through a full reset and firmware reboot ***Note**: This power mode is more similar to stop power mode in M0S8 platform.*
XRES	XRES is the state of the device when external reset is applied
OFF	OFF state is the state of the device when no power is applied to it

The SMIF is an active functionality. In Deep Sleep power mode, the retention MMIO registers are retained. Note that the cache and the AXI interface buffer are not retained in Deep Sleep power mode. When exiting Deep Sleep power mode, the cache and buffer are reset.

When entering Deep Sleep power mode, it is desirable to put the external memory devices in low-power mode as well (if this mode is supported by the devices). Similarly, when exiting deep sleep mode, the external memory devices should exit their low-power mode. Entering and exiting low-power mode is device-specific. This is supported through the MMIO mode. This means that if SMIF is in XIP mode, a change to MMIO is required before entering the device low-power mode.

30.6 Sub block descriptions

30.6.1 Address space

SMIF has three AHB-Lite slave interfaces:

- Fast and slow XIP interfaces have a shared design time configurable address space. This address space supports XIP_MODE mode of operation and is (partially) populated by the external devices.
- The MMIO interface has a 4 KB address space. This address space supports MMIO_MODE of operation. This address space includes all the MMIO registers (which also provides access to the TX and RX FIFOs).

The following sections describe the address spaces in more detail.

30.6.1.1 XIP address space

AHB-Lite transfers to the XIP address space either access the cache or are translated on-the-fly into SPI transfers to the external device. SMIF exposes an address space located at XMC7000 address 0x6000:0000. The XIP address capacity is design time configurable:

- The capacity is 2^n bytes, with n in the range [16, 32]. This allows for a minimum capacity of 64 KB and a maximum capacity of 4 GB.

The cache SRAM memory is used to cache read data (write data is not cached).

An access outside external device spaces result in an AHB-Lite error.

The location of the external devices in the XIP address space is programmable. Each external device i = 0, 1 (up to two external devices are supported) has an associated set of MMIO Device registers that specify their location and size in the XIP address space:

- ADDR bit field in ADDR register specifies the device location within the XIP address space. The device location should be a multiple of the device capacity.
- MASK bit field in ADDR register specifies the device capacity. The device capacity is 2^m bytes, with m in the range [8, n] (with n specifying the XIP address capacity).

For example, for a 16-Mbyte XIP address space from 0x6000:0000 to 0x60FF:FFFF and a 64-KB device at 0x6001:0000 to 0x6001:FFF, the MMIO device registers are programmed as follows:

- ADDR[23:8] bits in ADDR register is set to 0x0100 (location in XIP address space)
- MASK[23:8] bits in MASK register is set to 0xFF00 (device capacity)

For dual-quad SPI mode, it is required to program the same MMIO device register values for the two external devices that are connected in parallel to the SMIF I/O signal interface.

Write support to external devices is programmable. This is to support nonvolatile devices that do not support write accesses directly, but require a dedicated programming operation:

- WR_EN bit is set to '0' in CTL register: write accesses are not supported. An XIP write transfer results in an AHB-Lite bus error. Typically used for nonvolatile devices without write support.
- WR_EN bit is set to '1' in CTL register: write accesses are supported. Typically used for SRAMs.

30.6.1.2 MMIO address space

AHB-Lite transfers to the MMIO address space access the MMIO registers. The MMIO registers include registers to access the FIFOs.

Whereas the XIP address space supports highly efficient read and write access to external devices (through on-the-fly translation of AHB-Lite transfers into SPI transfers), the MMIO address space provides flexibility in the construction of SPI transfers.

30.6.2 TX and RX FIFOs

The SMIF has two TX FIFOs and one RX FIFO. These FIFOs provide an asynchronous clock domain transfer between CLK_mem logic and CLK_if_tx/CLK_if_rx memory interface logic. The memory interface logic is completely controlled through the TX and RX FIFOs. Additionally, SMIF has an RX data MMIO FIFO, which is used only in MMIO mode and which is logically an extension of the RX data FIFO enabling an easy-to-use RX data handling in software.

- The TX command FIFO transmits memory commands to the memory interface logic.
- The TX data FIFO transmits write data to the memory interface transmit logic.
- The RX data FIFO receives read data from the memory interface receive logic.

30.6.2.1 TX command FIFO

The FIFO consists of eight 27-bit entries. Each entry holds a command. A memory transfer consists of a series of commands. In other words, a command specifies the phase of a memory transfer. Five different types of commands are supported:

- TX command: A memory transfer must start with a TX command. The TX command includes one or two bytes to be transmitted over the memory interface. The TX command specifies the width of the data transfer (single, dual, quad, or octal data transfer). The TX command specifies the data transfer mode (SDR or DDR). The TX command specifies if the command is for the last phase of the memory transfer (explicit “last command” indication). The TX command specifies which of the two external devices are

selected (multiple devices can be selected simultaneously); that is, the device selection as encoded by the TX command is used for the complete memory transfer.

- The number of bytes included in the TX command depends on the data transfer width and the data transfer mode (SDR or DDR). Two bytes per TX command are transmitted for 8-bit width and DDR (HYPERBUS™/octal SPI with DDR), when 2 bytes per cycle are transmitted by the memory interface. This way a throughput bottleneck at the TX command FIFO is avoided. In other cases, only one byte per TX command is transmitted to allow byte granularity.
- TX_COUNT command: The TX_COUNT command relies on the TX data FIFO to provide the bytes to be transmitted over the memory interface. The TX_COUNT command specifies the number of memory data units to be transmitted. For SPI (except octal SPI with DDR) one memory data unit is a byte; for octal SPI with DDR and HYPERBUS™ one memory data unit is a 2-byte word. The TX_COUNT command specifies the width of the data transfer. The TX_COUNT command specifies the data transfer mode (SDR or DDR). The TX command specifies if the command is for the last phase of the memory transfer (explicit “last command” indication).
- RX_COUNT command: The RX_COUNT command relies on the RX data FIFO to accept the bytes that are received over the memory interface. The RX_COUNT command specifies the number of memory data units to be received. For SPI (except octal SPI with DDR) one memory data unit is a byte; for octal SPI with DDR and HYPERBUS™ one memory data unit is a 2-byte word. The RX_COUNT command specifies the width of the data transfer. The RX_COUNT command specifies the data transfer mode (SDR or DDR). The TX command specifies if the command is for the last phase of the memory transfer (explicit “last command” indication).
- DUMMY_COUNT command: The DUMMY_COUNT command specifies a number of dummy cycles. Dummy cycles are used to implement the turn-around (TAR) time in which the memory master changes from a transmitter driving the data lines to a receiver receiving on the same data lines. The DUMMY_COUNT command specifies the number of dummy cycles. The DUMMY_COUNT command specifies if the variable latency mode for HYPERRAM™ (indicated by an active RWDS input) is enabled causing the double number of dummy cycles. The DUMMY_COUNT command specifies whether the RWDS output signal should be driven starting in the last dummy cycle until deselection. The DUMMY_COUNT command never constitutes the last phase of the memory transfer (implicit not “last command” indication); that is, it needs to be followed by another command.
- DESELECT command: The DESELECT command causes the memory interface transmit logic to finish a transfer and deselect the memory device. The DESELECT command always constitutes the last phase of the memory transfer (implicit “last command” indication).

Together, the five command types can be used to construct any SPI or HYPERBUS™ (HB) transfer.

The TX command FIFO is used by both the memory interface transmit and receive logic. This ensures lockstep operation.

The software can read the number of used TX command FIFO entries through the USED[2:0] bit field in the MMIO TX_CMD_FF_STATUS register.

The software can write to the TX command FIFO through the MMIO TX_CMD_FIFO_WR register. If software attempts to write an entry of a full TX command FIFO, the BLOCK bit field in the MMIO CTL register specifies the behavior:

- If BLOCK is 0, an AHB-Lite bus error is generated.
- If BLOCK is 1, the AHB-Lite write transfer is extended until an entry is available.

30.6.2.2 TX data FIFO

The FIFO consists of eight 18-bit entries. Each entry holds a memory data unit (one or two bytes) to be transmitted over the memory interface. Additionally, each entry can hold a byte mask (which is used to mask bytes in HYPERBUS™ write transactions). A FIFO TX_COUNT command specifies the number of data units to be transmitted (bytes or 2-byte words); that is, the number of TX data FIFO entries to be used.

The TX data FIFO is used by the memory interface transmit logic.

The software can read the number of used TX data FIFO entries through the USED[3:0] bit field in the MMIO TX_DATA_FF_STATUS register.

In MMIO mode, the software can write to the TX data FIFO through the MMIO TX_DATA_FIFO_WR1, TX_DATA_FIFO_WR1ODD, TX_DATA_FIFO_WR2, and TX_DATA_FIFO_WR4 registers:

- The TX_DATA_FIFO_WR1 register supports write of a single byte to the FIFO.
  - For data transfer width smaller than eight or SDR mode a single byte is written to one FIFO entry.
  - For data transfer width of eight and DDR mode a single byte is written to the low byte of the FIFO entry; the high byte is masked.
- The TX_DATA_FIFO_WR1ODD register supports write of a single byte to the FIFO. This register provides the functionality to write a single byte for a 16-bit word-based memory interface (HYPERBUS™).
  - For data transfer width smaller than eight or SDR mode this register is not used.
  - For data transfer width of eight and DDR mode a single byte is written to the high byte of the FIFO entry; the low byte is masked.
- The TX_DATA_FIFO_WR2 register supports a write of two bytes to the FIFO.
  - For data transfer width smaller than eight or SDR mode a single byte is written to each of two FIFO entries.
  - For data transfer width of eight and DDR mode two bytes are written to one FIFO entry.
- The TX_DATA_FIFO_WR4 register supports a write of four bytes to the FIFO.
  - For data transfer width smaller than eight or SDR mode a single byte is written to each of four FIFO entries.
  - For data transfer width of eight and DDR mode two bytes are written to each of two FIFO entries.

The MMIO interface logic gets the information of the transfer width and transfer rate mode (SDR/DDR) from the previous TX COUNT or RX COUNT command written to the TX command FIFO. Therefore, the related TX COUNT command must be written before writing the transmit data to the TX DATA FIFO.

If software attempts to write more bytes than available entries in the TX data FIFO, the MMIO CTL.BLOCK field specifies the behavior:

- If BLOCK is 0, an AHB-Lite bus error is generated.
- If BLOCK is 1, the AHB-Lite write transfer is extended until the required entries are available.

30.6.2.3 RX data FIFO

The FIFO consists of sixteen 16-bit entries. Each entry holds up to 2 bytes, which are received over the memory interface. For SDR capturing, only the lower byte is used; for DDR capturing both bytes are used. For SDR capturing with an interface width of less than eight (single, dual, or quad SPI) only 1, 2, or 4 LSBs of the lower byte are used. For DDR capturing with an interface width of less than eight (single, dual, or quad SPI) only 1, 2, or 4 LSBs of both bytes are used.

A FIFO RX_COUNT command specifies the number of data units to be received (bytes or 2-byte words); that is, the number of RX data FIFO entries to be used.

The RX data FIFO is used by both the memory interface transmit and receive logic. This may appear unusual because the memory interface transmit logic does not receive bytes. However, the memory interface transmit logic is responsible for generating the memory interface clock SPIHB_CLK_out, and this clock should not be generated (should be kept low/0) when the RX data FIFO is full. Therefore, although the memory transmit logic does not receive bytes, it should keep track of the RX data FIFO state.

Software can read the number of used RX data FIFO entries through the USED[3:0] bit field in the MMIO RX_DATA_FF_STATUS register.

In MMIO mode, software can read from RX data FIFO through the MMIO RX_DATA_FIFO_RD1, RX_DATA_FIFO_RD2, and RX_DATA_FIFO_RD4 registers.

- The RX_DATA_FIFO_RD1 register supports a read of a single byte from the FIFO.
  - For data transfer width smaller than eight or SDR mode a single byte is read from one FIFO entry.
  - For data transfer width of eight and DDR mode a single byte is read from the low byte of the FIFO entry, the high byte is discarded.
- The RX_DATA_FIFO_RD2 register supports a read of two bytes from the FIFO.
  - For data transfer width smaller than eight or SDR mode a single byte is read from each of two FIFO entries.
  - For data transfer width of eight and DDR mode two bytes are read from one FIFO entry.
- The RX_DATA_FIFO_RD4 register supports a read of four bytes from the FIFO.
  - For data transfer width smaller than eight or SDR mode a single byte is read from each of four FIFO entries.
  - For data transfer width of eight and DDR mode two bytes are read from each of two FIFO entries.

The MMIO interface logic gets the information of the transfer width and transfer rate mode (SDR/DDR) from the previous TX COUNT or RX COUNT command written to the TX command FIFO. Therefore, the related RX COUNT command must be written before reading the receive data from the RX DATA FIFO.

If the software attempts to read more bytes than available in the RX data FIFO, the BLOCK bit field in the MMIO CTL register specifies the behavior:

- If BLOCK is 0, an AHB-Lite bus error is generated.
- If BLOCK is 1, the AHB-Lite read transfer is extended until the bytes are available.

30.6.3 Interrupts and triggers

The SMIF has a single interrupt line. The following interrupt causes are associated with this interrupt:

- TR_TX_REQ triggers when SMIF_TX_TR_OUT is activated. This interrupt is generated only in MMIO mode.
- TR_RX_REQ triggers when SMIF_RX_TR_OUT is activated. This interrupt is generated only in MMIO mode.
- XIP_ADDR_ERROR is activated in XIP mode, if the selected device's DIV2 bit is set to '1' in the ADDR_CTL register and the AHB-Lite bus transfer address is not a multiple of 2. This interrupt is generated only in XIP mode.
- TX_CMD_FIFO_OVERFLOW is activated in MMIO mode on an AHB-Lite write transfer to the TX command FIFO (TX_CMD_FIFO_WR) with not enough free entries available.
- TX_DATA_FIFO_OVERFLOW is activated in MMIO mode on an AHB-Lite write transfer to the TX data FIFO (TX_DATA_FIFO_WR1, TX_DATA_FIFO_WR2, TX_DATA_FIFO_WR4) with not enough free entries available.
- RX_DATA_FIFO_UNDERFLOW is activated in MMIO mode on an AHB-Lite read transfer from the RX data FIFO (RX_DATA_FIFO_RD1, RX_DATA_FIFO_RD2, RX_DATA_FIFO_RD4) with not enough entries available.

The SMIF has two triggers that allow for DW/DMA controller functionality:

- SMIF_TX_TR_OUT is activated when the TX data FIFO has 1, 2, 4, or 8 free/available entries. This is a state trigger, and will be (automatically) deactivated when data elements are written into the FIFO.
- SMIF_RX_TR_OUT is activated when the RX data FIFO has 1, 2, 4, or 8 used/available entries. This is a state trigger, and will be (automatically) deactivated when data elements are read from the FIFO.

The triggers are activated only in MMIO mode of operation.

30.6.4 Cache

To improve XIP performance, the XIP AHB-Lite interface has a cache. The cache is defined as follows:

- 4-KB capacity.

- Read-only cache. Write transfers bypass the cache.
- Four-way set associative with an LRU replacement scheme. This cache design provides a better hit rate than a direct mapped cache design at the same cache capacity.
- Sequential cache design. The cache tag functionality is performed before the cache data access. A sequential cache design has lower power consumption than a parallel cache design.
- 256 B line/sector, with sixteen 16 B subsectors each. For a 4-KB capacity, this results in a total of 16 lines, distributed over four sets. The subsector design allows for low overhead tag information, as the 16 subsectors in a line/sector share the tag and only have dedicated valid bits.

For each read transfer, the cache tag structure is evaluated before the cache data structure is accessed. The subsector design results in a relatively low number of 16 lines. The 16 associated tags are implemented in flip-flops. The cache data structure is implemented using one 128-bit wide SRAM memory.

Each cache set has an associated 6-bit LRU field, which keeps track of the access history (from the least recently used to most recently used) of the lines in the set.

Each cache line has an associated cache tag and 16 valid bits: one valid bit for each subsector in the cache line. The cache tag identifies the location of the line in system memory.

- The address bits that identify a byte in a cache line are not part of the cache tag (byte address bits 7 down to 0).
- The address bits that identify a cache set are not part of the cache tag (byte address bits 9 and 8).
- The address bits that are not part of the SMIF XIP address region are not part of the tag.

The above omissions of address bits result in small tags. As a result, the cache tag structure can be evaluated quickly.

Figure 315 gives an overview of the cache design.

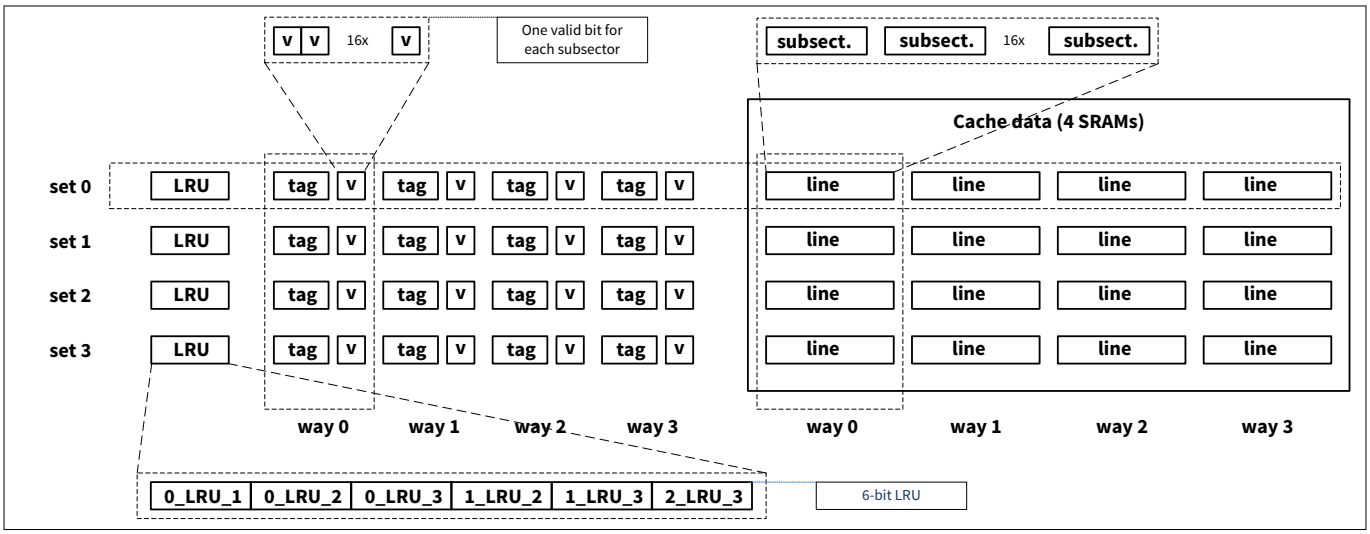


Figure 315 Cache organization

Read transfers that “hit” are processed by the cache.

Read transfers that “miss” result in a XIP SPI read transfer. A miss results in a 16 B (subsector) refill. The cache data structure is updated with the 16 B of refilled data. Two cases are considered:

- The refilled data is a subsector of a resident cache line. This case refills the data to the cache way that is used by the resident cache line. The subsector's valid field is set to '1' (the valid fields of all other subsectors in the cache line remain unchanged).
- The refilled data is not a subsector of a resident cache line. This case refills the data to the cache way that is identified by the LRU scheme. The cache line address bits are updated, and the subsector's valid field is set to '1' (the valid fields of all other subsectors in the cache line are set to '0'). Note that this case replaces a resident cache line.

The cache has a LRU replacement scheme. Each cache set has an associated 6-bit LRU field:

- LRU[5]: ‘1’ when way 0 is less recently used than way 1, '0' otherwise.
- LRU[4]: ‘1’ when way 0 is less recently used than way 2, '0' otherwise.
- LRU[3]: ‘1’ when way 0 is less recently used than way 3, '0' otherwise.
- LRU[2]: ‘1’ when way 1 is less recently used than way 2, '0' otherwise.
- LRU[1]: ‘1’ when way 1 is less recently used than way 3, '0' otherwise.
- LRU[0]: ‘1’ when way 2 is less recently used than way 3, '0' otherwise.

Although six bits allow for 26 = 64 bit patterns, only 4*3*2*1 = 24 bit patterns are legal LRU representations. The LRU set information is reset to all '1' or 0b111111, representing a set in which way 0 is less recently used than way 1, which is less recently used than way 2, which is less recently used than way 3. In this case, the line in way 0 is replaced when a new line is brought into the set. A line is made the most recently line of its set, when it is brought into the set, or when its line data is used because of an AHB-Lite read transfer.

A write to an address in the read-only cache, invalidates the associated cache subsector, but does not affect the LRU.

If ENABLED bit is set to '1' in the CA_CTL register, the cache is enabled and if ENABLED is set to '0', the cache is disabled.

If PREF_EN bit is set to '1' in the CA_CTL register, prefetching is enabled and if PREF_EN bit is set to '0', prefetching is disabled. If prefetch is enabled, a cache miss results in a 16 B (subsector) refill for the missing data and a 16 B prefetch for the next sequential data (independent of whether this data is already in the cache or not). The data of the 16 B prefetch is stored in a temporary buffer and only copied into the cache when a future read transfer “misses” in the cache and requires the buffered data.

For debug purposes, the tag and 16 valid bits of a cache line are readable through MMIO registers. The LRU information of a cache set is readable through MMIO registers.

30.6.5 Arbitration

The SMIF provides two AHB-Lite slave interfaces to CPUSS (one fast interface and one slow interface) or one AHB-Lite slave interface (slow interface) and one AXI slave interface. These AHB or AXI interfaces can generate XIP requests to the external memory devices.

30.6.6 Cryptography

In XIP mode, a cryptography component supports on-the-fly encryption for write data and on-the-fly decryption for read data. The use of on-the-fly cryptography is determined by a device's CRYPTO_EN bit field in the MMIO CTL register. In MMIO mode, the cryptography component is accessible through a MMIO register interface to support off-line encryption and decryption.

The rationale for this component is as follows: data is encrypted in the external memory devices and is not encrypted in the device. Therefore, SPI read and write data transfers require decryption and encryption functionality respectively. By storing data encrypted in the external memory devices (possibly nonvolatile devices), leakage of sensitive data is addressed.

Encryption and decryption are based on the AES-128 forward block cipher: advanced encryption standard block cipher with a 128-bit key. KEY[127:0] is a secret key, which is programmed into the MMIO CRYPTO_KEY3 to CRYPTO_KEY0 registers. These MMIO registers are software write-only: a software read returns “0”. By applying AES-128 with KEY[127:0] on a plaintext PT[127:0], we get ciphertext CT[127:0].

In XIP mode, the XIP address is used as the plaintext PT[]. The resulting ciphertext CT[] is used on-the-fly and not software accessible. The XIP address is extended with the MMIO CRYPTO_INPUT3 to CRYPTO_INPUT0 registers.

In MMIO mode, the MMIO CRYPTO_INPUT3 to CRYPTO_INPUT0 registers provide the plaintext PT[]. The resulting ciphertext CT[] is provided through the MMIO CRYPTO_OUTPUT3 to CRYPTO_OUTPUT0 registers.

Figure 316 illustrates the functionality in XIP and MMIO modes.

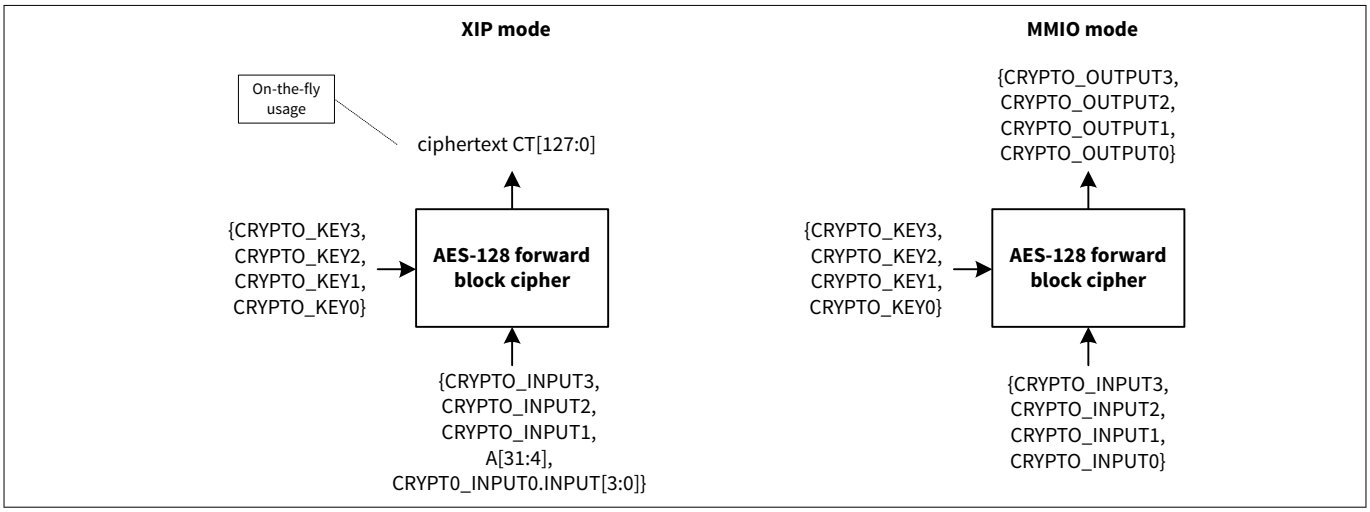


Figure 316 Cryptography in XIP and MMIO mode

In XIP mode, the resulting ciphertext CT[] is XOR'd with the SPI transfer's read data or write data. Note that the AES-128 block cipher is on the address of the data and not on the data itself. For SPI read transfers, this means that as long as the latency of the SPI transfer's read data is longer than the AES-128 block cipher latency, the on-the-fly decryption does not add any delay. Figure 317 illustrates the complete XIP mode functionality.

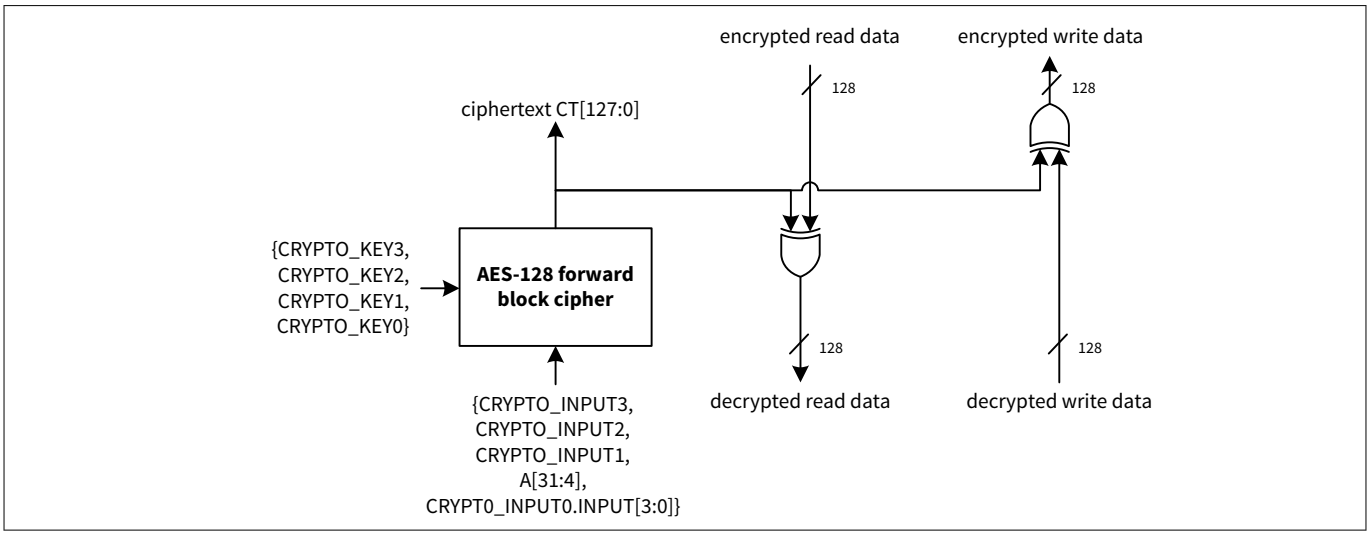


Figure 317 Cryptography in XIP mode

The SMIF supports on-the-fly cryptography. As a result, the external memory content is encrypted.

The encryption/decryption uses the AES block cipher with a 128-bit key in counter mode (CTR mode).

The CTR mode requires a nonce and a counter. The nonce is provided by a programmable SMIF MMIO register, and the system interconnect bus address is used as the counter (the lower 4 bits of the bus address are not used). The nonce and counter values are concatenated to provide the input to the block cipher.

The on-the-fly cryptography provides confidentiality for constant/read-only data in the external SPI memory devices. However, confidentiality is less for non-constant/write data. This is explained as follows. Consider an address A for which the block cipher output is AES(A). First, we store the plain data p0 as cipher data c0 = AES(A) ^ p0. Next, we store the plain data p1 as cipher data c1 = AES(A) ^ p1.

Both c0 and c1 can be observed on the device interface. We know that c0 ^ c1 = AES(A) ^ p0 ^ AES(A) ^ p1 = p0 ^ p1. If there are statistical information on plain data pi samples (for example, the plain data “0” is frequently written), we can deduce pi, if we have enough cipher data ci samples available.

If the SMIF on-the-fly cryptography is used for write data transfers, the nonce should be changed between write transfers to the same address to ensure confidentiality.

For dynamic data storage in the SMIF area with fixed CRYPTO_INPUT and KEY, SMIF block encryption is vulnerable to known-plaintext attacks (KPA).

Therefore, content in the SMIF memory area:

- For code storage, read-only and constant, the current scheme is good
- For data storage, the user can change CRYPTO_INPUT and KEY if necessary
- For sensitive data, the user can use mxcrypto for real AES encryption/decryption

30.6.6.1 Mapping of bytes

The plain text address, cipher text address, plain text data, and cipher data are represented as 128-bit values: PA[127:0], CA[127:0], PD[127:0], CD[127:0]. Each 128-bit value consists of sixteen bytes. The mapping between bytes and the 128-bit vector value V[127:0] is as follows:

- Vector byte 0: V[7:0]
- Vector byte 1: V[15:8]
- …
- Vector byte 15: V[127:120]

Given a XMC7000 address SoC_A[31:0], we derive an external device address A[m-1:0] = {SoC[m-1:4], 4'b0000}. The mapping between data bytes is as follows:

- Plain text byte 0 (PD[7:0]) is located at XMC7000 address {SoC_A[31:4], 4'b0000} and cipher text byte 0 (CD[7:0]) is located at external device address {A[m-1:4], 4'b0000}.
- Plain text byte 1 (PD[15:8]) is located at XMC7000 address {SoC_A[31:4], 4'b0001} and cipher text byte 1 (CD[15:8]) is located at external device address {A[m-1:4], 4'b0001}.
- …

Plain text byte 15 (PD[127:120]) is located at XMC7000 address {SoC_A[31:4], 4'b1111} and cipher text byte 15 (CD[127:120]) is located at external device address {A[m-1:4], 4'b1111}.

30.6.6.2 Software and MMIO register requirements

To ensure maximum protection of the XIP encryption key KEY[127:0], the following MMIO requirements should be met:

- The trusted software writes the encryption key in CRYPTO_KEY0 to CRYPTO_KEY3.
- The hardware provides “write only” access to the encryption key in CRYPTO_KEY0 to CRYPTO_KEY3. Software always reads CRYPTO_KEY0 to CRYPTO_KEY3 as “0” (in both XIP_MODE and MMIO_MODE).
- Software can read CRYPTO_RESULT0, to CRYPTO_RESULT3 in MMIO_MODE. It reads CRYPTO_RESULT0 to CRYPTO_RESULT3 as “0” in XIP_MODE.

To ensure that DAP test accesses cannot access decrypted data from the XIP address space, the following protection mode requirements should be met (these are the same requirements that apply for the SRAM memories in the CPUSS).

Table 257 XIP address space protection

Mode	Usage	XIP address space
VIRGIN	CPU, DW/DMA	Yes
	DAP	Yes (no write in privileged execution mode)
OPEN	CPU, DW/DMA	Yes

(table continues...)

Table 257 (continued) XIP address space protection

Mode	Usage	XIP address space
	DAP	Yes (no write in privileged execution mode)
PROTECTED	CPU, DW/DMA	Yes
	DAP	No
KILL	CPU, DW/DMA	Yes
	DAP	No
BOOT	CPU, DW/DMA	Yes
	DAP	No

***Note**: The XIP address space is a user memory (no privileged memory is present).*

30.6.6.3 Cryptography support for AXI interface

The AXI interface in XIP mode can generate the next read or write transfer request while the previous read or write transfer is currently processed by the memory interface logic. The exact time of this next transfer request is

- Whenever the next transfer is available at the AXI interface and
- The XIP block is ready to consume the next transfer request; that is, does not need the information (especially the address) of the previous transfer anymore.

To support data encryption/decryption for such outstanding transactions, the SMIF Crypto block contains:

- An input FIFO (with a depth of one entry), which acts as a buffer to keep the address (used as part of the plain text) stable for an encryption.
- An output FIFO (with a depth of two entries), which stores the encrypted address of the first transfer while the encrypted address for the next transfer is already calculated.

For read transfers, this allows the Crypto block to calculate the encrypted address of the next transfer while the previous memory transfer is currently in the data phase or even while the memory latency cycles are generated. This way the memory latency time can be used to calculate not only the current but also the next address encryption. This ensures that no delay is added by on-the-fly decryption for CM7 cache line fills (2 x 16-byte read transfers), even when using a fast HYPERBUS™ memory device with merged transfers (only 1 time the address and latency cycles).

30.6.7 Serial memory interface logic

30.6.7.1 TX and RX logic

The memory interface logic is implemented as two independent pieces:

- **The memory interface transmit logic:** This logic operates on the interface transmitter clocks clk_if_tx_div, clk_if_tx_data_out, and clk_if_tx_int. This logic is responsible for:
  - Generating and driving the sphb_clk_out clock.

- Driving the low active select signals, sphb_select_out[3:0].
- Driving the outgoing data on sphb_data_out[7:0].

- **The memory interface receiver logic:** This logic operates on the interface receiver clocks clk_if_rx_capture_ff, clk_if_rx_capture, clk_if_rx_capture_del[], and clk_if_rx_int. This logic captures incoming data on sphb_data_in[7:0] based on the following capture schemes:
  - (Legacy) output/feedback clock-based data capture.
  - RWDS-based data capture.

The different capture schemes are described in Data capture. Figure 318 shows an overview of the interface logic; the details are discussed in the following sections.

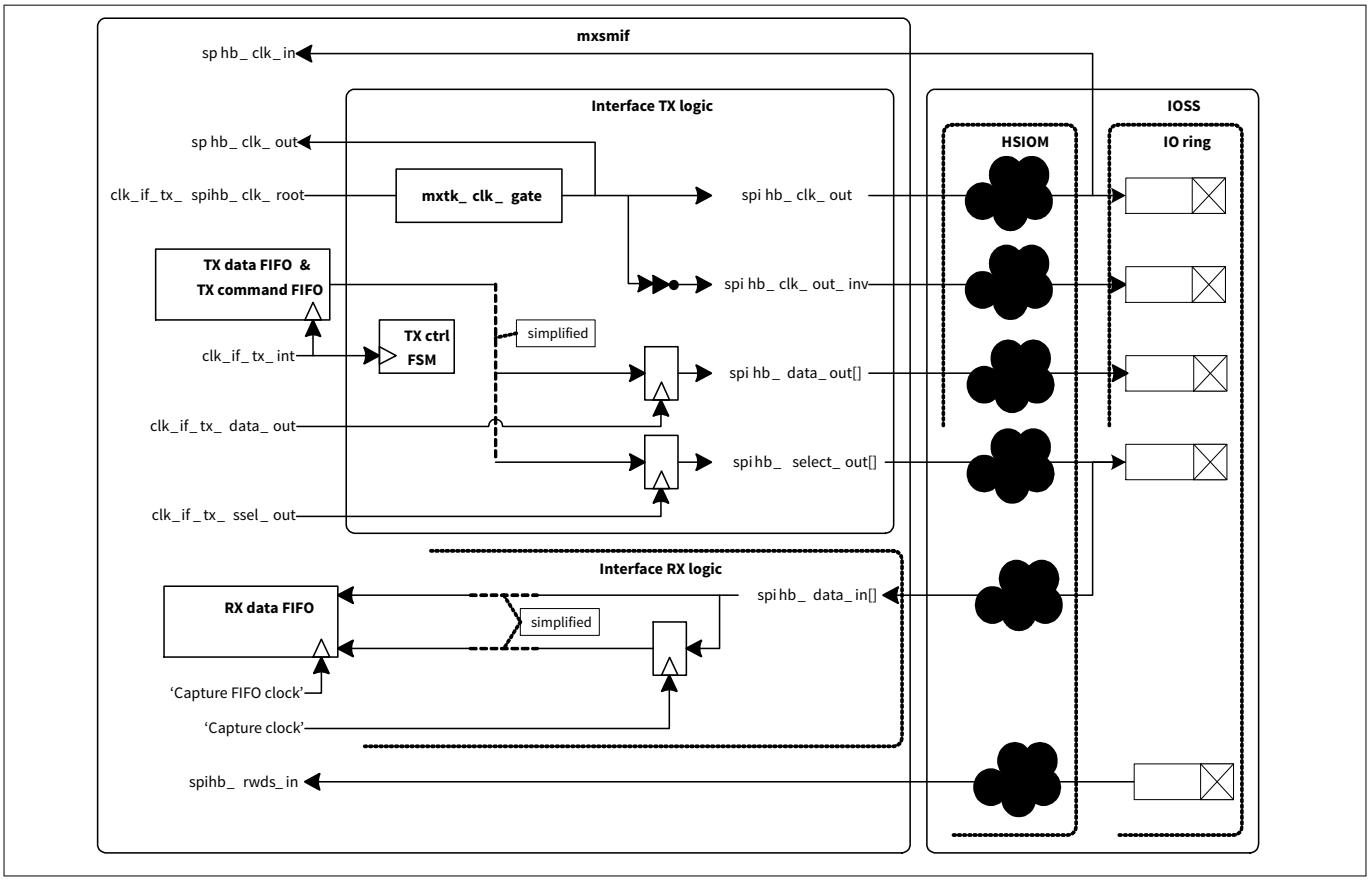


Figure 318 TX/RX interface to external memory devices

The TX command FIFO is used by both the memory interface transmit and receive logic. This ensures lockstep operation. Both pieces of logic are implemented as state machines that are driven by the TX command FIFO entries.

Note that the memory interface transmit logic clock generation of spihb_clk_out affects the memory interface receiver logic clocks. If spihb_clk_out is turned off, the memory interface receiver logic clocks will turn off.

The memory interface clock sphb_clk_out may turn off during a memory transfer in the following situations:

- The memory transfer produces read data (data from the memory device) and the RX data FIFO is full. To detect this situation, the TX interface transmitter logic needs to have access to the RX data FIFO level.
- The memory transfer consumes write data (data for the memory device) and the TX data FIFO is empty.

30.6.7.2 Flow control

Flow control prevents overflow and underflow for both directions – TX (SMIF writes to memory) and RX (SMIF reads from memory).

- TX underflow:
  - SMIF stops memory clock until TX cmd/data FIFO has data available
- TX overflow:
  - No flow control mechanism is provided at the memory interface. The memory device prevents TX overflow.
  - Flash devices receive data in write buffer at bus speed or use word write. This excludes overflow.
  - RAM writes the RAM array at the speed of external bus, which excludes overflow.
- RX overflow:
  - SMIF stops memory clock.
- RX underflow:
  - HYPERBUS™/OSPI memories stop RWDS/DQS. The TX interface logic needs to know how many memory interface clock cycles are required to generate without listening to RWDS/DQS. A synchronization from RX to TX interface clock domain causes a latency, which leads to overclocking (providing more memory clock cycles than needed). This leads to a potential mismatch between the number of cycles and therefore between the number of bytes/words read from memory and the number of bytes/words used in the SMIF. This creates an issue for any read side effects in the memory such as Bus CRC generation (potentially added to SMIF in the future).

    Therefore, the SMIF generates the number of latency cycles for initial and page boundary crossing based on MMIO registers (reflecting the memory requirements depending on memory interface clock frequency).The only exception is the variable initial latency for HYPERRAM™ based on the RWDS refresh indicator. This signal can be captured safely with the TX interface clock at the end of the address phase. This is because at that time this signal is stable for at least three memory interface clock cycles (per HYPERBUS™ protocol).
  - All other SPI memories have no flow control.

30.6.7.3 Data capture

SMIF supports data capturing with the SMIF output or output feedback clock for SDR and DDR timing:

- (Legacy) output/feedback clock-based data capture

Capture scheme supported by the SMIF:

- Internal clock-based data capture
- RWDS-based data capture

Table 258 shows the (RWDS) capture schemes required for serial memory.

Table 258 Transfer types and capture schemes

Memory bus	Feature	Loading	SMIF features for M7 core	
			Support for capture via	Frequency MHz
HYPERBUS™	HB	DDR	RWDS	100
xSPI, x=2,4,8 8-8-8	OSPI	DDR	RWDS	100
8-8-8	–	SDR	RWDS	100
1-8-8	Octal I/O	DDR	RWDS	100

(table continues...)

Table 258 (continued) Transfer types and capture schemes

Memory bus	Feature	Loading	SMIF features for M7 core	
			Support for capture via	Frequency MHz
1-8-8	–	SDR	RWDS	100
1-1-8	Octal data	DDR	RWDS	100
1-1-8	–	SDR	RWDS	100

In the following sections, the principles of the three capture schemes and their capture timing are discussed.

Output/feedback clock-based capture

The output/feedback clock-based capture scheme uses the memory output clock (spihb_clk_out), the inverted memory output clock, the memory output feedback clock (spihb_clk_in), or the inverted memory output feedback clock as the capture clock (clock selection CLOCK_IF_RX_SEL[3:0] in the CTL register).

To allow sampling later and to adjust the sample time with a finer granularity, the option of using the delay line for delaying the output or feedback clock has been added to this capture scheme. The delay line is already available for the RWDS capture scheme.

For SDR timing, any of the clock selections above can be used as capture clock (depending on the delay of the RX data). The data is directly captured in the RX data FIFO.

For DDR timing, the inverted version of the selected capture clock (memory output clock, memory output feedback clock, or an inverted and/or delayed version) also needs to be used as capture clock. The data driven in the first memory half-cycle is captured in the memory interface RX logic with this inverted selected clock intermediately. The data driven at the second memory half-cycle is directly captured by the RX data FIFO (as data low part); this and the first capture data are handed over to the RX data FIFO (as data high part). The RX data FIFO is 16-bit wide.

Figure 319 illustrates the output/feedback clock-based capture scheme.

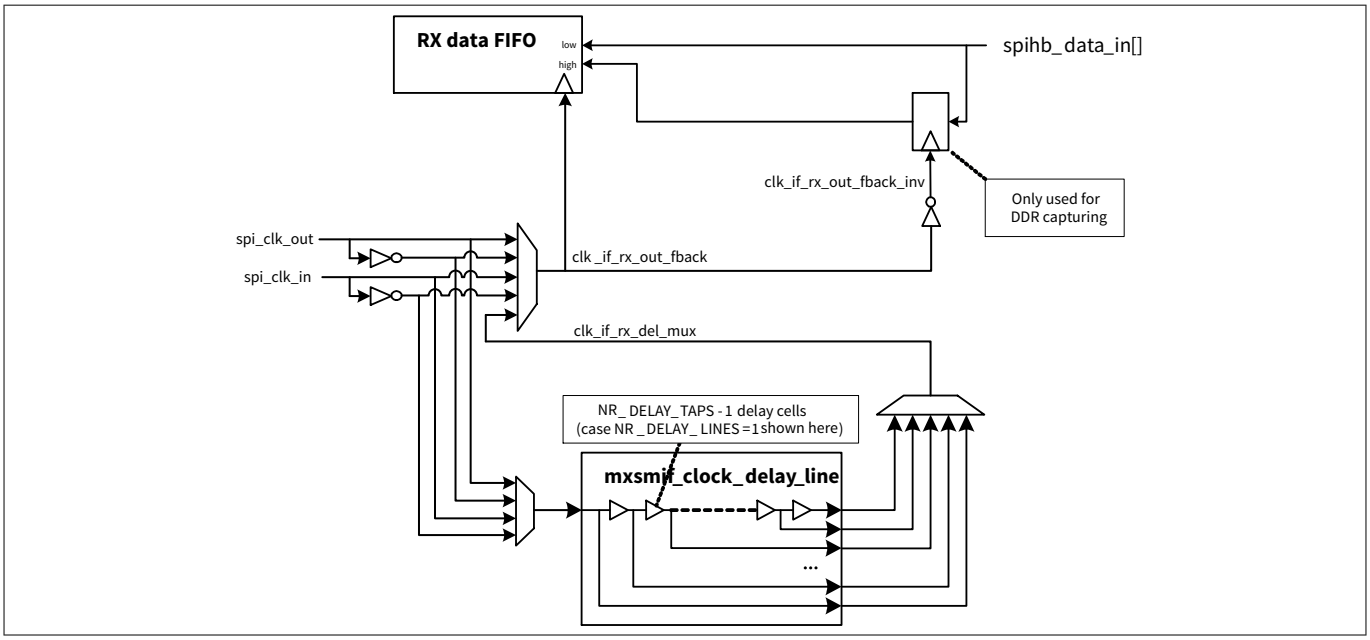


Figure 319 Output/feedback clock-based capture scheme

The output/feedback clock-based capture scheme includes an RX interface FSM, which tracks the memory protocol and generates a ‘capture enable’ signal for storing the received data in the RX data FIFO. This RX interface FSM runs with the same clock, clk_if_rx_out_fback.

RWDS-based capture

In the RWDS-based capture scheme, the RWDS signal is used as a clock to capture the input data. The RWDS signal edges are aligned with the start of the data valid time in the memory device.

The SMIF contains a clock multiplexer, which selects one of the delay line taps based on INT_CLOCK_DELAY_TAP_SEL0/1 registers. This allows it to calibrate the delay.

The calibration works as follows:

- The software can use a known data sequence located at an arbitrary address of the memory to “paint” the data eye. A flash image is known by the customer, in case of the RAM the data has to be written before. Even in the case that there is no known data sequence, software can read one safely by configuring a low-memory clock frequency and a late delay line tap (to meet hold timing).
- Now, software reads the data sequence for each delay tap setting using the SMIF MMIO mode. For some adjacent delay line taps it will receive the correct data.
- Note that the selectable delay per delay line tap can also be used in the RWDS-based capture scheme. Software can select that the delay per delay line tap in the CTL_DELAY_LINE_SEL register. The delay line with the most adjacent delay line taps resulting in correct received data is preferred.
- Then, it configures the middle tap of all the taps providing the correct data.

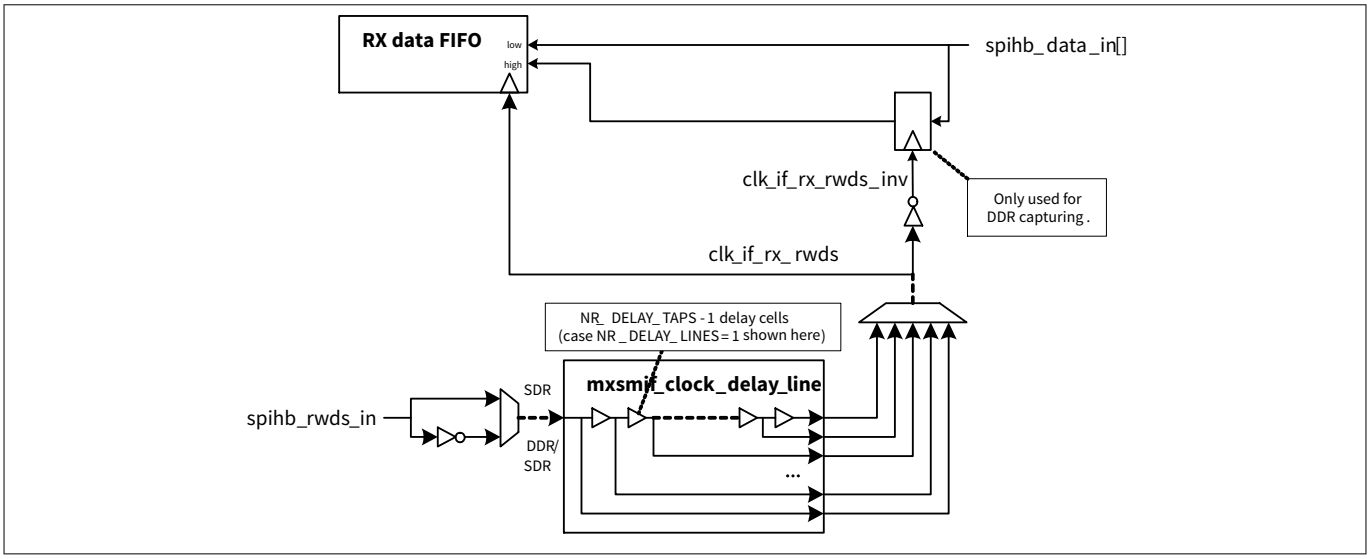


Figure 320 RWDS-based capture scheme

Figure 320 shows the timing of the RWDS-based capture scheme.

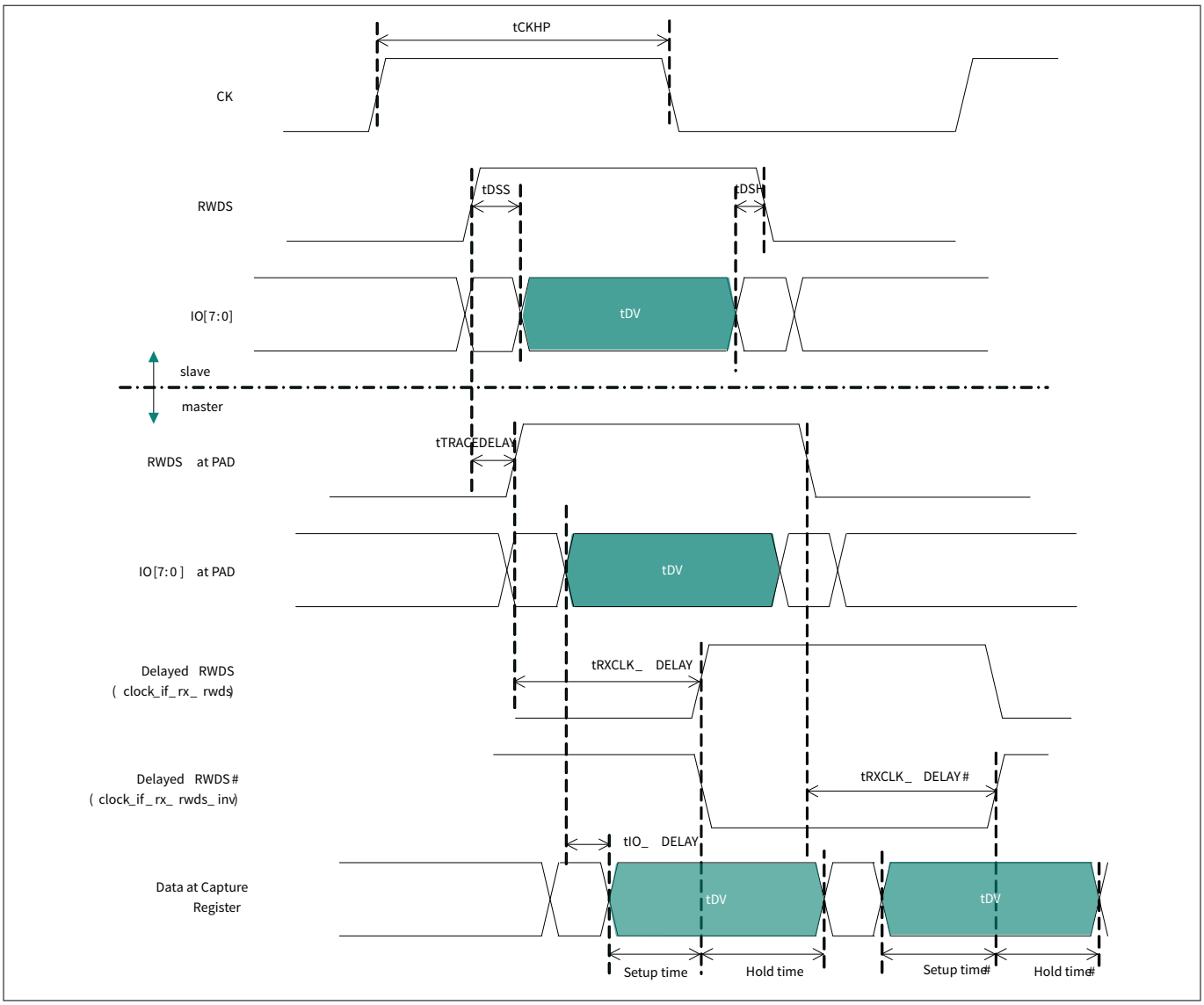


Figure 321 Capturing data with RWDS

Note that the RWDS signal (sphb_rwds) is not only used to capture data from the memory device, and in HYPERBUS™ mode when writing to and reading from the HYPERBUS™ RAM.

30.7 Registers

Table 259 List of SMIF registers

Register	Name	Description
SMIFx_CTL	Control	Control register
SMIFx_STATUS	Status	Busy status of AHB cache, AXI interface, cryptography, XIP device interface, and other logic in SMIF.
SMIFx_DELAY_TAP_SEL	RX Clock Delay Tap Select	Shift the strobe signal into the data eye.
SMIFx_TX_CMD_FIFO_STATUS	Transmitter Command FIFO Status	Number of entries used in the TX command FIFO.
SMIFx_TX_CMD_FIFO_WR	Transmitter Command FIFO Write	Command data
SMIFx_TX_DATA_FIFO_CTL	Transmitter Data FIFO Control	Determines when the TX data FIFO tr_tx_req trigger is activated.
SMIFx_TX_DATA_FIFO_STATUS	Transmitter Data FIFO Status	Number of entries used in the TX data FIFO.
SMIFx_TX_DATA_FIFO_WR1	Transmitter Data FIFO Write 1	Write a single byte to the TX FIFO.
SMIFx_TX_DATA_FIFO_WR2	Transmitter Data FIFO Write 2	Write two simultaneous bytes to the TX FIFO.
SMIFx_TX_DATA_FIFO_WR4	Transmitter Data FIFO Write 4	Write four simultaneous bytes to the TX FIFO.
SMIFx_TX_DATA_FIFO_WR1ODD	Transmitter Data FIFO Write	Write a single byte to the TX FIFO for odd byte addresses when using a word-based memory protocol (HYPERBUS™/ octal SPI).
SMIFx_RX_DATA_MMIO_FIFO_CTL	Receiver Data MMIO FIFO Control	Determines when RX data FIFO tr_rx_req trigger is activated.
SMIFx_RX_DATA_MMIO_FIFO_STATUS	Receiver Data MMIO FIFO Status	Number of entries used in the RX data MMIO FIFO.
SMIFx_RX_DATA_FIFO_STATUS	Receiver Data FIFO Status	Number of entries used in the RX data FIFO.
SMIFx_RX_DATA_MMIO_FIFO_RD1	Receiver Data MMIO FIFO Read 1	Provides a single byte from the RX data MMIO FIFO.
SMIFx_RX_DATA_MMIO_FIFO_RD2	Receiver Data MMIO FIFO Read 2	Provides two bytes from the RX data MMIO FIFO.
SMIFx_RX_DATA_MMIO_FIFO_RD4	Receiver Data MMIO FIFO Read 4	Provides four bytes from the RX data MMIO FIFO.
SMIFx_RX_DATA_MMIO_FIFO_RD1_SILENT	Receiver Data MMIO FIFO Silent Read	This register is similar to RX_DATA_MMIO_FIFO_RD1, with the exception that the entry is not removed from the FIFO.

(table continues...)

Table 259 (continued) List of SMIF registers

Register	Name	Description
SMIFx_SLOW_CA_CTL	Slow Cache Control	The slow interface is used for the CPUSS CM0+, Crypto, and DataWire components.
SMIFx_SLOW_CA_CMD	Slow Cache Command	Cache and prefetch buffer invalidation.
SMIFx_CRYPTO_CMD	Cryptography Command	Starts an encryption operation.
SMIFx_CRYPTO_INPUT0	Cryptography Input 0	Four bytes of the plaintext PT[31:0].
SMIFx_CRYPTO_INPUT1	Cryptography Input 1	Four bytes of the plaintext PT[63:32].
SMIFx_CRYPTO_INPUT2	Cryptography Input 2	Four bytes of the plaintext PT[95:64].
SMIFx_CRYPTO_INPUT3	Cryptography Input 3	Four bytes of the plaintext PT[127:96].
SMIFx_CRYPTO_KEY0	Cryptography Key 0	Four bytes of the key KEY[31:0].
SMIFx_CRYPTO_KEY1	Cryptography Key 1	Four bytes of the key KEY[63:32].
SMIFx_CRYPTO_KEY2	Cryptography Key 2	Four bytes of the key KEY[95:64].
SMIFx_CRYPTO_KEY3	Cryptography Key 3	Four bytes of the key KEY[127:96].
SMIFx_CRYPTO_OUTPUT0	Cryptography Output 0	Four bytes of the ciphertext CT[31:0].
SMIFx_CRYPTO_OUTPUT1	Cryptography Output 1	Four bytes of the ciphertext CT[63:32].
SMIFx_CRYPTO_OUTPUT2	Cryptography Output 2	Four bytes of the ciphertext CT[95:64].
SMIFx_CRYPTO_OUTPUT3	Cryptography Output 3	Four bytes of the ciphertext CT[127:96].
SMIFx_INTR	Interrupt	Indicates the interrupt causes.
SMIFx_INTR_SET	Interrupt Set	Write with '1' to set the corresponding bit in interrupt request register.
SMIFx_INTR_MASK	Interrupt Mask	Mask bit for the corresponding bit in interrupt request register.
SMIFx_INTR_MASKED	Interrupt Masked	Reflects a bitwise 'AND' between the Interrupt Request and Mask Registers.
SMIFx_DEVICEy_CTL	Device Control	Control Register
SMIFx_DEVICEy_ADDR	Device Region Base Address	Device region base address
SMIFx_DEVICEy_MASK	Device Region Mask	Device region mask
SMIFx_DEVICEy_ADDR_CTL	Address Control	Address control
SMIFx_DEVICEy_RD_CMD_CTL	Read Command Control	Read command control
SMIFx_DEVICEy_RD_ADDR_CTL	Read Address Control	Read address control
SMIFx_DEVICEy_RD_MODE_CTL	Read Mode Control	Read mode control
SMIFx_DEVICEy_RD_DUMMY_CTL	Read Dummy Control	Read dummy control
SMIFx_DEVICEy_RD_DATA_CTL	Read Data Control	Read data control
SMIFx_DEVICEy_RD_BOUND_CTL	Read Bound Control	Read bound control
SMIFx_DEVICEy_WR_CMD_CTL	Write Command Control	Write command control

(table continues...)

Table 259 (continued) List of SMIF registers

Register	Name	Description
SMIFx_DEVICEy_WR_ADDR_CTL	Write Address Control	Write address control
SMIFx_DEVICEy_WR_MODE_CTL	Write Mode Control	Write mode control
SMIFx_DEVICEy_WR_DUMMY_CTL	Write Dummy Control	Write dummy control
SMIFx_DEVICEy_WR_DATA_CTL	Write Data Control	Write data control

Note that overwriting the same value on each register has different effects; this is explained in the register map by the software access attributes. For SMIF registers, the following access restrictions must be mentioned:

- All Status registers are not SW-writable.
- INTR is SW-clear and HW-set (or set by writing '1' to INTR_SET).
- Read INTR_SET will return the value of INTR.
- Other registers are normal and can be overwritten with the same value.

31 SDHC host controller

The secure digital high capacity (SDHC) host controller in XMC7000 MCU allows interfacing with embedded multimedia card (eMMC)-based memory devices, secure digital (SD) cards, and secure digital input output (SDIO) cards. Figure 322 illustrates a typical application using the SDHC block.

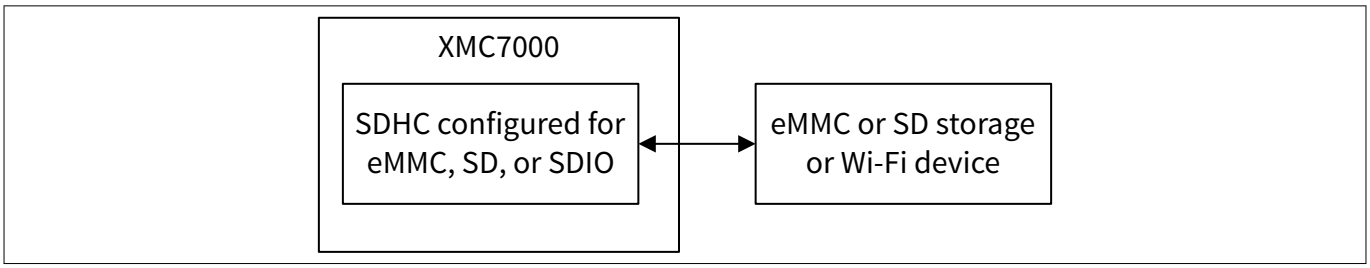


Figure 322 Typical SDHC application

31.1 Features

- Complies with eMMC 5.1, SD 6.0, and SDIO 4.10 standards
- Supports host controller interface (HCI) 4.2 shared by eMMC and SD
- SD interface supports 1-bit and 4-bit bus interfaces, and the following speed modes. The specified data rate is for a 4-bit bus.
  - 3.3-V signal voltage: Default speed (12.5 MB/s at 25 MHz) and high speed (25 MB/s at 50 MHz)
- eMMC interface supports 1-bit, 4-bit, and 8-bit bus interfaces, and the following speed modes. The specified data rate is for an 8-bit bus.
  - Legacy (26 MB/s at 26 MHz), high-speed SDR (52 MB/s at 52 MHz), and high-speed DDR (104 MB/s at 52 MHz)
- Supports three DMA modes – SDMA, ADMA2, and ADMA3 – through a dedicated DMA engine
- Provides 1 KB SRAM for buffering up to two 512-byte blocks
- Provides I/O interfaces for functions such as card detection and mechanical write protection

31.1.1 Features not supported

The SDHC block does not support the following features.

- SD/SDIO operation in UHS-II mode
- Command queuing engine (CQE)
- eMMC boot operation in dual data rate mode
- Read wait operation by DAT[2] signaling in an SDIO card
- Suspend/resume operation in an SDIO card
- Interrupt input pins for embedded SD systems
- SPI protocol mode of operation
- SD UHS-I mode using 1.8-V signal voltage: SDR, SDR25, SDR50, and DDR50

31.2 Block diagram

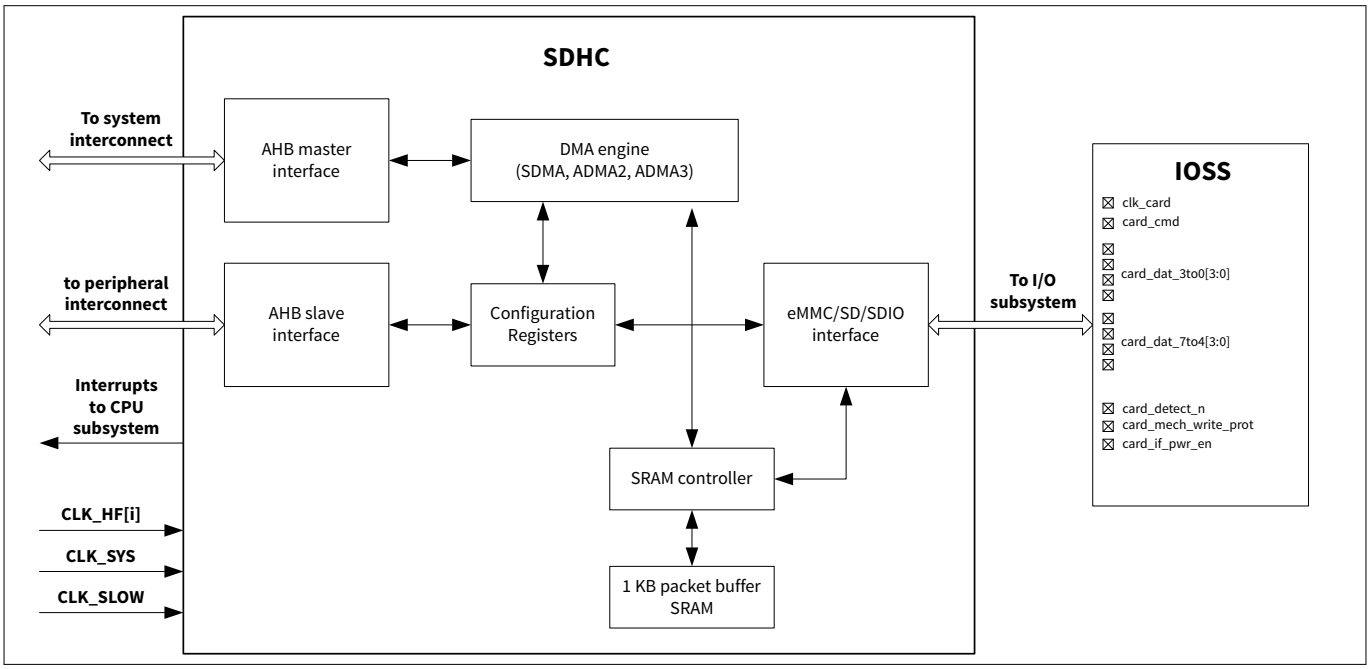


Figure 323 SDHC block diagram

The SDHC block supports all three interfaces – SD, SDIO, and eMMC. It does not have the eMMC reset signal, which is not a mandatory signal for eMMC operation. The AHB master interface helps to transfer data to and from the system memory and the AHB slave interface provides access to the Configuration Registers. The register set comprises the standard SD Host Controller Interface (HCI) Registers as specified in the SD specifications part A2 SD host controller standard specification. These registers are described in the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual. The DMA engine handles direct data transfer between the SDHC logic and system memory. It supports SDMA, ADMA2, and ADMA3 modes based on the configuration.

The SDHC block complies with the following standards. Refer to the specifications documents for more information on the protocol and operations.

- SD specifications part 1 physical layer specification version 6.00
- SD specifications part A2 SD host controller standard specification version 4.20
- SD specifications part E1 SDIO specifications version 4.10
- Embedded multi-media card (eMMC) electrical standard 5.1

31.3 Clocking

Table 260 lists the different clocks used in the SDHC block. While configuring the clock for SDHC make sure that CLK_SLOW ≥ CLK_GRx ≥ CLK_CARD.

Table 260 Clocks in SDHC

Source	SDHC clock	Function
CLK_SLOW	Core SDHC lock	Used for core SDHC functions including the packet buffer SRAM; it is sourced from the slow clock (CLK_SLOW); it must be ≥ AHB slave clock.
	AHB master interface clock	Used by the AHB master interface; it is sourced from the slow clock (CLK_SLOW); it must be ≥ AHB slave clock.

(table continues...)

Table 260 (continued) Clocks in SDHC

Source	SDHC clock	Function
CLK_GRx	AHB slave interface clock	Used by the AHB slave interface; it is clocked by the PERI group clock (CLK_GRx); it must be ≥ CLK_CARD. The group clock is derived from the PERI clock (CLK_PERI) using a divider. Because this divider can remain at the default value of '1' for most applications, CLK_PERI can be considered as CLK_GRx for SDHC. See Clocking system for information on CLK_GRx and CLK_PERI.
CLK_HFx	Base clock/card clock	Used for sourcing the SD/eMMC interface clock (CLK_CARD); it is derived from CLK_HFx; it must be set to 100 MHz to be compatible with the Capabilities register. See Base clock (CLK_HFx) configuration for details. See High-frequency root clocks to know which CLK_HFx drives an SDHC instance.
	Timer clock	Used for command and data timeout functions; it is derived from CLK_HFx.

31.3.1 Clock gating

All the clocks except the slave interface clock can be gated internally to enter standby mode (See Power modes). In standby mode, you can also stop the clocks externally if required. The slave clock cannot be gated because it is used for wakeup logic (see Interrupts to CPU) during the standby mode. The card clock is gated by clearing the SDHCx_CORE_CLK_CTRL_R.SD_CLK_EN bit and other clocks are gated by clearing the SDHCx_CORE_CLK_CTRL_R.INTERNAL_CLK_EN bit. See Clock setup for the sequence to be followed while modifying this bit.

31.3.2 Base clock (CLK_HFx) configuration

The SDHCx_CORE_CAPABILITIES1_R Register has a read-only field (BASE_CLK_FREQ) to indicate the base clock frequency so that an SD HCI-compatible driver can easily configure the divider for the required bus speed. This value is set to 0x64 (100 MHz) and hence CLK_HFx must be set to 100 MHz. If this compatibility is not required, CLK_HFx can be set to any value. See Timeout (TOUT) configuration.

31.3.3 Card clock (SDCLK) configuration

The SDCLK or card clock frequency is set by configuring the 10-bit divider in SDHCx_CORE_CLK_CTRL_R and selecting the 10-bit divided clock mode by clearing the SDHCx_CORE_CLK_CTRL_R.CLK_GEN_SELECT bit. The default value of this bit is zero. The SDHCx_CORE_CLK_CTRL_R.UPPER_FREQ_SEL field holds the upper two bits (9:8) and the SDHCx_CORE_CLK_CTRL_R.FREQ_SEL field holds the lower eight bits (7:0) of the divider. Base clock frequency is sourced from CLK_HFx as explained in Table 260. SDCLK frequency is equal to base clock frequency when the divider value is zero.

SDCLK frequency = Base clock frequency / (2 × 10-bit divider value)

These fields are set automatically, based on the selected Bus Speed mode, to a value specified in one of the preset registers when SDHCx_CORE_HOST_CTRL2_R.PRESET_VAL_ENABLE is set. The preset registers are selected according to Table 261.

31.3.4 Timeout (TOUT) configuration

An internal timer is used for command and data timeouts. The timeout value is specified through the SDHCx_CORE_TOUT_CTRL_R.TOUT_CNT Register field. The timer clock (TMCLK) frequency indicated by the SDHCx_CORE_CAPABILITIES1_R.TOUT_CLK_FREQ and SDHCx_CORE_CAPABILITIES1_R.TOUT_CLK_UNIT read-

only fields is '1' MHz. Timer clock is derived by dividing the CLK_HFx, which means that CLK_HFx must be set to 100 MHz to be compatible with the Capabilities register.

31.4 Bus speed modes

The SDHC block can operate in either SD/SDIO mode or in eMMC mode. The SDHC block operates in eMMC mode when the SDHCx_CORE_EMMC_CTRL_R.CARD_IS_EMMC bit is set; otherwise, it operates in SD/SDIO mode. The speed mode is configured through SDHCx_CORE_HOST_CTRL1_R and SDHCx_CORE_HOST_CTRL2_R Registers as shown in Table 261. The SDHCx_CORE_HOST_CTRL2_R.UHS2_IF_ENABLE bit should remain at its default value of zero because the block does not support UHS-II mode. The card clock must be configured according to the selected speed mode through the SDHCx_CORE_CLK_CTRL1/2_R Register. See Card clock (SDCLK) configuration for more information.

Table 261 Bus speed mode configuration

Bus speed mode	SDHCx_CORE_HOST_CTRL1_R field	SDHCx_CORE_HOST_CTRL2_R Fields		Selected preset Register
	HIGH_SPEED_EN	SIGNALING_EN	UHS_MODE_SEL	
SD default speed (DS)	0	0	Don't care	PRESET_DS_R
SD high speed (HS)	1	0	Don't care	PRESET_HS_R
eMMC legacy	Don't care	0	000b	PRESET_SDR12_R
eMMC high speed	Don't care	0	001b	PRESET_SDR25_R
eMMC high speed DDR	Don't care	0	100b	PRESET_DDR50_R

31.5 Power modes

The block can operate during active and sleep system power modes. It does not support Deep sleep mode and cannot wake up from events such as card insertion and removal when the system is in deep sleep. Only the SDHCx_WRAP_CTL.ENABLE Register is retained when the system enters Deep sleep mode and the SRAM is switched off to save power. Make sure that no AHB traffic (such as register read/write and DMA operation) is present, the SD/SDIO/eMMC bus interface is idle, and no data packets are pending in the packet buffer SRAM when the system transitions into Deep sleep mode.

31.5.1 Standby mode

The block can be put into standby mode to save power during the active and sleep system power modes by turning off the clocks. See Clock gating for details. The block can detect wakeup interrupts (see Interrupts to CPU) in standby mode.

31.6 Interrupts to CPU

The block provides two interrupt signals to CPUSS:

- Wakeup interrupt signal – Triggered on events such as card insertion, removal, and SDIO card interrupt. This interrupt source cannot wake up the system from Deep sleep mode and is provided so that a host driver can take appropriate action on those events. For example, resuming operation from standby mode on card insertion. See Standby mode for details. As card insertion and removal is not applicable to an embedded device, wakeup interrupt should not be used in this case. However it can still be used for SDIO card interrupt.
- General interrupt signal – Triggered on all other events, in either normal conditions or error conditions.

A host driver must not enable the wakeup and general interrupt signals at the same time.

To use only the wakeup interrupt signal, clear the SDHCx_CORE_NORMAL_INT_STAT_R and SDHCx_CORE_NORMAL_INT_SIGNAL_EN_R Registers, and then set the enable bits of the required wakeup events in the SDHCx_CORE_WUP_CTRL_R and SDHCx_CORE_NORMAL_INT_STAT_EN Registers.

To use only the general interrupt signal, clear the SDHCx_CORE_WUP_CTRL_R and SDHCx_CORE_NORMAL_INT_STAT_R Registers. Then, set the required bits in SDHCx_CORE_NORMAL_INT_SIGNAL_EN_R and SDHCx_CORE_NORMAL_INT_STAT_EN Registers.

These interrupts remain asserted until the CPU clears the interrupt status through one of the Status Registers – SDHCx_CORE_NORMAL_INT_STAT_R and SDHCx_CORE_ERROR_INT_STAT_R.

The SDIO card interrupt status bit, SDHCx_CORE_NORMAL_INT_STAT_R.CARD_INTERRUPT, is a read-only bit. The host driver may clear the SDHCx_CORE_NORMAL_INT_STAT_EN_R.CARD_INTERRUPT_STAT_EN bit before servicing the SDIO card interrupt and may set this bit again after all interrupt requests from the card are cleared to prevent inadvertent interrupts.

Following is the list of registers used in interrupt configuration.

Table 262 Interrupt Control Registers

Register	Description
SDHCx_CORE_WUP_CTRL_R	Enables or disables different wakeup interrupts. Host driver must maintain voltage on the SD bus by setting SDHCx_CORE_PWR_CTRL_R.SD_BUS_PWR_VDD1 bit for these interrupts to occur. These interrupts cannot wakeup the device from deep sleep.
SDHCx_CORE_NORMAL_INT_STAT_R	Reflects the status of wakeup interrupts and non-error general interrupts. It also has a bit to indicate whether any of the bits in SDHCx_CORE_ERROR_INT_STAT_R is set.
SDHCx_CORE_ERROR_INT_STAT_R	Reflects the status of general interrupts that are triggered by error conditions.
SDHCx_CORE_NORMAL_INT_STAT_EN_R	Provides mask bits for wakeup interrupts and non-error general interrupts.
SDHCx_CORE_ERROR_INT_STAT_EN_R	Provides mask bits for general interrupts that are triggered by error conditions.
SDHCx_CORE_NORMAL_INT_SIGNAL_EN_R	Setting any of these bits to ‘1’ enables interrupt generation for wakeup interrupts and non-error general interrupts.
SDHCx_CORE_ERROR_INT_SIGNAL_EN_R	Setting any of these bits to ‘1’ enables interrupt generation for general interrupts that are triggered by error conditions.
SDHCx_CORE_FORCE_ERROR_INT_STAT_R	Forces an error interrupt to occur when the corresponding bit is set.

31.6.1 SDIO interrupt

The SDIO interrupt function is supported on SDHCx_CARD_DAT_3TO0_[1] line (SD pin 8). See the SDIO specifications for details on this feature. The SDHCx_CORE_NORMAL_INT_STAT_EN_R.CARD_INTERRUPT_STAT_EN bit and the SDHCx_CORE_NORMAL_INT_SIGNAL_EN_R.CARD_INTERRUPT_SIGNAL_EN bit must be set to enable this interrupt. To use this interrupt as wakeup interrupt, use SDHCx_CORE_WUP_CTRL_R.WUP_CARD_INT instead of SDHCx_CORE_NORMAL_INT_SIGNAL_EN_R.

31.7 I/O interface

SDHC block provides the signals shown in Table 263, which can be routed to pins through the I/O subsystem (IOSS). Refer to the I/O system to configure the I/Os, and the device datasheet for specific pins available for each signal. SDHC also supports SDIO interrupt on DAT[1] line (SDHCx_CARD_DAT_3TO0_[1]). The output signals must be configured in strong drive mode, bi-directional signals in strong drive with the input buffer ON, and the input pins in high-impedance mode when an external pull-up resistor is available; otherwise, they must be configured in internal pull-up mode. Input buffer must be enabled for the input pins. The drive mode of the DAT lines must be set to high impedance after card removal. See Card detection for details. In addition to configuring the drive mode and HSIOM Registers in IOSS, the SDHCx_CORE_GP_OUT_R Register must be configured to enable the required signals. See Table 263. The SDHCx_CARD_DETECT_N and SDHCx_CARD_MECH_WRITE_PROT should be connected to ground if an eMMC or an embedded SDIO device is connected.

Table 263 I/O signal interface

Signal	Function	Register configuration
SDHCx_CLK_CARD	Clock output	SDHCx_CORE_GP_OUT_R.CARD_CLOCK_OE
SDHCx_CARD_CMD	Command (bi-directional)	Always enabled
SDHCx_CARD_DAT_3TO0_[3:0]	Data (bi-directional)	SDHCx_CORE_HOST_CTRL1_R.DAT_XFER_WIDTH
SDHCx_CARD_DAT_7TO4_[3:0]	Data (bi-directional)	SDHCx_CORE_HOST_CTRL1_R.EXT_DAT_XFER
SDHCx_CARD_DETECT_N	Card detect signal input, active low	SDHCx_CORE_GP_OUT_R.CARD_DETECT_EN
SDHCx_CARD_MECH_WRITE_PROT	Mechanical write protect signal input, active low	SDHCx_CORE_GP_OUT_R.CARD_MECH_WRITE_PROT_EN
SDHCx_IF_PWR_EN	Card interface power enable output	SDHCx_CORE_GP_OUT_R.CARD_IF_PWR_EN_OE

31.8 Packet buffer SRAM

SRAM that is internal to the SDHC block is used as a packet buffer to store data packets while carrying out data transfer to and from the card. The size of the SRAM is 1 KB to support buffering of two 512 bytes blocks. As write and read transfers to the cards do not occur simultaneously, a single shared buffer is used for read and write operations. During the data transfer command handshake, the read/write bit of the command register is sampled and stored. This internal bit defines whether the SDHC is in read or write mode.

Figure 324 shows how data flows from the card interface to the AHB master interface through the packet buffer for a card read transfer. Received data from the card interface is written into packet buffer. When one block of data is received, DMA starts transmitting that data to the system by reading it from the packet buffer. For a card write transfer, data flows in the reverse direction. DMA writes data into a packet buffer that is subsequently read by the card interface logic. DMA and card interface logic can work simultaneously because read and write to packet buffer can be interleaved. For card read, DMA can send out the previous block while card interface logic is receiving the current block. For card write, DMA can write the current block into packet buffer while card interface logic is sending out the previous block.

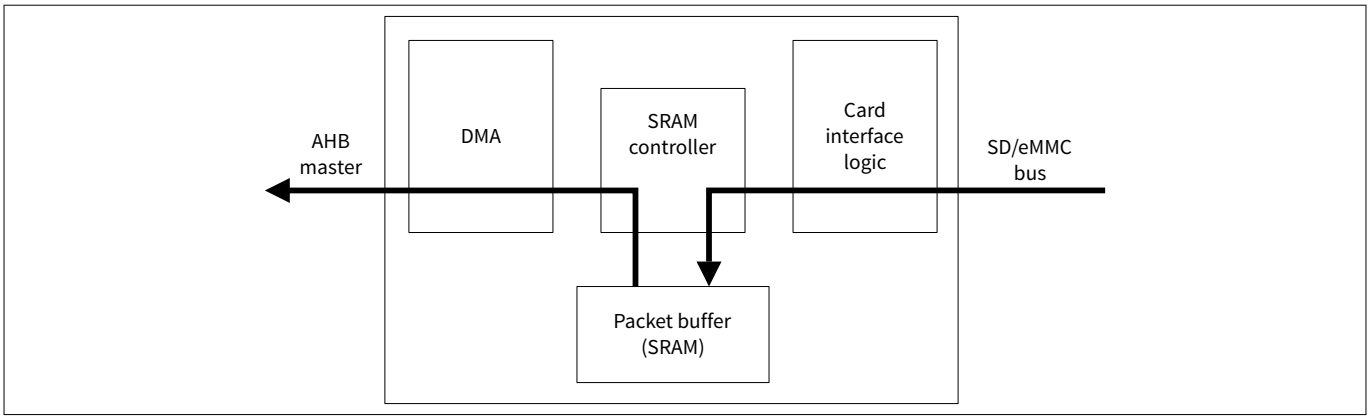


Figure 324 Data flow in a read transfer

31.8.1 Packet buffer full/empty

When the packet buffer becomes full in card read, the clock to the card is stopped to prevent the card from sending the next data block. When packet buffer is empty, data block is not sent. In both cases, card interface logic is idle. SDHC does not support SDIO Read Wait signaling through DAT[2]. Therefore, the I/O command (CMD52) cannot be performed during a multiple read cycle because the card clock is stopped.

31.9 DMA engine

The DMA engine handles data transfer between SDHC and system memory. Following are the features of this unit:

- Supports SDMA, ADMA2, and ADMA3 modes based on the configuration.
- The same DMA engine is used to interleave data transfer and descriptor fetch. This enables new task descriptor fetches (for CMD44 and CMD45) while DMA is moving data during task execution (for CMD46 and CMD47).
- Prefetches data for back-to-back eMMC write commands.
- Writes back the received data packets to system memory.

Figure 325 shows the data flow between the host driver and SD bus. The host driver can transfer data using either a programmed I/O (PIO) method in which the internal buffer is accessed through the buffer data port (SDHCx_CORE_BUF_DATA_R) Register or using any of the defined DMA methods. PIO mode is much slower and burdens the processor. Do not use the PIO mode for large transfers.

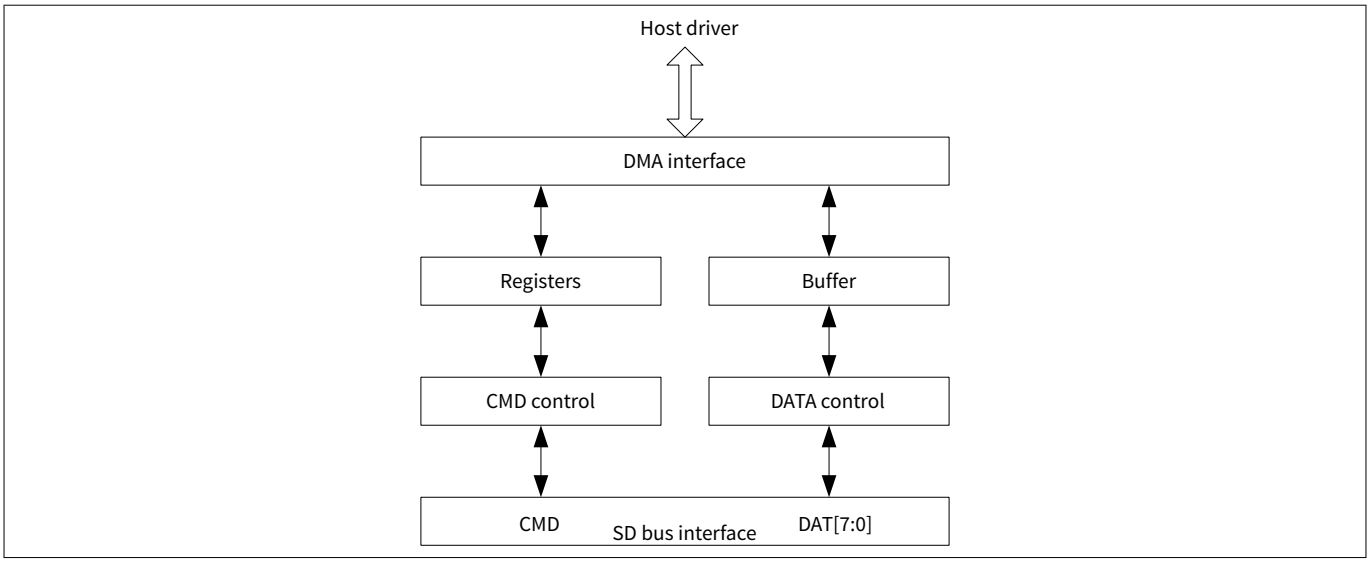


Figure 325 Data flow

DMA supports both single block and multi-block transfers. The control bits in the block gap control (SDHCx_CORE_BGAP_CTRL_R) Register is used to stop and restart a DMA operation. SDMA mode is used for short data transfer because it generates interrupts at page boundaries. These interrupts disturb the CPU to reprogram the new system address. Only one SD command transaction can be executed for every SDMA operation.

ADMA2 and ADMA3 are used for long data transfers. They adopt scatter gather algorithm so that higher data transfer speed is available. The host driver can program a list of data transfers between system memory and SD card to the descriptor table. ADMA2 performs one read/write SD command operation at a time. ADMA3 can program multiple read/write SD command operation in a descriptor table.

In SDMA and ADMA2 modes, writing the SDHCx_CORE_CMD_R Register triggers the DMA operation. In ADMA3 mode, writing SDHCx_CORE_ADMA_ID_LOW_R Register triggers the DMA operation.

SD mode commands are generated by writing into the following registers – system address (SDHCx_CORE_SDMASA_R), block size (SDHCx_CORE_BLOCKSIZE_R), block count (SDHCx_CORE_BLOCKCOUNT_R), transfer mode (SDHCx_CORE_XFER_MODE_R), and command (SDHCx_CORE_CMD_R). When SDHCx_CORE_HOST_CTRL2_R.HOST_VER4_EN = 0, SDMA uses SDHCx_CORE_SDMASA_R as system address register and hence Auto CMD23 cannot be used with SDMA because this register is assigned for Auto CMD23 as the 32-bit block count register. When SDHCx_CORE_HOST_CTRL2_R.HOST_VER4_EN = 1, SDMA uses SDHCx_CORE_ADMA_SA_LOW_R as system address register and SDHCx_CORE_SDMASA_R is reassigned to 32-bit block count and hence SDMA may use Auto CMD23.

To use the 32-bit block count register when SDHCx_CORE_HOST_CTRL2_R.HOST_VER4_EN = 1, it must be programmed with a non-zero value and the value of the 16-bit block count register SDHCx_CORE_BLOCKCOUNT_R must be zero. Refer to the respective specifications documents listed in Block diagram to learn more about the DMA operation.

31.10 Initialization sequence

Figure 326 shows the sequence for initializing SDHC to work with SD/SDIO/eMMC cards. Subsequent sections describe each step. After initialization, SDHC is ready to communicate with the card. Refer to the corresponding specifications document for information on other sequences such as card initialization and identification, changing bus speed mode, signal voltage switch procedure, transaction generation, and error recovery.

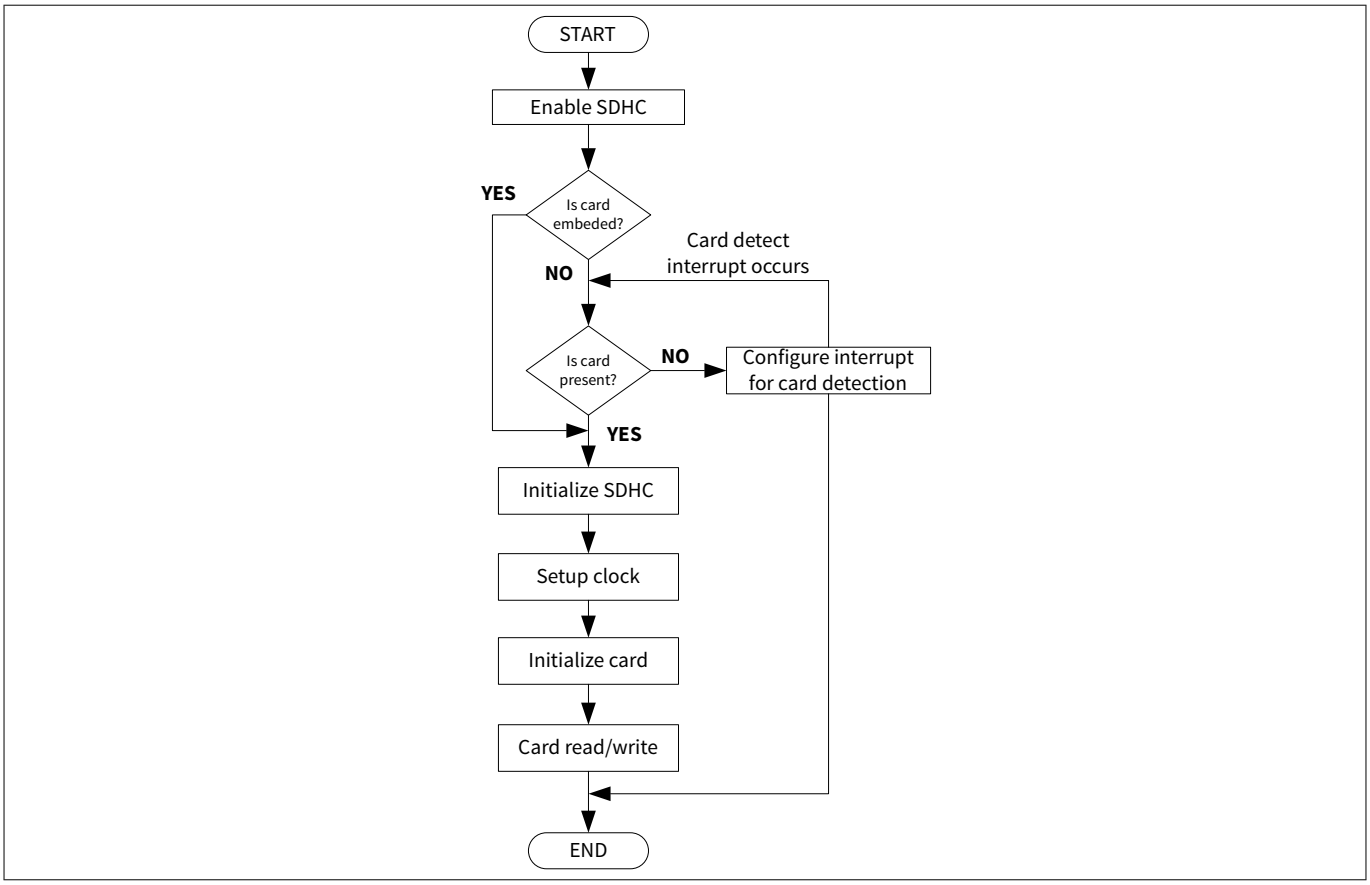


Figure 326 SDHC programming sequence

31.10.1 Enabling SDHC

Ensure CLK_GRx is configured to be greater than or equal to CLK_CARD and is running. Then, follow the sequence in Figure 327 to enable the block. The internal clock can also be enabled later during clock setup. It must be enabled to detect card insertion or removal through general interrupts when SDHC is not in standby mode. See Card detection for details.

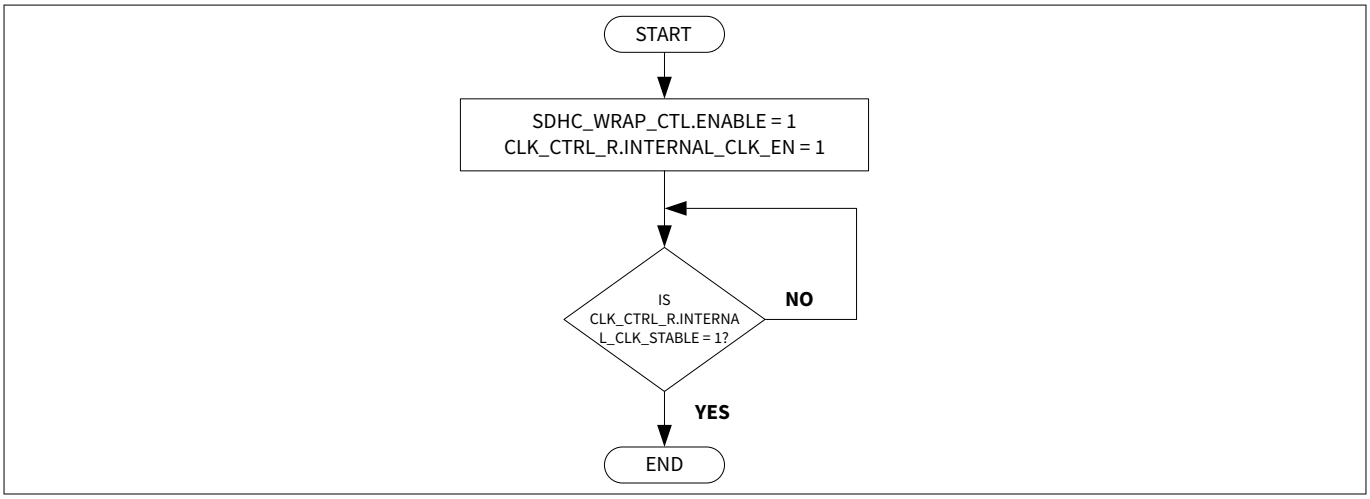


Figure 327 SDHC enable sequence

31.10.2 Card detection

Check if the card is already inserted by following the sequence shown in Figure 328. This step is required for a removable card. After the card is detected, the host driver can supply power and clock to the card. If the card is inserted, then proceed with SDHC initialization. To detect card insertion or removal through interrupt when the internal clock is already enabled, follow the sequence shown in Figure 329. To detect the card status through interrupt when the internal clock is disabled (when SDHC is in standby mode), the bits in the SDHCx_CORE_WUP_CTRL_R Register must be set and the SDHCx_CORE_NORMAL_INT_SIGNAL_EN Register must be cleared. See Interrupts to CPU for details. To detect SDIO card interrupt on DAT[1] line, a separate bit is provided in these registers, which must be configured.

SDHC clears the SDHCx_CORE_PWR_CTRL_R.SD_BUS_PWR_VDD1 bit when the card is removed and drives the DAT lines low. Therefore, the drive mode of the DAT lines must be changed from strong (with input buffer ON) to HI-Z when the card is removed to keep the lines pulled high. After detecting card insertion, the drive mode must be configured back to strong (with input buffer ON) mode only after SDHCx_CORE_PWR_CTRL_R.SD_BUS_PWR_VDD1 is set to 1.

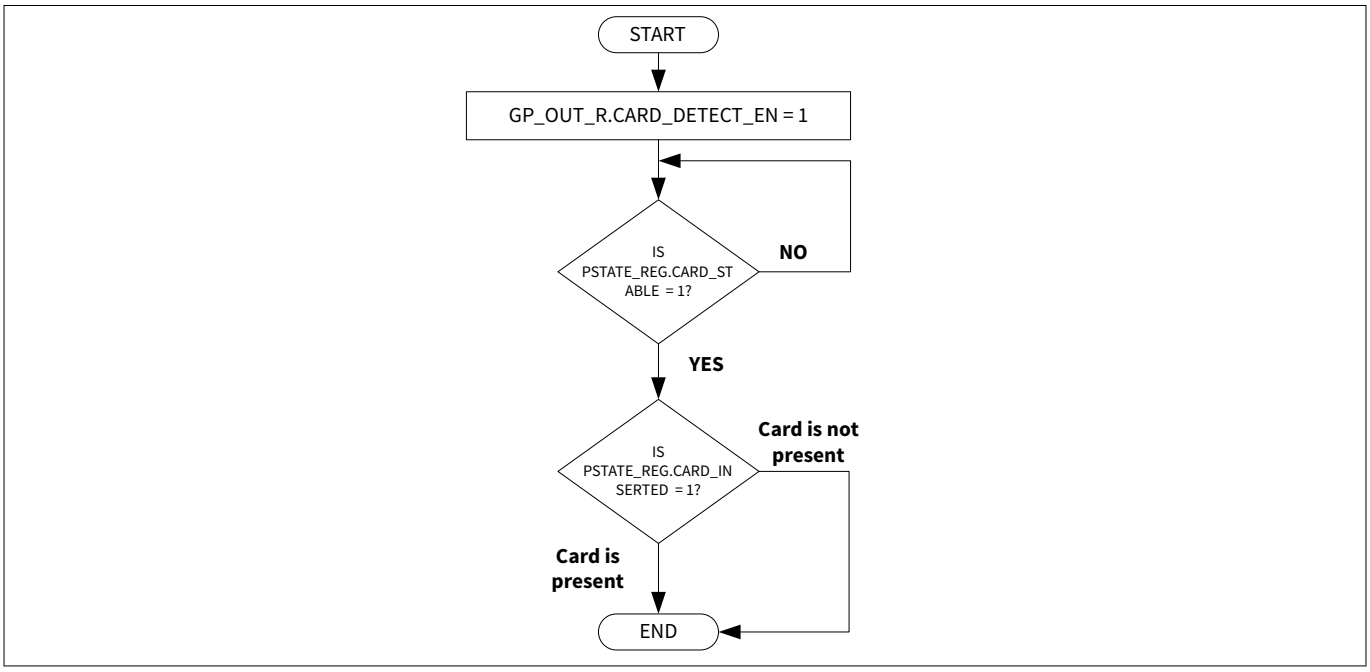


Figure 328 Card status check sequence

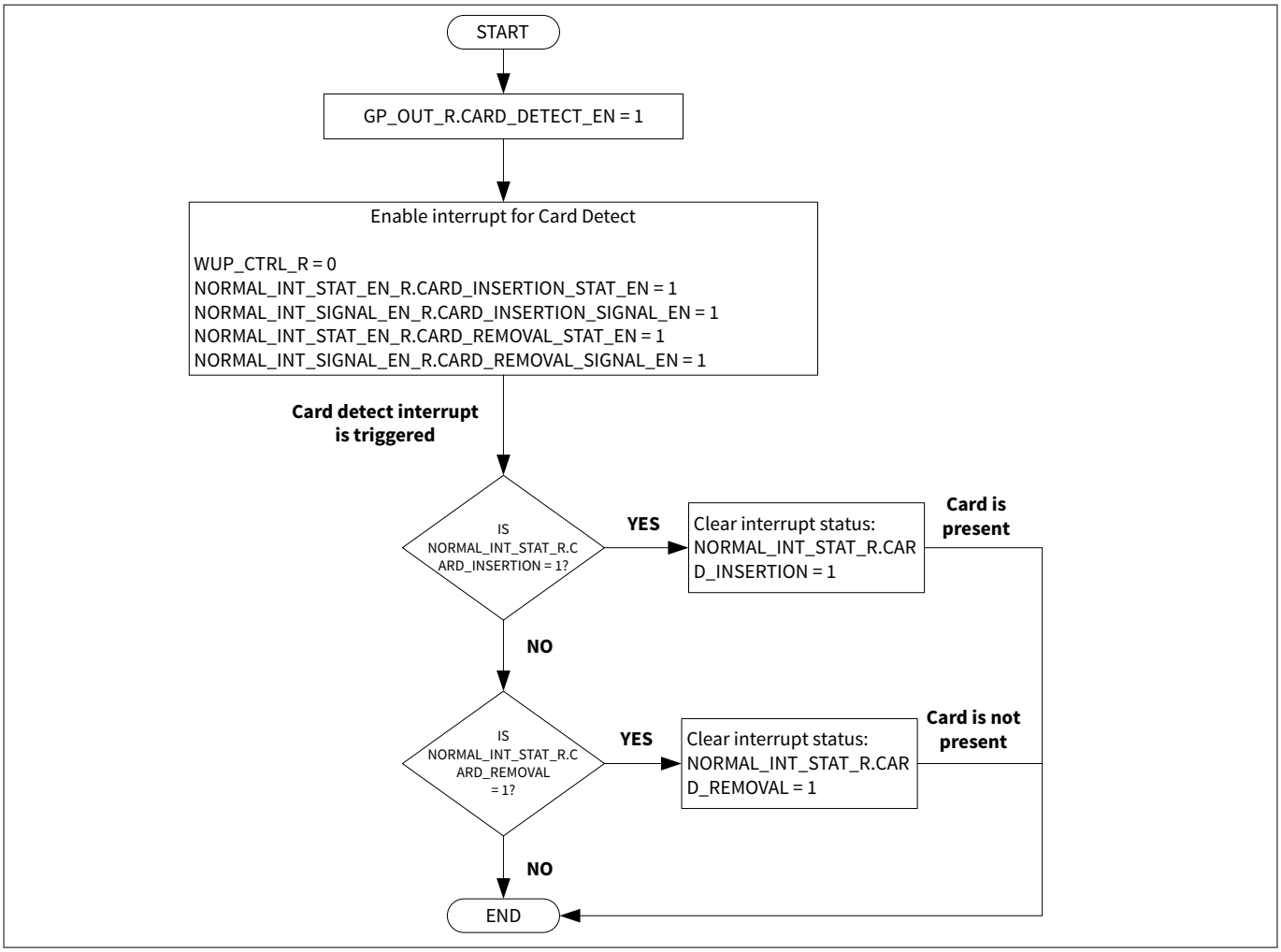


Figure 329 Card detection through interrupt

31.10.3 SDHC initialization

To initialize SDHC, configure the basic settings as shown in Figure 330. This step can also be executed immediately after enabling SDHC.

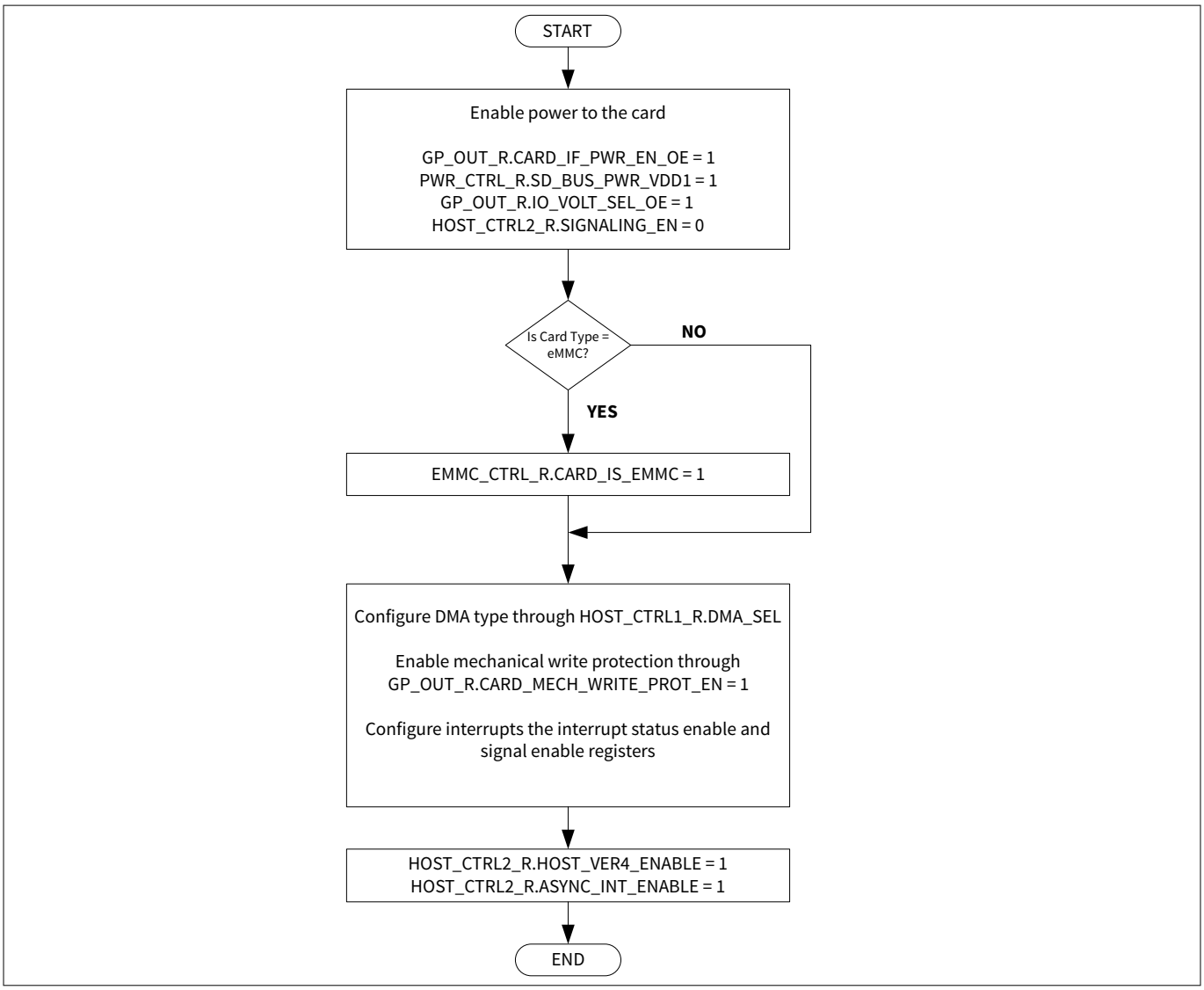


Figure 330 SDHC setup

31.10.4 Clock setup

Enable the internal clock followed by the card clock (SD clock) by following the sequence shown in Figure 331. The SD clock frequency must be 100 kHz to 400 kHz during the card initialization. See Card clock (SDCLK) configuration for details. SD clock can be started and stopped by toggling the SDHCx_CORE_CLK_CTRL_R.SD_CLK_EN bit. The same sequence excluding the step of enabling the internal clock can be used to change the SD clock frequency. The SD clock must be stopped before changing its frequency. Note that SDHCx_CORE_GP_OUT_R.CARD_CLOCK_OE should have been set to '1' for the card clock to appear on the pin.

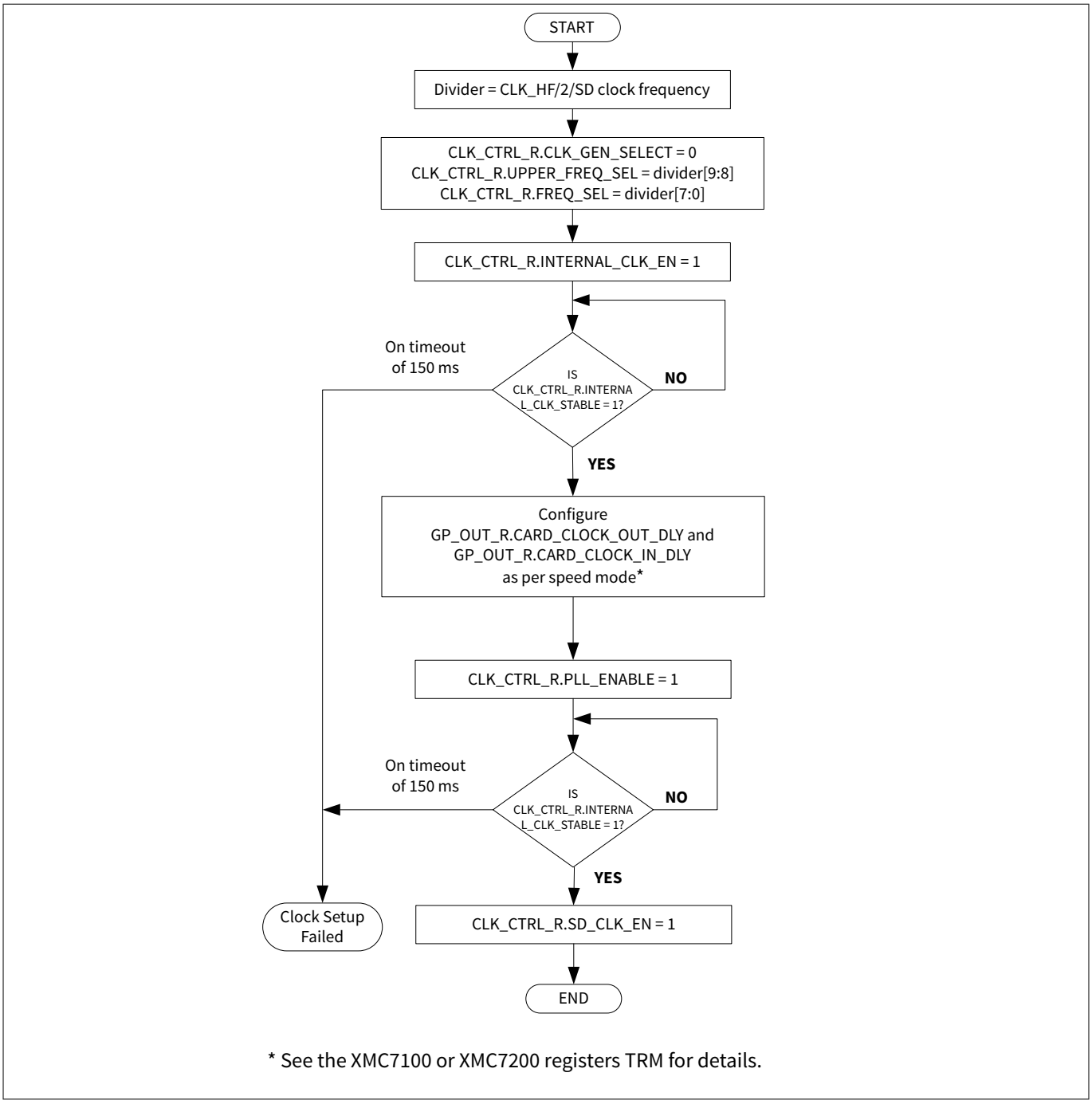


Figure 331 Clock setup

31.11 Error detection

SDHC can detect different types of errors in SD and eMMC transactions. Error is detected in either the command or data portion of the transaction. When an error is detected, SDHCx_CORE_NORMAL_INT_STAT_R.ERR_INTERRUPT bit is set. The exact error can then be identified through the SDHCx_CORE_ERROR_INT_STAT_R Register. The Abort command is used to recover from an error detected during data transfer. In addition to these two registers, SDHC has two other error status registers – Auto CMD error status (SDHCx_CORE_AUTO_CMD_STAT_R) and ADMA error status (SDHCx_CORE_ADMA_ERR_STAT_R). The following table lists the errors detected by SDHC.

Table 264 Errors detected by SHDC

Type	Error
Command errors	Command timeout error
	Command CRC error
	Command end bit error
	Command index error
	Command conflict error
	Response error
Auto command errors	Command not issued by auto CMD12 error
	Auto command timeout error
	Auto command CRC error
	Auto command end bit error
	Auto command index error
	Auto command conflict error
	Auto CMD response error
Data errors	Data timeout error
	Data CRC error
	Data end bit error
	ADMA error
	Tuning error

31.12 Registers

When the SDHC block is enabled and a DMA operation is initiated, there is a brief time during which the SDHC hardware accesses the SDHC Registers from address 0x00 through 0x0F. During this time, writes are ignored and reads return zero. The following registers are accessed:

- SDHCx_CORE_SDMASA_R
- SDHCx_CORE_BLOCKSIZE_R
- SDHCx_CORE_BLOCKCOUNT_R
- SDHCx_CORE_ARGUMENT_R
- SDHCx_CORE_XFER_MODE_R
- SDHCx_CORE_CMD_R

If the SDHC block is enabled, any writes to these registers should be read back to ensure the write was accepted. If the read back does not match the write, a retry is required.

Table 265 SDHC Register list

Register	Name	Description
SDHCx_WRAP_CTL	Control Register	Enables SDHC
SDHCx_CORE_SDMASA_R	SDMA System address Register	This register is used to configure a 32-bit block count or an SDMA system address based on the host version 4 enable bit in the Host Control 2 Register. This register is applicable to both SD and eMMC modes.
SDHCx_CORE_BLOCKSIZE_R	Block Size Register	This register is used to configure an SDMA buffer boundary and the number of bytes in a data block. This register is applicable to both SD and eMMC modes.
SDHCx_CORE_BLOCKCOUNT_R	16-bit Block Count Register	This register is used to configure the number of data blocks. This register is applicable to both SD and eMMC modes.
SDHCx_CORE_ARGUMENT_R	Argument Register	This register is used to configure the SD/eMMC command argument.
SDHCx_CORE_XFER_MODE_R	Transfer mode Register	This register is used to control the operation of data transfers for the SD/eMMC mode.
SDHCx_CORE_CMD_R	Command Register	This register is used to provide the information related to a command and a response packet. This register is applicable to the SD/eMMC mode.
SDHCx_CORE_RESP01_R	Response Register 0/1	This register stores 39-8 bits of the response field for the SD/eMMC mode.
SDHCx_CORE_RESP23_R	Response Register 2/3	This register stores 71-40 bits of the response field for the SD/eMMC mode.
SDHCx_CORE_RESP45_R	Response Register 4/5	This register stores 103-72 bits of the response field for the SD/eMMC mode.
SDHCx_CORE_RESP67_R	Response Register 6/7	This register stores 135-104 bits of the response field for the SD/eMMC mode.
SDHCx_CORE_BUF_DATA_R	Buffer Data Port Register	This register is used to access the packet buffer. This register is applicable to the SD/eMMC mode.
SDHCx_CORE_PSTATE_REG	Present State Register	This register indicates the present status of the host controller. This register is applicable to the SD/eMMC mode.

(table continues...)

Table 265 (continued) SDHC Register list

Register	Name	Description
SDHCx_CORE_HOST_CTRL1_R	Host Control 1 Register	This register is used to control the operation of the host controller. This register is applicable to the SD/eMMC mode.
SDHCx_CORE_PWR_CTRL_R	Power Control Register	This register is used to control the bus power for the card. This register is applicable to the SD/eMMC mode.
SDHCx_CORE_BGAP_CTRL_R	Block Gap Control Register	This register is used by the host driver to control any operation related to block gap. This register is applicable to the SD/eMMC mode.
SDHCx_CORE_WUP_CTRL_R	Wakeup Control Register	The register wakes up an otherwise idle host driver. It does NOT wake up from deep sleep.
SDHCx_CORE_CLK_CTRL_R	Clock Control Register	This register controls SDCLK (card clock) in the SD/eMMC mode. This register is applicable to the SD/eMMC mode.
SDHCx_CORE_TOUT_CTRL_R	Timeout Control Register	This register is used to set the data timeout counter value for the SD/eMMC mode according to the timer clock defined by the capabilities register, while initializing the host controller.
SDHCx_CORE_SW_RST_R	Software Reset Register	This register is used to generate a reset. This register is applicable to the SD/eMMC mode.
SDHCx_CORE_NORMAL_INT_STAT_R	Normal Interrupt Status Register	This register reflects the status of the normal interrupt. This register is applicable to the SD/eMMC mode.
SDHCx_CORE_ERROR_INT_STAT_R	Error Interrupt Signal Enable Register	This register enables an interrupt when the error interrupt status enable is enabled and at least one of the statuses is set to 1. This register is applicable to the SD/eMMC mode.
SDHCx_CORE_NORMAL_INT_STAT_EN_R	Normal Interrupt Status Enable Register	This register enables the interrupt status for normal interrupt status register (SDHCx_CORE_NORMAL_INT_STAT_R) when SDHCx_CORE_NORMAL_INT_STAT_R is set to 1. This register is applicable to the SD/eMMC mode.

(table continues...)

Table 265 (continued) SDHC Register list

Register	Name	Description
SDHCx_CORE_ERROR_INT_STAT_EN_R	Error Interrupt Status Enable Register	This register sets the interrupt status for error interrupt status register (SDHCx_CORE_ERROR_INT_STAT_R), when SDHCx_CORE_ERROR_INT_STAT_EN_R is set to '1'. This register is applicable to the SD/eMMC mode.
SDHCx_CORE_NORMAL_INT_SIGNAL_EN_R	Normal Interrupt Signal Enable Register	This register is used to select the interrupt status that is indicated to the host system as the interrupt. This register is applicable to the SD/eMMC mode.
SDHCx_CORE_ERROR_INT_SIGNAL_EN_R	Error Interrupt Signal Enable Register	This register is used to select the interrupt status that is notified to the host system as an interrupt. This register is applicable to the SD/eMMC mode.
SDHCx_CORE_AUTO_CMD_STAT_R	Auto CMD Status Register	This register is used to indicate the CMD12 response error of Auto CMD12, and the CMD23 response error of Auto CMD23. This register is valid only when Auto CMD Error is set. This register is applicable to the SD/eMMC mode.
SDHCx_CORE_HOST_CTRL2_R	Host Control 2 Register	This register is used to control how the host controller operates. This register is applicable to the SD/eMMC mode.
SDHCx_CORE_CAPABILITIES1_R	Capabilities 1 Register - 0 to 31	This register provides the host driver with information specific to the host controller implementation.
SDHCx_CORE_CAPABILITIES2_R	Capabilities 2 Register - 32 to 63	This register provides the host driver with information specific to the host controller implementation.
SDHCx_CORE_CURR_CAPABILITIES1_R	Current Capabilities Register - 0 to 31	This register indicates the maximum current capability for each voltage, for VDD1.
SDHCx_CORE_CURR_CAPABILITIES2_R	Maximum Current Capabilities Register - 32 to 63	This register indicates the maximum current capability for each voltage, for VDD2.
SDHCx_CORE_FORCE_AUTO_CMD_STAT_R	Force Event register for Auto CMD Error Status Register	The register is not physically implemented but is an address at which the Auto CMD Error Status Register can be written. This register is applicable to the SD/eMMC mode.

(table continues...)

Table 265 (continued) SDHC Register list

Register	Name	Description
SDHCx_CORE_FORCE_ERROR_INT_STAT_R	Force Event Register for Error Interrupt Status	This register is not physically implemented but is an address at which the Error Interrupt Status Register can be written. This register is applicable to the SD/eMMC mode.
SDHCx_CORE_ADMA_ERR_STAT_R	ADMA Error Status Register	This register stores the ADMA state during an ADMA error. This register is applicable to the SD/eMMC mode.
SDHCx_CORE_ADMA_SA_LOW_R	ADMA System Address Register - Low	This register holds the lower 32-bit system address for DMA transfer. This register is applicable to the SD/eMMC mode.
SDHCx_CORE_ADMA_ID_LOW_R	ADMA3 Integrated Descriptor address Register - Low	This register holds the lower 32-bit integrated descriptor address. This register is applicable to the SD/eMMC mode.
SDHCx_CORE_HOST_CNTRL_VERS_R	Host Controller Version Register	This register is used to indicate the host controller version number.
SDHCx_CORE_MSHC_VER_ID_R	MSHC Version Register	This register reflects the current release number.
SDHCx_CORE_MSHC_VER_TYPE_R	MSHC Version Type Register	This register reflects the current release type.
SDHCx_CORE_MSHC_CTRL_R	MSHC Control Register	This register is used to control the operation of MSHC host controller.
SDHCx_CORE_MBIU_CTRL_R	MBIU Control Register	This register is used to select the valid burst types that the AHB master bus interface can generate.
SDHCx_CORE_EMMC_CTRL_R	eMMC Control Register	This register is used to control the eMMC operation.
SDHCx_CORE_BOOT_CTRL_R	eMMC Boot Control Register	This register is used to control the eMMC boot operation.
SDHCx_CORE_GP_IN_R	General Purpose Input Register	This register is used as a General-Purpose Input Register.
SDHCx_CORE_GP_OUT_R	General Purpose Output Register	This register is used as a General-Purpose Output Register.

32 Audio subsystem

The inter-IC sound bus (I^2S) is a serial bus interface standard used to connect digital audio devices together. In addition to the standard I^2S format, the I^2S block also supports the left justified (LJ) format and the time division multiplexed (TDM) format.

32.1 Features

- Supports standard I^2S, LJ, and eight-channel TDM digital audio interface formats
- Supports both master and slave mode operation in all the digital audio formats
- Supports independent operation of receive (RX) and transmit (TX) directions
- Supports operating from an external master clock provided through an external IC such as audio codec
- Provides Configurable Clock Divider Registers to generate the required sample rates
- Supports data word length of 8-bit, 16-bit, 18-bit, 20-bit, 24-bit, and 32-bit per channel
- Supports channel length of 8-bit, 16-bit, 18-bit, 20-bit, 24-bit, and 32-bit per channel (channel length fixed at 32-bit in TDM format)
- Provides two hardware FIFO buffers, one each for the TX block and RX block, respectively
- Supports both DMA- and CPU-based data transfers

32.2 Architecture

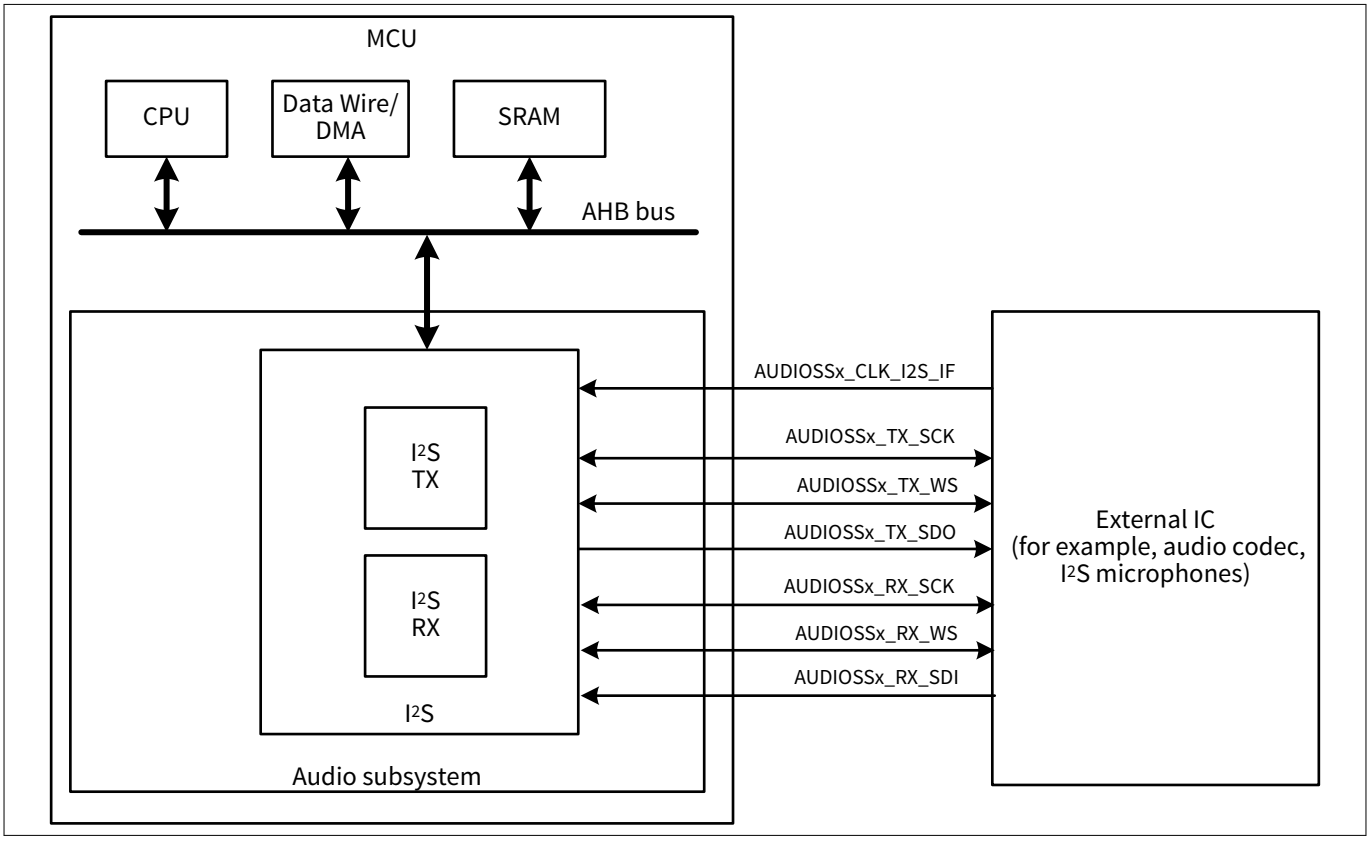


Figure 332 I^2S block diagram

Figure 332 shows the high-level block diagram of the I²S block, which consists of two sub-blocks – I²S transmitter (TX) and I²S receiver (RX). The digital audio interface format and master/slave mode configuration can be done independently for the TX and RX blocks. In the master mode, the word select (WS) and serial data clock (SCK) are generated by the I²S block in the MCU. In the slave mode, the WS and SCK signals are inputs signals to the MCU, and are generated by the external master device. The I²S block configuration, control, and Status Registers, along with the FIFO data buffers are accessible through the AHB bus. AHB bus masters such as CPU and DMA can access the I²S Registers through the AHB interface. Refer to the device datasheet for information on port pin assignments of the I²S block signals and AC/DC electrical specifications.

32.3 Digital audio interface formats

The I²S block supports the following digital audio interface formats.

- Standard I²S format
- Left justified format
- Time division multiplexed (TDM) format

The TX and RX sub-blocks can be independently configured to support one of the above formats in either master or slave mode. The I2S_MODE bits in the I2Sx_TX_CTL and I2Sx_RX_CTL Registers are used to configure the digital audio interface format for the TX and RX blocks respectively. The MS (Master/Slave) bit in the I2Sx_TX_CTL and I2Sx_RX_CTL Registers is used to configure the blocks in master or slave mode.

32.3.1 Standard I²S format

Figure 333 shows the timing diagrams for the different word length and channel length combinations in the standard I²S digital audio format. In the standard I²S format, the word select signal (WS) is low for left channel data, and high for right channel data. The WS signal transitions one bit-clock (SCK) early relative to the start of the left/right channel data. All the serial data (SD), WS signal transitions on the falling edge of the SCK signal, and the read operations on the WS and sd lines are usually done on the rising edge of SCK. Therefore, the I²S TX block writes to the serial data (AUDIOSSx_TX_SDO) line on the falling edge of AUDIOSSx_TX_SCK, and the I²S RX block reads the data (AUDIOSSx_RX_SDI) on the rising edge of AUDIOSSx_RX_SCK. The serial data is transmitted most significant bit (MSb) first. Depending on whether the block is in master or slave mode, the WS/SCK signals are either generated by the block (master mode) or input signals to the block (slave mode).

The I²S block supports configurable word length and channel length selection options. The word length for the TX and RX blocks can be configured using the WORD_LEN bits in the I2Sx_TX_CTL and I2Sx_RX_CTL Registers, respectively. The channel length for the TX and RX blocks can be configured using the CH_LEN bits in the I2Sx_TX_CTL and I2Sx_RX_CTL Registers respectively. The channel length configuration should always be greater than or equal to the word length configuration. Ensure that when the I²S RX block is operated in slave mode, the master TX device ensures that its channel length configuration aligns with the I²S RX block channel length setting. If there is channel length mismatch, the I²S RX block in slave mode will not operate correctly.

In the TX block, when the channel length is greater than the word length, the unused bits can be transmitted either as '0' or '1'. This selection is made using the OVHDATA bit in the I2Sx_TX_CTL Register. In the RX block, when the word length is less than 32 bits, the unused most significant bits written to the 32-bit RX FIFO Register can either be set to '0' or sign bit extended. This selection is made using the BIT_EXTENSION bit in the I2Sx_RX_CTL Register.



Figure 333 Standard I²S format (word length and channel length combination timing diagrams)

Table 266 lists the supported word length and channel length combinations.

Table 266 Word length and channel length combinations

		Word length					
		8-bit	**16-bit**	**18-bit**	**20-bit**	**24-bit**	**32-bit**
Channel length	32-bit	Valid	Valid	Valid	Valid	Valid	Valid
	24-bit	Valid	Valid	Valid	Valid	Valid	Invalid
	30-bit	Valid	Valid	Valid	Valid	Invalid	Invalid
	18-bit	Valid	Valid	Valid	Invalid	Invalid	Invalid
	16-bit	Valid	Valid	Invalid	Invalid	Invalid	Invalid
	8-bit	Valid	Invalid	Invalid	Invalid	Invalid	Invalid

32.3.2 Left justified (LJ) format

Figure 334 shows the timing diagrams for the Left Justified interface format using the 32-bit channel length and 32-bit word length configuration as an example. The only differences between the standard I²S and LJ formats are:

- In the standard I²S format, WS signal is low for left channel data and high for right channel data. In the LJ format, WS signal is high for left channel data and low for right channel data.
- In the standard I²S format, WS signal transitions one bit-clock (SCK) early relative to the start of the channel data (coincides with LSb of the previous channel). In the LJ format, there is no early transition, and the WS signal transitions coincide with the start of the channel data.

Apart from these differences, all the features explained in Standard I²S format apply to the LJ format as well.

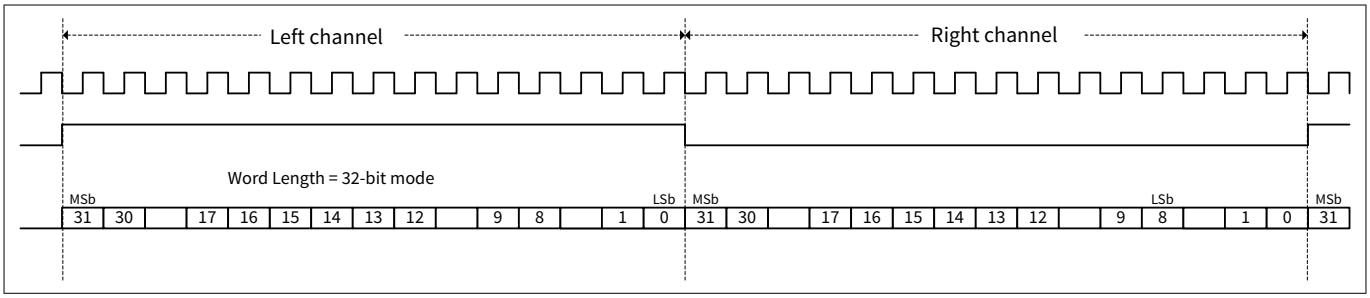


Figure 334 Left justified digital audio format

32.3.3 Time division multiplexed (TDM) format

Figure 335 shows the timing diagrams for the two types of time division multiplexed (TDM) formats supported by the I²S block. The differences between the standard I²S/LJ formats and the TDM format are as follows:

- Standard I²S/LJ formats support only two channels (left/right) per frame, while TDM format supports up to eight channels per frame.
- In the TDM format, channel length for all eight channels is fixed at 32 bits. In the standard I²S/LJ formats, the channel length is configurable. The word length per channel is configurable similar to the standard I²S and the data is also transmitted most significant bit first. Similar to I²S, when the word length per channel is less than the 32-bit channel length for TX block, the OVHDATA bit in the I2Sx_TX_CTL Register is used to fill the unused least significant channel data bits with either all zeros or all ones
- In the TDM format, all eight channels of data are always present in a frame, and thus the frame width is fixed at 256 bits. You have the option to configure the number of active channels in a frame by configuring

the CH_NR bits in the I2Sx_TX_CTL and I2Sx_RX_CTL Registers. In the standard I²S/LJ format, the CH_NR should always be configured for two channels. The number of active channels in the TDM format can be less than or equal to eight channels. The unused (inactive) channels always follow the active channels in a frame. As an example, if CH_NR is set for four channels, CH0 to CH3 are the active channels and CH4 to CH7 are the unused channels. The OVHDATA bit in the I2Sx_TX_CTL Register is used to fill the unused channels with either all zeros or all ones.

- The pulse width of the word select (WS) signal in the TDM format can be configured to be either one bit clock (SCK) wide or one channel wide. The selection is made using the WS_PULSE bit in the I2Sx_TX_CTL and I2Sx_RX_CTL Registers. The pulse width is fixed to one channel width in the I²S/LJ format.
- Two types of TDM formats are supported. In TDM mode A, the WS rising edge signal to signify the start of frame coincides with the start of CH0 data. In TDM mode B, the WS rising edge signal to signify the start of frame is one bit clock (SCK) early, relative to the start of CH0 data (coincides with the last bit of the previous frame). The selection between the two TDM formats is made using the I2S_MODE bits in the I2Sx_TX_CTL and I2Sx_RX_CTL Registers.

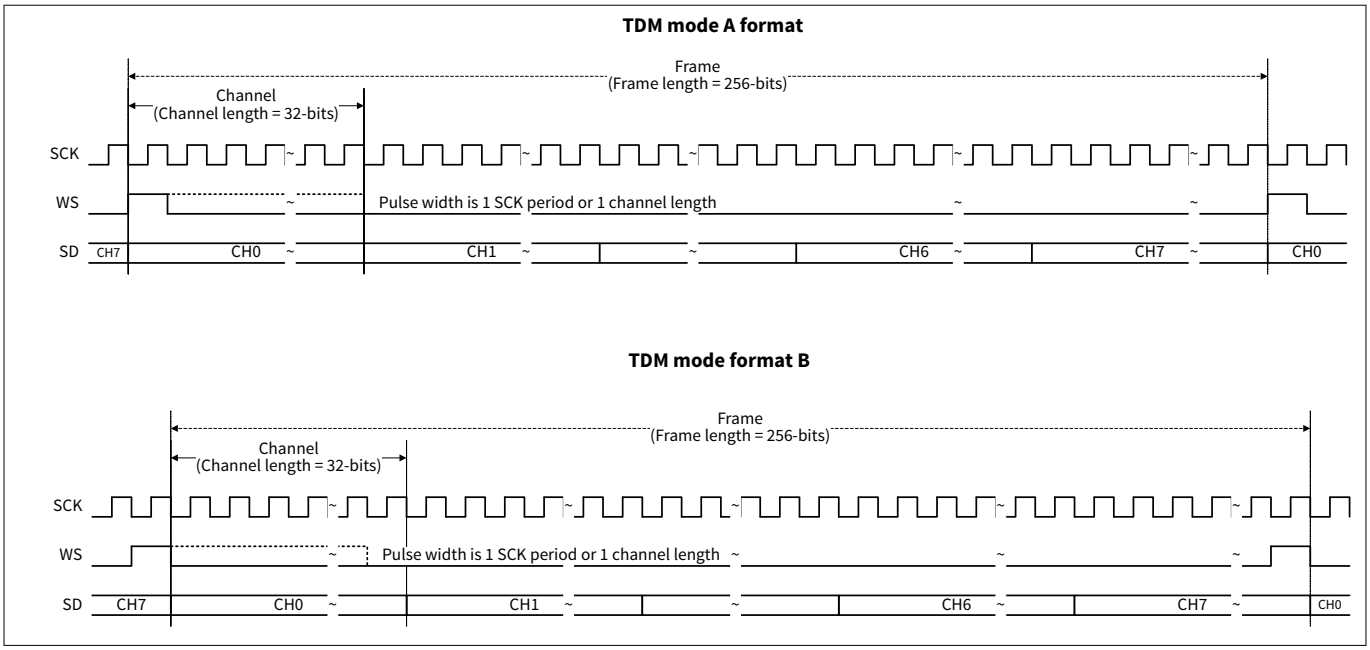


Figure 335 TDM digital audio interface format

32.4 Clocking polarity and delay options

The I²S block supports configurable clock polarity and delay options to alleviate any timing issues in the system involving PCB signal propagation delays, and delays associated with internal device signal routing.

When the I²S TX block operates in the slave mode, the AUDIOSSx_TX_SCK and AUDIOSSx_TX_WS signals are input signals to the MCU, and the AUDIOSSx_TX_SDO output signal is transmitted off the AUDIOSSx_TX_SCK falling edge. The AUDIOSSx_TX_SDO signal is sampled by the external master device RX block on the subsequent AUDIOSSx_TX_SCK rising edge. Timing issues arise if the AUDIOSSx_TX_SDO signal reaching the master side RX block does not meet the setup and hold time requirements for input data on the master side. The I²S TX block in the MCU has an option to advance the serial data transmission by 0.5 SCK cycles when the B_CLOCK_INV bit in the I2Sx_TX_CTL Register is set. This feature can be used if there are timing issues while operating the I²S TX block in slave mode.

Table 267 TX block configuration for master mode

TX block configuration	Clock polarity Register	Description
	I2Sx_TX_CTL.SCKO_POL	
(1)	0	Serial data is transmitted off the SCK falling edge
(2)	1	Serial data is transmitted off the SCK rising edge

Table 268 TX block configuration for slave mode

TX block configuration	Clock polarity Register	Delay option Register	Description
	I2Sx_TX_CTL.SCKI_POL	I2Sx_TX_CTL.B_CLOCK_INV	
(1)	0	0	Serial data is transmitted OFF the SCK falling edge
(2)	0	1	Serial data is transmitted OFF the SCK rising edge that is 0.5 SCK cycles before (1)
(3)	1	0	Serial data is transmitted OFF the SCK rising edge
(4)	1	1	Serial data is transmitted OFF the SCK falling edge that is 0.5 SCK cycles before (3)

Similarly, when the I^2S RX block operates in the master mode, the AUDIOSSx_RX_SCK and AUDIOSSx_RX_WS signals are output signals from the MCU, and the AUDIOSSx_RX_SDI signal is transmitted by the external master device on the falling edge of AUDIOSSx_RX_SCK. The I^2S RX block samples the AUDIOSSx_RX_SDI signal on the subsequent AUDIOSSx_RX_SCK rising edge. Timing issues arise if the AUDIOSSx_RX_SDI signal reaching the MCU block does not meet the setup and hold time requirements for input data. The I^2S RX block has an option to delay the serial data capture by 0.5 SCK cycles when the B_CLOCK_INV bit in the I2Sx_RX_CTL Register is set. This feature can be used if there are timing issues while operating the I^2S RX block in master mode.

In addition to these clock delay options, there is also an option to invert the outgoing bit clock (SCK) in master mode by setting the SCKO_POL bit in the I2Sx_TX_CTL and I2Sx_RX_CTL Registers. Similarly, in the slave mode, there is an option to invert the incoming bit clock (SCK) by setting the SCKI_POL bit in the I2Sx_TX_CTL and I2Sx_RX_CTL Registers.

Refer to the XMC7100 registers technical reference manual and XMC7200 registers technical reference manual for detailed description of the B_CLOCK_INV, SCKI_POL, and SCKO_POL Register configurations.

Table 269 RX block configuration for master mode

RX block configuration	Clock polarity Register	Delay option Register	Description
	I2Sx_RX_CTL.SCKO_POL	I2Sx_RX_CTL.B_CLOCK_INV	
(1)	0	0	Serial data is captured by the SCK rising edge
(2)	0	1	Serial data is captured by the SCK falling edge that is 0.5 SCK cycles after (1)

(table continues...)

Table 269 (continued) RX block configuration for master mode

RX block configuration	Clock polarity Register	Delay option Register	Description
	I2Sx_RX_CTL.SCKO_POL	I2Sx_RX_CTL.B_CLOCK_INV	
(3)	1	0	Serial data is captured by the SCK falling edge
(4)	1	1	Serial data is captured by the SCK rising edge that is 0.5 SCK cycles after (3)

Table 270 RX block configuration for slave mode

RX block configuration	Clock polarity Register	Description
	I2Sx_RX_CTL.SCKI_POL	
(1)	0	Serial data is captured by SCK rising edge
(2)	1	Serial data is captured by SCK falling edge

32.5 Interfacing with audio codecs

The I^{2}S block in the MCU interfaces with an audio codec device based on the choice of codec device and the end application requirements. Some scenarios and the connection diagrams are as follows:

- Codecs with separate WS and SCK signals for the RX and TX directions: To interface with these codecs, the connections between the I^{2}S block and the codec device will be as shown in Figure 336 where the I^{2}S TX signals (AUDIOSSx_TX_SCK, AUDIOSSx_TX_WS, AUDIOSSx_TX_SDO) connect to the codec RX signals, and the I^{2}S RX signals (AUDIOSSx_RX_SCK, AUDIOSSx_RX_WS, AUDIOSSx_RX_SDI) connect to the codec TX signals. The direction of SCK (AUDIOSSx_TX_SCK, AUDIOSSx_RX_SCK) and WS (AUDIOSSx_TX_WS, AUDIOSSx_RX_WS) signals depends on which device is the master and which device is the slave.
- Codecs with common WS and SCK signals for both RX and TX directions: There are two possible configurations to interface these codecs with the MCU as shown in Figure 336. In both configurations, the SCK signals (AUDIOSSx_TX_SCK, AUDIOSSx_RX_SCK, codec_sck) are shorted externally. The same goes for the WS signal connections as well (AUDIOSSx_TX_WS, AUDIOSSx_RX_WS, codec_ws). Ensure that only one block is driving the SCK and WS lines. So when the codec acts as the slave device, the I^{2}S RX block should be in the master mode, and the I^{2}S TX block should be in the slave mode (or I^{2}S RX as slave and I^{2}S TX as master). When the codec acts as the master device, both the I^{2}S RX and I^{2}S TX blocks should be in slave mode.

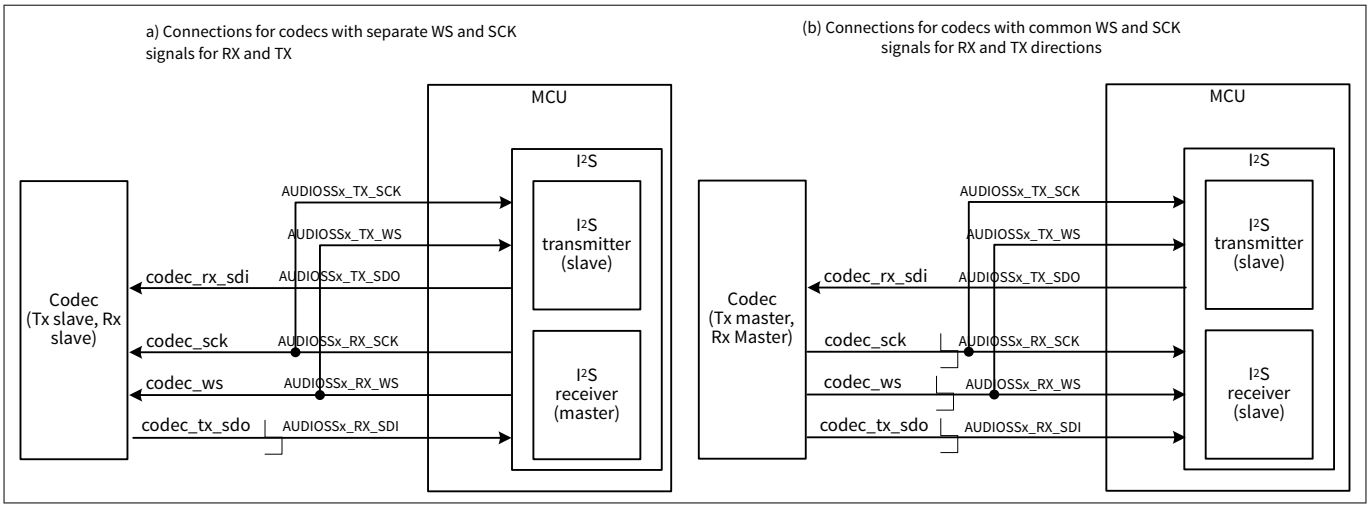


Figure 336 Interfacing with codecs having common WS and SCK signals

32.6 Clocking features

The I²S unit has three clock inputs.

Table 271 Clock inputs

Signal	Description
CLK_SYS_I2S	System clock. This clock is used for the AHB slave Interface, control, status, and Interrupt Registers, and also clocks the DMA trigger control logic.
CLK_AUDIO_I2S	I²S internal clock. This clock is used for I²S transmitter (TX)/receiver (RX) blocks; it is asynchronous with the CLK_SYS_I2S. This clock is connected to the CLK_HFx high-frequency clock in the device. Refer to the Clocking system for more details on high frequency clocks.
AUDIOSSx_CLK_I2S_IF	I²S external clock. This clock is provided from an external I²S bus host through a port pin. It is used in place of the CLK_AUDIO_I2S clock to synchronize I²S data to the clock used by the external I²S bus host.

Figure 337 shows the clocking divider structure in the I²S block. In the master mode, the SCK and WS signals are generated either using the CLK_AUDIO_I2S internal clock or the AUDIOSSx_CLK_I2S_IF external clock. Refer to the device datasheet for the port pin assignment of AUDIOSSx_CLK_I2S_IF clock. The CLOCK_SEL bit in the I2Sx_CLOCK_CTL Register controls the selection between internal and external clocks.

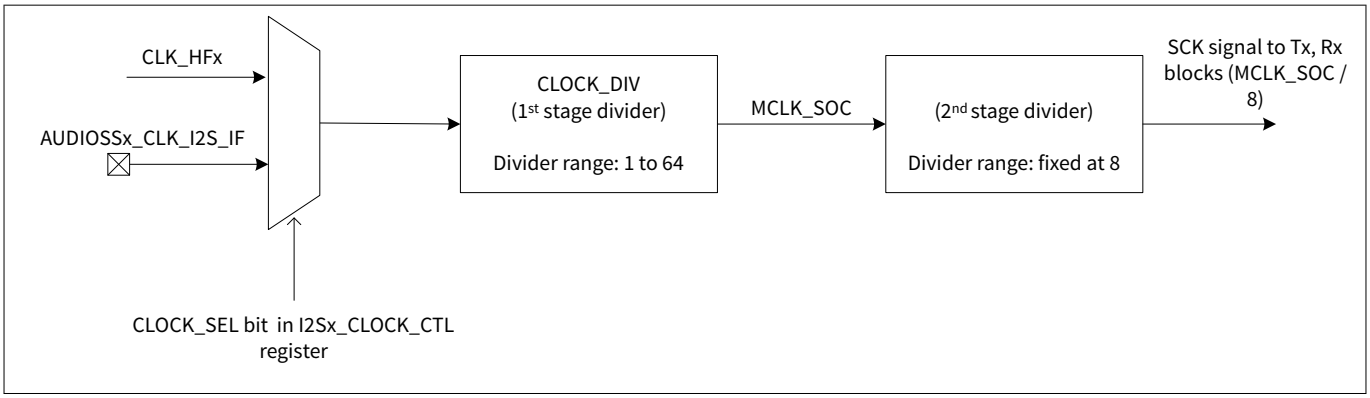


Figure 337 Clocking divider structure

There are two stages of clock dividers in the I²S block as follows.

- The first stage clock divider is used to generate the internal I²S master clock (MCLK_SOC). The input clock to the first stage divider is either CLK_AUDIO_I2S or AUDIOSSx_CLK_I2S_IF. The first stage clock divider is configured using the CLOCK_DIV bits in I2Sx_CLOCK_CTL Register. Divider values from 1 to 64 are supported.
- The second stage clock divider is used to generate the SCK signals. The input clock is the output from the first stage clock divider. This divider value is fixed at '8' (FTX_SCK = FRX_SCK = FMCLK_SOC/8). The word select (WS) signal frequency depends on the SCK frequency, and the configured channel length value.

When in slave mode, the internal clock (MCLK_SOC) frequency should still be eight times the frequency of the input serial clock. You must choose the appropriate clock source and the CLOCK_DIV divider value to guarantee this condition is met in the slave mode of operation. Usually, when the I²S block operates in the slave mode, the host sends a master clock which is an integral multiple of the sampling rate. This master clock can be routed to the AUDIOSSx_CLK_I2S_IF port pin. The CLOCK_DIV divider value can then be adjusted to ensure that the MCLK_SOC is eight times the input SCK frequency.

Table 272 gives an example of the clock divider settings for operating the I²S block at the standard sampling rates in the standard I²S format. Note that the first stage divider values in the table are the register field values – the actual divider values are one more than the configured register values as explained in the clock divider section. Refer to the device datasheet for details on maximum values of SCK frequency, and the output sampling rates.

Table 272 I²S divider values for standard audio sampling rates in standard I²S format

Sampling rate (SR) (kHz)	WORD_LEN (bits)	SCK (2*WORD_LEN *SR) (MHz)	CLK_HFx (MHz)	CLK_HFx/SCK (total divider ratio)	CLK_CLOCK_DIV (first divider)	Second stage divider (fixed at 8)
8	32	0.512	49.152	96	11	8
16	32	1.024	49.152	48	5	8
32	32	2.048	49.152	24	2	8
48	32	3.072	49.152	16	1	8
44.1	32	2.8224	45.1584	16	1	8

32.7 FIFO buffer and DMA support

The I²S block has two FIFO buffers – one each for the TX block and RX block, respectively. The ordering format of the channel data in both TX and RX FIFOs depends on the configured digital audio format. This ordering format should be considered when writing to the TX FIFO or reading from the RX FIFO. In the standard I²S and LJ digital audio formats, the ordering of the data is (L, R, L, R, L, ...) where L refers to the left channel data and R refers to the right channel data. In the TDM format with the number of active channels set to four, the data order will be (CH0, CH1, CH2, CH3, CH0, CH1, CH2, CH3, CH0, ...). If the number of active channels is set to eight, the cycle will repeat after CH0–CH7 data.

Table 273 I²S FIFO buffers

Feature	TX FIFO	RX FIFO
Architecture	256 depth FIFOs for up to 32-bit data elements	
Data Register	I2Sx_TX_FIFO_WR	I2Sx_RX_FIFO_RD
Data format	Right-aligned	Right-aligned Receive data is extended by zeros or the sign-bit

(table continues...)

Table 273 (continued) I²S FIFO buffers

Feature	TX FIFO	RX FIFO
Trigger control Register for DMA	I2Sx_TR_CTL.TX_REQ_EN	I2Sx_TR_CTL.RX_REQ_EN
Trigger level	When the TX FIFO has less entries than I2Sx_TX_CTL.TRIGGER_LEVEL, a transmitter trigger event is generated	When the RX FIFO has more entries than I2Sx_RX_CTL.TRIGGER_LEVEL, a receiver trigger event is generated

I²S TX FIFO: The I²S TX block has a hardware FIFO of depth 256 elements where each element is 32-bit wide. In addition to this 256-element FIFO, the I²S block has an internal transmit buffer that can store four 32-bit data to be transmitted. This four-element buffer is used as an intermediary to hold data to be transferred on the I²S bus, and is not exposed to the AHB BUS interface.

The I2Sx_TX_FIFO_CTL Register is used for FIFO control operations. The TRIGGER_LEVEL bits in the I2Sx_TX_FIFO_CTL Register can be used to generate a transmit trigger event when the TX FIFO has less entries than the value configured in the TRIGGER_LEVEL bits.

The FIFO freeze operation can be enabled by setting the FREEZE bit in the I2Sx_TX_FIFO_CTL Register. When the FREEZE bit is set and the TX block is operational (TX_START bit in I2Sx_CMD is set), hardware reads from the TX FIFO do not remove the FIFO entries. Also, the TX FIFO read pointer will not be advanced. Any writes to the I2Sx_TX_FIFO Register will increment the TX FIFO write pointer; when the TX FIFO becomes full, the internal write pointer stops incrementing. The freeze operation may be used for firmware debug purposes. This operation is not intended for normal operation. To return to normal operation after using the freeze operation, the I²S must be reset by clearing the TX_ENABLED bit in the I2Sx_CTL Register, and then setting the bit again.

The CLEAR bit in the I2Sx_TX_FIFO_CTL Register is used to clear the TX FIFO by resetting the read/write pointers associated with the FIFO. Write access to the TX FIFO using the I2Sx_TX_FIFO_WR Register is not allowed while the CLEAR bit is set.

The I2Sx_TX_FIFO_STATUS Register provides FIFO status information. This includes number of used entries in the TX FIFO and the current values of the TX FIFO read/write pointers. This register can be used for debug purposes. The I²S TX FIFO read pointer is updated whenever the data is transferred from the TX FIFO to the internal transmit buffer. TX FIFO write pointer is updated whenever the data is written to the I2Sx_TX_FIFO_WR Register, either through the CPU or the DMA controller.

For TX FIFO data writes using the CPU, the hardware can be used to trigger an interrupt event for any of the FIFO conditions such as TX_TRIGGER, TX_NOT_FULL, and TX_EMPTY. As part of the interrupt handler, the CPU can write to the I2Sx_TX_FIFO_WR Register. The recommended method is to write (256 – TRIGGER_LEVEL) words to the I2Sx_TX_FIFO_WR Register every time the TX_TRIGGER interrupt event is triggered. In addition, interrupt events can be generated for FIFO overflow/underflow conditions.

For DMA-based TX data transfers, the I²S TX DMA trigger signal (tr_i2s_tx_req) can be enabled by writing '1' to the TX_REQ_EN bit in I2Sx_TR_CTL Register. The trigger signal output will become high whenever the TX FIFO has less entries than that configured in the TRIGGER_LEVEL field. The DMA channel can be configured to transfer up to (256 – TRIGGER_LEVEL) words from the applicable source address (such as flash and SRAM regions). The destination address of the DMA should always be the I2Sx_TX_FIFO_WR Register address, with the destination address increment feature disabled in the DMA channel configuration. This FIFO address increment logic is handled internally to adjust the write pointer, and the DMA should not increment the destination address. For more details on DMA channel configuration, refer to the Direct memory access.

The data in the I2Sx_TX_FIFO_WR is always right-aligned. The I2Sx_TX_FIFO_WR format for different word length configurations is provided in Figure 338.

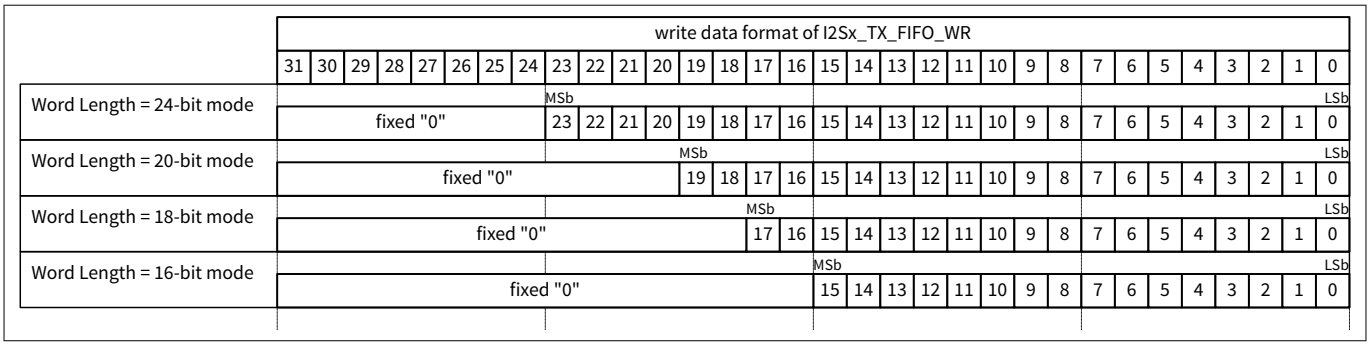


Figure 338 I2Sx_TX_FIFO_WR Register format for different word lengths

I²S RX FIFO: The I²S RX block has a hardware FIFO of depth 256 elements where each element is 32-bit wide. In addition to this 256-element FIFO, the I²S block has an internal receive buffer that can store four 32-bit data to be received. This four-element buffer is used as an intermediary to hold data received on the I²S bus, and is not exposed to the AHB BUS interface.

The I2Sx_RX_FIFO_CTL Register is used for FIFO control operations. The TRIGGER_LEVEL bits in the I2Sx_RX_FIFO_CTL Register is used to generate a receive trigger event when the RX FIFO has more entries than the value configured in the TRIGGER_LEVEL bits. In the standard I²S/LJ format, the TRIGGER_LEVEL bits can be configured up to the allowed maximum value of 253. In the TDM format, the maximum value of TRIGGER_LEVEL is [254-CH_NR) where CH_NR is the number of active channels in the TDM frame.

The FIFO freeze operation can be enabled by setting the FREEZE bit in the I2Sx_RX_FIFO_CTL Register. When the FREEZE bit is set and the RX block is operational (RX_START bit in the I2Sx_CMD Register is set), hardware will not write to the RX FIFO. Also, the RX FIFO write pointer will not be advanced. Any reads from the I2Sx_RX_FIFO register will increment the RX FIFO read pointer; when the RX FIFO becomes empty, the internal read pointer stops incrementing. The freeze operation may be used for firmware debug purposes. This operation is not intended for normal operation. To return to normal operation after using the freeze operation, the I²S must be reset by clearing the RX_ENABLED bit in the I2Sx_CTL Register and then setting the bit again.

The CLEAR bit in I2Sx_RX_FIFO_CTL Register is used to clear the RX FIFO by resetting the read/write pointers associated with the FIFO. Read access from RX FIFO using the I2Sx_RX_FIFO_RD or I2Sx_RX_FIFO_RD_SILENT Registers are not allowed while the CLEAR bit is set.

The I2Sx_RX_FIFO_STATUS Register provides FIFO status information. This includes number of used entries in the RX FIFO and the current values of the RX FIFO read/write pointers. This register can be used for debug purposes. The I²S RX FIFO write pointer is updated whenever the data is transferred to the RX FIFO from the internal receive buffer. RX FIFO read pointer is updated whenever the data is read from the I2Sx_RX_FIFO_RD Register, either through the CPU or the DMA controller. For debug purposes, the I2Sx_RX_FIFO_RD_SILENT Register is available, which always returns the top element of the RX FIFO without updating the read pointer.

For RX FIFO data reads using the CPU, the hardware can be used to trigger an interrupt event for any of the FIFO conditions such as RX_TRIGGER, RX_NOT_EMPTY, and RX_FULL. As part of the interrupt handler, the CPU can read from the I2Sx_RX_FIFO_RD Register. The recommended method is to read (TRIGGER_LEVEL + 1) words from the I2Sx_RX_FIFO_RD Register every time the RX_TRIGGER interrupt event is triggered. In addition, interrupt events can be generated for FIFO overflow/underflow conditions.

For DMA-based RX data transfers, the I²S RX DMA trigger signal (tr_i2s_rx_req) can be enabled by writing '1' to the RX_REQ_EN bit in the I2Sx_TR_CTL Register. The trigger signal output will become high whenever the RX FIFO has more entries than that configured in the TRIGGER_LEVEL field. The DMA channel can be configured to transfer up to (TRIGGER_LEVEL + 1) words to the applicable destination address (such as SRAM regions). The source address of the DMA should always be the I2Sx_RX_FIFO_RD Register address, with the source address increment feature disabled in the DMA channel configuration. This FIFO address increment logic is handled internally to adjust the read pointer, and the DMA should not increment the source address. For more details on DMA channel configuration, refer to the Direct memory access.

The data in the I2Sx_RX_FIFO_RD is always right aligned. The I2Sx_RX_FIFO_RD format for different word length configurations is provided in Figure 339. Note that the unused most significant bits are either set as '0' or sign-bit extended depending on the BIT_EXTENSION bit in the I2Sx_RX_CTL Register.

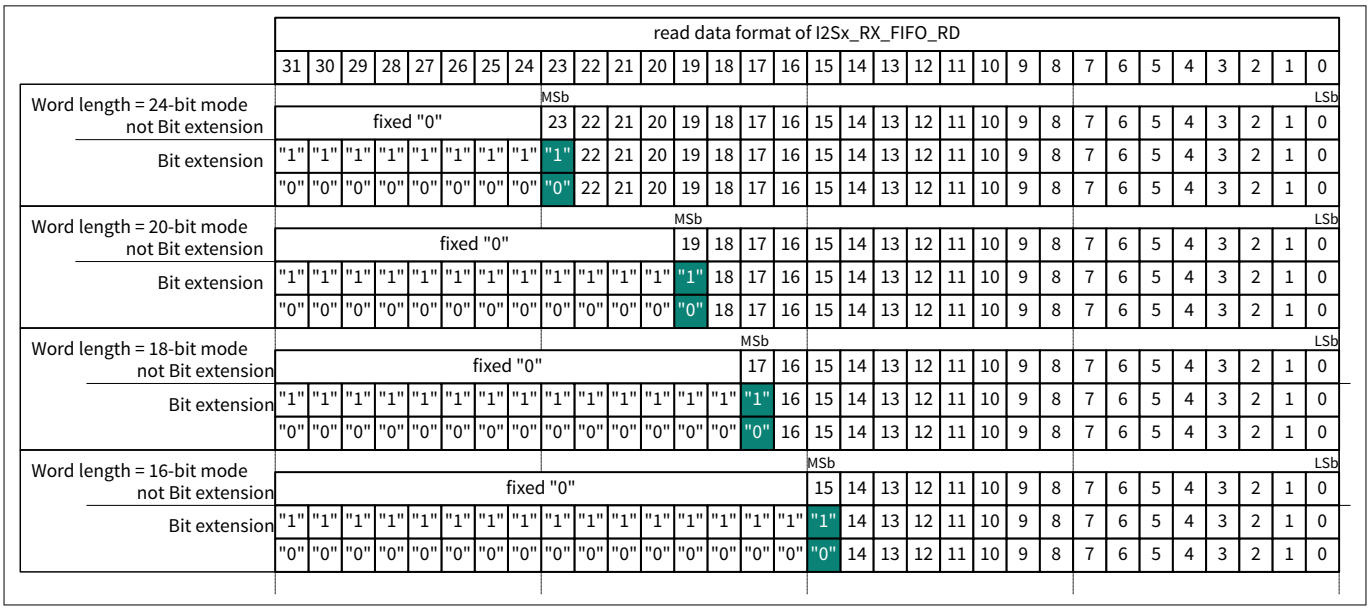


Figure 339 I2Sx_RX_FIFO_RD Register format for different word lengths

32.8 Interrupt support

The I²S block has one interrupt output signal that goes to the interrupt controller in the CPU. Refer to the Interrupts for details on the vector number of the I²S interrupt and the procedure to configure the interrupt priority, vector address, and enabling/disabling. The I²S interrupt can be triggered under any of the following events – TX_TRIGGER, TX_NOT_FULL, TX_EMPTY, TX_OVERFLOW, TX_UNDERFLOW, TX_WD, RX_TRIGGER, RX_NOT_EMPTY, RX_FULL, RX_OVERFLOW, RX_UNDERFLOW, or RX_WD. Each of the interrupt events can be individually enabled/disabled to generate an interrupt condition. The I2Sx_INTR_MASK Register is used to enable the required events by writing '1' to the corresponding bit. Irrespective of the INTR_MASK settings, if any of the events occur, the corresponding event status bit will be set by the hardware in the I2Sx_INTR Register. The I2Sx_INTR_MASKED Register is the bitwise AND of the I2Sx_INTR_MASK and I2Sx_INTR Registers. The final I²S interrupt signal is the logical OR of all the bits in the I2Sx_INTR_MASKED Register. So only those events that are enabled in the I2Sx_INTR_MASK Register are propagated as interrupt events to the interrupt controller.

Interrupts can also be triggered in software by writing to the corresponding bits in I2Sx_INTR_SET Register. Figure 340 illustrates the interrupt signal generation.

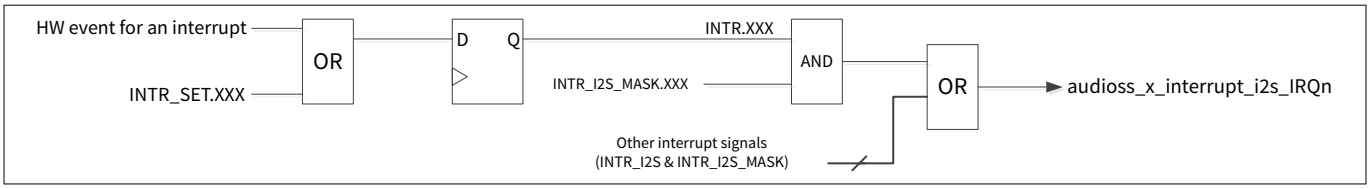


Figure 340 Interrupt signal generation

In the interrupt service routine (ISR), the I2Sx_INTR_MASKED Register should be read to know the events that triggered the interrupt event. Multiple events can trigger the interrupt because the final interrupt signal is the logical OR output of the events. The ISR should do the tasks corresponding to each interrupt event that was triggered. At the end of the ISR, the value read in the I2Sx_INTR_MASKED Register earlier should be written to the I2Sx_INTR Register to clear the bits whose interrupt events were processed in the ISR. Unless the bits are

not cleared by writing '1' to the I2Sx_INTR Register, the interrupt signal will always be high. A dummy read of the I2Sx_INTR Register should be done for the earlier register write to take effect.

32.9 Watchdog timer

The TX and RX blocks have independent watchdog timers, which can be used to generate an interrupt event if the Word Select (WS) input is idle for more than the configured time period. This feature is available only in the slave mode of operation where the external master drives the WS input lines (AUDIOSSx_TX_WS, AUDIOSSx_RX_WS). This feature can be used to detect any signal transmission issues, master device issues, or if the master has halted communication. If the master drives the same word select signal to both the AUDIOSSx_TX_WS and AUDIOSSx_RX_WS lines, then only one of the watchdog timers can be enabled to cause the interrupt event. Although the following explanation covers TX watchdog, the same explanation applies to RX watchdog as well.

To enable the TX watchdog timer feature, WD_EN bit in the I2Sx_TX_CTL Register should be set. The watchdog timer reload value (32-bit timer) is configured by writing to the I2Sx_TX_WATCHDOG Register. A value of zero written to the I2Sx_TX_WATCHDOG Register will also disable the watchdog timer. Figure 341 illustrates the watchdog behavior when the timer is enabled. The timer runs off the CLK_PERI system clock. Refer to the Clocking system for details on generation of CLK_PERI. The timer starts running when WD_EN and TX_START bits are set. The timer reload happens either on a rising edge event on AUDIOSSx_TX_WS input signal, or when the timer values reaches zero. When the timer value reaches zero, the TX_WD interrupt event is generated. The TX_WD bit in the I2Sx_INTR_MASK Register should be set to enable interrupt generation by the watchdog timer interrupt event. The interrupt event can be cleared by writing '1' to the TX_WD bit in the I2Sx_INTR Register.

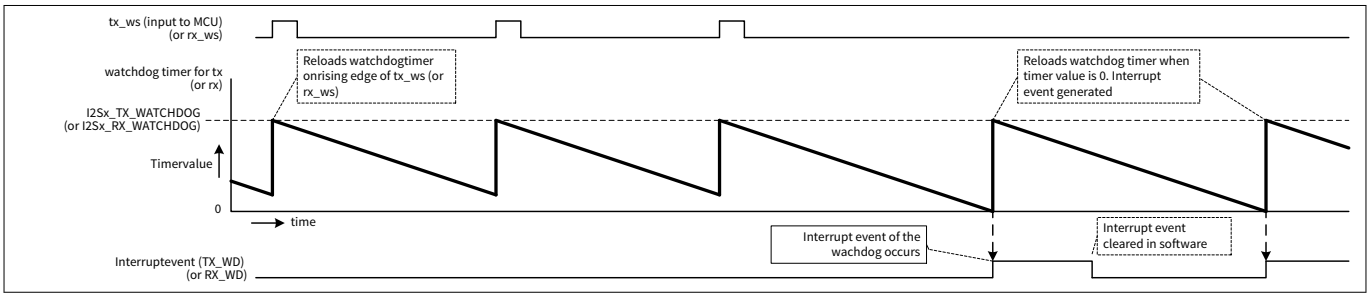


Figure 341 Watchdog timer working

32.10 MCLK output function

The I^2S unit generates MCLK output signal for an external audio codec. The MCLK output signal is generated only when the following conditions are met:

- I2Sx_CTL.TX_ENABLE or RX_ENABLE = 1 (I^2S is enabled)
- I2Sx_CLOCK_CTL.CLOCK_SEL = 0 (I^2S clock is from internal clock: CLK_HFx)
- I2Sx_CLOCK_CTL.MCLK_DIV = 0, 1, 2, or 3 (Division ratio: 1, 2, 4, or 8)
- I2Sx_CLOCK_CTL.MCLK_EN = 1 (MCLK output enabled)

Figure 342 illustrates the using of MCLK output signal for the external audio codec.

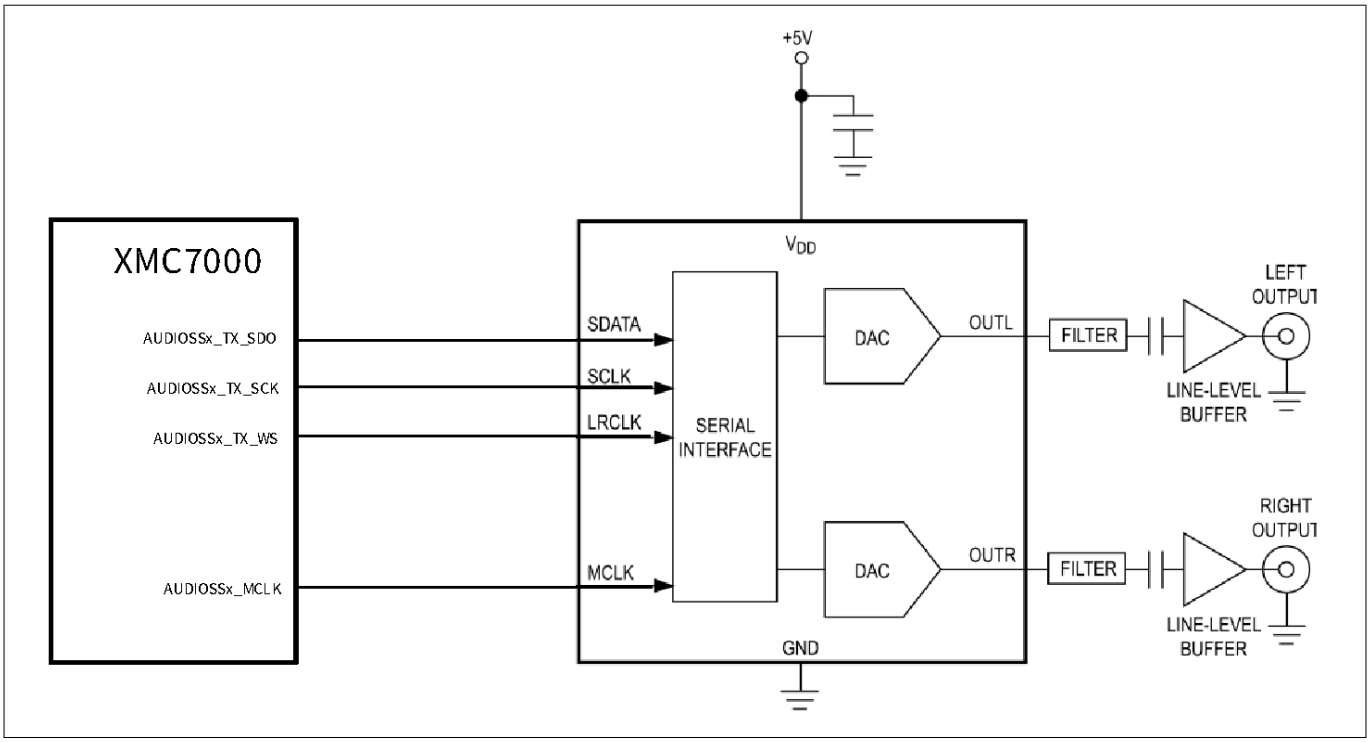


Figure 342 MCLK output signal for external audio codec

32.11 Registers

Table 274 Registers

Register	Name	Description
I2Sx_CTL	I²S Control Register	This register is used to control the I²S block
I2Sx_CLOCK_CTL	I²S Clock Control Register	This register can configure frequencies for I²S block and master clock. It should not be changed during communication
I2Sx_CLOCK_STAT	I²S Clock Status Register	This register indicates the status of the MCLK divider and is used to allow a controlled shutdown of the clock
I2Sx_CMD	I²S Command Register	This register can control I²S functions
I2Sx_TR_CTL	I²S Trigger Control Register	This register can enable DMA for I²S. It should not be changed during communication
I2Sx_TX_CTL	I²S Transmitter Control Register	This register can configure the transmitter for I²S functions. It should not be changed during communication
I2Sx_TX_WATCHDOG	I²S Transmitter Watchdog Register	This register can configure counter value for the TX watchdog. It should not be changed during the operation
I2Sx_RX_CTL	I²S Receiver Control Register	This register can configure the receiver for I²S functions. It should not be changed during communication

(table continues...)

Table 274 (continued) Registers

Register	Name	Description
I2Sx_RX_WATCHDOG	I^2S Receiver Watchdog Register	This register can configure counter value for the RX watchdog. It should not be changed during the operation
I2Sx_TX_FIFO_CTL	I^2S TX FIFO Control Register	This register can configure TX FIFO for I^2S
I2Sx_TX_FIFO_STATUS	I^2S TX FIFO Status Register	This register displays the status of TX FIFO for I^2S
I2Sx_TX_FIFO_WR	I^2S TX FIFO Write Register	This register stores transmission data for I^2S transfer
I2Sx_RX_FIFO_CTL	I^2S RX FIFO Control Register	This register can configure RX FIFO for I^2S
I2Sx_RX_FIFO_STATUS	I^2S RX FIFO Status Register	This register displays the status of RX FIFO for I^2S
I2Sx_RX_FIFO_RD	I^2S RX FIFO Read Register	This register stores received data for I^2S transfer
I2Sx_RX_FIFO_RD_SILENT	I^2S RX FIFO Silent Read Register	This register stores received data for I^2S transfer
I2Sx_INTR	I^2S Interrupt Register	This register displays the interrupt status
I2Sx_INTR_SET	I^2S Interrupt set Register	This register can be used to trigger an interrupt for firmware testing.
I2Sx_INTR_MASK	I^2S Interrupt Mask Register	This register controls whether the I^2S interrupt is forwarded to the corresponding processor
I2Sx_INTR_MASKED	I^2S interrupt Masked Register	Bitwise AND between the Interrupt Request and Mask Registers, so firmware can read the status of all mask enabled interrupt causes with a single load operation

33 SAR ADC

Top-level architecture

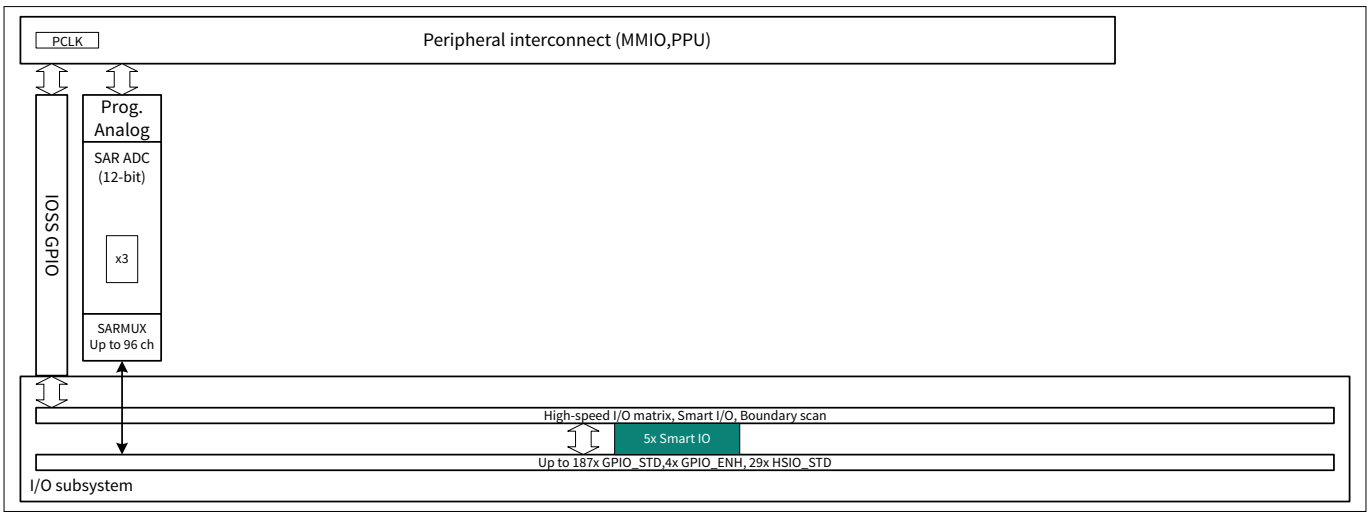


Figure 343 Analog system block diagram

XMC7000 features a successive approximation register analog-to-digital converter (SAR ADC), The SAR ADC is designed for applications that require a moderate resolution and high data rate. It consists of the following blocks:

- SARADC Core
- SARMUX
- SAR sequencer
- Diagnostic reference
- Reference buffer

SARMUX is an analog multiplexer to connect the signal sources to the ADC input; SARADC core then performs analog-to-digital conversion. A SAR sequencer is responsible for prioritizing the triggers requests, enable the appropriate analog channel, and control the sampling.

A single-ended SAR ADC system is capable of scanning up to 40 analog inputs (32 GPIOs and eight internal signals) as shown in the block diagram (see Figure 344).

33.1 Features

- SAR ADC core
  - 12-bit resolution with a maximum sample rate of 1 Msps
- 32 logical channels with the same capabilities
- Each logical channel can select input from
  - 32 analog input pins
  - Diagnostic signals
  - Analog input pins of other ADC units
  - Support for external mux (three select bits)
  - AMUXBUSA/B
- Scans triggered by timer, software, continuous, pins, or system triggers
  - Multiple ADC units can be triggered by the same trigger to ensure lock-step operation

  - Triggers can be cleared by software
  - Optional debug pause
- Double buffering of output data
- Programmable sample time for each channel
- Programmable post processing options for each channel
  - Sign/zero extension to 16-bit
  - Left/right alignment
  - Averaging: first order accumulate and dump, up to 256 samples
  - Programmable right shift
  - Range detection: below/above threshold, in/out-side range
  - Pulse detection: programmable positive and negative event counters
- Channels can be individual or grouped
  - Flexible grouping: from 32 groups with one channel to one group with 32 channels
- Group scans are dynamically scheduled by the hardware
  - Eight priorities, programmable per group
  - Four preemption types: resume, restart, cancel, or finish
  - Optional automatic idle power down
- Interrupt generation
  - Group scan done
  - Group scan done overflow detect
  - Group scan canceled
  - Per channel range detect
  - Per channel pulse detect
  - Per channel pulse/range overflow detect
- Output trigger generation per channel
  - Data ready/completion (each channel can trigger DW transfer)
  - Range violation detected
- Digital and analog calibration available
- Programmable offset and gain calibration
  - Non-intrusive background recalibration
  - Coherent calibration update
- Support for diagnostic measurements including broken wire detection. This includes:
  - ADC sampling capacitor preconditioning feature
  - Selectable current source or sink on selected ADC input while sampling
  - Support for LED diagnostics (see Trigger outputs for details)
- On-chip temperature sensor and power monitoring

33.2 Block diagram

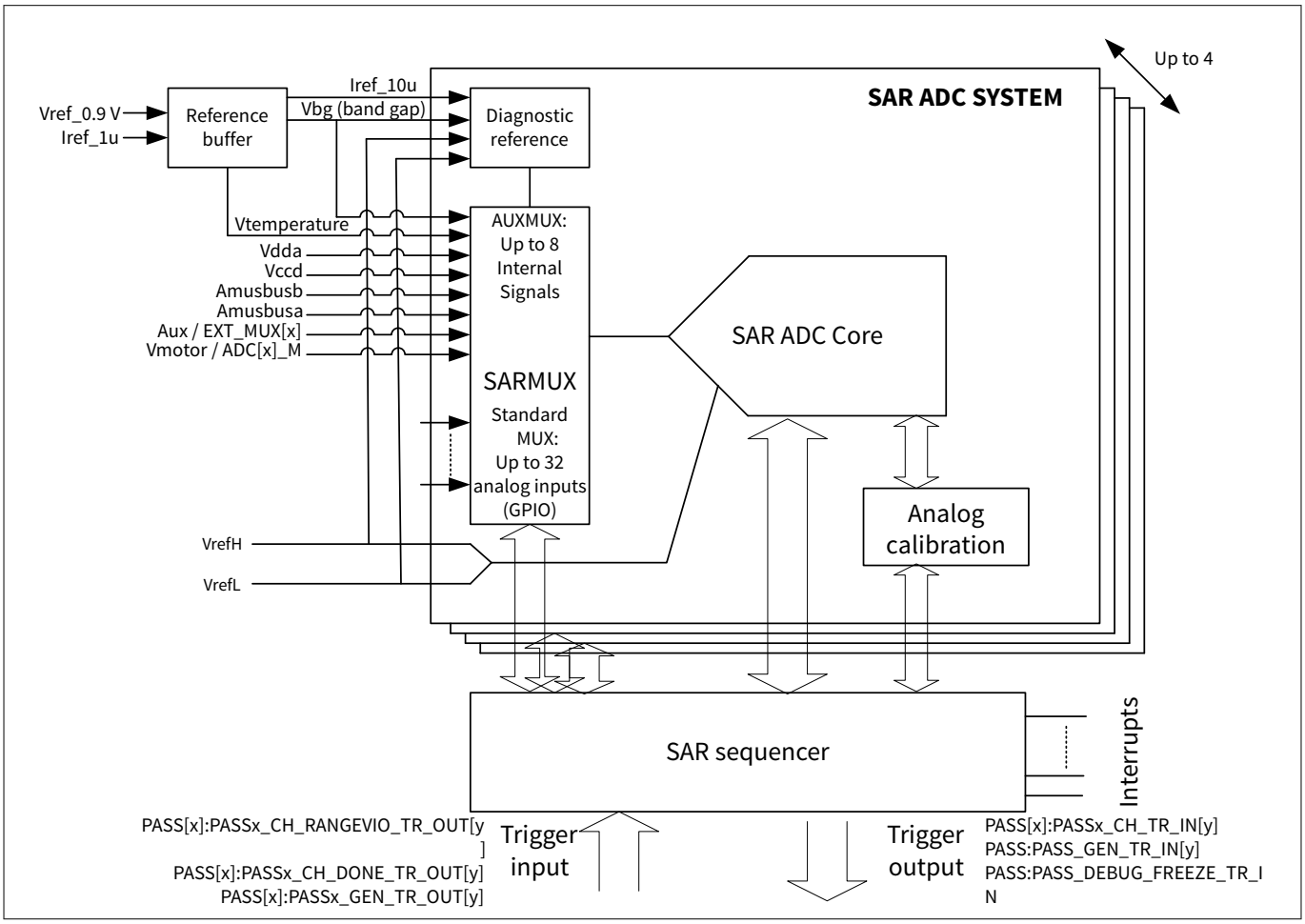


Figure 344 SAR ADC system block diagram

33.3 Operation

SAR conversion begins when a trigger signal is received by the SAR sequencer. The sequencer selects the appropriate analog input for the logical channel to be converted and triggers the analog-to-digital conversion and manages conversion results. If the sequencer is performing a group conversion, it proceeds to the next channel in the group and starts another conversion.

To perform a conversion, the ADC starts with optional preconditioning interval, which pre-charges or discharges the sampling capacitor, if preconditioning is enabled. The appropriate analog switches are then enabled, and the desired input is connected to the ADC. Signal sampling occurs when the ADC core start conversion signal (STC) goes high (from sequencer). STC remains high during the signal sampling window for a user specified number of clocks. At the end of the sampling window, STC goes low starting a 13 to 15 clock conversion cycle. At the end of conversion, the results can be post-processed; for example, averaging, range detection, and pulse detection. Results are stored in dedicated double-buffered locations for each virtual channel. Figure 345 shows a simple block diagram of the ADC core.

Several SARs can operate in lock-step for simultaneous conversion of analog inputs. This configuration is useful for brushless motor control, multi-phase power conversion, and other applications where simultaneous sampling is needed.

The analog input multiplexer (SARMUX) is implemented with two-levels of transmission gate switches. Input selection is performed by addresses stored in the logical channel configuration. Each channel can select an input signal, a diagnostic reference signal, or both.

The standard analog multiplexer has 32 inputs for signals from I/O pins, and supports another eight channels to measure special internal signals such as a bandgap reference voltage, a temperature sensor voltage, power supply pins, and long-reach signals to other input pads through GPIO AMUXBUSA/B signals. One channel is also reserved for motor sensing inputs. The SARMUX can use an expansion signal to reach other SARMUXs when their ADCs are not in use.

The following sections will include detailed descriptions of various aspects of the ADC system including:

- SAR ADC core
- SARMUX
- SAR sequencer
- Triggering and scheduling
- Output triggers and interrupts
- Diagnostic reference generator
- Reference buffer

***Note**:*
- *The peripheral clock divider for ADC (CLK_PERI) must be at least 2. The ePass SAR requires a 50/50 duty cycle clock; this is generated only when CLK_PERI is at least 2.*
- *Do not divide CLK_GR9; this makes CLK_GR9 = CLK_PERI, keeping all clocks coming to SAR ADC at the same frequency. If these clocks are not equal, it can cause the GRP_CANCELLED bit to be set incorrectly during ABORT_CANCEL preemption.*

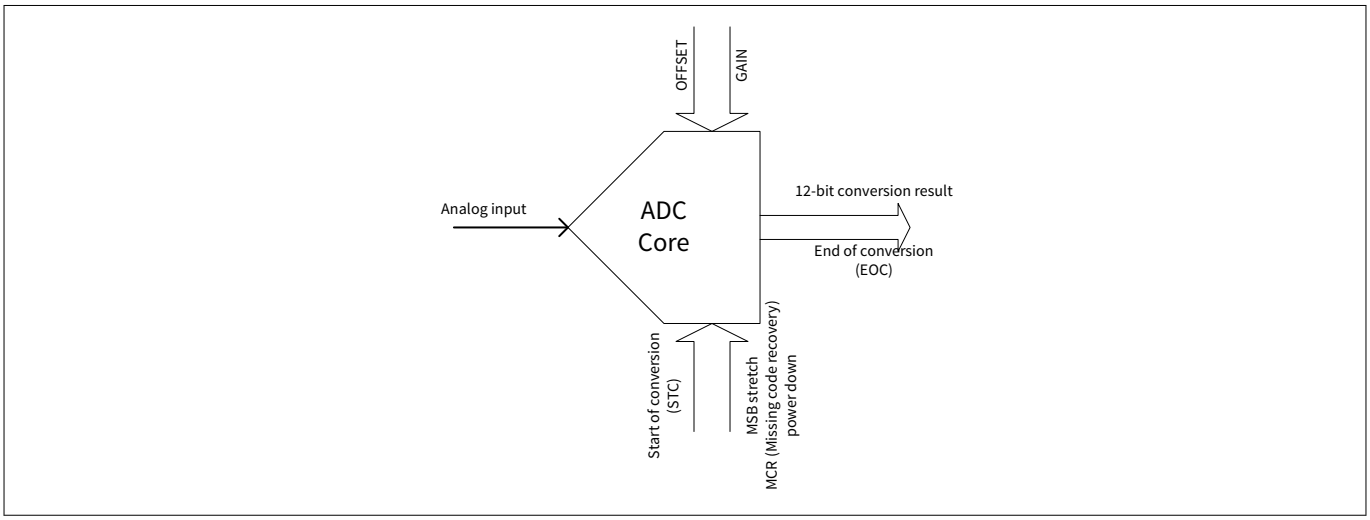


Figure 345 ADC core block diagram

33.3.1 SAR ADC conversion flow

Analog-to-digital conversion starts with a trigger received by the sequencer. All the channels in the group are then sequentially scanned and converted until the end of the group. After the end channel of the group is converted, the group conversion done interrupt is set. The size of the group can vary from one to 31 channels. Figure 346 shows the flow of the analog-to-digital conversion, and the important steps are explained here:

1. Analog-to-digital conversion request trigger and the trigger pending bit (PASSx_SARy_TR_PEND.TR_PEND) is set for the first logical channel of the group. The first channel should be triggered to scan the complete group.
2. If the ADC is in power-down state, a latency is added before the first conversion due to power up settling time. This latency is software configurable (PASSx_SARy_CTL.PWRUP_TIME).
3. Sampling the analog input to the mapped logical channel; sample time is configurable for each channel PASSx_SARy_CHz_SAMPLE_CTL.SAMPLE_TIME.

4. Analog-to-digital conversion of the sampled input.
5. Sample and analog-to-digital conversion for the next channels in the group (repeat steps 3 and 4).
6. Conversion of the last channel in the group is completed. The new result is coherently updated in the PASSx_SARy_CHz_RESULT Register of all the channels. Group conversion done interrupt is set and trigger pending bit is cleared.
7. The group conversion done interrupt is cleared by writing '1' to the Interrupt Request Register PASSx_SARy_CHz_INTR.
8. Start the next group conversion if there is a trigger or go to power-down state if idle and auto idle power-down PASSx_SARy_CTL.IDLE_PWRDWN is enabled.

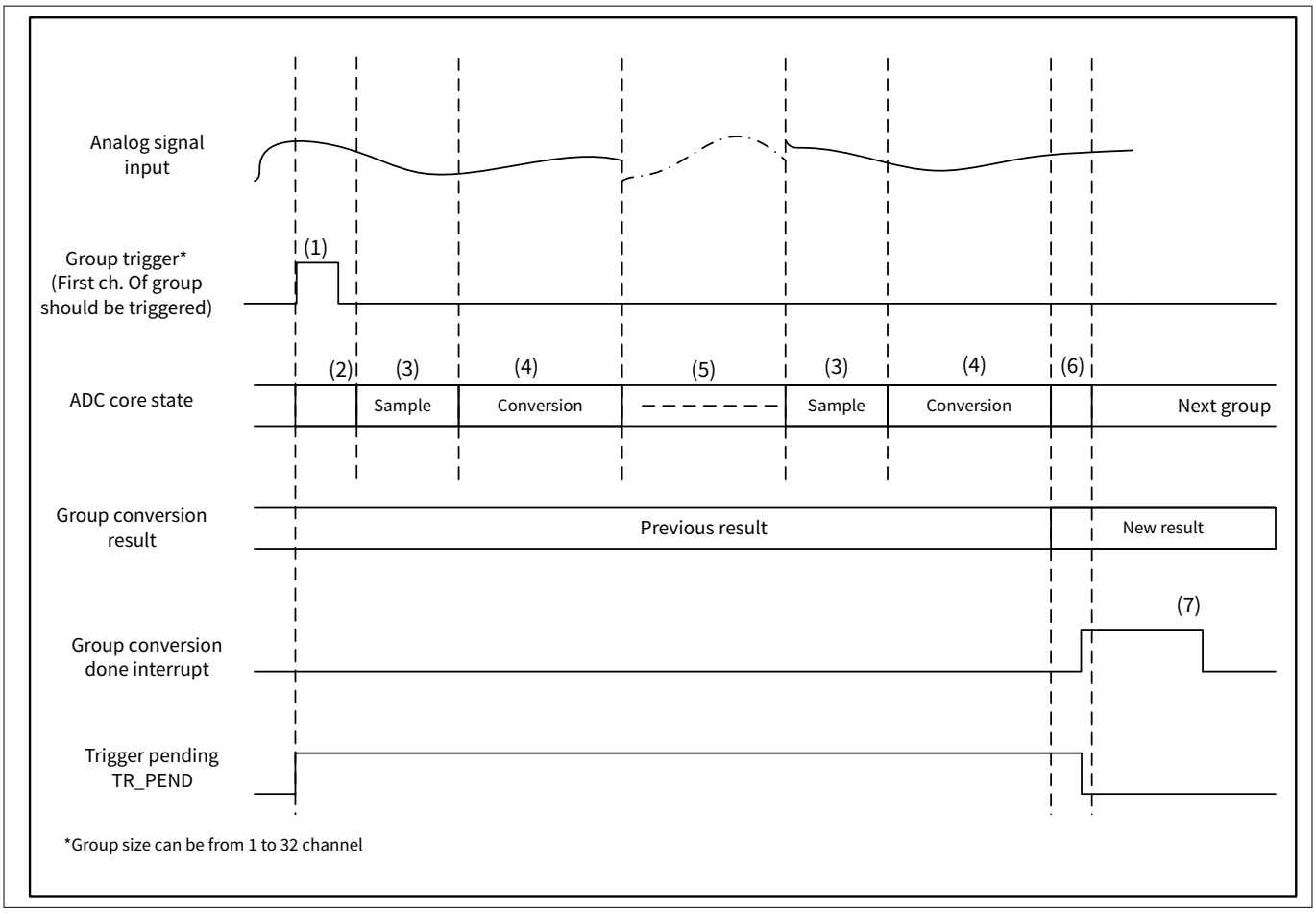


Figure 346 ADC conversion timing flow

33.3.2 Result data format

The result after conversion is stored in the lower 16 bits of a 32-bit register, PASSx_SARy_CHz_RESULT (see Table 276). The upper 16 bits store the mirror bits of certain flags. Depending on the configuration the possible result data formats are shown Figure 344.

33.3.2.1 Signed/unsigned result

Conversion results can be treated as signed or unsigned. The PASSx_SARy_CHz_POST_CTL.SIGN_EXT bit is used to set the format. Unsigned is the default and is effectively a 12-bit value zero-extended. Considering the result, signed can be useful when VREFH/2 is the virtual analog ground. The 12-bit code for a signal at VREFH/2 is 0x800. This means 0x800 is considered 0, any value below 0x800 is considered negative, and values above 0x800 are considered positive. Therefore, when ‘signed’ is set, the MSb (bit 11) is inverted and sign extended.

33.3.2.2 Alignment

A 12-bit result can either be right-aligned (default) or left-aligned within the lower 16 bits of the PASSx_SARy_CHz_RESULT Register. This is configured per channel with the PASSx_SARy_CHz_POST_CTL.LEFT_ALIGN bit. This feature is sometimes used for fixed point arithmetic. For example, this allows for a 12-bit conversion results to be compared to a 16-bit result from a conversion with averaging.

This post processing step takes the 16-bit output from the Right shift step. When PASSx_SARy_CHz_POST_CTL.LEFT_ALIGN is used the 16-bit data is shifted four bits to the left; this is done with the assumption that only the lower 12 of the 16 bits are used. The output from the step is still 16-bit.

Table 275 Result data format

Alignment	Signed/ unsigned	Result Register															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Right	Unsigned	-	-	-	-	11	10	9	8	7	6	5	4	3	2	1	0
Right	Signed	11	11	11	11	11	10	9	8	7	6	5	4	3	2	1	0
Left	-	11	10	9	8	7	6	5	4	3	2	1	0	-	-	-	-

Table 276 Channel Result Register

SARx_CHy_RESULT

Field	Bits	Access	Default	Description
RESULT	15:0	R	-	Conversion result of the channel. Data is copied from the WORK Register after all the enabled channel of the group are sampled.
ABOVE_HI_MIR	28	R	-	Set if the result of range detect was above RANGE_HI or cleared otherwise. Note this is only done for the OUTSIDE_RANGE mode, for all other range detection modes this bit is undefined.
RANGE_INTR_MIR	29	R		Mirror bit of INTR_CH_RANGE bit
PULSE_INTR_MIR	30	R		Mirror bit of INTR_CH_PULSE bit
VALID_MIR	31	R		Mirror bit of the corresponding bit in RESULT_VALID Register

33.3.3 Acquisition/sample time

The SAR ADC acquisition consists of two steps. In the first step, the sample window, the analog input signal is sampled on the sampling capacitor in the ADC core and in the second step that voltage value is converted to the corresponding digital code.

To get an accurate conversion the analog input signals need to have sufficient time to charge the sampling capacitor. This time is called the 'sample time'. The right sample time depends on the drive strength of the signal source and the RC delay of the whole signal path, including the chip pin, SARMUX, sample capacitor, and wiring. As a result, the required sample time may be different for each signal.

In this SAR ADC, each channel configuration has its own sample time definition (PASSx_SARy_CHz_SAMPLE_CTL.SAMPLE_TIME). This enables optimizing the sample time for each separate analog signal, which in turn enables optimal use of the ADC resource (and power).

Given the fixed on-chip signal path and maximum allowed current draw from the signal source, the minimum sample time can be calculated. This time needs to be translated to the number of SAR clock cycles. PASSx_SARy_CHz_SAMPLE_CTL.SAMPLE_TIME (Table 277) is a 12-bit field and the legal values are [1….4095] (0

will interpreted as 1). At the SAR clock frequency of say, 20 MHz, this corresponds to a sample window range of 50 ns to 0.2 ms. The recommended minimum sampling time for proper settling of the signal is 412 ns; refer to the device datasheet for the exact minimum sample time requirement. The maximum clock frequency for SAR ADC is 26.7 MHz (80/3 MHz) to achieve the 1 MS/s throughput; the recommended sample time corresponds to ~11 clock cycles at this frequency.

The converter requires 13 to 15 clock cycles to perform successive approximation of sampled voltages and present the conversion results. Basic conversion takes 13 cycles with an extra cycle required if the MSB stretch option is enabled (PASSx_SARy_CTL.MSB_STRETCH bit) and another clock is required if a missing code recovery mode (MCR) is enabled (PASSx_SARy_CTL.HALF_LSB bit). To simplify the use, the SAR sequencer may typically use 15 clocks regardless of options, with unused clocks transparently added to sampling clocks.

***Note**: See the device datasheet to calculate the sampling time.*

Table 277 Channel sample time setting field

SARx_CHy_SAMPLE_CTL				
Field	**Field**	**Field**	**Field**	**Field**
SAMPLE_TIME	27:16	RW	-	Sample time in ADC clock cycles. Minimum value is ‘1’ (Setting 0 gives same result as ‘1’)

33.4 SARMUX

The SARMUX is the analog multiplexer to route the signal to be converted to the ADC core input. The number of GPIO inputs can be up to 32 but the actual connected pins may vary with the device variants. In addition, it can support up to eight internal signals. The selection of input signal is controlled by the physical address field of each virtual channel PASSx_SARy_CHz_SAMPLE_CTL.PIN_ADDR. SARMUX can access and route the analog signal from other SARMUXs through an expansion signal. Figure 347 shows a high-level block diagram of the SARMUX.

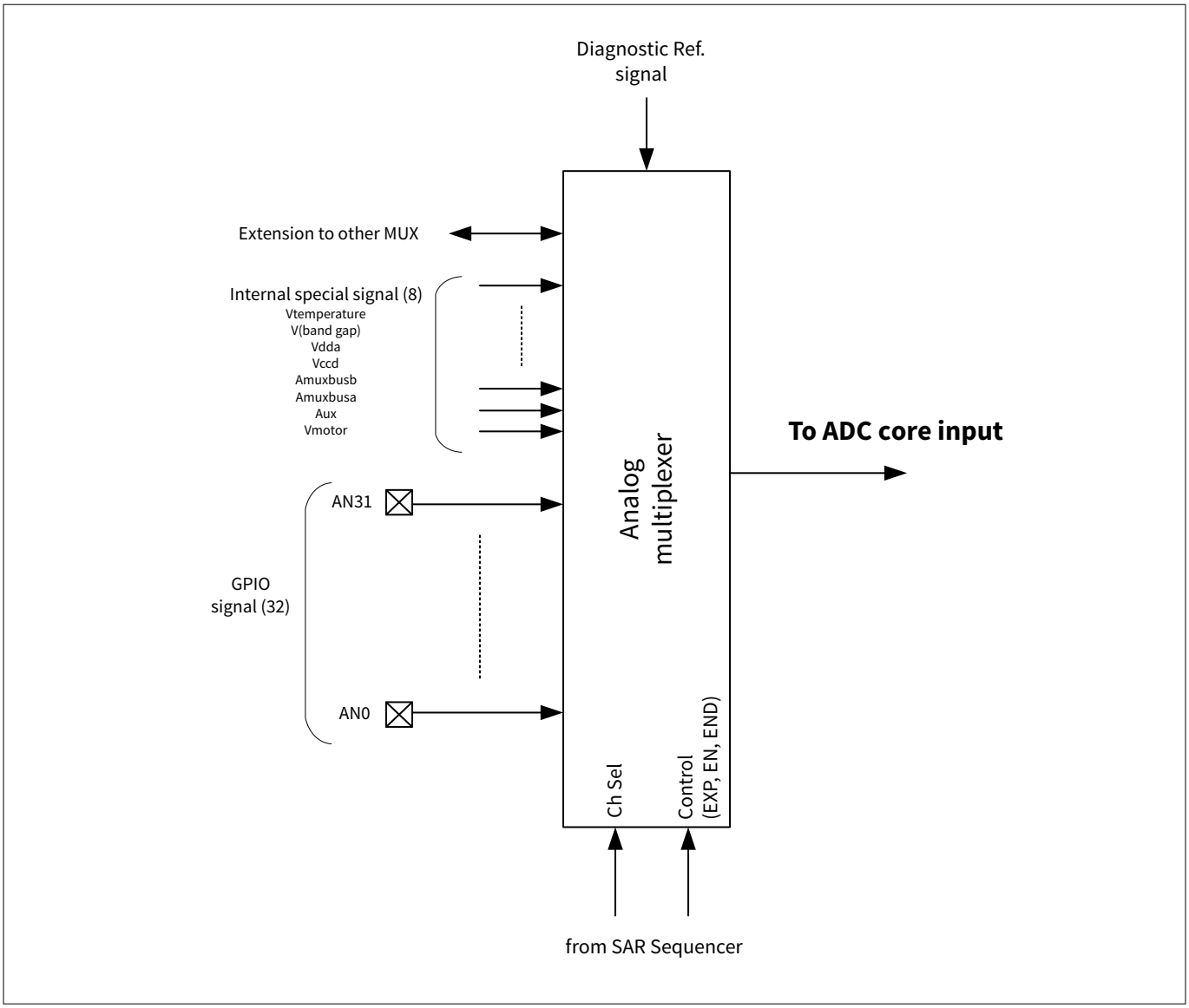


Figure 347 SARMUX block diagram

33.4.1 Preconditioning

For functional safety and diagnostics, the SAR sequencer supports preconditioning, which enables broken wire detection by charging or discharging the ADC sampling capacitor before sampling the input signal. The use of this feature is optional and is defined by the PASSx_SARy_CHz_SAMPLE_CTL.PRECOND_MODE field (see Table 278) of the channel configuration. There are four possible selections:

- OFF – no preconditioning
- VREFL – discharge to VREFL
- VREFH – charge to VREFH
- DIAG – connect to the diagnostic reference output during preconditioning

When DIAG preconditioning is used, the diagnostic reference should be configured to output a reference voltage. Note that for overlap diagnostics, the diagnostic reference needs to be configured to supply an i_{bias} current. There is one diagnostic reference per ADC with a global (not per channel) configuration; therefore, DIAG preconditioning is mutually exclusive with overlap diagnostics. The duration of preconditioning is configurable through the Register PASSx_SARy_PRECOND_CTL.PRECOND_TIME. This time is specified in SAR clock cycles.

33.4.2 Overlap diagnostic

Overlap diagnostics is another functional safety feature. For overlap diagnostics, the diagnostic reference typically sources or sinks a small i_{bias} current. In this case, the diagnostic reference output and the analog input signal are both connected to the ADC sampling capacitor at the same time.

The use of this feature is optional and is defined by the PASSx_SARy_CHz_SAMPLE_CTL.OVERLAP_DIAG field of the channel configuration. There are three overlap diagnostics modes:

- OFF– No overlap diagnostics
- HALF – Overlapping diagnostic reference for the first half of the sample window
- FULL – Overlapping diagnostic reference for the full sample window

For FULL, the sample window duration is defined by the PASSx_SARy_CHz_SAMPLE_CTL.SAMPLE_TIME. However, for the HALF overlap diagnostic mode, the PASSx_SARy_CHz_SAMPLE_CTL.SAMPLE_TIME defines the duration of only half the sample window.

33.4.3 SARMUX diagnostics

SARMUX diagnostics is a functional safety feature, used to verify the connection from the selected SARMUX input to ADC sampling capacitor. This is done by connecting only the diagnostic reference output to the selected SARMUX input. Note that this does not disturb the analog input signal (for analog details, see SARMUX).

The SARMUX diagnostics mode is a per channel optional feature, which is selected by setting the PASSx_SARy_CHz_SAMPLE_CTL.OVERLAP_DIAG field. The diagnostic reference should be configured to provide one of the available reference voltages.

Table 278 Preconditioning mode selection

SARx_CHy_SAMPLE_CTL

Field	Bits	Access	Default	Description
PRECOND_MODE	13:12	RW	–	Select the Preconditioning mode for the channel. 00 - No preconditioning 01 - Discharge to VREFL 10 - Charge to VREFH 11 - Connect to Diagnostic reference output (the diagnostic reference generator must be configured accordingly)

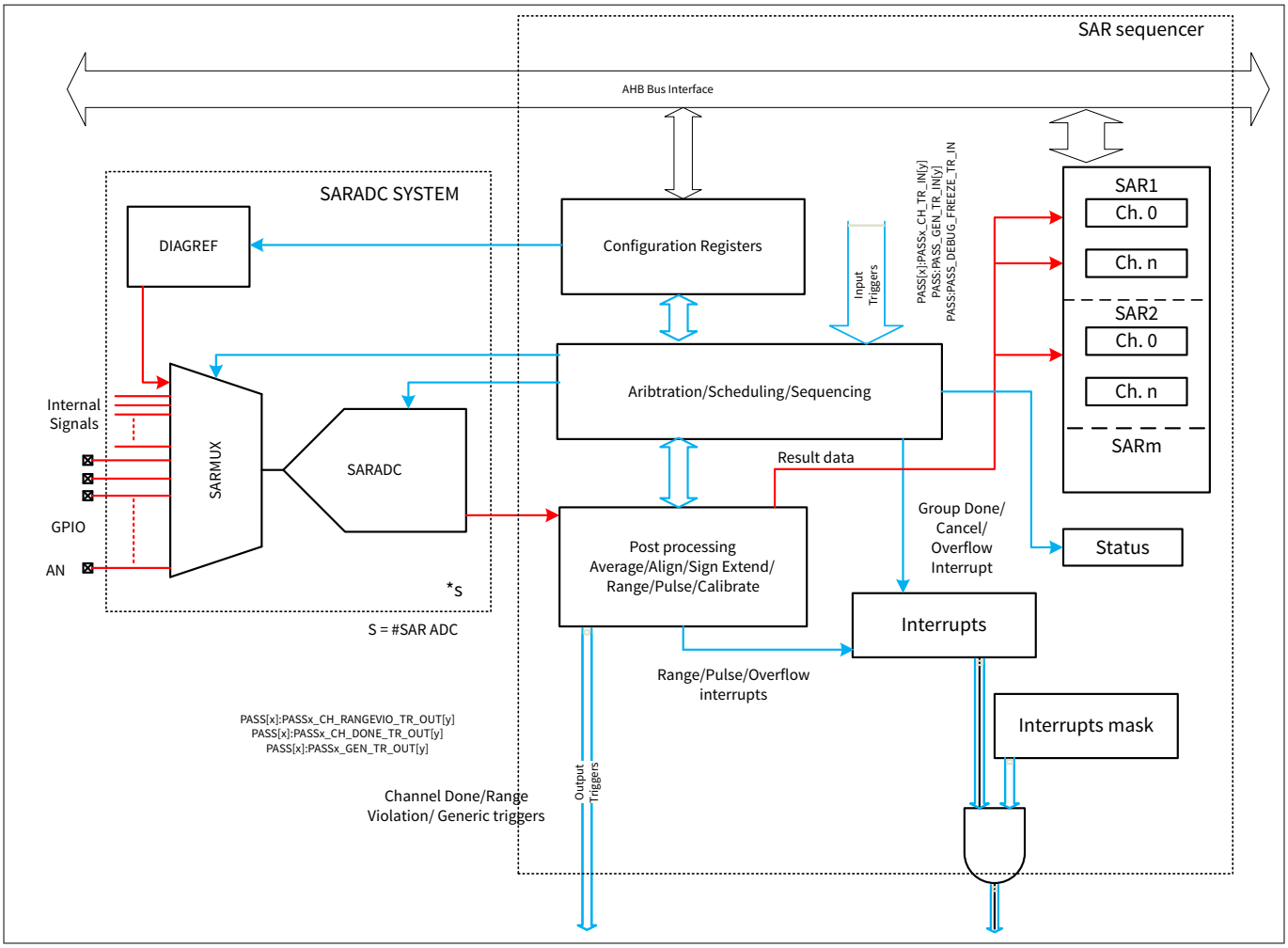


Figure 348 SAR sequencer block diagram

33.5 SAR sequencer

The SAR subsystem is largely autonomous, arbitrating acquisition requests (triggers), performing acquisitions, and optionally post processing results without firmware intervention. This important characteristic enables real-time measurements without loading the CPU. The SAR sequencer optionally generates various interrupts to enable further CPU processing of the results or to handle errors. The SAR sequencer also generates several triggers to enable low-latency handling of a detected error or to have the data picked up by DataWire.

33.5.1 Analog input selection

There are up to 32 analog pins connected to the regular SARMUX inputs. The number of analog pins is device-specific (refer to the device datasheet for details). In addition, 10 special analog signals can be selected. Eight are selected through the SARMUX and the remaining two are VREFL and VREFH, which bypass the SARMUX and are directly selected at the ADC core input (see Figure 344). These two signals are used for calibration.

One of the eight special analog signal is an on-chip temperature sensor. There is only one temperature sensor, which is shared between all ADCs. For correct operation, the temperature sensor should not be connected to more than one ADC at any given time.

One of the 42 analog signals can be selected by setting the PASSx_SARy_CHz_SAMPLE_CTL.PIN_ADDR field (see Table 279) of the respective channel. The selected analog signal will only be connected to the ADC during the sample window. If an acquisition is aborted during the sample window, then it is guaranteed that there will be at least one break-before-make cycle (SAR clock) before the new signal is connected to the ADC.

33.5.2 External analog multiplexer

The SAR sequencer supports the use of an external analog mux. This can be used to expand the number of analog signals that can be sampled beyond the number of analog pins. Each channel configuration has its own 3-bit wide external mux select value (PASSx_SARy_CHz_SAMPLE_CTL.EXT_MUX_SEL in Table 279). This allows up to eight channels to use the same analog input pin with different select values. The three external mux select signals are connected at chip level to GPIO digital outputs.

The per channel external mux enable bit (PASSx_SARy_CHz_SAMPLE_CTL.EXT_MUX_EN) is next to the external mux select. This can be used as a chip select for the external mux select device. It is also connected at chip level to a GPIO digital output. Note this enable is not used as output enable for the GPIO digital output drivers of the external mux select. Additionally, when the PASSx_SARy_CHz_SAMPLE_CTL.EXT_MUX_EN bit is low, the PASSx_SARy_CHz_SAMPLE_CTL.EXT_MUX_SEL field will be ignored.

Table 279 External analog multiplexer select and enable

SARx_CHy_SAMPLE_CTL

Field	Bits	Access	Default	Description
EXT_MUX_SEL	10:8	RW	–	External analog multiplexer select bits
EXT_MUX_EN	11	RW	–	External analog mux enable This can be used as enable (chip select) for the external analog mux (it is not used as enable for the GPIO output driver).

33.5.3 Port selection

Each ADC is preceded by its own SARMUX, which connects to a distinct set of up to 32 analog pins. This means that ADC1 cannot sample the analog pins connected to ADC2. In some cases, it may be desirable to have one ADC being able to reach all the analog inputs of the chip. To support this use-case the ADC0 channels have an additional PASSx_SARy_CHz_SAMPLE_CTL.PORT_ADDR field.

With this field, ADC0 can be connected to the output of the SARMUXes of the other ADCs. This is done through the 'expansion' bus shown in Figure 347.

Note that ADC0 can only use the SARMUX of another ADC if that ADC is disabled (PASSx_SARy_CTL.ADC_EN = 0), while SARMUX for that ADC is enabled (PASSx_SARy_CTL.ENABLED = 1 and PASSx_SARy_CTL.SARMUX_EN = 1).

When ADC0 borrows another SARMUX it may need a longer sample time due to the additional on-chip wiring and connected switches.

33.5.4 Averaging

The SAR sequencer includes basic averaging functionality for every channel. When enabled for a channel the SAR sequencer will do back-to-back acquisitions of the same signal and accumulate the results (after sign extension) in a 20-bit accumulator. This is also referred to as a “first order accumulate and dump” filter.

Averaging is fully configured per channel by the PASSx_SARy_CHz_POST_CTL Register (see Table 282). The number of samples averaged is determined by the 8-bit PASSx_SARy_CHz_POST_CTL.AVG_CNT field. The number of samples averaged is AVG_CNT+1, which gives a range of [1...256].

For true averaging, the averaging count needs to be a power of 2 and the right shift needs to be set to the corresponding value. For non-power of 2 averaging counts the right shift can only approximate the required divide. If a true averaging result is required, the software will need to do a divide. Note that the acquisitions for averaging are considered to be atomic, i.e. when the channel is aborted due to a preemption then the results are discarded and on return the averaging starts from scratch. On the flip side when the FINISH_RESUME preemption type is used, or in case of a debug freeze trigger, all averaging acquisitions are completed before the preemption or freeze happens.

Also note that using averaging for a low priority channel (such as background re-calibration) is problematic when the ADC is highly loaded as it is unlikely the averaging will ever complete. In such a scenario it is more practical to do the averaging in software.

Note that averaging is mutually exclusive with pulse detection.

33.5.5 Right shifting

The right shift post processing step, originally only intended for averaging, allows the up to 20-bit averaging result to be shifted right so that it fits in the 16-bit RESULT Register. Now that it is independent of averaging, it can also be used to make a regular 12-bit result fit in 8-bit. The right shift step is configured per channel by the SHIFT_R field. The SHIFT_R is a 5-bit field, but the legal values are only [0..12]. This is sufficient to allow a 20-bit result to be shifted right to fit into the lower 8-bits. The right shift is an arithmetic shift to the right, i.e. depending on the SIGN_EXT configuration, sign extension or zero-extension will be used.The right shift post processing step takes the 20-bit output from the averaging step, then right shift by 4, resulting in an output of a 16-bit result by eliminating the 4 least significant bits.

33.5.6 Range detect

The SAR sequencer supports optional range detection feature. Range detection enables a check against up to two programmable threshold values (see Table 281) without CPU involvement. The result is a fast, fixed latency, response time, which is a critical requirement for some use-cases.

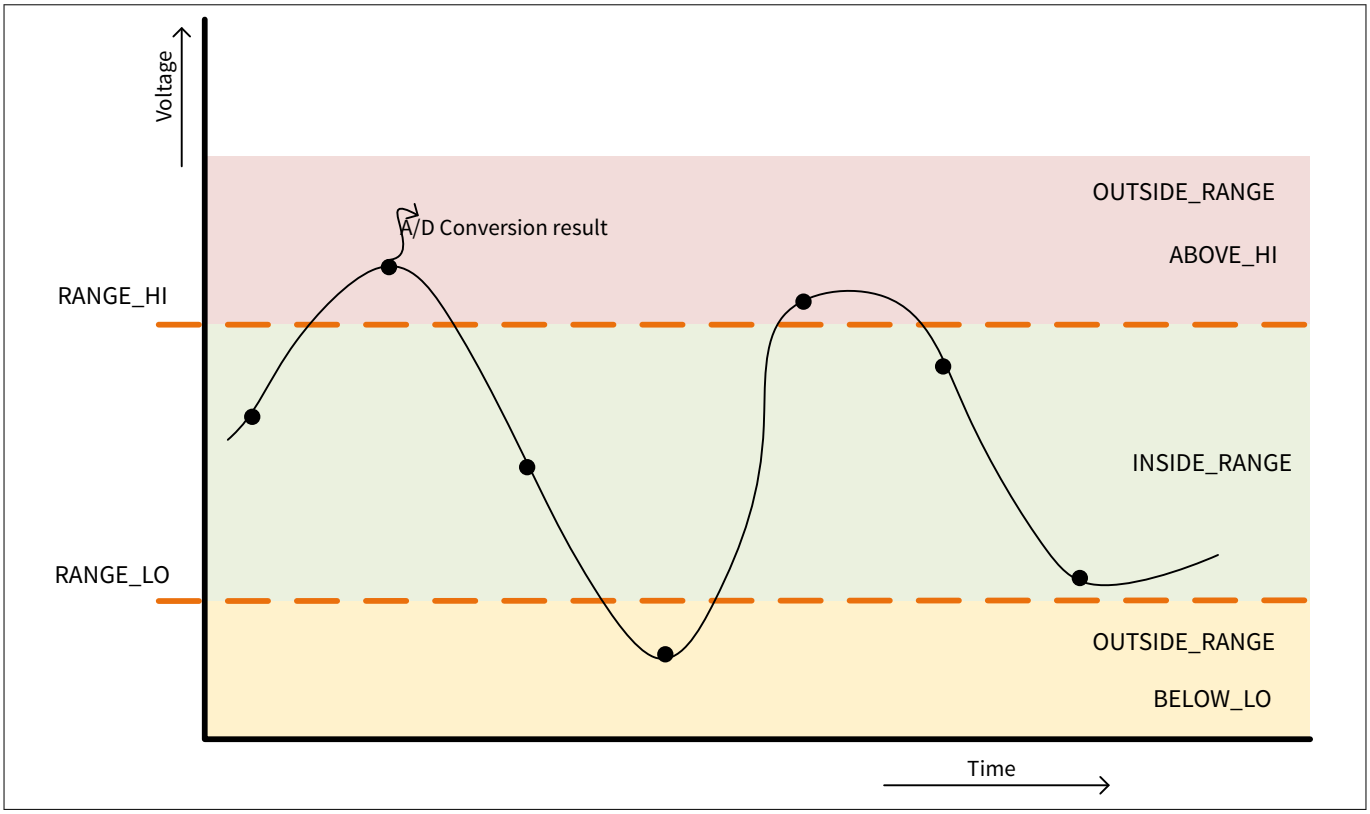


Figure 349 Range detection, threshold, and events

Range detection is defined by two 16-bit threshold values and a mode field selecting one of four possible modes. Both the mode (PASSx_SARy_CHz_POST_CTL.RANGE_MODE) and the two thresholds (PASSx_SARy_CHz_RANGE_CTL.RANGE_LO and PASSx_SARy_CHz_RANGE_CTL.RANGE_HI) are configured per channel. The available range detection modes are:

- BELOW_LO (RESULT < RANGE_LO)
- INSIDE_RANGE (RANGE_LO ≤ RESULT < RANGE_HI)
- ABOVE_HI (RANGE_HI ≤ RESULT)
- OUTSIDE_RANGE (RESULT < RANGE_LO) || (RANGE_HI ≤ RESULT)

Range detection uses the 16-bit PASSx_SARy_CHz_RESULT.RESULT from the left-/right-align step. This means that the threshold values need to be in the same format as the PASSx_SARy_CHz_RESULT.RESULT after all the preceding post processing steps (including averaging). The event flag will be set when the range mode condition evaluates to true.

When the event flag is set, the PASSx_SARy_CHz_INTR.CH_RANGE interrupt will be set and a pulse is output on the range violation trigger (see Output triggers and interrupts).

Note that the range detection results (trigger and interrupt) are in addition to the data result.

However, if pulse detection is also enabled for the channel then neither the range detection results nor the data result will be visible; instead, only the range detect event flag is forwarded to the pulse detect feature.

33.5.7 Pulse detect

The SAR sequencer supports optional pulse detection. Pulse detection is used to filter the events resulting from range detection. The pulse detection filter counts events to detect 'sufficiently long' high pulses while ignoring 'short enough' low spikes.

The pulse detection filter consists of an 8-bit positive event counter and a 5-bit negative event counter. These event counters decrement and/or reload based on the range detection event. When the event is high it is called a positive event and when low, it is called a negative event. The reload values are per channel pulse detection configuration settings. The positive reload value determines what is considered a 'sufficiently long' high pulse and the negative reload value determines which low spikes are 'short enough' to ignore (which is equivalent to; too long not to ignore).

Note that both the pulse detection filter and averaging are used to filter noise. Only one of these two methods can be used for a channel; these two features are mutually exclusive, and their configuration fields are mapped to the same bits.

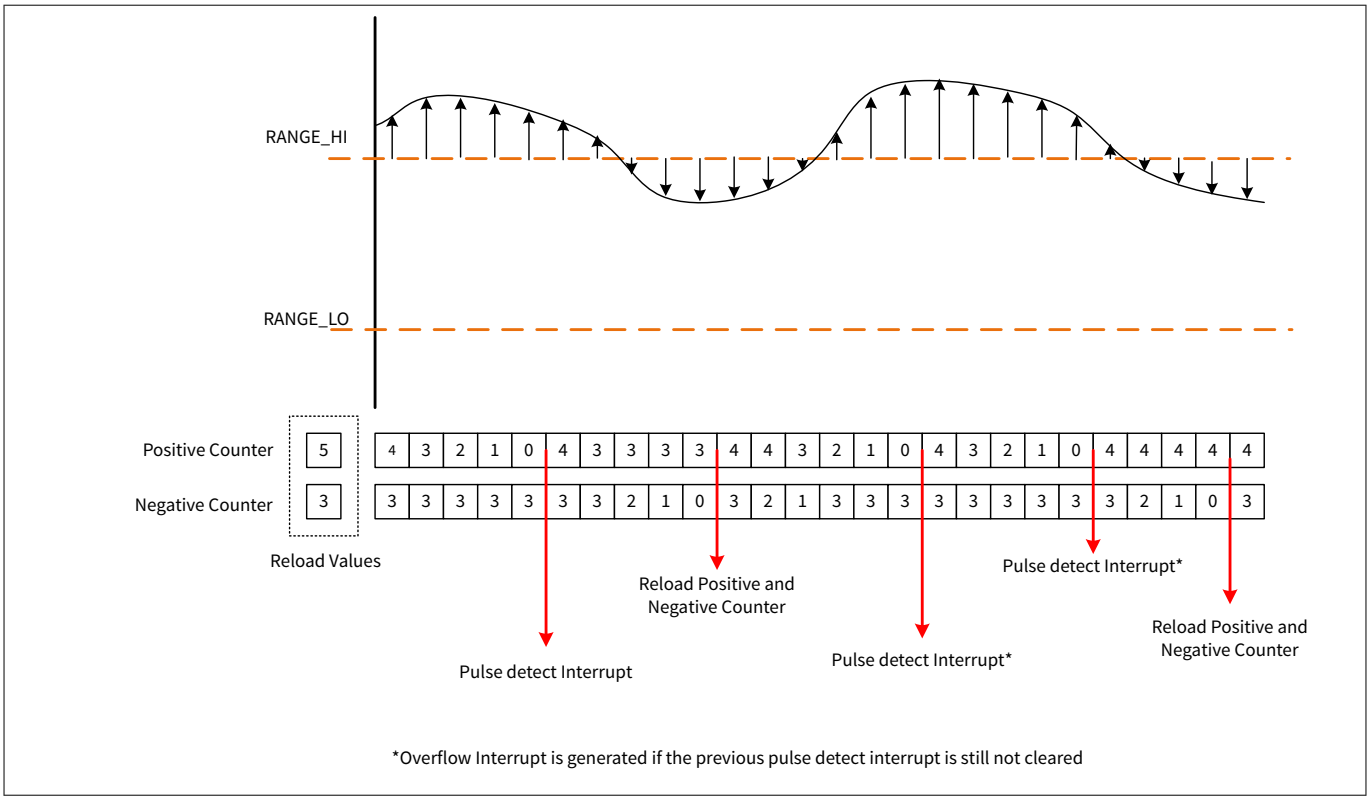


Figure 350 Pulse detection

Table 280 Channel Sample Control Register

SARx_CHy_SAMPLE_CTL

Field	Bits	Access	Default	Description
PIN_ADDR	5:0	RW	–	Address of the analog signal (pin) to be sampled by this channel 0..31 - AN0…AN31 32 - Select motor input 33 - Select auxiliary input 34 - AMUXBUSA 35 - AMUXBUSB 36 - Digital power supply (VCCD) 37 - Analog power supply (VDDA) 38 - Bandgap voltage from SRSS 39 - Temperature sensor 40..61 - Reserved 62 - VREFL 63 - VREFH
PORT_ADDR	7:6	RW	–	Select Physical Port. This field is only valid for SAR0 (or ADC0) 00 - SARMUX0 (SAR0 uses its own MUX) 01 - SARMUX1 (SAR0 uses MUX of SAR1) 10 - SARMUX2 (SAR0 uses MUX of SAR2) 11 - SARMUX3 (SAR0 uses MUX of SAR3)

Table 281 Channel Range Control Register

SARx_CHy_RANGE_CTL				
Field	**Bits**	**Access**	**Default**	**Description**
RANGE_LO	15:0	RW	–	Range detect low threshold
RANGE_HI	31:16	RW	–	Range detect high threshold

33.5.8 Double buffer

For each channel the SAR sequencer has two complete sets of registers to hold the acquired data and derived flags. The first set of registers are the Working Registers (PASSx_SARy_CHz_WORK). The Working Registers are used to store preliminary results, after post processing, from newly completed channel acquisitions.

The second set of registers are the result registers (PASSx_SARy_CHz_RESULT). When a group scan completes, the contents of the Working Registers are copied (committed) to the corresponding result registers and the Group Done interrupt is set. A group scan completes when the acquisition for the last channel of the group successfully completes. Also, the bits corresponding to the channels are set in the PASSx_SARy_RESULTy_VALID Register.

When the results are copied to the result registers, the Working Registers are immediately available for the SAR sequencer to start a new group scan (for example, in a continuous trigger). In parallel to the new group scan, the software can process the results of the just completed group scan. This double buffering maximizes the time that the software has to pick up the results.

Note that software should never use information from the PASSx_SARy_CHz_WORK Registers, as that information is not coherent (see Group coherency). The PASSx_SARy_CHz_WORK Registers are only visible to software to provide the status of a group scan in progress, which may be helpful for debug. The corresponding channel bits are set in the PASSx_SARy_WORK_VALID Register as soon as the conversion of a channel is completed and result is stored in the PASSx_SARy_CHz_WORK Register.

Note that for both the result and working registers the lower 16 bits contain the data (or pulse-detect counters) and in the upper 16 bits the flags/interrupts are mirrored. Double buffering also ensures that preliminary results from a canceled or restarted group scan are discarded; that is, they are never copied in the result registers and thus are not made available to the software.

Table 282 Post Processing Control Register

SARx_CHy_POST_CTL				
Field	**Bits**	**Access**	**Default**	**Description**
POST_PROC	2:0	RW	–	Post processing 000 - No processing 001 - Averaging 010 - Averaging followed by range detect 011 - Range detect 100 - Range detect followed by pulse detect 101 - Reserved 110 - Reserved 111 - Reserved

(table continues...)

Table 282 (continued) Post Processing Control Register

SARx_CHy_POST_CTL

Field	Bits	Access	Default	Description
LEFT_ALIGN	6	RW	–	Alignment 0 - Data is right aligned RESULT[11:0], 1 - Data is left aligned RESULT[15:4]
SIGN_EXT	7	RW	–	Sign extension of result 0 - Result is unsigned (0- extended if needed) 1 - Result is signed
AVG_CNT	15:8	RW	–	Either average count for channel or pulse positive reload value (if pulse detection is enabled). Averaging count for channels that have averaging enabled. A channel will be sampled (AVG_CNT+1) = [1...256] time
SHIFT_R	20:16	RW	–	Either shift right (no pulse detection) or pulse negative reload value (if pulse detection is enabled)
RANGE_MODE	23:22	RW	–	Range detection mode 00 - BELOW_LO (RESULT < RANGE_LO) 01 - INSIDE_RANGE (RANGE_HI > RESULT> RANGE_LO) 10 - ABOVE_HI (RESULT>RANGE_HI_ 11 - OUTSIDE_RANGE (RANGE_HI<RESULT or RESULT<RANGE_LO)

33.5.9 Group coherency

For software, it is important that all the results of a group scan are coherent. Coherent results means that all information for all the channels in one group are guaranteed to have been acquired during the same group scan. The SAR sequencer achieves this coherency by making sure that the copy from PASSx_SARy_CHz_WORK to PASSx_SARy_CHz_RESULT Registers, of all the channels in the group, happens on a single clock edge. The information for a group scan includes the following:

- RESULT data
- Data valid flags
- All interrupt flags
- Range detect ABOVE_HI flags
- Pulse detect counters
- Channel done triggers

Note that the range violation trigger is not coherent (however, the range detect interrupt is coherent). The range violation trigger is required to have a low latency in the group trigger; therefore, this trigger is set immediately after the range violation is detected.

33.5.10 Status

The SAR sequencer has several Status Registers to allow software to observe what it is doing. Most of these registers are only intended for debug purposes. Some status registers can also be used for polling.

The following status registers are available:

- A generic status register (PASSx_SARy_STATUS) that shows:
  - If the ADC is busy or not (BUSY)
  - If not busy why not (PWRUP_BUSY, DBG_PAUSE)
  - Or if busy shows with which channel (CUR_CHAN) and the priority (CUR_PRIO) and the preemption (CUR_PREEMPT_TYPE) attributes for that channel
- An averaging status register (PASSx_SARy_AVG_STAT) that shows:
  - Current averaging counter
  - Current value of the accumulator – sum of averaging samples acquired so far
- A register to show which input triggers are currently pending (PASSx_SARy_TR_PEND)
- A group status register (PASSx_SARy_CHz_GRP_STAT) that only gathers copies of bits from other registers (PASSx_SARy_CHz_INTR, PASSx_SARy_TR_PEND)

33.6 Triggering and scheduling

The industrial SAR sequencer has several specific features required for the industrial market. Most of these unique features are related to how acquisitions are scheduled. For example, this SAR sequencer supports the creation of several signal acquisition groups each with their own trigger and priority, which potentially can preempt each other.

33.6.1 Channel grouping

All the available channels (up to 32) of the SAR ADC can be grouped. A group consists of several consecutive channels of which only the last channel has the 'PASSx_SARy_CHz_TR_CTL.GROUP_END' flag set (see Table 283). The number of channels in a group can be anywhere from one (single channel) to 32. Separate groups can have different number of channels.

The first channel of a group defines which trigger is used for that group.

Note:
- *The first channel, which defines the trigger in the group must be enabled*
- *The last channel, which defines the end in the group must be enabled*
- *A channel in the group may be disabled in which case it will be skipped*
- *A group implicitly ends at the last existing and enabled channel*

Table 283 Channel group end flag field

SARx_CHy_TR_CTL

Field	Bit	Access	Default	Description
GROUP_END	11	RW	1	0 - Continue group with next channel 1 - Last channel of a group

33.6.2 Triggers

A trigger for a group will cause the acquisitions, as defined by the configurations of the channels in the group, to be executed. There is one dedicated (one-to-one) trigger input for each channel of each of the ADCs connected to the trigger outputs from corresponding TCPWM.

In addition to the TCPWM triggers there are (3 × Number of SAR instances) generic trigger inputs that are shared between ADC channels. Any five of these generic triggers can be routed to any ADC channel. PASS_SAR_TR_IN_SEL_x Register can forward any five of all the available generic triggers (up to 16).

Note that the synchronization of the generic trigger inputs happens at a level before the trigger routing selection. This means that the generic triggers arrive synchronously at each ADC, which is an essential feature that enables the synchronized ADC triggering (and thus lock-step execution) that is required for the motor control.

The trigger for a channel group is selected by the configuration (PASSx_SARy_CHz_TR_CTL.SEL, see Table 284) of the first channel of the group. There are seven possible hardware triggers and a software trigger. The hardware trigger options are:

- TCPWM – one-to-one trigger output from a corresponding TCPWM
- GENERIC0-4 – five generic input triggers routed to this ADC
- CONTINUOUS – this trigger is always high, making the group always triggered or in other words Idle trigger
- OFF – no hardware trigger

A group can be software-triggered by setting the PASSx_SARy_CHz_TR_CMD.START bit. This software trigger can be used even if the group is configured to use a hardware trigger.

Note:
- *Setting the pending bit has priority over clearing, so if the hardware trigger input is still high when a trigger clear is received then the pending bit will remain pending.*
- *If a new trigger is received while the pending bit is already set, then effectively the new trigger is ignored.*
- *Only the first channel of a group should ever be triggered.*

The input trigger signal naming convention is given below.TCPWM: PASS[x]: PASSx_CH_TR_IN[y]where x is instance and y is the channely varies between 0-19 for TCPWM0 group 0 andy varies between 20-31 for TCPWM0 group 1GENERIC: PASS:PASS_GEN_TR_IN[y] where y varies between 0 to 3.Freeze Pass0 during Debug:PASS:PASS_DEBUG_FREEZE_TR_IN.

Table 284 Trigger Control Register

SARx_CHy_TR_CTL

Field	Bits	Access	Default	Description
SEL	2:0	RW	0	Analog-to-digital conversion trigger select for the channel 000 - OFF 001 - TCPWM 010 - GENERIC TRIGGER 0 011 - GENERIC TRIGGER 1 100 - GENERIC TRIGGER 2 101 - GENERIC TRIGGER 3 110 - GENERIC TRIGGER 4 111 - CONTINOUS

33.6.3 Arbitration, preemption, and acquisition scheduling

When a trigger occurs (pending bit high), then the corresponding group of acquisitions (group scan) needs to be executed. When triggers for multiple groups happen, then arbitration is needed to determine which of the pending group scans will be executed first.

The arbitration of the pending triggers is based on both an explicit and an implicit priority. The explicit priority is set, as a trigger attribute (PASSx_SARy_CHz_TR_CTL.PRIO, see Table 285), by software. There are eight explicit priority levels and priority level 0 is the highest.

The implicit priority is defined by the channel ordering as follows: a pending trigger for a lower channel has a higher priority than a higher channel with the same explicit priority.

When a group scan is ongoing and a new higher priority trigger arrives, then it can cause the preemption of the ongoing lower priority group scan. Whether preemption happens is determined by the scheduler based on the explicit priority level and the trigger preemption type of the ongoing group scan. The trigger preemption type is another trigger attribute (PASSx_SARy_CHz_TR_CTL.PREEMPT_TYPE) set by software.

The trigger preemption type determines both when preemption is allowed and what happens with the preempted group scan on return – after the preempting group scan is done.

The following four preemption types are available:

- ABORT_RESUME
  - Immediately abort the ongoing acquisition and on return resume the group scan starting with the aborted channel.
  - Keep the pending trigger of the aborted group.
- ABORT_RESTART
  - Immediately abort the ongoing acquisition and on return restart the group scan from the first channel of the group.
  - Keep the pending trigger of the aborted group.
- ABORT_CANCEL
  - Immediately abort the ongoing acquisition and do not return.
  - Clear pending trigger of the aborted group and set the canceled interrupt for the last channel of the aborted group.
- FINISH_RESUME
  - Before preempting, complete the ongoing acquisition (including averaging) and on return resume the group scan starting with the next channel.
  - Keep the pending trigger of the aborted group.

Figure 351 and Figure 352 show the behavior of these preemption types when a high priority group trigger arrives.

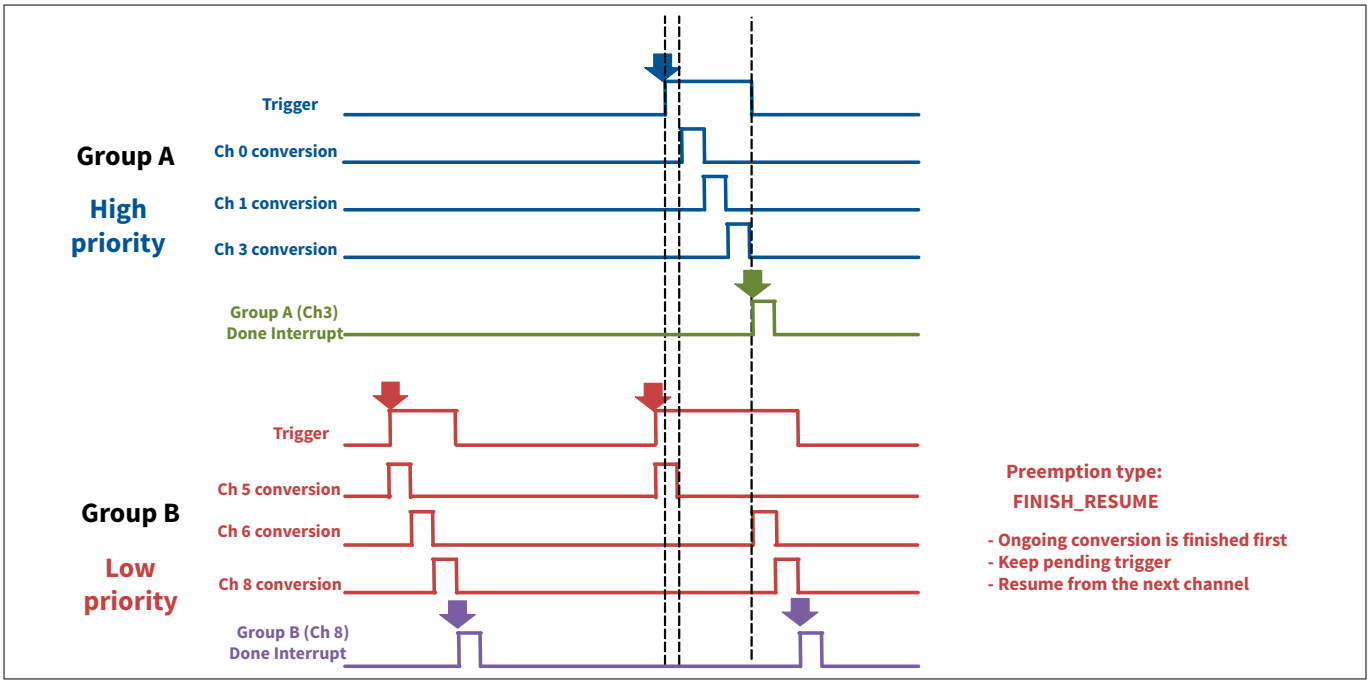


Figure 351 Preemption types: A low priority group B behavior with FINISH_RESUME preemption type

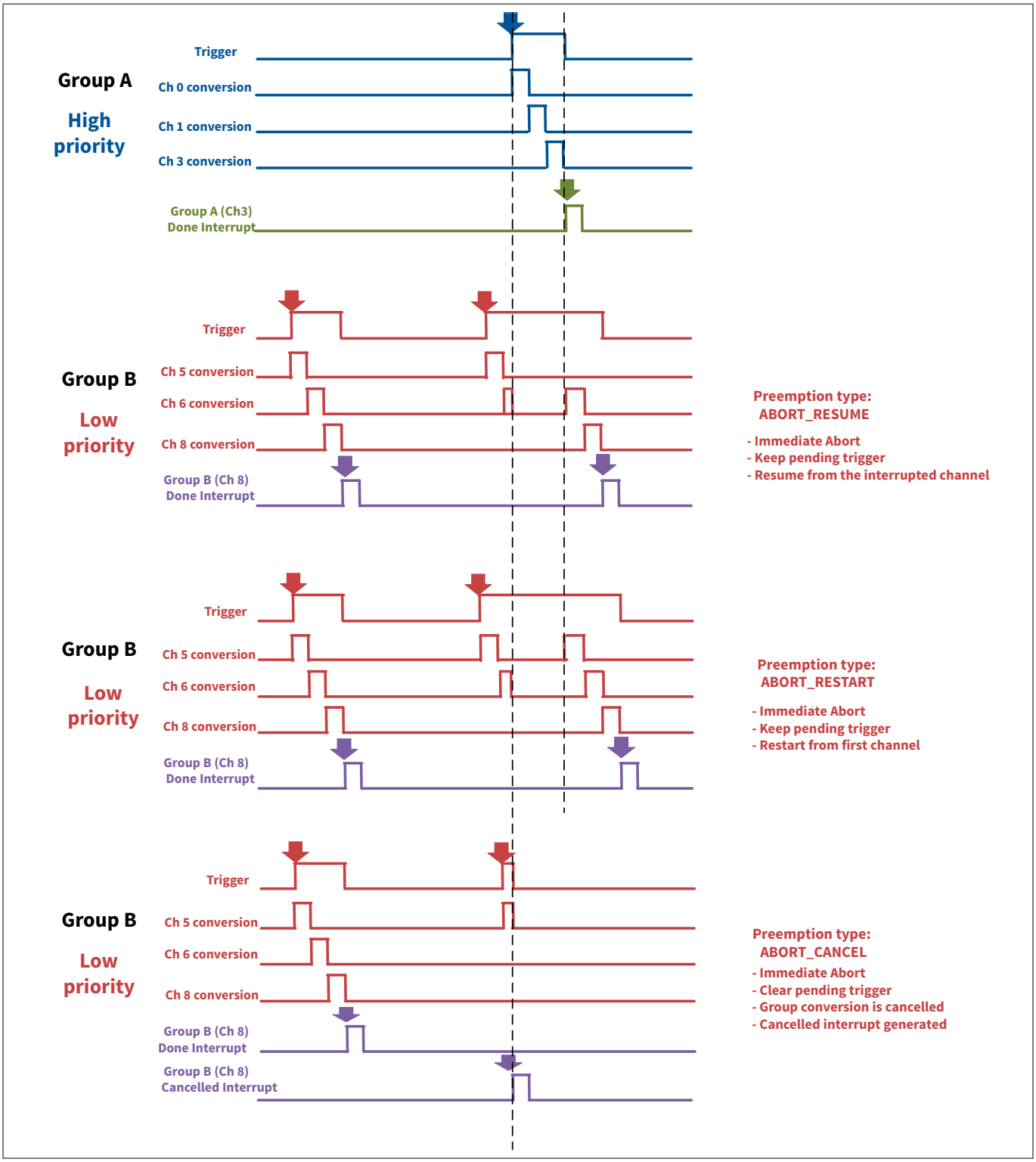


Figure 352 Preemption types: A low priority group B behavior with different preemption types

Table 285 Trigger control register

SARx_CHy_TR_CTL				
Field	**Bits**	**Access**	**Default**	**Description**
PRIO	6:4	RW	0	Channel priority 0 Highest priority 1 ….. 6 7 Lowest priority
PREEMPT_TYPE	9:8	RW	0	Preemption type of the group 00 - ABORT_CANCEL 01 - ABORT_RESTART 10 - ABORT_RESUME 11 - FINISH_RESUME

33.6.4 Debug freeze

When enabled, the assertion of the debug freeze trigger prevents the scheduler from starting acquisitions for a new channel. Note that averaging, if started, will complete even if the debug pause trigger is asserted. The SAR ADC system has only one debug freeze trigger. However, there is a separate debug freeze enable for each ADC (PASS_PASS_CTL.DBG_FREEZE_EN).

33.6.5 Auto idle power down

The SAR sequencer can optionally be configured (mask bit in the PASSx_SARy_CHz_INTR_MASK Register) to automatically power down the analog when the ADC is idle. When this feature is used, the analog will also automatically power up when a trigger arrives. However, after power-up the analog circuit must settle for some time before it can make accurate acquisitions. The required power-up time needs to be configured by software (PASSx_SARy_CTL.PWRUP_TIME).

33.6.6 Channel disable/software abort

When a group is activated, it is no longer allowed to change the configuration settings of the channels in the group. It is undefined what will happen when this rule is violated. When a group or channel needs to be reconfigured it should be disabled first by clearing PASSx_SARy_CHz_ENABLE.ENABLE. All channels in a group need to be disabled together. The channels in a group should be disabled in order, from first to last. If these rules are violated some undefined output may be produced, but no lock up will occur.

Disabling a channel has the following consequences:

- Immediately abort the acquisition if it happens to be in progress for that channel
- Clear the pending trigger for the channel (if present)
- Discard preliminary results ('work' flags and data).

33.7 Output triggers and interrupts

For each channel, there are two trigger outputs, three channel interrupts, and three group interrupts. In addition, there are two generic output triggers per ADC. Only enabled channels can generate new triggers or interrupts. Interrupts are implemented compliant to the platform rules, which means:

- Each of the interrupts has a corresponding mask bit in the PASSx_SARy_CHz_INTR_MASK Register to individually enable or disable that interrupt source.
- Software needs to clear the interrupt by writing a '1' to the corresponding bit in the PASSx_SARy_CHz_INTR Register.

All enabled interrupts are consolidated into one interrupt output signal per channel. Note that disabling a channel does not clear already pending triggers or interrupts.

33.7.1 Trigger outputs

Two trigger outputs can be generated per enabled channel: Channel Done and Range Violation triggers.

The trigger output signal naming convention is given below:

x: instance, y: channel

Range violation: PASS[x]:PASSx_CH_RANGEVIO_TR_OUT[y]

Channel done: PASS[x]:PASSx_CH_DONE_TR_OUT[y]

Generic: PASS[x]:PASSx_GEN_TR_OUT[y]

33.7.1.1 Channel done trigger

This trigger indicates that the data for a channel is available in the result register. This means it is never asserted for a pulse-detect channel. This trigger is intended to be used to trigger a DataWire (DW) channel to pick up the result and copy it to system RAM. There is a one-to-one trigger connection from each ADC channel to a corresponding DW channel. The done trigger can be configured (PASSx_SARy_CHz_TR_CTL.DONE_LEVEL) as a level trigger or a pulse trigger. When triggering the DW, a level trigger is recommended. In this mode, the level trigger will remain asserted until the corresponding data is read; that is, the level trigger is de-asserted as a side effect of reading the data. Level trigger mode also enables channel overflow interrupt detection as described here.

Note that all the channel done triggers of the group only assert when the whole group scan is complete and not immediately after the channel acquisition is complete.

In addition, the done trigger of the last channel of a group can be used as a ‘group violation’ trigger (PASSx_SARy_CHz_POST_CTL.TR_DONE_GRP_VIO). In this mode, the done trigger is only set if at least one of the channels in the group detect a range violation. If none of the channels in the group have range detection enabled, then this trigger is never set.

33.7.1.2 Range violation trigger

This trigger generates a pulse in case the acquisition result for the channel causes a range detect event (see Range detect). This trigger is a one-to-one trigger connection from each ADC channel to a corresponding TCPWM channel. This trigger is typically used to 'kill' the TCPWM whenever the ADC acquisition results in a value that is outside the predefined allowable range. Note that range detection will not generate a trigger if pulse detection is also enabled for the channel. Range violation trigger asserts immediately after the channel acquisition is complete; unlike all the other ADC outputs it does not wait until the whole group scan is complete.

In addition to these two triggers, there are two generic triggers routed out to the generic trigger infrastructure. This enables the ADC to trigger another IP, other than DW, on completion of a group conversion.

One common use case for one-to-one trigger connection from ADC channel to a TCPWM channel is LED diagnostics. In this use case the LED is driven with a pulse generated by a TCPWM and the SAR is used to sample a diagnostic feedback signal from the LED driver to ensure that the LED is operating correctly. If the SAR result is outside a predefined range it needs to immediately stop the TCPWM from driving the LED.

33.7.1.3 Generic trigger output

For each SAR ADC, two channel triggers (done trigger or range violation trigger) can be routed to the generic trigger infrastructure. PASS_SAR_TR_OUT_SEL_x.OUT0_SEL and PASS_SAR_TR_OUT_SEL_x.OUT1_SEL Registers can be used to select channel triggers to forward. This enables the use of these triggers outside the dedicated 1-to-1 trigger connections to DW or TCPWM. For this use case, it may be better to configure the channel done trigger as a pulse (two cycles on CLK_SYS), to avoid the need for a data read to de-assert the trigger.

33.7.2 Channel interrupts

Each channel has a dedicated interrupt and associated Interrupt Registers – PASSx_SARy_CHz_INTR, PASSx_SARy_CHz_INTR_SET, PASSx_SARy_CHz_INTR_MASK, and PASSx_SARy_CHz_INTR_SET. The PASSx_SARy_CHz_INTR_MASK register is used to mask the interrupt source register, PASSx_SARy_CHz_INTR; only the masked interrupt flags are forwarded to the CPU.

Three types of channel interrupts can be generated for each enabled channel.

33.7.2.1 Range detect interrupt

This interrupt is set if the acquisition result for the channel causes a range detect event (see Range detect). Note that this interrupt is never set if pulse detection is also enabled for the channel – range and pulse detection interrupts are mutually exclusive.

33.7.2.2 Pulse detect interrupt

This interrupt is set if the acquisition result for the channel causes a pulse detection.

33.7.2.3 Channel overflow interrupt

The channel overflow interrupt is only set if a new group scan completes while the results from a previous completion have not yet been handled. There are three error situations for the channel that the hardware detects. The overflow interrupt is set when on completion of a group scan one of the following conditions is true:

- the range detect interrupt is enabled and still pending
- the pulse detect interrupt is enabled and still pending
- the channel done trigger is set to LEVEL and still asserted

For the first two cases software should have handled and cleared the interrupts before a new one is set.

Similarly, the channel done trigger should have been cleared by the reading the result. If this is not the case, because DataWire is too slow, then the previous data will be overwritten and thus is lost.

33.7.3 Group interrupts

These are interrupts that can only be set for the last channel of a group (which must be an enabled channel). There are three group interrupts: group done, group canceled, and group overflow interrupts.

33.7.3.1 Group done interrupt

This interrupt is set every time a group scan completes.

33.7.3.2 Group canceled interrupt

This interrupt can only be set for an enabled group with the ABORT_CANCEL preemption type. As explained in Arbitration, preemption, and acquisition scheduling, this interrupt is set when the group scan is aborted due to preemption or if a new trigger arrives but it does not immediately result in starting the corresponding group scan.

33.7.3.3 Group overflow interrupt

This interrupt is set when a new group scan completes and the group done interrupt is enabled, and still pending from a previous completion. This is an error situation that occurs when software is too slow to pick up the previous results and clear the group done interrupt.

33.8 Calibration

33.8.1 Analog calibration

Analog calibration is used to make the actual ADC transfer curve come closer to the Ideal transfer curve. Analog calibration can correct an offset and a gain error – linear errors as shown by the 'Actual curve' as shown in Figure 353.

Analog calibration is controlled by the configuration register PASSx_SARy_ANA_CAL, which contains an 8-bit analog offset (PASSx_SARy_ANA_CAL.AOFFSET) and 5-bit analog gain (PASSx_SARy_ANA_CAL.AGAIN) calibration value. Before the ADC is used for acquisitions, these values need to be set to the correct values; that is, the ADC needs to be calibrated. The following steps are recommended to find the optimal analog calibration values[7].

As shown in Figure 353, the ideal transfer curve has the following two characteristics:

- Transition between values 0x000 and 0x001 for VREFL + 0.5LSB input voltage
- Transition between values 0xFFE and 0xFFF for VREFH – 1.5LSB input voltage

If this is not the case, then the ADC needs to be calibrated, which can be done by performing the following steps:

- Set the analog gain correction value (PASSx_SARy_ANA_CAL.AGAIN) to ‘0’
- Configure a channel to convert VREFL
- Do several software-triggered acquisitions using different PASSx_SARy_ANA_CAL.AOFFSET values
- Do this until the AOFFSET value X is found for which the converted value transitions from 0x000 to 0x001
- Change the channel configuration to convert VREFH
- Do several software triggered acquisitions using different AOFFSET values
- Do this until the AOFFSET value Y is found for which the converted value transitions from 0x001 to 0x000
  - Use averaging when the search approaches the desired target

[7] Based on the XMC7000 manual.

- Set AOFFSET to (X+Y)/2 + 2
- Change the channel configuration back to converting VREFL
- Do several software triggered acquisitions using different AGAIN values
- Do this until the AGAIN value Z is found for which the converted value transitions from 0x001 to 0x000 (using averaging for the final acquisitions)
- Set AGAIN to Z+1

Figure 354 shows the flow chart to calibrate the ADC.

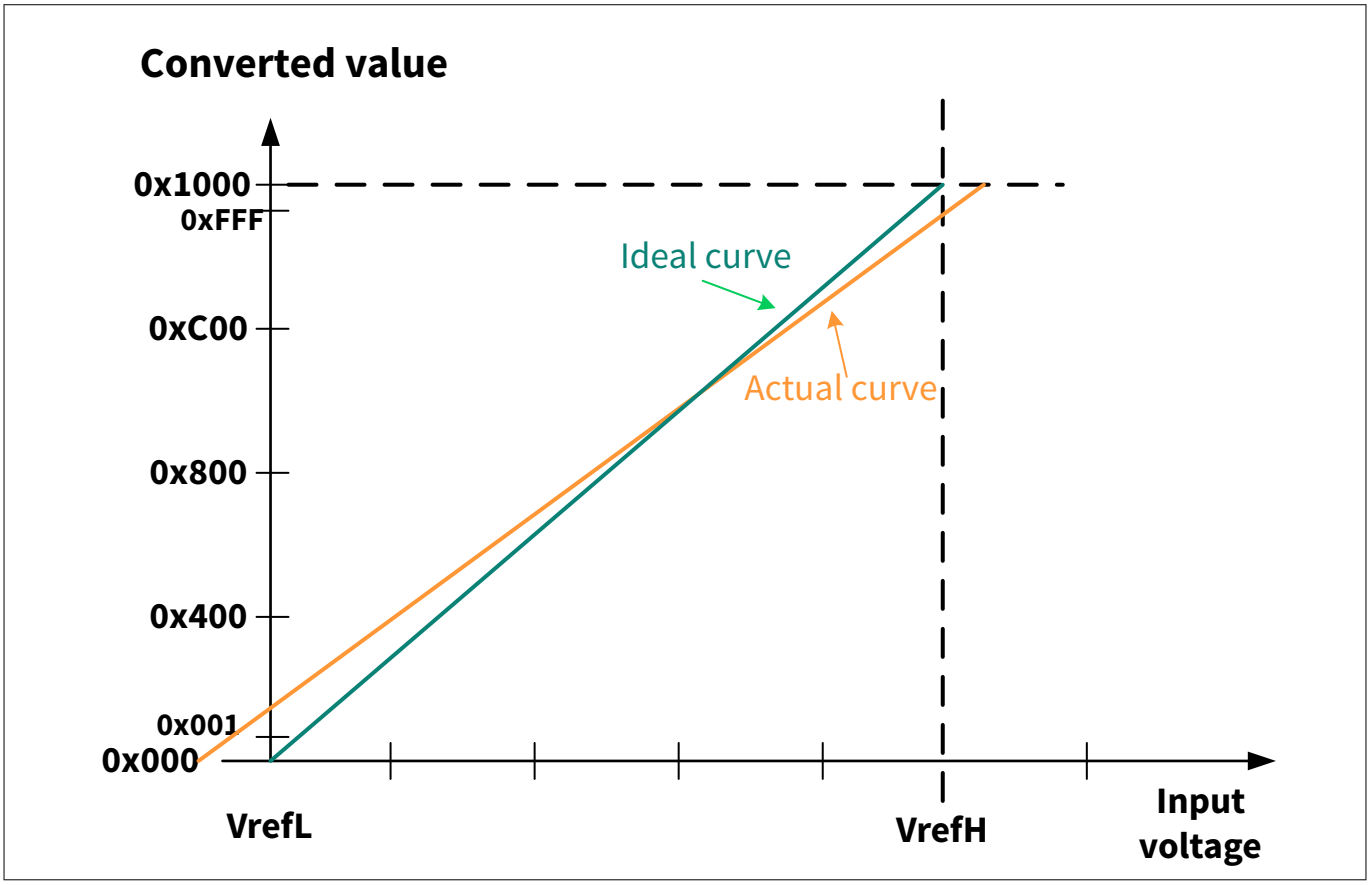


Figure 353 Analog to digital transfer curve

33.8.2 Alternate calibration

A potential problem with calibration is that over time the ADC error drifts; for example, due to temperature changes. To counter that problem, periodically re-calibrate the ADC.

Running the calibration algorithm requires experimenting with the global calibration configurations. As a result, all other acquisitions should be paused until the calibration algorithm is finished. Considering the number of acquisitions needed for calibration such a pause is unacceptably long.

To solve that problem, a second set of alternate calibration values are added (PASSx_SARy_ANA_CAL_ALT). Each channel can choose to use the alternate calibration values by setting the PASSx_SARy_CHz_SAMPLE_CTL.ALT_CAL bit. With this in place the periodic recalibration algorithm can quietly run in the background while the main application can keep on running undisturbed using the active calibration values.

33.8.3 Coherent calibration update

After the new calibration values are established, the next step is to coherently deploy the new values. Changing the calibration values while an acquisition is in progress will result in undefined results for that acquisition and is therefore not allowed.

Similarly, changing the calibration in the middle of a group scan will lead to incoherent results within that group scan. Due to preemption, it is troublesome to determine if some group scan is still waiting to resume and complete.

The PASSx_SARy_CAL_UPD_CMD.UPDATE bit solves this problem. When this bit is set, the sequencer will wait for the ‘right moment’ to coherently copy values from the alternate calibration registers to the regular calibration registers. At the same time, the PASSx_SARy_CAL_UPD_CMD.UPDATE bit will also be cleared. The right moment for a coherent calibration update is when the ADC becomes idle, or a 'continuous' triggered group completes. This ensures that all acquisitions within a group scan (even if preempted) are done with the same calibration values.

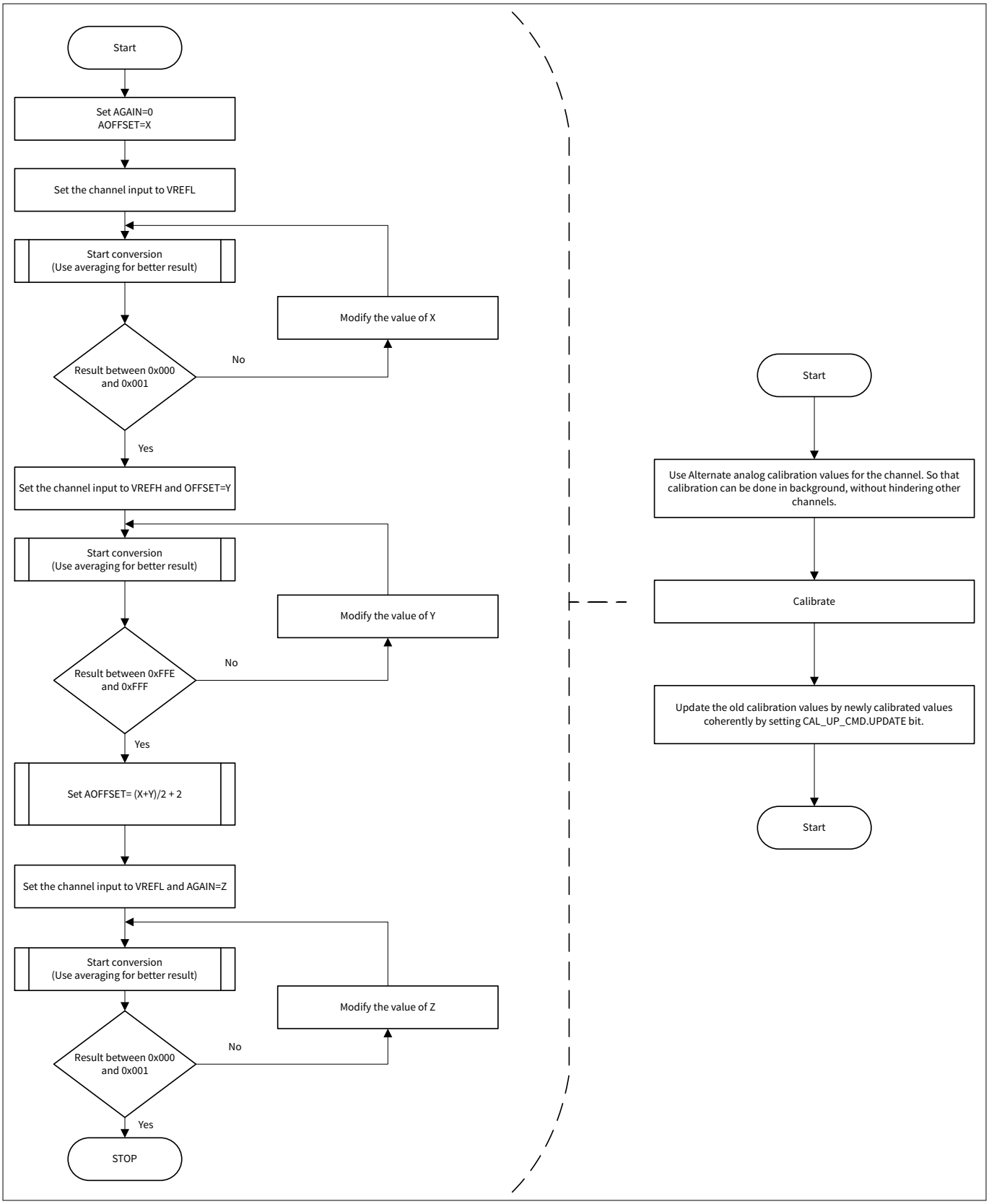


Figure 354 Calibration flow diagram

33.9 Temperature measurement

XMC7000 devices are equipped with a built-in temperature sensor to measure the chip temperature. To accurately measure the temperature at runtime, use the reference measurement done during production. This reference data is stored in SFlash along with other calibration data. Refer to the device datasheet for the exact address to read this data.

Two types of reference data are stored in SFlash – SORT and CLASS. SORT data is more accurate because it is measured and taken from the actual die. CLASS data is less accurate (distorted) because it is taken when the silicon is in the package. The temperature measurement is done by the user in the CLASS manner; therefore, CLASS data needs to be fitted using the least square approximation (LSA) algorithm. The CLASS fitting should also be adjusted based on the SORT curvature to compensate the package effect.

33.9.1 Example measurement flow

The second order polynomial equation is described as follows.

$$y(x) = a1 + a1x + a2x^2 \tag{12}$$

The second order polynomial fitting needs at least three data values to produce three polynomial coefficients. Because CLASS data consists of only two values (25°C and 125°C), one coefficient is taken from the second order polynomial of SORT data based on the assumption that the parabolic curvature of the SORT and CLASS data are the same. It means the quadratic coefficient (a2) for SORT and CLASS are the same. The location of the parabolic vertex determined by a1 and parabolic offset by a0 may be different due to the package effect. Then, the two unknown variables a1 and a0 can be found using the two known CLASS data. The V_{BG} data is used to remove the ADC output dependency on the ADC reference voltage. Thus, the user does not have to consider the ADC voltage reference when the algorithm is executed. Figure 355 shows the conceptual relation between SORT and CLASS data.

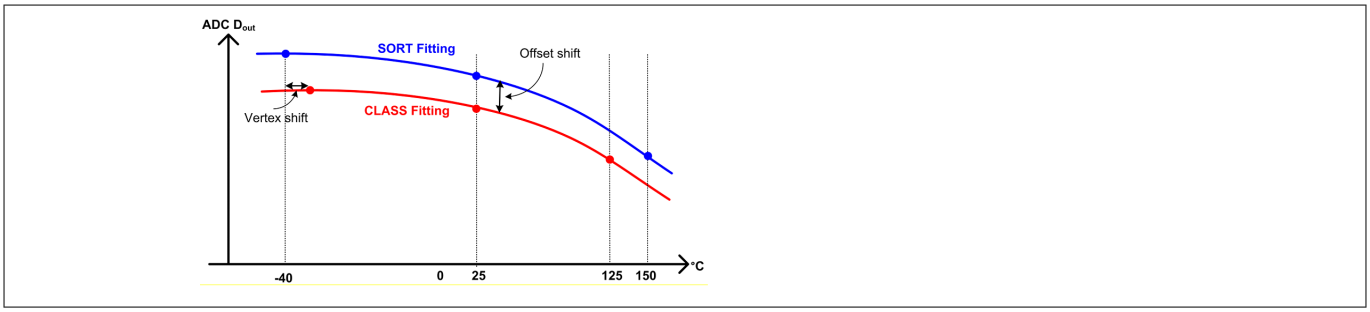


Figure 355 SORT and CLASS relationship

33.9.2 Temperature sensor calibration and SFlash address

To measure the accurate temperature, a calibration procedure needs to be done. The calibration data are stored in the SFlash. During the temperature sensor calibration, the following measurements need to be taken:

- Measure the die temperature (T_{CHIP}) using external currents and external ADC
- Measure on-chip diode voltage (V_{BE}) with EPASS ADC
- Measure on-chip bandgap reference (V_{BG}) using EPASS ADC

The calibration is applied for

- Two sets of supplies (3.3 V and 5 V)
- Three different temperatures (CLASS HOT, SORT2, and SORT3)

Table 286 SFLASH DATA SET#0: for die temperature

SFlash address	SFlash name	Parameter	Ambient temperature (C)	Description
1700_0654, 1700_0655	EPASS_TEMP_TRIM_TEMP_COLDSORT	T_{CHIP_S2}	~-40	On-chip temperature measured using external currents and external ADC
1700_064E, 1700_064F	EPASS_TEMP_TRIM_TEMP_ROOMSORT	T_{CHIP_S3}	~25	
1700_065A, 1700_065B	EPASS_TEMP_TRIM_TEMP_HOTCLASS	T_{CHIP_CHI}	~130	

Table 287 SFLASH DATA SET#1: V_{BE} and V_{BG} at VDDA = 3.3 V

SFlash address	SFlash name	Parameter	Description
1700_0656, 1700_0657	EPASS_TEMP_TRIM_DIODE_COLDSORT	V_{BE_S2}	Temperature sensor diode voltage at COLD using EPASS ADC
1700_0658, 1700_0659	EPASS_TEMP_TRIM_VBG_COLDSORT	V_{BG_S2}	Bandgap voltage at COLD using EPASS ADC
1700_0650, 1700_0651	EPASS_TEMP_TRIM_DIODE_ROOMSORT	V_{BE_S3}	Temperature sensor diode voltage at ROOM using EPASS ADC
1700_0652, 1700_0653	EPASS_TEMP_TRIM_VBG_ROOMSORT	V_{BG_S3}	Bandgap voltage at ROOM using EPASS ADC
1700_065C, 1700_065D	EPASS_TEMP_TRIM_DIODE_HOTCLASS	V_{BE_CHI}	Temperature sensor diode voltage at HOT using EPASS ADC
1700_065E, 1700_065F	EPASS_TEMP_TRIM_VBG_HOTCLASS	V_{BG_CHI}	Bandgap voltage at HOT using EPASS ADC

Table 288 SFLASH DATA SET#2: V_{BE} and V_{BG} at VDDA = 5 V

SFlash address	SFlash name	Parameter	Description
1700_066E, 1700_066F	EPASS_TEMP_TRIM_DIODE_COLDSORT_5V	$V_{BE_S2_5V}$	Temperature sensor diode voltage at COLD using EPASS ADC
1700_0670, 1700_0671	EPASS_TEMP_TRIM_VBG_COLDSORT_5V	$V_{BG_S2_5V}$	Bandgap voltage at COLD using EPASS ADC
1700_066A, 1700_066B	EPASS_TEMP_TRIM_DIODE_ROOMSORT_5V	$V_{BE_S3_5V}$	Temperature sensor diode voltage at ROOM using EPASS ADC
1700_066C, 1700_066D	EPASS_TEMP_TRIM_VBG_ROOMSORT_5V	$V_{BG_S3_5V}$	Bandgap voltage at ROOM using EPASS ADC
1700_0672, 1700_0673	EPASS_TEMP_TRIM_DIODE_HOTCLASS_5V	$V_{BE_CHI_5V}$	Temperature sensor diode voltage at HOT using EPASS ADC
1700_0674, 1700_0675	EPASS_TEMP_TRIM_VBG_HOTCLASS_5V	$V_{BG_CHI_5V}$	Bandgap voltage at HOT using EPASS ADC

33.9.3 Temperature calculation

The die temperature can be calculated using the on-chip temperature sensor and EPASS ADC, along with the calibration data stored in the SFlash.

SFlash data entries are classified by the values of analog supply VDDA and I/O supply to ADC VDDIO. The values of the supply are 3.3 V and 5 V. Based on the application, the respective data needs to be fetched from the SFlash. The following three cases should be considered.

Table 289 Use case table

Case	Condition	SFlash address	When to use this data
1	VDDA = VDDIO = 3.3 V ± 10%	SET#0 and SET#1	3 V ≤ VDDA ≤ 3.6 V and 3 V ≤ VDDIO ≤ 3.6 V
2	VDDA = VDDIO = 5 V ± 10%	SET#0 and SET#2	4.5 V ≤ VDDA ≤ 5.5 V and 4.5 V ≤ VDDIO ≤ 5.5 V
3	VDDA = VDDIO = 2.7 V ~ 5.5 V	SET#0 and SET#1	VDDA ≠ VDDIO

33.9.3.1 Procedure to calculate the temperature

The steps to measure die temperature (T_{CHIP}) using calibration data stored in SFlash are as follows:

1. V_{BE} (temperature sensor output) has a second order dependency on temperature and can be described by this equation.

   $$V_{BE} = aT^2 + bT + c$$

   To calculate temperature from V_{BE}, we need to know the coefficients (a, b, and c) of the above equation. These coefficients can be calculated using the data (V_{BE} measured at three different temperatures) stored in SFlash. Refer to Table 289, Table 287, and Table 288 for details. The three combinations of (V_{BE}, T) to be used depends on the supply voltage (VDDA). For example, to calculate polynomial coefficients:

   - When VDDA = 3.3 V, use data in Table 286 and Table 287
   - When VDDA = 5 V, use data in Table 286 and Table 288
2. After determining the coefficients, the next step is to trim the EPASS ADC for OFFSET/GAIN
3. When EPASS ADC is trimmed for OFFSET/ GAIN, measure TS output (V_{BE}) and bandgap reference (V_{BG}) using this ADC.
4. Since the ADC reference voltage may have changed from calibration, V_{BE} must be scaled using the ratio of V_{BG_S3} and V_{BG}, where $V_{BE_new} = V_{BE} \times (V_{BG_S3}/V_{BG})$ before it is used to calculate the temperature.
5. Calculate temperature using the above polynomial

   $$V_{BE_new} = aT^2 + bT + c$$
6. After calculating temperature, we can improve accuracy by using the bandgap reference from the nearest temperature in step 4. Essentially, if the temperature calculated in step 5 is close to
   a. COLD (–40), repeat step 4 with V_{BG_S2} and recalculate temperature using step 5.
   b. HOT (150), repeat step 4 with V_{BG_CHI} and recalculate temp using step 5.
   c. ROOM (27), temperature calculated in step 5 is the final temperature.

***Note**: See the device datasheet for temperature sensor sampling time.*

***Note**: The following procedure is applicable for the XMC7000 i.e., devices in order to gain the accuracy of the temperature sensor.*

After a reset or deep sleep wakeup, set bit 9, 8, and 6 of PASS_TEST_CTL Register (Address: 0x409F0080) to 1 while keeping the other bits unchanged.

33.10 Diagnostic reference generator

This block provides voltages and currents for chip-level use to test internal and external signal connections. Manipulating happens by preconditioning or by the overlapping feature. Injecting voltages near pads allows testing the continuity of internal routing to the ADC. Provisions also exist to connect reference voltages and currents to pads whether independently or simultaneously with ADC inputs.

It generates selectable output voltages. It also includes the option to add 10-µA sink or source currents. The voltages can be derived from the supply (VDDA) or can be a buffered version of the bandgap (VBG). The 10-µA sink/source currents used by the diagnostic reference are generated by the reference buffer described in the next section.

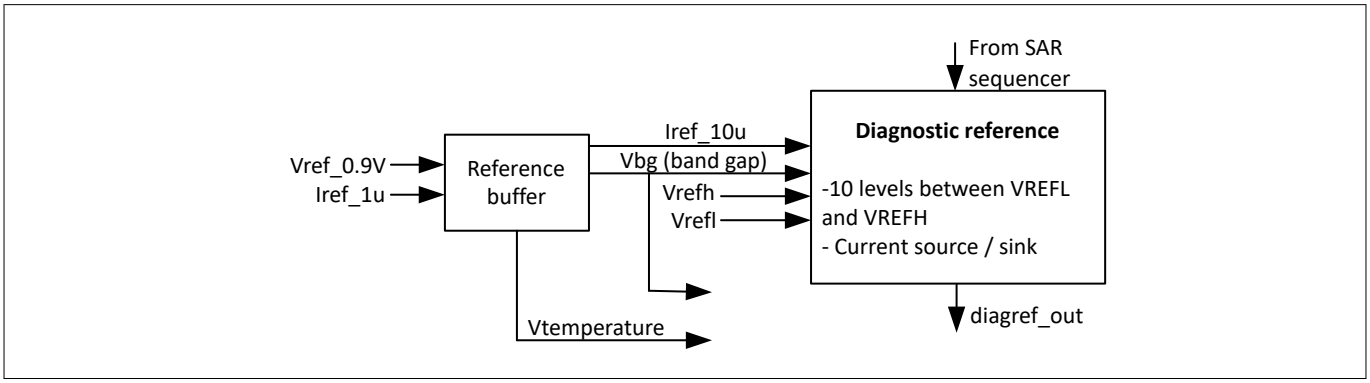


Figure 356 Diagnostic reference block diagram

33.10.1 Diagnostic reference configuration

Each ADC has one diagnostic reference block (see Figure 356), which consists of:

- An RDAC providing 10 voltage levels from VREFL to VREFH
- Four other voltage references
- A current source and sink function
- An analog mux to select one of these signals

The diagnostic reference block is configured through the DIAG_CTL Register. This is a global (per ADC) configuration.

When the diagnostic reference block is not used by any channel it should be disabled, to save power, by clearing the DIAG_EN bit.

33.11 Reference buffer

The reference buffer contains control logic, a voltage follower amplifier (the same amplifier circuit used in the SRSS to buffer Vbg), a current multiplier and array of current sources and sinks, and a temperature sensor. Additionally, it also provides power monitoring block. In principle it provides four functions:

- Power supply monitoring
- Buffering the 0.9-V bandgap signal from the SRSS.SET_PASS_PASS_CTL.REFBUF_MODE = 1 when bandgap signal is used.

- Scaling 1 µA (4x 250-nA currents in parallel) from the SRSS to 10 µA and replicating this current (both source and sink) for diagnostic reference generators.
- Providing a temperature dependent voltage for on-die temperature sensing.

The buffered bandgap voltage connects to SAR and diagnostic reference generator multiplexer inputs. The current source/sinks are used by the diagnostic reference generators for broken wire detection.

Only one ADC at a time should perform temperature measurements as sampling of another ADC may disturb the temperature sensor output voltage.

The power supply monitoring portion of the reference buffer provides termination, which forms voltage dividers between power supply voltages routed on the AMUXBUSA/B signals and the ADC. Resistors and switches contained in power and ground pins connect these supplies to the AMUXBUSA/B as controlled by the IOSS (HSIOM_MONITOR_CTL Register), with the termination of the signals controlled by the reference buffer power supply monitor block. The midpoint of the signal (AMUXBUSA/B) is connected to the SARMUX (internal signals) and can be selected for analog-to-digital conversion by a channel.

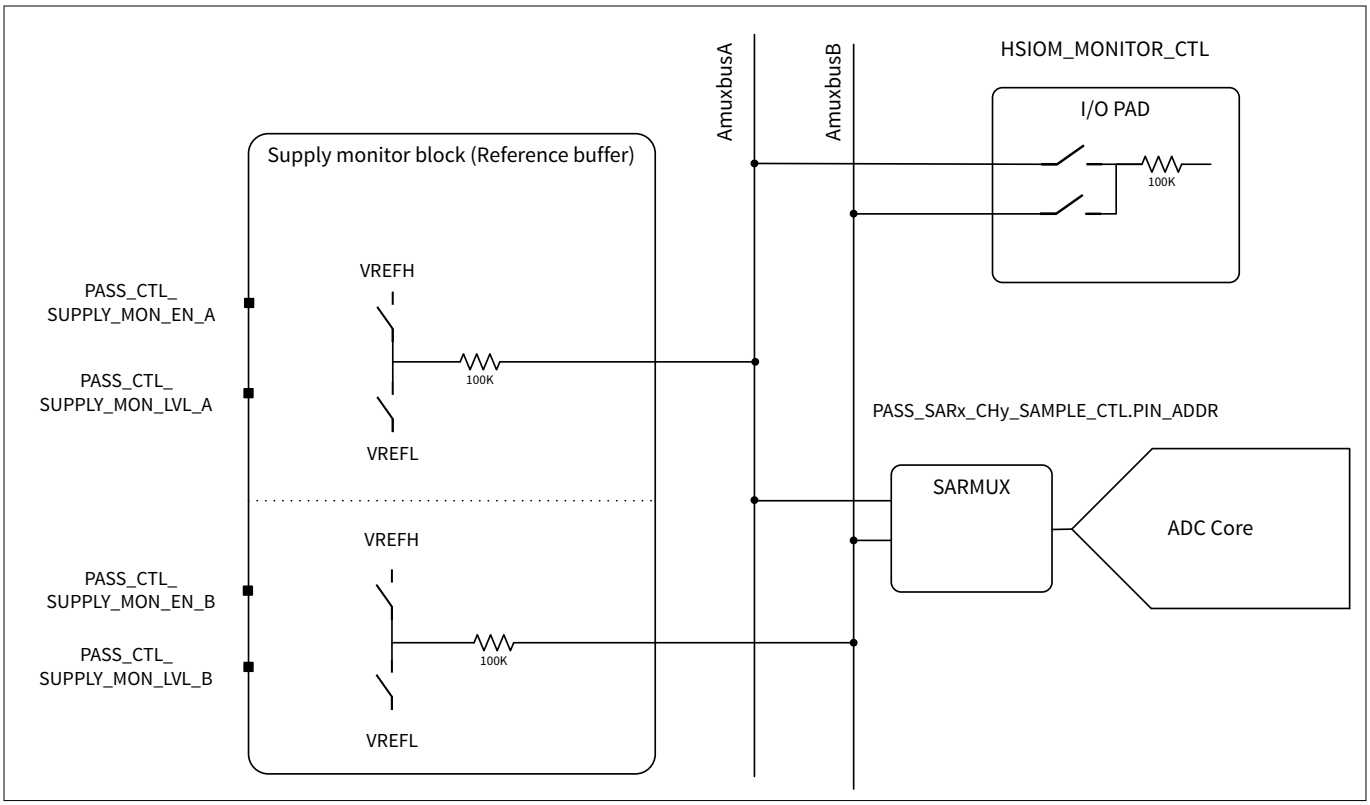


Figure 357 Power monitoring block diagram

Table 290 PASS Control Register (PASS_CTL)

Field	Bits	Access	Default	Description
SUPPLY_MON_EN_A	0	RW	0	Supply monitor enable for AMUXBUS_A
SUPPLY_MON_LVL_A	1	RW	0	Supply monitor level select for AMUXBUS_A 0 - VRL 1 - VRH
SUPPLY_MON_EN_B	4	RW	0	Supply monitor enable for AMUXBUS_B

(table continues...)

Table 290 (continued) PASS Control Register (PASS_CTL)

Field	Bits	Access	Default	Description
SUPPLY_MON_LVL_B	5	RW	0	Supply monitor level select for AMUXBUS_B 0 - VRL 1 - VRH
REFBUF_MODE	22:21	RW	0	The reference needs to be present when using TEMP sensor or diagnostic reference (in addition to SAR.DIAG_CTL.DIAG_EN). ***Note**: Setting this mode is not required for the ADC operation itself.* 00 - OFF - No Reference (disabled) 01 - ON - Reference enabled with buffered Vbg from SRSS. 10 - Reserved 11 - BYPASS - Reference enabled with unbuffered Vbg from SRSS

33.12 Registers

Table 291 Registers

Symbol	Name	Description
PASSx_SARy_CTL	Analog Control Register	This register controls the power and configuration of SAR ADC instance.
PASSx_SARy_DIAG_CTL	Diagnostic Reference Control Register	This register configures the diagnostic reference generator.
PASSx_SARy_PRECOND_CTL	Preconditioning Control Register	This register set the time for precondition. The value is set in number of clock cycles.
PASSx_SARy_ANA_CAL	Current Analog Calibration Values	This register stores the value of offset and gain for the ADC core.
PASSx_SARy_DIG_CAL	Current Digital Calibration Values	This register stores the value of digital offset and gain.
PASSx_SARy_ANA_CAL_ALT	Alternate Analog Calibration Values	This register stores the offset and gain; it enables the background calibration.
PASSx_SARy_DIG_CAL_ALT	Alternate Digital Calibration Values	It stores the digital offset and gain and enables the background calibration.
PASSx_SARy_CAL_UPD_CMD	Calibration Update Command Register	Coherently updates the calibration registers with the value from alternate calibration register.
PASSx_SARy_TR_PEND	Channel Trigger Pending Status Register	Channel trigger pending status bit is set when the trigger is received for the channel.

(table continues...)

Table 291 (continued) Registers

Symbol	Name	Description
PASSx_SARy_WORK_VALID	WORK Data Valid Flag Register	This is set when the data in the WORK register of channel is valid or the conversion is successfully completed.
PASSx_SARy_WORK_RANGE	WORK Range Detect Flag Register	Channel range detect flag register.
PASSx_SARy_WORK_RANGE_HI	WORK Outside Range Detect Flag Register	This bit is set when the range violation detected in OUTSIDE_RANGE mode.
PASSx_SARy_WORK_PULSE	Channel Pulse Detect	Pulse detect flag register.
PASSx_SARy_RESULT_VALID	Channel Result Data register 'valid' bits	Channel RESULT register data valid flags.
PASSx_SARy_RESULT_RANGE_HI	Channel Range Above Hi flags	This bit is set when the range violation detected in OUTSIDE_RANGE mode.
PASSx_SARy_STATUS	SAR Status register	Reads the status of SAR and the currently scanning channel.
PASSx_SARy_AVG_STAT	Current Averaging status	Reads the current value of the accumulator and counter
PASSx_SARy_CHz_TR_CTL	Channel Trigger Control Register	Sets the channel triggers, priority, preempt, and group related configurations.
PASSx_SARy_CHz_SAMPLE_CTL	Channel Sample Control Register	Configure the analog sampling related configurations such as physical pin address, sample time, and preconditioning.
PASSx_SARy_CHz_POST_CTL	Channel Post Processing Control Register	Configures the post processing of the converted analog value such as averaging, alignment, and range mode.
PASSx_SARy_CHz_RANGE_CTL	Channel Range Threshold Register	Stores the low and high range thresholds for range detection of the channel.
PASSx_SARy_CHz_INTR	Channel Interrupt Request Register	Channel interrupt request register clears the interrupt request by writing ‘1’.
PASSx_SARy_CHz_INTR_SET	Channel Interrupt Set Request Register	Sets the INTR by writing '1'.
PASSx_SARy_CHz_INTR_MASK	Channel Interrupt Mask Register	Channel interrupt masking register.
PASSx_SARy_CHz_INTR_MASKED	Channel Interrupt Masked Request Register	INTR register after the masking.
PASSx_SARy_CHz_WORK	Channel Working Data Register	Stores the conversion result as soon as it is completed for the channel
PASSx_SARy_CHz_RESULT	Channel Result Data Register	Data is copied from the work register after all the channels in the current group are sampled.

(table continues...)

Table 291 (continued) Registers

Symbol	Name	Description
PASSx_SARy_CHz_GRP_STAT	Channel Group Status Register	Reads the status of the current scanning group.
PASSx_SARy_CHz_ENABLE	Channel Enable Register	Channel enable/disable, resets trigger, and valid flags immediately if disabled.
PASSx_SARy_CHz_TR_CMD	Channel Software trigger	Channel software trigger, triggered by setting '1'. Always reads '0'.
PASS_PASS_CTL	PASS Control register	Debugs freeze control of all the ADC instances and reference buffer control.
PASS_SAR_TR_IN_SEL_x	Generic Input Trigger Select Register	Selection of five generic input triggers for the ADC.
PASS_SAR_TR_OUT_SEL_x	Generic Output Trigger Select Register	Selection of output trigger for the two generic output triggers.

***Note**: In PASSx_SARy_CHz, 'x' signifies the PASS instance, 'y' signifies the SAR instance, and 'z' signifies the SAR channel. Refer to the device datasheet for the specifications.*

34 Program and debug interface

Top-level architecture

Program and debug modules

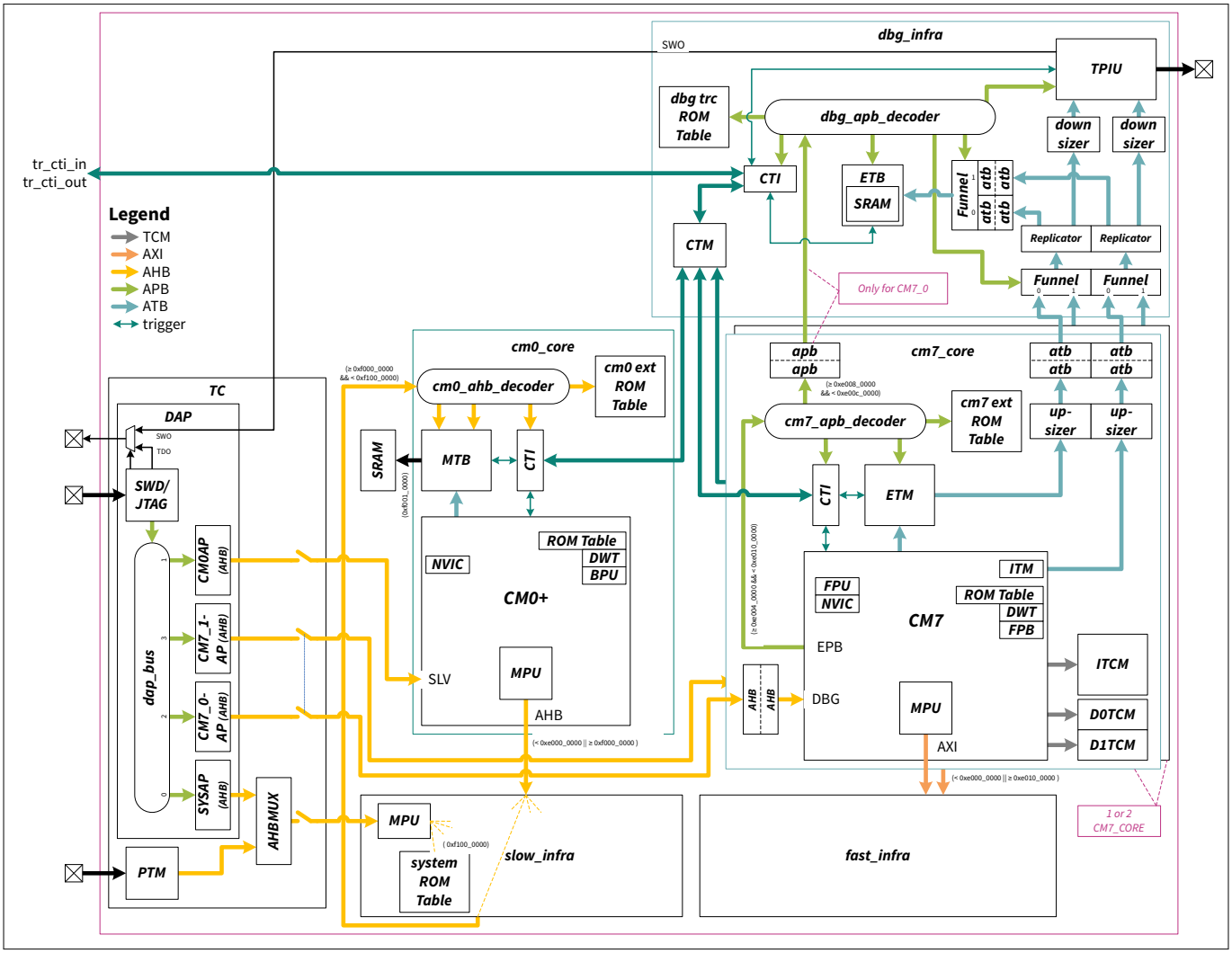


Figure 358 Program and debug modules

The XMC7000 program and debug interface provides a communication gateway for an external device to perform programming and debugging. The external device can be a Infineon-supplied programmer and debugger, or a third-party device that supports programming and debugging. The serial wire debug (SWD) or the JTAG interface can be used as the communication protocol between the external device and the XMC7000 device.

34.1 Features

The XMC7000 program and debug interface has the following features:

- Supports programming and debugging through the JTAG or SWD interface
- CM7 supports ETM/ITM tracing through a 4-bit TPIU, embedded trace buffer (ETB) with 8KB dedicated RAM. CM0+ supports micro trace buffer (MTB) with 4 KB dedicated RAM.
- Supports cross-trigger interface (CTI) and cross-trigger matrix (CTM)
- CM0+ supports four hardware breakpoints and two watchpoints. CM7 supports six hardware breakpoints and four watchpoints
- Provides read and write access to all memory and registers in the system while debugging, including the Cortex®-M7 and Cortex®-M0+ Register banks when the core is running or halted

34.2 Functional description

The 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 XMC7000 “target” using either the SWD or JTAG interface. The debug physical port pins communicate with the DAP through the high-speed I/O matrix (HSIOM). See the I/O system for details on HSIOM.

The debug infrastructure is organized into the following four groups:

- DAP (provides pin interfaces through which the debug host can connect to the chip)
- Cortex®-M0+ core debug components
- Cortex®-M7 core debug components, tightly related to the CPU and running at the same frequency
- Other debug infrastructure (includes the low-speed section of the CM7 tracing, the cross-triggering matrix, and the system ROM table)

The debug and trace infrastructure is built mostly using the library of CoreSight compliant components from Arm® CoreSight-400 IP. The following are the various debug and trace components.

The following are the various debug and trace components:

- Debug components
  - JTAG and SWD for debug control and access
- Trace source components
  - Micro trace buffer (MTB-M0+) to trace Cortex®-M0+ program execution
  - Embedded trace macro (ETM-M7) to trace Cortex®-M7 program execution
- Trace sink components
  - ETB for on-chip storage of the trace
  - Trace port interface unit (CoreSight-TPIU) to drive the trace information out of the device to an external trace port analyzer (TPA)
- Trace link components
  - ATB (AMBA trace bus) funnel
  - ATB replicator
- Cross-triggering components
  - CTI
  - CTM
- ROM tables

34.2.1 Debug access port (DAP)

The DAP consists of a combined SWD/JTAG interface (SWJ) that also includes the SWD listener. The SWD listener decides if the JTAG interface (default) or SWD interface is active. Note that JTAG and SWD are mutually exclusive because they share pins.

The debug port (DP) connects to the DAP bus, which in turn connects to one of three access ports (AP), namely:

- The CM0-AP, which connects directly to the AHB debug slave port (SLV) of the CM0+ and gives access to the CM0+ internal debug components. This also allows access to the rest of the system through the CM0+ AHB master interface. This provides the debug host the same view as an application running on the CM0+. This includes access to the MMIO of other debug components of the Cortex® M0+ subsystem. These debug components can also be accessed by the CM0+ CPU, but cannot be reached through the other APs or by the CM7 core.
- The CM7_0-AP/CM7_1-AP, which connects directly to AHB-Lite debug slave port pf CM7_0/CM7_1 (DBG) through an AHB2AHB bridge gives access to the CM7_0/CM7_1 internal debug components. The CM7_0-AP/CM7_1-AP also allows access to the rest of the system through the CM7_0/CM7_1 AHB-Lite master interfaces. This provides the debug host the same view as an application running on the CM7_0/CM7_1 core. Additionally, the CM7_0-AP/CM7_1-AP provides access to the debug components in the CM7_0/CM7_1 core through the external peripheral bus (EPB). These debug components can also be accessed by the CM7_0/CM7_1 CPU, but cannot be reached through the other APs or by the CM0+ core.
- The system-AP gives access to the rest of the system through an AHB mux. This allows access to the system ROM table, which cannot be reached any other way. The system ROM table provides the chip ID but is otherwise empty.

34.2.1.1 DAP security

For security reasons all three APs each can be independently disabled. Each AP disable is controlled by two MMIO bits. The DAP_CTL.xxx_AP_DISABLE bit (where xxx can be CM0 or CM7_0 or CM7_1 or SYS), can be set during boot, before the debugger can connect, based on eFuse settings. After this bit is set it cannot be cleared.

The second bit, DAP_CTL.xxx_AP_ENABLE (where xxx can be CM0 or CM7_0/CM7_1 or SYS), is a regular read/write bit. This bit also resets to zero and is set to '1' by either the ROM boot code or the flash boot code depending on the life-cycle stage. This feature can be used to block debug access during normal operation, but re-enable some debug access after a successful authentication.

In addition, the system AP is also protected by an MPU. This can be used to give the debugger limited access to the rest of the system. For chip identification, access to the system ROM table should be allowed. If debug access is restored after successful authentication, this MPU needs to be configured to allow authentication requests.

***Note**: The debug slave interfaces of all the CPUs bypass the internal CPU MPU.*

34.2.1.2 DAP power domain

Almost all the debug components are part of the active power domain. The only exception is the SWD/JTAG-DP, which is part of the deep sleep power domain. This allows the debug host to connect during deep sleep, while the application is 'running' or powered down. This enables in-field debugging for low-power applications in which the chip is mostly in deep sleep.

After the debugger is connected to the device, it needs to bring the device to the active state before any operation. For this, the SWD/JTAG-DP has a register (CTRL/STAT) with two power request bits. The two bits, CDBGPWRUPREQ and CSYSPWRUPREQ, request for debug power and system power respectively. These bits need to remain set for the duration of the debug session.

Note that only the two SWD pins (SWCLKTCK and SWDIOTMS) are operational during the Deep sleep mode - the JTAG pins are only operational in active mode. The JTAG debug and JTAG boundary scan are not available when the system is in deep sleep. JTAG functionality is only available after a device power-on-reset.

A system reset (XRES_L pin or AIRCR.SYSRESETREQ) will reset the I/O configuration and cause the host connection to be lost.

34.2.2 ROM tables

The ROM tables are organized in a tree hierarchy. Each AP has a register that contains a 32-bit address pointer to the base of the root ROM table for that AP. XMC7000 has four such root ROM tables.

Each ROM table contains 32-bit entries with an address pointer that either points to the base of the next level ROM table or a leaf debug component. Each ROM table also contains a set of ID Registers that hold JEDEC compliant identifiers to identify the manufacturer, part number, and major and minor revision numbers. These IDs are the same for all ROM tables in XMC7000. Each ROM table and CoreSight compliant component also contain Component Identification Registers.

34.2.3 Trace

XMC7000 supports instruction tracing for all the CPUs (design time configurable). The CM7 also includes an optional instrumentation trace macrocell (ITM). The CM0+ trace can only be captured in the MTB; that is, CM0+ tracing is not connected to the trace infrastructure.

The CM7 trace uses the standard AMBA trace bus (ATB) for both ITM and ETM output. The trace streams from the two CM7 CPUs are first combined using ATB funnel components to create a single trace stream. This trace output is then replicated, which enables the traces to go to two trace sink components, namely the trace port interface unit (TPIU) and the embedded trace buffer (ETB).

The ETB is similar to the MTB and captures the trace information in an on-chip SRAM, which can be retrieved by a debugger. The TPIU brings the trace off-chip through asynchronous interface with up to four pins. The trace sources live in the CM7 clock domain and have a width of 8-bit.

34.2.4 Embedded cross-triggering

The Arm® CoreSight includes embedded cross-triggering (ECT) to communicate events between debug components. These events are particularly useful with tracing and multicore platforms. For example, trigger events can be used to:

- Start or stop both CPUs at (almost) the same time
- Start or stop instruction tracing based on trace buffer being full or not or based on other events

CoreSight uses two components to support ECT, namely a CTI and a CTM, both of which are used in XMC7000.

The CTI component interfaces with other debug components, sending triggers back and forth and synchronizing them as needed. The CTM connects several CTIs, thus allowing events to be communicated from one CTI to another.

The XMC7000 has four CTIs, one for each CPU and one for the trace components in the debug structure. These three CTIs are connected together through the CTM. The CM7_0/CM7_1 CTI is located in the fast clock domain and the other two CTIs and the CTM are all located in the same slow-frequency clock domain. The list of the triggers connected to each CTI are as follows:

CM0+ CTI

- Input triggers:
  - 0=cm0p.halted // CM0+ is in debug mode
- Output triggers:
  - cm0p.edbgr // CM0+ to enter debug mode
  - 2 = sys.cm0_cti_irq[0] // Interrupt request
  - 3 = sys.cm0_cti_irq[1] // Interrupt request
  - 4 = mtb.tstart // Request MTB to start tracing
  - 5 = mtb.tstop // Request MTB to stop tracing
  - 7 = cm0p.dbgrestart // Request CM0+ to exit debug mode

CM7 CTI

- Input triggers:
  - 0 = cm7.halted // CM7 entered debug mode
  - 1 = cm7.dwtmatch[0] || cm7.dwtmatch[4] // CM7 DWT comparator outputs
  - 2 = cm7.dwtmacth[1] || cm7.dwtmatch[5] // CM7 DWT comparator outputs
  - 3 = cm7.dwtmacth[2] || cm7.dwtmatch[6] // CM7 DWT comparator outputs
  - 4 = etm.eventm[0] // ETM event output
  - 5 = etm.eventm[1] // ETM event output
  - 6 = etm.eventm[2] // ETM event output
  - 7 = etm.eventm[3] // ETM event output
- Output triggers:
  - 0 = cm7.edbgrq // Request CM7 to enter debug mode
  - 1 = sys.cm7_cti_irq[0] // interrupt request TBD which system interrupts?
  - 2 = sys.cm7_cti_irq[1] // interrupt request
  - 3 = etm.events[0] // ETM event input
  - 4 = etm.events[1] // ETM event input
  - 5 = etm.events[2] // ETM event input
  - 6 = etm.events[3] // ETM event input
  - 7 = cm7.dbgrestart // Request CM7 to exit debug mode

TRC CTI

- Input triggers:
  - 0 = cm0p.halted // CM0+ is in debug mode (level)
  - 2 = etb_full // Flag that ETB is full
  - 3 = etb_acqomp // Flag trace acquisition complete
  - 4 = cm7_0.halted // CM7_0 is in debug mode (level)
  - 5 = cm7_1.halted // CM7_1 is in debug mode (level)
  - 6 = CTI_TR_IN[0] // CTI trigger input from system triggers
  - 7 = CTI_TR_IN[1] // CTI trigger input from system triggers
- Output triggers:
  - 0 = etb_flushin // Request ETB to flush
  - 1 = etb_trigin // Request ETB to stop tracing
  - 2 = tpiu_flushin // Request TPIU to flush
  - 3 = tpiu_trigin // Request TPIU to stop tracing
  - 6 = CTI_TR_OUT[0] // CTI trigger output to system triggers
  - 7 = CTI_TR_OUT[1] // CTI trigger output to system triggers

Note that CoreSight cross-triggering is mostly separate from the peripheral trigger multiplexer. The only connection between the two are the four CTI_TR_IN/CTI_TR_OUT signals mentioned above.

***Note**: The CTI Registers are only accessible if a debugger is connected. Any code that needs to access CTI Registers should first ensure the presence of a debugger; for example, by using the CPUSS_DP_STATUS Register.*

34.3 Serial wire debug (SWD) interface

The XMC7000 device 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 XMC7000 target.
- **Target acknowledge response phase** – The XMC7000 target sends an acknowledgment 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 359 shows the timing diagrams of read and write SWD packets.

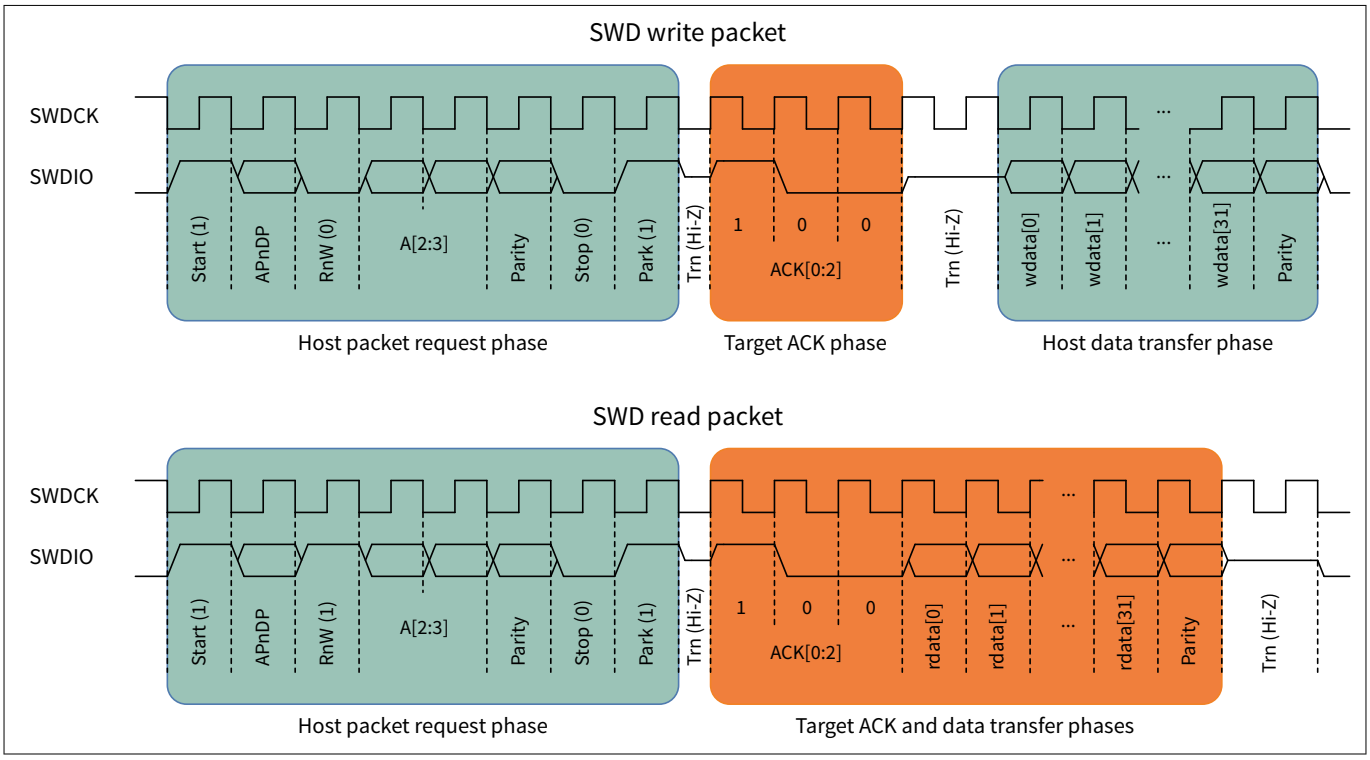


Figure 359 SWD write and read packet timing diagrams

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[2:3]) are register select bits for AP or DP, depending on the APnDP bit value.

  ***Note**: Address bits are transmitted with the LSb first.*

- **e.** The parity bit contains the parity of APnDP, RnW, and A[2:3] bits. It is an even parity bit; this means, when XORed with the other bits, the result will be 0.

  If the parity bit is not correct, the header is ignored by XMC7000; there is no ACK response (ACK = 3b111).

  The programming operation should be aborted and retried again by following a device reset.
- **f.** The stop bit is always logic 0.
- **g.** The park bit is always logic 1.

2. Target acknowledge response phase: SWDIO driven by the target

- **a.** The ACK[0:2] bits represent the target to host response, indicating failure or success, among other results.

  See Table 293 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 followed by at least two idle cycles.

34.3.1 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 292 and Figure 359 illustrate the timing of SWDIO bit writes and reads.

Table 292 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		

34.3.2 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. For a FAULT status, the programming operation should be aborted immediately. Table 293 shows the ACK bit-field decoding details.

Table 293 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 XMC7000. However, 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 XMC7000 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.
- For a FAULT response, the programming operation should be aborted and retried by doing a device reset.

34.3.3 Turnaround (Trn) period details

There is a turnaround period between the packet request and ACK phases, as well as between the ACK and data phases for host write transfers, as shown in Figure 359. According to the SWD protocol, the Trn period is used by both the host and target to change the drive modes on their respective SWDIO lines. During the first Trn period after the packet request, the target starts driving the ACK data on the SWDIO line on the rising edge of SWDCK. This action ensures that the host can read the ACK data on the next falling edge. Thus, the first Trn period lasts only one-half cycle. The second Trn period of the SWD packet is one and a half cycles. Neither the host nor the XMC7000 device should drive the SWDIO line during the Trn period.

34.4 JTAG interface

In response to higher pin densities on microcontrollers, the joint test action group (JTAG) proposed a method to test circuit boards by controlling the pins on the microcontrollers (and reading their values) via a separate test interface. The solution, later formalized as IEEE standard 1149.1-2001, is based on the concept of a serial shift register routed across all of the pins – hence the name boundary scan. The circuitry at each pin is supplemented with a multipurpose element called a boundary scan cell. In XMC7000 devices, most GPIO port pins have a boundary scan cell associated with them (see the GPIO block diagrams in the I/O system). The interface used to control the values in the boundary scan cells is called the test access port (TAP) and is commonly known as the JTAG interface. It consists of three signals: test data in (TDI), test data out (TDO), and test mode select (TMS). Also included is a clock signal (TCK) that clocks the other signals. TDI, TMS, and TCK are all inputs to the device and TDO is the output from the device. This interface enables testing multiple devices on a circuit board, in a daisy-chain fashion, as shown in Figure 360.

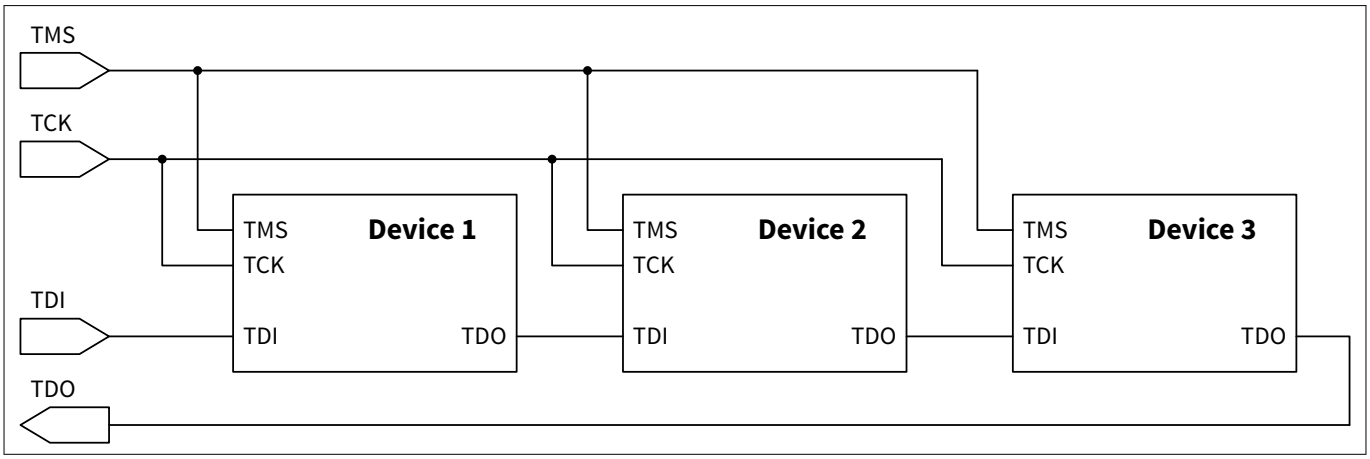


Figure 360 JTAG interface to multiple devices on a circuit board

Figure 361 shows the JTAG interface architecture within each device. Data at TDI is shifted in, through one of several available registers, and out to TDO.

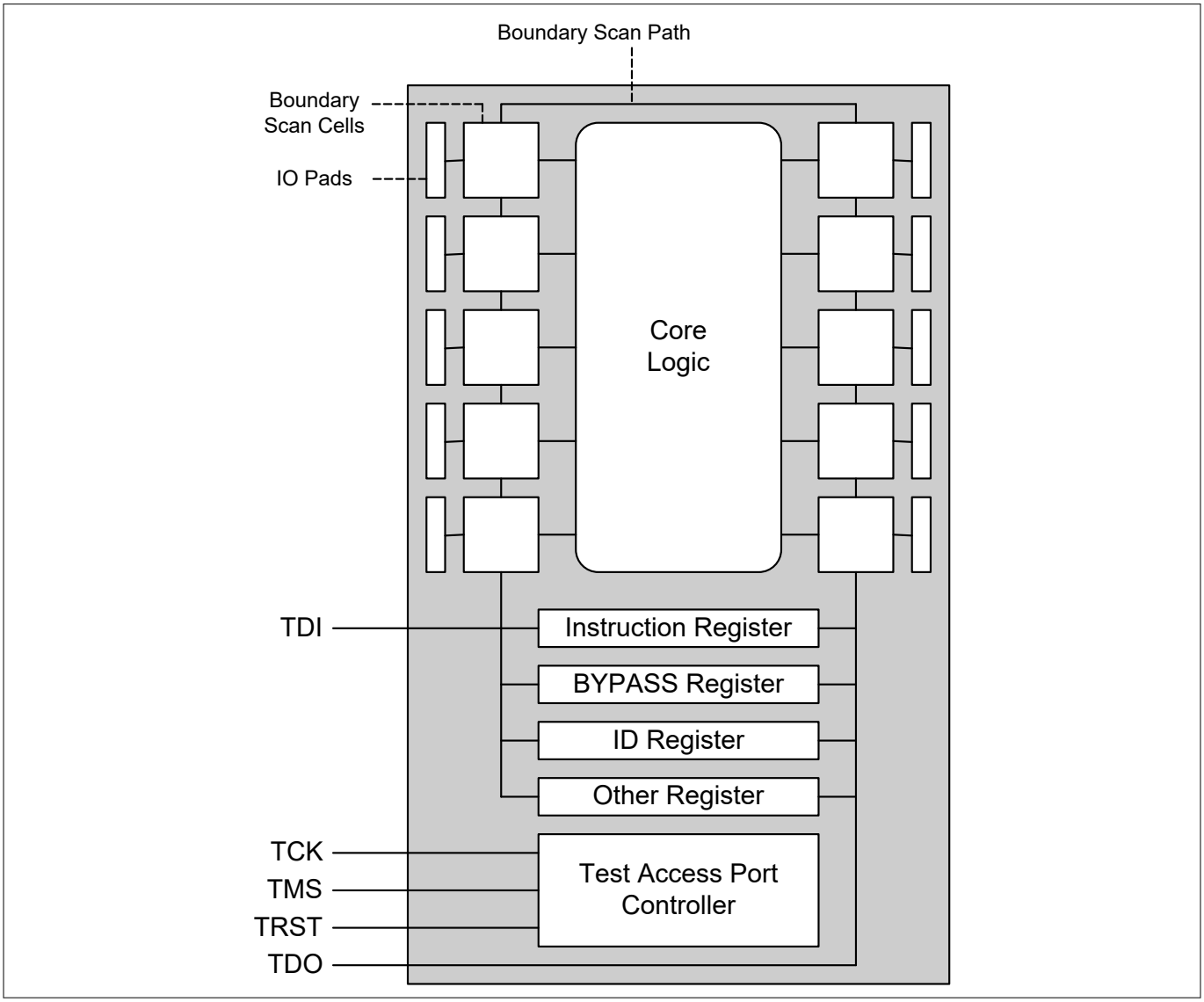


Figure 361 JTAG interface architecture

The TMS signal controls a state machine in the TAP. The state machine controls which register (including the boundary scan path) is in the TDI-to-TDO shift path, as shown in Figure 362. The following terms apply:

- ir - the Instruction Register
- dr - one of the other registers (including the boundary scan path), as determined by the contents of the Instruction Register
- capture - transfer the contents of a dr to a Shift Register, to be shifted out on TDO (read the dr)
- update - transfer the contents of a Shift Register, shifted in from TDI, to a dr (write the dr)

***Note**: Flash boot configures the JTAG reset input pin to SWJ_TRSTN mode upon reset (refer to the related device datasheet for the pin number). The JTAG TRSTn pin is optional for the JTAG debug protocol; the user application can configure it to some other mode (such as GPIO).*

If the debug protocol is JTAG and this pin is configured, the debug session will crash. To avoid this issue, modify the JTAG reset input to GPIO within the scope of the debug script.

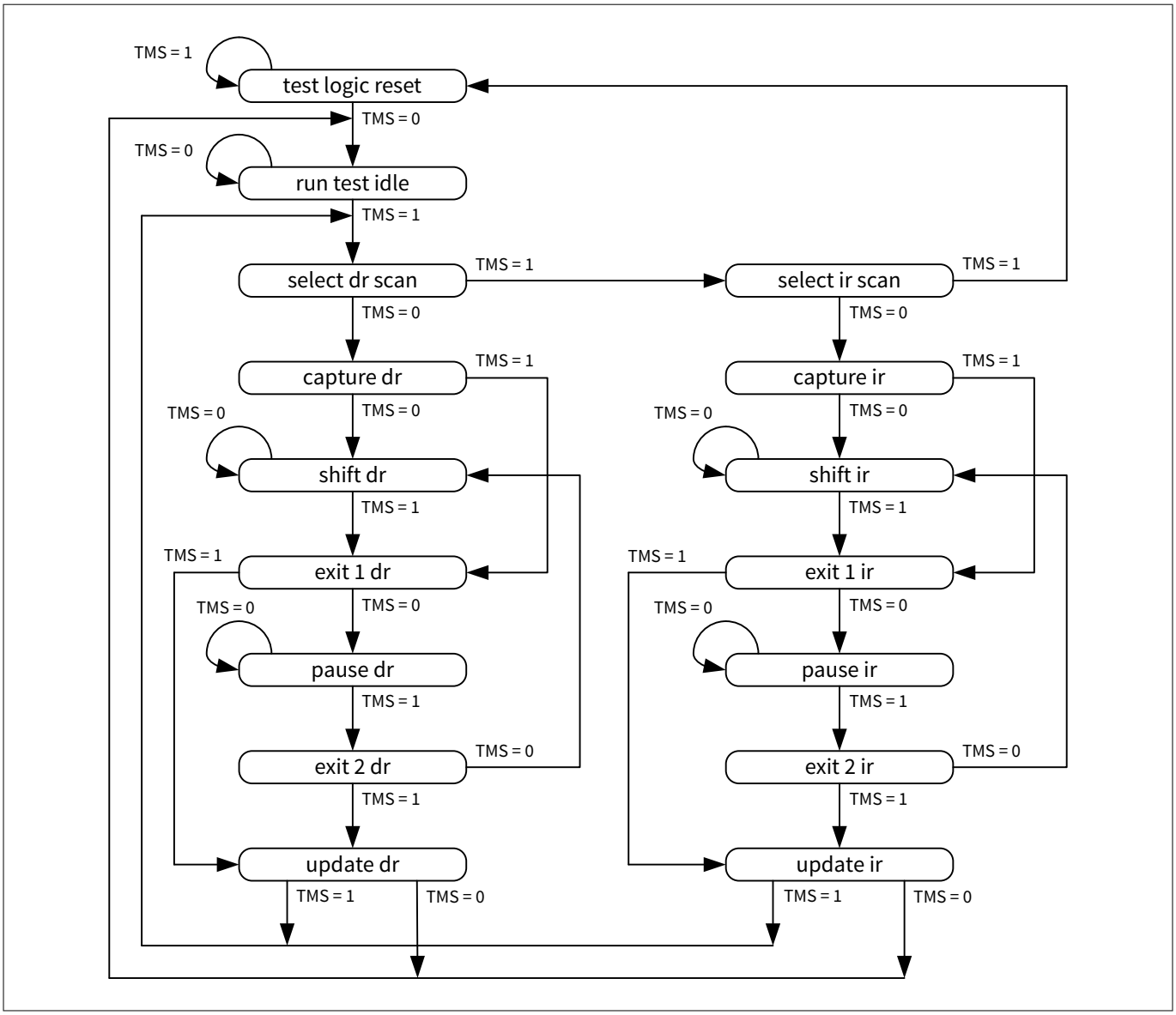


Figure 362 TAP state machine

The registers in the TAP are:

- Instruction – Typically two to four bits wide, holds the current instruction that defines which data register is placed in the TDI-to-TDO shift path.
- Bypass – one bit wide, directly connects TDI with TDO, causing the device to be bypassed for JTAG purposes.
- ID – 32 bits wide, used to read the JTAG manufacturer/part number ID of the device.
- Boundary Scan Path (BSR) – Width equals the number of I/O pins that have boundary scan cells, used to set or read the states of those I/O pins.

Other registers may be included according to the device manufacturer specifications. The standard set of instructions (values that can be shifted into the Instruction Register), as specified in IEEE 1149, are:

- EXTEST – Causes TDI and TDO to be connected to the BSR. The device is changed from its normal operating mode to a test mode. Then, the device's pin states can be sampled using the capture dr JTAG state. New values can be applied to the pins of the device using the update dr state.

- SAMPLE – Causes TDI and TDO to be connected to the BSR, but the device is left in its normal operating mode. During this instruction, the BSR can be read by the capture dr JTAG state to take a sample of the functional data entering and leaving the device.
- PRELOAD – Causes TDI and TDO to be connected to the BSR, but device is left in its normal operating mode. The instruction is used to preload test data into the BSR before loading an EXTEST instruction.

Optional, but commonly available, instructions are:

- IDCODE – Causes TDI and TDO to be connected to an IDCODE Register.
- INTEST – Causes TDI and TDO to be connected to the BSR. While the EXTEST instruction allows access to the device pins, INTEST enables similar access to the corelogic signals of a device

For more information, see the IEEE Standard, available at www.ieee.org.

***Note**: The boundary scan TAP is daisy-chained with the CoreSight DAP in the following order: TDI → Boundary Scan TAP (IR length = 4) → CoreSight DAP (IR length = 4) → TDO. To get boundary scan working, an additional BSDL file is needed that sets the CoreSight DAP into BYPASS mode.*

34.5 Pin configuration of debug interface on BootROM

After reset, the debug pins remain in high-impedance mode until the BootROM initializes them in the following way:

Pin name	Input enable	Drive mode
swj_trstn	Yes	Internal pull-up
swj_swo_tdo	Yes	Strong (output)
swj_swdoe_tdi	Yes	Internal pull-up
swj_swdio_tms	Yes	Internal pull-up
swj_swclk_tclk	Yes	Internal pull-down

34.6 Programming the XMC7000 device

XMC7000 is programmed using the following sequence. Refer to the XMC7000 MCU programming specification for complete details on the programming algorithm, timing specifications, and hardware configuration required for programming.

1. Acquire the SWD port in XMC7000.
2. Enter the programming mode.
3. Execute the device programming routines such as silicon ID check, flash programming, flash verification, and checksum verification.

34.6.1 SWD port acquisition

34.6.1.1 SWD port acquire sequence

The default interface on power-on reset is JTAG; to switch to SWD, use a transition through dormant state.

The first step in device programming is for the host to acquire the target's SWD port. The host performs a device reset by asserting XRES_L pin. After removing the XRES_L 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 debug access port must be reset using the standard Arm® command. The DAP reset command consists of more than 49 SWDCK clock cycles with SWDIO asserted 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 (2 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.

34.6.2 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 XMC7000 MCU programming specification document.

34.6.3 SWD programming routines executions

When the device is in programming mode, the external programmer can start sending the SWD packet sequence to perform programming operations such as flash erase, flash program, checksum verification, and so on. The programming routines are explained in the Non-volatile memory programming. The exact sequence of calling the programming routines is given in the XMC7000 MCU programming specification document.

34.7 Registers

Table 294 List of Registers

Register name	Description
SYSAP_ROM	System debug access port ROM-table with Infineon vendor/silicon ID
CM0P_DWT	Cortex® M0+ data watchpoint and trace (DWT) Registers
CM0P_BP	Cortex® M0+ system control space (SCS) Registers
CM0P_SCS	Cortex® M0+ breakpoint (BP) Registers
CM0P_ROM	Cortex® M0+ CPU coresight ROM table
CM0P_EXT_ROM	Cortex®-M0+ system ROM-table with Infineon vendor/silicon ID
CM0_MTB	Cortex®-M0+ MTB Registers
CM0P_MTB_SRAM	Cortex®-M0+ MTB SRAM
CM0P_CTI	Cortex® M0+ CTI Registers
CM7_ITM	Cortex®-M7 instrumentation trace macrocell (ITM) Registers
CM7_DWT	Cortex®-M7 data watchpoint and trace (DWT) Registers
CM7_FPB	Cortex®-M7 flash patch and breakpoint (FPB) Registers
CM7_SCS	Cortex®-M7 system control space (SCS) Registers
CM7_ETM	Cortex®-M7 embedded trace macrocell (ETM) Registers
CM7_CTI	Cortex®-M7 CTI Registers
TRC_CTI	System trace CTI
TRC_ITM_CSTF	System trace ITM coresight trace funnel (CSTF) Registers

(table continues...)

Table 294 (continued) List of Registers

Register name	Description
TRC_ETM_CSTF	System trace ETM coresight trace funnel (CSTF) Registers
TRC_ETB_CSTF	System trace ETB coresight trace funnel (CSTF) Registers
TRC_ETB	System trace embedded trace buffer (ETB) Registers
TRC_TPIU	System trace coresight trace port interface unit (TPIU) Registers
CM7_EXT_ROM	Cortex®-M7 system ROM-table with Infineon vendor/silicon ID
CM7_ROM	Cortex®-M7 CPU coresight ROM-table

35 Non-volatile memory programming

Non-volatile memory programming refers to the programming of flash memory in the XMC7000 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 XMC7000 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.

The XMC7000 device supports programming through the debug and access port (DAP) of Cortex®-M7 (CM7_0 and CM7_1) and Cortex®-M0+ CPUs.

35.1 Functional description

The user software must set up CM0+ IRQ0 and IRQ1 interrupts correctly for system call management. The boot code automatically sets CPUSS_CM0_SYSTEM_INT_CTL0.CPU_INT_VALID bit to 1 and CPUSS_CM0_SYSTEM_INT_CTL0.CPU_INT_IDX [2:0] bits to b'000. Hence, the mapping of System Interrupt 0 (IPC Interrupt Structure 0 interrupt) to CM0+ IRQ0 for system calls is done by boot code and CM0+ IRQ0 is triggered by IPC Interrupt Structure 0 interrupt.

However, the software needs to ensure that CM0+ IRQ0 and IRQ1 are enabled and they are configured with the correct priorities as this is not automatically done by the boot code. Also, the software must ensure that IRQ0 and IRQ1 vector entries in the user CM0+ vector table are identical to the vector entries in the default SROM vector table (addresses 0x00000040 and 0x00000044, respectively). This is achieved by copying values from the SROM vector table to the user vector table during runtime if it is located in RAM; otherwise, hard-coded values need to be used and reconfirmed if the target MCU or revision changes. Note that the software must ensure that the default hardfault vector entry is replaced with the specific user handler. The default SROM handler is only designed to be used during boot.

As explained in Operating modes and privilege levels, when the CPU is executing code in thread mode, the CONTROL Register can be configured to use the process stack pointer (PSP) or main stack pointer (MSP). In handler mode, the MSP is always used. Note that the CPU enters thread mode and uses MSP when it comes out of reset. The software must take special care while setting up the system call interrupts because this is dependent on the CPU mode (thread or handler) of CM0+ and the stack pointer (MSP or PSP) used when the system call was triggered.

Case 1: If the software triggers system calls only when CM0+ is in Handler mode, then ensure that the software sets CM0+ IRQ1 with a higher priority than IRQ0. To do this, set the IRQ0 priority to '1'. By default, IRQ1 priority will be set to '0'.

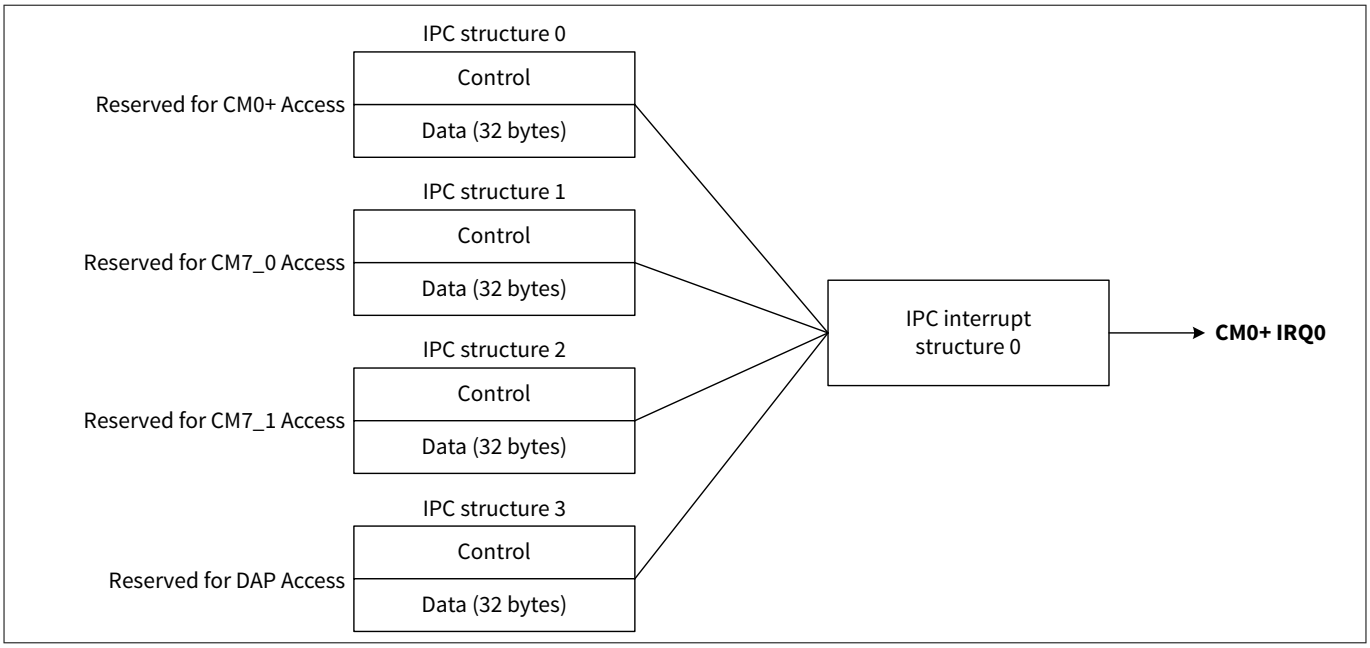


Figure 363 System call interface using IPC

Case 2: If the software triggers system calls with any other CPU states for CM0+ (for example, if thread mode and PSP is used), then the software additionally needs to use another CM0+ interrupt (such as IRQ2), which acts as a manager for system calls. This approach in principle can also be used for any CPU state (including handler mode). This is a more generic approach to manage system calls under all CPU states.

The following steps are involved in the CM0+ IRQ setup with this approach.

1. Set up the system call manager function (Sys_Call_Manager) as IRQ handler for CM0+ IRQ2 in the user vector table.
2. Map the IPC interrupt structure 0 interrupt CM0+ IRQ2.
3. IRQ2 must be given the lowest priority in relation with IRQ0 and IRQ1. The same highest priority is given for both IRQ0 and IRQ1. For example, set IRQ2 priority to 1; by default, IRQ0 and IRQ1 priority will be 0.
4. IRQ2 handler triggers IRQ0 in software.
5. IRQ2 handler clears the pending bit of IRQ0.

Thus, the CM0+ vector table will have the entries for the first three interrupts as shown in Table 295.

Table 295 CM0+ vector table

Interrupt number	Handler
…	…
IRQ0	Contents of address (0x00000040)
IRQ1	Contents of address (0x00000044)
IRQ2	Sys_Call_Manager
…	…

Note that instead of directly assigning Sys_Call_Manager as CM0+ IRQ2 handler, it can be combined with multiple other system interrupts when using a dispatcher implementation. This is explained in the Interrupts.

A pseudo code for the interrupt configuration needed for system call for case 2 is as follows.

```
/* IRQ2 handler function for IPC Interrupt structure 0 interrupt. This is the system call
manager function */
void Sys_Call_Manager()
{
    /* Trigger IRQ0 in Software by writing to ISPR Register */
    CM0P_SCS_ISPR = 1;
/* Read back the register to ensure that the write has happened */
    CM0P_SCS_ISPR;
/* Clear the NVIC Pending bit of IRQ0. This is done as a fallback in case the system call was
suppressed (e.g., by disabled interrupts) */
    CM0P_SCS_ICPR = 1;
/* Read back the register to ensure that the write has happened */
    CM0P_SCS_ICPR;
}
/* Application function for interrupt configurations */
void interrupt_configure()
{
    /* Enable CM0+ IRQ0, IRQ1 and IRQ2 */
    CM0P_SCS_ISER = 7;
/* Set Priority 0 for IRQ0, IRQ1 and Priority 1 for IRQ2 */
    CM0P_SCS_IPR0 = 0x00400000;
/* Connect IPC Interrupt Structure 0 Interrupt (System Interrupt 0) to IRQ2. The interrupt
triggers Sys_Call_Manager */
    CPUSS_CM0_SYSTEM_INT_CTL0.CPU_INT_IDX = 2;
/* Clear the PRIMASK Register to enable the interrupts. This could also be done by the
application at a later point in time */
    __ASM("cpsie i");
}
```

Note that in both cases, the System Call Interrupt IRQ0 execution should not be interrupted by another interrupt with higher or equal priority.

All system call requests from the master can arrive at the same time; the requests are prioritized at CM0+ > CM7_0 > CM7_1 > DAP.

The XMC7000 device IPC component implements two 32-bit data registers, but only one of these two registers is used to pass parameters to the system calls. This argument is either a pointer to SRAM or a formatted opcode or argument value that cannot be a valid SRAM address. The encoding used for DAP and the CM7_0, CM7_1, or CM0+ is slightly different.

- DAP: If (opcode + argument) is less than or equal to 31 bits, store them in the data field and set the LSb of the data field as ‘1’. Upon completion of the call, a return value is passed in the IPC Data Register. For calls that need more argument data, the data field is a pointer to a structure in SRAM (aligned on a word boundary) that has the opcode and the argument. Therefore, it is a pointer if and only if the LSb is 0.
- CM7_0, CM7_1, or CM0+: A pointer is always used to a structure in SRAM. Commands that are issued as a single word by DAP can still be issued by CM0+ or CM7_0/CM7_1, but use an SRAM structure instead.

The IRQ0 interrupt handler for system calls works as follows.

- If the ROM boot process code is not initialized in the protection state (PROTECTION is still at its default/reset value UNKOWN), the IRQ0 calls have no effect and the handler returns.
- A jump table is used to point to the code in ROM or flash. This jump table is in ROM or flash (as configured in supervisory flash).

The IPC mechanism is used to return the result of the system call. Two factors need to be considered.

- The result is to be passed in SRAM: CM0+ writes the result in the SRAM location provided by the requester and releases the IPC structure. The requester knows that the result is ready from the RELEASE interrupt.
- The result is scalar (32 bits): CM0+ writes the result to the data field of the IPC structure and releases it. The requester can read the data when the IPC structure lock is released. The requester polls the IPC structure to know when it is released.

External programmers program the flash memory of XMC7000-- using the JTAG or SWD protocol by sending the commands to the DAP. The programming sequence for XMC7000 with an external programmer is given in the XMC7000 MCU programming specifications. Flash memory can also be programmed by the CM7_0/CM7_1/CM0+ CPU by accessing the IPC 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 CM0+.

35.2 System call implementation

35.2.1 System call via CM0+ or CM7_0 or CM7_1

System calls can be made from the CM0+ or CM7_0/CM7_1 at any point during code execution. CM0+ or CM7_0/CM7_1 should acquire the IPC_STRUCT reserved for them and provide arguments in either of the methods described earlier and notify IPC interrupt 0 to trigger a system call.

35.2.1.1 System call via DAP

When the debug interface is acquired, then the boot ROM enters “busy-wait loop” and waits for commands issued by the DAP. For a detailed description on acquiring the debug interface see the XMC7000 MCU Programming Specifications.

35.2.2 Exiting from a system call

When the API operation is complete, CM0+ will release the IPC structure that initiated the system call. If an interrupt is required upon release, then the corresponding mask bit should be set in IPC_INTR_STRUCT.INTR_MASK.RELEASE[i].

35.3 SROM API library

SROM has two categories of APIs:

- Flash management APIs – These APIs provide the ability to program, erase, and test the flash macro.
- System management APIs – These APIs provide the ability to perform system tasks such as blowing eFuse and checksum.

Table 296 shows a summary of the APIs.

Table 296 List of system calls

System call	Opcode	Description	Access allowed		
			Normal[1)]	Secure	Dead
BlankCheck	0x2A	Performs blank check on the addressed work flash	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP

(table continues...)

Table 296 (continued) List of system calls

System call	Opcode	Description	Access allowed		
			Normal[1)]	Secure	Dead
BlowFuseBit	0x01	Blows an eFuse bit	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1	
CheckFactoryHash	0x27	Generates the FACTORY_HASH according to TOC1 and compares with the FACTORY1_HASH fuses	CM0+, CM7_0, CM7_1, DAP		
CheckFMStatus	0x07	Returns the status of the flash operation	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP
Checksum	0x0B	Calculates the checksum of a flash region	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP
ComputeBasicHash	0x0D	Computes the hash value of a flash region	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP
ConfigureFMInterrupt	0x08	Configures the flash macro interrupt	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP
EraseAll	0x0A	Erases all flash	CM0+, CM7_0, CM7_1, DAP		DAP
EraseResume	0x23	Resumes a suspended erase operation	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP
EraseSector	0x14	Erases a flash sector	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP
EraseSuspend	0x22	Suspends and ongoing erase operation	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP
GenerateHash	0x1E	Returns the truncated SHA-256 of the flash boot programmed in SFlash	CM0+, CM7_0, CM7_1, DAP	–	–
SwitchOverRegulators	0x11	Switches between REGHC and linear regulators	CM0+	CM0+	–
ConfigureRegulator	0x15	Configures high-current regulator (REGHC) for devices that include REGHC, or PMIC for devices that use PMIC control without REGHC	CM0+	CM0+	–
ProgramRow	0x06	Programs the addressed flash page	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP
ProgramWorkFlash	0x30	Programs the addressed work flash page	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP
ReadFuseByte	0x03	Reads addressed eFuse byte	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP	–

(table continues...)

Table 296 (continued) List of system calls

System call	Opcode	Description	Access allowed		
			Normal[1)]	Secure	Dead
ReadFuseByteMargin	0x2B	Reads addressed eFuse byte marginally	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP	–
ReadSWPU	0x2C	Reads the identified SWPU from SRAM	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP	–
ReadUniqueID	0x1F	Reads the unique ID of the die from flash	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP
SetEnforcedApproval	0x2E	Sets the enforced approval bit in SRAM	CM0+	CM0+	CM0+
SiliconID	0x00	Returns family ID, revision ID, silicon ID and protection state	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP
SoftReset	0x1B	Provides system reset or CM7_0, CM7_1 only reset	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP
TransitiontoRMA	0x28	Converts parts from SECURE to RMA life-cycle stage	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP	–
TransitiontoSecure	0x2F	Converts parts to Secure life-cycle stage	CM0+, CM7_0, CM7_1, DAP	–	–
DirectExecute	0x0F	Directly executes code located at a configurable address	DAP	–	–
WriteRow	0x05	Programs SFlash	CM0+, CM7_0, CM7_1, DAP	–	CM0+, CM7_0, CM7_1, DAP
WriteSWPU	0x2D	Updates the identified SWPU in SRAM	CM0+, CM7_0, CM7_1, DAP	CM0+, CM7_0, CM7_1, DAP	–
DebugPowerUpDown	0x12	Enables/disables CM7 debugging	CM0+, DAP	CM0+, DAP	CM0+, DAP
LoadRegulatorTrims	0x16	Sets proper trims to PWR_TRIM_HT_PWRSYS_CTL	CM0+	CM0+	–

1) See the Chip operational modes for more details.

35.4 System calls

Table 296 lists all the system calls supported in XMC7000 device along with the function description and availability in device protection modes. See the Device security for more information on the device protection settings. Note that some system calls cannot be called by the CM7_0, CM7_1, CM0+, or DAP as given in the table. Detailed information on each system call follows the table.

***Note**: System calls that require more than 32-bit arguments, such as program row and erase sector APIs, will first fetch the parameter address from IPC_DATA0 to derive further arguments and expect it to be a 32-bit aligned address in SRAM; if this is not followed, then the SROM API will trigger a HardFault.*

If NC ECC fault(s) are pending in the SRAM controller #0 and SWPUs are populated in the design, then Flash and eFuse system calls (for example, BlowFuseBit, ProgramRow and so on) will return a misleading status of 0xf0000005 ("Page is write protected") even for the non-protected row or 0xf0000002 ("Invalid Fuse address") for valid efuse address. To workaround this problem, if the NC ECC fault(s) are not due to HW malfunction (i.e. if the faults are due to usage of non-initialized SRAM or improper SRAM initialization), then clearing of these pending faults will resolve the issue. Pending faults can be cleared by making use of any of the Fault Structures. See Fault Subsystem chapter on page 182 for details on capturing and clearing pending faults.

***Note**: For Flash system calls requested by DAP (e.g. ProgramRow), SYS DAP MPU is also checked for Flash write accessibility. The API returns an error status if the DAP MPU does not allow Flash access on the requested region.*

The MPU of SYS-AP is configured based on the access restrictions encoding stored in SFlash or eFuse.

***Note**: System calls targeting Flash (such as ProgramRow, EraseSector) can be triggered in two modes - Blocking mode and non-blocking mode.*

When Blocking mode is used, CM0+ core is “blocked“ inside SROM till the flash operation is complete. This means that the CM0+ core cannot execute any user application code while the flash operation is in progress.

When non-blocking mode is used, the CM0+ core just initiates the flash controller to start the flash operations and immediately comes out of SROM. This means that the CM0+ core is now free to execute its own tasks while the flash operation is in progress. However, CM0+ cannot execute code or read data from the same logical bank where the flash operation is currently going on. For example, the CM0+ core can execute code out of SRAM while the flash operation is in progress.

***Note**: The ReadSWPU or WriteSWPU system calls shall not be called while executing EraseSector or ProgramRow in non-blocking mode on bank #0. ReadSWPU or WriteSWPU read data from bank #0 in SFLASH. While doing that, the check for active non-blocking erase or program of bank #0 is not performed. Therefore, reading bank #0 while there is an active erase/ program operation will trigger a bus error which can result in a hardfault occurrence based on the FLASHC_FLASH_CTL register settings.*

35.4.1 BlankCheck

Performs blank check on the addressed work flash in blocking mode.

Table 297 Parameters

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x2A	Blank Check opcode
Bits [23:16]	–	Not used
Bits [15:8]	–	Not used
Bits [7:0]	–	Not used
SRAM_SCRATCH_ADDR + 0x04		
Bits [31:0]	–	Work flash address whose blank check needs to be performed. It should be provided in 32-bit system address format.
SRAM_SCRATCH_ADDR + 0x08		
Bits [31:16]	–	Not used
Bits [15:0]	0: 1 word 1: 2 words …	Number of words to be checked

Table 298 Return

Address	Return value	Description
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS/program command ongoing in background 0xF = ERROR	Status code (see System call status for details)
Bits [27:24]	–	Not used
Bits [23:8]	–	In case of fail, first failed word number
Bits [7:0]	–	In case of fail, error code (see SROM API status codes)

35.4.2 BlowFuseBit

This function blows the addressed eFuse bit. The read value of a blown eFuse bit is '1'. Parameters and result are described here.

APIs that target blowing of eFuses, such as BlowFuseBit and TransitionToSecure, have some requirements for clk_hf0. To avoid complications, these APIs can be triggered with any of the clock settings used by internal boot (specified by TOC2_FLAGS.CLOCK_CONFIG) before the application changes the clk_hf0 settings.

If the application changes the configurations used by boot for clk_hf0, then ensure that the source clock for clk_hf0 is FLL and the frequency of clk_hf0 is restricted to 100 MHz maximum. If clk_hf0 is not sourced from FLL and FLL is disabled, then the maximum frequency of clk_hf0 should only be 8 MHz.

Table 299 Arguments if IPC_DATA[0] = 1

Address	Value to be written	Description
IPC_DATA0 register		
Bits [31:24]	0x01	Blow fuse bit opcode.
Bits [23:16]	Byte Address	Refer to the Device security for more details.
Bits [15:12]	Macro Address	
Bits [11]	Not used	
Bits [10:8]	Bit Address	
Bits [7:1]	Not used	–
Bit [0]	0x1	Indicates that all the arguments are passed in the IPC_DATA0 Register.

Table 300 Arguments if IPC_DATA[0] = 0

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x01	Blow fuse bit opcode.
Bits [23:16]	Byte address	Refer to the Device security for more details.
Bits [15:12]	Macro address	
Bits [11]	Not used.	
Bits [10:8]	Bit address	
Bits [7:0]	Not used.	–

***Note**: Because the SRAM address is 32-bit aligned, the last two bits of the address are 0. Therefore, IPC_DATA[0] is 0.*

Refer to Customer eFuses for details about macro address/byte address calculation.

Table 301 Return if DAP invoked the system call

Address	Return value	Description
IPC_DATA0 Register		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	Error code (if any)	See System call status for details.

Table 302 Return if IPC_STRUCT.DATA[0] = 0

Address	Return value	Description
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	Error code (if any)	See System call status for details.

35.4.3 CheckFactoryHash

Generates the FACTORY_HASH according to TOC1 and compares with the FACTORY1_HASH fuses.

Table 303 Arguments if IPC_STRUCT.DATA[0] = 1

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:24]	0x27	Check factory hash opcode
Bits [23:1]	Not used	–
Bit [0]	0x1	Indicates that all the arguments are passed in the IPC_DATA0 Register.

Table 304 Parameters if CM0+/CM7_0/CM7_1 is master

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x27	Check Factory Hash opcode
Bits [23:0]	Not used	–

Table 305 Return if IPC_STRUCT.DATA[0] = 1

Address	Return value	Description
IPC_DATA0 Register		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR (INVALID_FACTORY_HASH)	Status code (see System call status for details).
Bits [23:0]	Error code (if any)	See System call status for details.

Table 306 Return if CM0+/CM7_0/CM7_1 invoked the system call

Address	Return value	Description
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR (INVALID_FACTORY_HASH)	Status code (see System call status for details).
Bits [23:0]	Error code (if any)	See System call status for details.

35.4.4 CheckFMStatus

This API returns the status of the flash operation.

***Note**: The flash operation status can be retrieved by directly reading the FLASHC_STATUS Register without the use of the CheckFMStatus API.*

Table 307 Arguments if IPC_STRUCT.DATA[0] = 1

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:24]	0x07	Check FM status opcode
Bits [23:1]	Not used	–
Bit [0]	0x1	Indicates that all the arguments are passed in the IPC_DATA0 Register.

Table 308 Arguments if IPC_STRUCT.DATA[0] = 0

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x07	Check FM Status opcode.
Bits [23:0]	Not used.	–

Table 309 Return if IPC_STRUCT.DATA[0] = 1

Address	Return value	Description
IPC_DATA0 Register		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [8:0]	Error code (0x1) or status	See System call status for details.

Table 310 Return if IPC_STRUCT.DATA[0] = 0

Address	Return value	Description
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	Error code or status	See System call status for details.

Table 311 Status

Bits	Name	Description
0	PGM_CODE	Indicates if active PGM operation to the code flash is taking place 0: not running 1: running
1	PGM_WORK	Indicates if active PGM operation to the work flash is taking place 0: not running 1: running
2	ERASE_CODE	Indicates if active erase operation to the code flash is taking place 0: not running 1: running
3	ERASE_WORK	Indicates if active erase operation to the work flash is taking place 0: not running 1: running
4	ERS_SUSPEND	Indicates if erase operation (code/work) is currently being suspended 0: not suspended 1: suspended
5	BLANK_CHECK_WORK	Indicates if blank check mode is currently running on the work flash 0: not running 1: running

(table continues...)

Table 311 (continued) Status

Bits	Name	Description
6	BLANK_CHCEK_PASS	Indicates the blank check command result is PASS (Blank) 0: Not blank 1: Blank (PASS)+G76
7	HANG	After embedded operation (pgm/erase) this flag will tell if it was successful or failed 0: PASS 1: FAIL
8	BUSY	Whenever the device is in embedded mode the RDY goes low. Should be the same as the c_interrupt pin (but inverted) 1: busy in embedded 0: rdy (high also in erase suspend)

35.4.5 Checksum

Checksum reads either the whole flash or a row of flash, and returns the sum of each byte read.

Bytes 1 and 2 of the parameter select whether the checksum is performed on the whole flash, or a row of flash. The row of SFlash or main or work flash is determined by the Row Id Lo and Row Id Hi parameters.

This API checks if the client has read access to the requested memory region by looking into DAP MPUs and SMPUs. If the client does not have read access, then STATUS_ROW_PROTECTED status is returned.

If the flash is configured in dual bank mode, then the appropriate bank needs to be provided when the whole flash option is selected. If bank 1 is selected in single bank mode, then API will return invalid argument status. Note that only one bank of SFlash is exposed.

Parameters and result are described here.

Table 312 Arguments if IPC_STRUCT.DATA[0] = 1

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:24]	0x0B	Checksum opcode
Bits [23:22]	0 - code 1 - work Other - supervisory	Flash region
Bit [21]	0 - page 1 - whole memory	Page or whole memory
Bits [20:8]	–	Row ID
Bit [7]	0 - Bank 0 1 - Bank 1	Bank (Only for dual bank device)
Bits [6:1]	–	Not used
Bit [0]	1	Indicates that all the arguments are passed in the IPC_DATA0 Register

Table 313 Arguments if IPC_STRUCT.DATA[0] = 0

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x0B	Checksum opcode
Bits [23:22]	0 - code 1 - work Other - supervisory	Flash region
Bit [21]	0 - page 1 - whole memory	Page or whole memory
Bits [20:8]	–	Row ID
Bit [7]	0 - Bank 0 1 - Bank 1	Bank (Only for dual bank device)
Bits [6:1]	–	Not used.
Bit [0]	0	–

Table 314 Return if IPC_STRUCT.DATA[0] = 1

Address	Return value	Description
IPC_DATA1 Register		
Bits [31:0]	Checksum	–
IPC_DATA0 Register		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	Error code (if any)	See System call status for details.

Table 315 Return if IPC_STRUCT.DATA[0] = 0

Address	Return value	Description
IPC_DATA1 Register		
Bits [31:0]	Checksum	–
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	Error code (if any)	See System call status for details.

35.4.6 Compute basic hash

This function generates the hash of the flash region provided using the formula:

H(n+1) = {H(n) × 2+Byte}% 127; where H(0) = 0

This function returns an invalid address status if called on an out-of-bound flash region.

This function checks if the client has read access to the requested memory region by looking into DAP MPUs and SMPUs. If the client does not have read access, then STATUS_ROW_PROTECTED status is returned.

The first byte of the parameter specifies if a CRC8SAE is computed based on the following polynomial

x^{8}+x^{4}+x^{3}+x^{2}+1

Table 316 Parameters

Address	Value to be written	Description
IPC_DATA Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x0D	Compute hash opcode.
Bits [23:16]	–	Not used.
Bits [15:8]	0x01 - CRC8SAE Other - Basic hash	Hash type
Bits [7:0]	–	Not used
SRAM_SCRATCH_ADDR + 0x04		
Bits [31:0]	–	Start address (32-bit system address of the first byte of the data).
SRAM_SCRATCH_ADDR + 0x08		
Bits [31:0]	0 - 1 byte 1 - 2 bytes, 2 - 3 bytes and so on	Number of bytes

Table 317 Return

Address	Return value	Description
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	Hash of the data	Hash of data if status is SUCCESS; otherwise, an error code.

35.4.7 ConfigureFMInterrupt

Configures the flash macro interrupt.

The following functionalities are provided:

- Set interrupt mask
- Clear interrupt mask
- Clear interrupt

Table 318 Arguments if IPC_STRUCT.DATA[0] = 1

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:24]	0x08	Configure flash macro interrupt opcode.
Bits [15:8]	0: Clear interrupt mask 1: Set interrupt mask Other: Clear interrupt	–
Bit [0]	0x1	Indicates that all the arguments are passed in the IPC_DATA0 Register.

Table 319 Arguments if IPC_STRUCT.DATA[0] = 0

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x08	Configure Flash Macro Interrupt opcode.
Bits [15:8]	0: Clear interrupt mask 1: Set interrupt mask Other: Clear interrupt	–
Bits [7:0]	Not used.	–

Table 320 Return if IPC_STRUCT.DATA[0] = 1

Address	Return value	Description
IPC_DATA0 Register		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [8:0]	Error code (0x1) or status	See System call status for details.

Table 321 Return if IPC_STRUCT.DATA[0] = 0

Address	Return value	Description
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [8:0]	Error code (0x1) or status	See System call status for details.

35.4.8 EraseAll

This function erases the whole flash macro specified. This API will erase only the code flash. The API returns fail status if user does not have write access to flash according to SMPU settings.

Note that when in dual bank mode, the API will always erase the alternate bank addressed from 0x12000000.

Table 322 Arguments if IPC_STRUCT.DATA[0] = 1

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:24]	0x0A	Erase all opcode
Bits [23:1]	–	Not used
Bit [0]	1	Indicates that all the arguments are passed in the IPC_DATA0 Register.

Table 323 Arguments if IPC_STRUCT.DATA[0] = 0

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x0A	Erase all opcode
Bits [23:0]	–	Not used

Table 324 Return if IPC_STRUCT.DATA[0] = 1

Address	Return value	Description
IPC_DATA0 Register		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	Error code (if any)	See System call status for details.

Table 325 Return if IPC_STRUCT.DATA[0] = 0

Address	Return value	Description
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	Error code (if any)	See System call status for details.

35.4.9 EraseResume

This function resumes a suspended erase operation.

Table 326 Arguments if IPC_STRUCT.DATA[0] = 1

Address	Value to be written	Description
IPC_ DATA0 Register		
Bits [31:24]	0x23	Erase resume opcode.
Bits [23:16]	0x01 - Set FM interrupt mask. Other - FM interrupt mask not set.	Interrupt mask, only applicable when non-blocking.
Bits [15:8]	0x01 - API blocks CM0+ Other - non-blocking	Blocking mode
Bits [7:1]	–	Not used
Bit [0]	0x1	Indicates all arguments are passed in the IPC_DATA0 Register.

Table 327 Arguments if IPC_STRUCT.DATA[0] = 0

Address	Value to be written	Description
IPC_ DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x23	Erase resume opcode.
Bits [23:16]	0x01 - Set FM interrupt mask. Other - FM interrupt mask not set.	Interrupt mask, only applicable when non-blocking.
Bits [15:8]	0x01 - API blocks CM0+	–
Other - non-blocking	Blocking mode	–
Bits [7:0]	–	Not used

Table 328 Return if IPC_STRUCT.DATA[0] = 1

Address	Return value	Description
IPC_ DATA0 Register		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	Error code (if any)	See System call status for details.

Table 329 Return if IPC_STRUCT.DATA[0] = 0

Address	Return value	Description
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	Error code (if any)	See System call status for details.

35.4.10 EraseSector

This function starts erase operation on the specified sector. This function cannot be called on SFlash; the API will return STATUS_INVALID_FLASH_ADDR if invoked on SFlash.

EraseSector is allowed on the sector that is erase suspended. If EraseSector is called on a sector other than the suspended one, then the new sector will be erased and the suspended sector will be in an unknown state. EraseSector can be called on the suspended sector to restore to blank state.

Note that user should perform a dummy read from Work FLASH after erase operation is complete if EraseSector is invoked in non-blocking mode. Dummy read is required to make the logical bank of Work FLASH ready for read operation after a program or erase operation. This is not applicable if EraseSector is invoked in Blocking mode.

Table 330 Parameters

Address	Value to be written	Description
IPC_DATA0 register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x14	Erase Sector opcode.
Bits [23:16]	0x01 - Set FM interrupt mask. Other - FM interrupt mask not set.	Interrupt mask, only applicable when non-blocking.
Bits [15:8]	0x01 - API blocks CM0+ Other - non-blocking	Blocking mode
Bits [7:0]	–	Not used

(table continues...)

Table 330 (continued) Parameters

Address	Value to be written	Description
SRAM_SCRATCH_ADDR + 0x04		
Bits [31:0]	–	Flash address to be erased. Should be provided in 32-bit system address format. For example, to erase the second sector you need to provide the 32-bit system address of any of the bytes lying in the second sector.

Table 331 Return

Address	Return value	Description
SRAM_SCRATCH register		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	Error code (if any)	See System call status for details.

35.4.11 EraseSuspend

This function suspends an ongoing erase operation. User should not read from a sector that is suspended. The Program Row API function will return error if invoked on suspended sector.

Table 332 Arguments if IPC_STRUCT.DATA[0] = 1

Address	Value to be written	Description
IPC_DATA0 register		
Bits [31:24]	0x22	Erase suspend opcode
Bits [23:1]	–	Not used
Bit [0]	0x1	Indicates all arguments are passed in the IPC_DATA0 Register.

Table 333 Parameters if CM0+/CM7_0/CM7_1 is master

Address	Value to be written	Description
IPC_DATA0 register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x22	Erase suspend opcode
Bits [23:0]	–	Not used

Table 334 Return if IPC_STRUCT.DATA[0] = 1

Address	Return value	Description
IPC_DATA0 register		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details)
Bits [27:0]	Error code (if any)	See System call status for details

Table 335 Return if CM0+/CM7_0/CM7_1 invoked the system call

Address	Return value	Description
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details)
Bits [27:0]	Error code (if any)	See System call status for details

35.4.12 GenerateHash

This API returns the truncated SHA-256 of the flash boot programmed in SFlash and optionally includes public key and other objects as indicated in table of contents (TOC).

This function gets the flash boot size from TOC.

Typically, this function will be called to check if the HASH to be blown into eFuse matches with what ROM boot expects it to be.

***Note**: If the TOC1/TOC2 hash object start address is wrong or is an unaligned address (not 4-byte aligned), a hard-fault will be generated.*

Table 336 Parameters

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x1E	Generate hash opcode
Bits [23:16]	–	Not used
Bits [15:8]	0x1: returns FACTORY_HASH Other: returns hash of all objects according to TOC1 and 2	Factory
Bits [7:0]	–	Not used

Table 337 Return

Address	Return value	Description
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS/program command ongoing in background 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	–	In case of fail, error code (see SROM API status codes)
SRAM_SCRATCH_ADDR + 0x4		
Bits [31:0]	HASH_WORD0	–
SRAM_SCRATCH_ADDR + 0x8		
Bits [31:0]	HASH_WORD1	–
SRAM_SCRATCH_ADDR + 0xC		
Bits [31:0]	HASH_WORD2	–
SRAM_SCRATCH_ADDR + 0x10		
Bits [31:0]	HASH_WORD3	–
SRAM_SCRATCH_ADDR + 0x14		
Bits [31:0]	HASH_ZEROS	–

35.4.13 SwitchOverRegulators

This function is used to switch between the high-current regulator (REGHC or PMIC without REGHC) required to run CM7 and the linear regulator (LDO). It should be called to switch from LDO to REGHC before enabling CM7. The Configure Regulator system call should be called before using this function.

***Note**: If the API is called in the blocking mode it will handle setting of the proper regulator trims. However, if the API is called in the non-blocking mode, then the proper trims will not be set for transition to the external regulator. Therefore, the trims should be set when transition is complete.*

For transition to LDO, the required trims will be set by the API before the transition is initialized.

Table 338 Arguments if IPC_STRUCT.DATA[0] = 1

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:24]	0x11	Opcode
Bit [3]	Blocking: 0- Non-blocking call 1- Blocks CM0+ until the transition completes	–

(table continues...)

Table 338 (continued) Arguments if IPC_STRUCT.DATA[0] = 1

Address	Value to be written	Description
Bit [2]	Regulator: 0 - Switch over to REGHC (PMIC without REGHC) 1 - Switch over to linear regulator	–
Bit [1]	Operating mode: 0 - External transistor 1 - External PMIC	For devices without REGHC, the operating mode is ignored.
Bit [0]	0x1	Indicates that all the arguments are passed in the IPC_DATA0 Register.

Table 339 Arguments if IPC_STRUCT.DATA[0] = 0

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x11	Opcode.
Bits [23:16]	Blocking: 0- Non-blocking call 1- Blocks CM0+ until the transition completes	–
Bits [15:8]	Select Regulator: 0 - Switch over to REGHC 1 - Switch over to LDO	–
Bit [1]	Operating mode: 0 - External transistor 1 - External PMIC	For devices without REGHC, the operating mode is ignored.
Bit [0]	Not used	–

Table 340 Return if IPC_STRUCT.DATA[0] = 1

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR 0xF1 = Computed hash	Status code (see System call status for details).
Bits [23:0]	Error code (if any) or computed hash	See System call status for details.

Table 341 Return if IPC_STRUCT.DATA[0] = 0

Address	Value to be written	Description
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR 0xF1 = Computed hash	Status code (see System call status for details).
Bits [23:0]	Error code (if any) or computed hash	See System call status for details.

35.4.14 ConfigureRegulator

This function is used to configure the high-current regulator (REGHC) for devices that include REGHC, or PMIC for devices that use PMIC control without REGHC. It should be called to configure the desired regulator only once before switching to the regulator using the switch over regulators system call.

Table 342 Arguments if IPC_STRUCT.DATA[0] = 1

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:24]	0x15	Opcode.
Bits [15:13]	Radj value	RADJ reset threshold value These bits are invalid in CYT4D and CYT3D series devices
Bits [12:8]	0x10	VADJ trim value (VadjTrim) used in the regulator output trim. This value should be fixed to 0x10 and is valid only for the “External transistor” operating mode.
Bit [7]	0 - Device generates VADJ when PMIC is enabled. 1 - Device does not generate VADJ, and it must not be part of the PMIC feedback loop	This bit configures REGHC_PMIC_VADJ_DIS
Bit [6]	0 - no Radj 1 - use Radj	Use Radj to generate a reset threshold for the PMIC
Bit [5]	0 - Internal active linear regulator is disabled after PMIC is enabled. OCD is disabled 1 - Internal active linear regulator is kept enabled. See the related datasheet for the minimum PMIC V_{CCD} input to prevent OCD	UseLinReg

(table continues...)

Table 342 (continued) Arguments if IPC_STRUCT.DATA[0] = 1

Address	Value to be written	Description
Bit [4]	0 - Allow HC regulator to operate in normal mode 1 - Allow HC regulator to operate in Deep sleep mode	This bit configures PMIC behavior in Deep sleep mode
Bit [3]	Reset polarity: 0 – Logic low is used for enable 1 – Logic high is used for enable	The polarity used to trigger a reset action based on the PMIC status input of the regulator
Bit [2]	Enable polarity: 0 – Logic low is used for enable 1 – Logic high is used for enable	Polarity used to enable the regulator
Bit [1]	0 – External transistor 1 – External PMIC	Operating mode For devices without REGHC, the operating mode is ignored.
Bit [0]	0x1	Indicates that all arguments are passed in the IPC_DATA0 Register
IPC_DATA1 Register		
Bit [8:0]	Wait Count	Wait count in steps of 4 µs after PMIC status is OK. This is used by the hardware sequencer to allow additional settling time before disabling the internal regulator. Note that the ConfigureRegulator API supports wait count field [8:0]; it does not support wait count bit [9]. Therefore, if the user needs wait count more than 0x1FF, set the PWR_REGHC_CTL.REGHC_PMIC_STATUS_WAIT[29:20] register after the ConfigureRegulator API is successful.

Table 343 Arguments if IPC_STRUCT.DATA[0] = 0

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR1	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
IPC_DATA1 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR2	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR1		
Bits [31:24]	0x15	Opcode.

(table continues...)

Table 343 (continued) Arguments if IPC_STRUCT.DATA[0] = 0

Address	Value to be written	Description
Bits [15:13]	Radj Value	RADJ reset threshold value These bits are invalid in CYT4D and CYT3D series devices
Bits [12:8]	0x10	VADJ trim value (VadjTrim) used in the regulator output trim. This value should be fixed to 0x10 and is valid only for the “External transistor” operating mode.
Bit [7]	0 - Device generates VADJ when PMIC is enabled. 1 - Device does not generate VADJ, and it must not be part of the PMIC feedback loop	This bit configures REGHC_PMIC_VADJ_DIS
Bit [6]	0 - no Radj 1 - use Radj	Use Radj to generate a reset threshold for the PMIC
Bit [5]	0 - Internal active linear regulator is disabled after PMIC is enabled. OCD is disabled 1 - Internal active linear regulator is kept enabled. See the datasheet for the minimum PMIC V_{CCD} input to prevent OCD	UseLinReg
Bit [4]	0 - Allow HC regulator to operate in normal mode 1 - Allow HC regulator to operate in Deep sleep mode	This bit configures PMIC behavior in Deep sleep mode
Bit [3]	Reset polarity: 0 – Logic low is used for enable 1 – Logic high is used for enable	The polarity used to trigger a reset action based on the PMIC status input of the regulator.
Bit [2]	Enable polarity: 0 – Logic low is used for enable 1 – Logic high is used for enable	Polarity used to enable the regulator.
Bit [1]	0 – External transistor 1 – External PMIC	Operating mode. For devices without REGHC, the operating mode is ignored.
Bit [0]	Not used.	–

(table continues...)

Table 343 (continued) Arguments if IPC_STRUCT.DATA[0] = 0

Address	Value to be written	Description
SRAM_SCRATCH_ADDR2		
Bit [8:0]	Wait count	Wait count in steps of 4 µs after PMIC status is OK. This is used by the hardware sequencer to allow additional settling time before disabling the internal regulator. Note that the ConfigureRegulator API supports wait count field [8:0]; it does not support wait count bit [9]. Therefore, if the user needs wait count more than 0x1FF, set the PWR_REGHC_CTL.REGHC_PMIC_STATUS_WAIT[29:20] register after the ConfigureRegulator API is successful.

Table 344 Return if IPC_STRUCT.DATA[0] = 1

Address	Return value	Description
IPC_DATA0 Register		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details)
Bits [23:0]	Error code (if any)	See System call status for details

Table 345 Return if IPC_STRUCT.DATA[0] = 0

Address	Return value	Description
SRAM_SCRATCH_ADDR1		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details)
Bits [23:0]	Error code (if any)	See System call status for details

35.4.15 ProgramRow

This function programs the addressed flash page (the flash can be code flash or work flash). The user needs to provide the data to be loaded and the flash address to be programmed. The flash page should be in the erased state before calling this function. Otherwise, it will return an error status. The function returns a fail status if the user does not have write access to flash according to SMPU/SWPU settings.

The FM interrupt mask option can be set to generate an interrupt from the flash macro when running with non-blocking option.

Note that the user should perform a dummy read from work flash after the program operation is complete if ProgramRow is invoked in non-blocking mode. Dummy read is required to make the logical bank of work flash ready for read operation after a program or erase operation. This is not applicable if ProgramRow is invoked in blocking mode.

Table 346 Parameters

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x06	Program Row opcode
Bits [23:16]	0x01 - Skips the blank check step. Other - Perform blank check	Skip blank check
Bits [15:8]	0x01 - API blocks CM0+ Other - non-blocking	Blocking mode
Bits [7:0]	–	Not used
SRAM_SCRATCH_ADDR + 0x04		
Bits [31:24]	0x01 - Set FM interrupt mask. Other - FM interrupt mask not set.	Interrupt mask, only applicable when non-blocking.
Bits [23:16]	–	Not used
Bits [15:8]	–	Not used
Bits [7:0]	3 - 64 bits 5 - 256 bits 9 - 4096 bits	Data size for code flash. For work flash the data size is always 32 bits.
SRAM_SCRATCH_ADDR + 0x08		
Bits [31:0]	–	Flash address to be programmed. This should be provided in 32-bit system address format.
SRAM_SCRATCH_ADDR + 0x0C		
Bits [31:0]	SRAM_SCRATCH_DATA_ADDR	Address of SRAM where data to be programmed is stored
SRAM_SCRATCH_DATA_ADDR		
Bits [31:24]	–	Data byte 3 to be programmed in flash
Bits [23:16]	–	Data byte 2 to be programmed in flash
Bits [15:8]	–	Data byte 1 to be programmed in flash
Bits [7:0]	–	Data byte 0 to be programmed in flash
SRAM_SCRATCH_DATA_ADDR + (n-3)		
Bits [31:24]	–	Data byte n to be programmed in flash
Bits [23:16]	–	Data byte n-1 to be programmed in flash

(table continues...)

Table 346 (continued) Parameters

Address	Value to be written	Description
Bits [15:8]	–	Data byte n-2 to be programmed in flash
Bits [7:0]	–	Data byte n-3 to be programmed in flash

Table 347 Return

Address	Return value	Description
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS/program command ongoing in background 0xF = ERROR	Status code (see System call status for details).
Bits [27:0]	Error code (if any)	See System call status for details. In case of success: • 0x0 indicates successful completion of API (second phase) • 0x9 indicates successful completion of first phase, program command is ongoing in the background.

35.4.16 ProgramWorkFlash

This function programs the addressed work flash page. The function is not applicable for programming of any flash other than work flash and will return an error status when called on non-work flash. The user must provide the data to be loaded and the work flash address to be programmed. The flash page should be in the erased state before calling this function. Otherwise, it will return an error status. The function returns a fail status if the user does not have write access to flash according to SMPU/SWPU settings.

Table 348 Parameters

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits[31:24]	0x30	ProgramWorkFlash opcode
Bits[23:16]	0x01 - Skips the blank check step. Other - Perform blank check	Skip blank check
Bits[15:8]	0x01 - API blocks CM0+	Only blocking mode is supported
Bits[7:0]	–	Reserved.
SRAM_SCRATCH_ADDR + 0x04		
Bits[31:24]	–	Reserved
Bits [23:16]	–	Reserved

(table continues...)

Table 348 (continued) Parameters

Address	Value to be written	Description
Bits [15:8]	–	Reserved
Bits[7:0]	2 - 32 bits 3 - 64 bits 4 - 128 bits 5 - 256 bits 6 - 512 bits 7 - 1024 bits 8 - 2048 bits 9 - 4096 bits	Data size
SRAM_SCRATCH_ADDR + 0x08		
Bits [31:0]	–	Work flash address to be programmed. This should be provided in 32-bit system address format.
SRAM_SCRATCH_ADDR + 0x0C		
Bits [31:0]	SRAM_SCRATCH_DATA_ADDR	Address of SRAM where data to be programmed is stored
SRAM_SCRATCH_DATA_ADDR		
Bits [31:24]	–	Data byte 3 to be programmed in work flash
Bits [23:16]	–	Data byte 2 to be programmed in work flash
Bits [15:8]	–	Data byte 1 to be programmed in work flash
Bits [7:0]	–	Data byte 0 to be programmed in work flash
SRAM_SCRATCH_DATA_ADDR + (n-3)		
Bits [31:24]	–	Data byte n to be programmed in work flash
Bits [23:16]	–	Data byte n-1 to be programmed in work flash
Bits [15:8]	–	Data byte n-2 to be programmed in work flash
Bits [7:0]	–	Data byte n-3 to be programmed in work flash

Table 349 Return

Address	Return value	Description
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS/program command ongoing in the background 0xF = ERROR	Status code (see System call status for details).

(table continues...)

Table 349 (continued) Return

Address	Return value	Description
Bits [27:0]	Error code (if any)	See System call status for details. In case of success: 0x0 indicates successful completion of API (second phase) 0x9 indicates successful completion of first phase; program command is ongoing in the background.

35.4.17 ReadFuseByte

This function returns the value of an eFuse. The read value of a blown eFuse bit is '1' and that of a not blown eFuse bit is '0'. This API inherits the client protection context.

Table 350 Arguments if IPC_STRUCT.DATA[0] = 1

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:24]	0x03	Read fuse byte opcode
Bits [23:8]	Value in the range [0,511]	eFuse address
Bits [7:0]	0x01	Indicates all arguments are passed in IPC_DATA0

Table 351 Arguments if IPC_STRUCT.DATA[0] = 0

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x03	Read fuse byte opcode.
Bits [23:8]	Value in the range [0,511]	eFuse address
Bits [7:0]	–	Not used

Table 352 Return if IPC_STRUCT.DATA[0] = 1

Address	Return value	Description
IPC_DATA0 Register		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	eFuse byte	Byte read from eFuse if status is success; otherwise, error code.

Table 353 Return if IPC_STRUCT.DATA[0] = 0

Address	Return value	Description
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	eFuse byte	Byte read from eFuse if status is success; otherwise, error code.

35.4.18 ReadFuseByteMargin

API returns the eFuse contents of the addressed byte read marginally.

The read value of a blown eFuse bit is '1' and that of not blown is '0'.

This API inherits client's protection context.

Table 354 Arguments if IPC_STRUCT.DATA[0] = 1

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:24]	0x2B	Read fuse byte margin opcode
Bits [23:20]	0: Low resistance,-50% from nominal 1: Nominal resistance (default read condition) 2: High resistance (+50% from nominal) Other: Higher resistance (+100% from nominal)	Margin control
Bits [19:8]	Value in the range [0,511]	eFuse address
Bit [0]	0x01	Indicates all arguments are passed in IPC_DATA0

Table 355 Arguments if IPC_STRUCT.DATA[0] = 0

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x2B	Read fuse byte margin opcode

(table continues...)

Table 355 (continued) Arguments if IPC_STRUCT.DATA[0] = 0

Address	Value to be written	Description
Bits [23:20]	0: Low resistance,-50% from nominal 1: Nominal resistance (default read condition) 2: High resistance (+50% from nominal) Other: Higher resistance (+100% from nominal)	Margin control
Bits [19:8]	Value in the range [0,511]	eFuse address
Bits [7:0]	–	Not used

Table 356 Return if IPC_STRUCT.DATA[0] = 1

Address	Return value	Description
IPC_DATA0 Register		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	eFuse byte	Byte read from eFuse if status is success; otherwise, error code.

Table 357 Return if IPC_STRUCT.DATA[0] = 0

Address	Return value	Description
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	eFuse byte	Byte read from eFuse if status is success; otherwise, error code.

35.4.19 ReadSWPU

Reads the identified SWPU from SRAM. The PU ID is based on the storage of SWPU in SFlash. There is only one contiguous SWPU indexing in SFlash even though there are two physically separate storage in SFlash.

Table 358 Parameters

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x2C	Read SWPU opcode

(table continues...)

Table 358 (continued) Parameters

Address	Value to be written	Description
Bits [23:16]	1: eFuse write 2: eFuse read Other: Flash write	PU type
Bits [15:8]	Structure ID to be read. Indexed from 0	PU ID
Bits [7:0]	–	Not used
SRAM_SCRATCH_ADDR + 0x04		
Bits [31:0]	SRAM_DATA_ADDRESS	–

Table 359 Return

Address	Return value	Description
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS/program command ongoing in background] 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	–	In case of fail, error code (see SROM API status codes)
SRAM_DATA_ADDRESS		
Bits [31:0]	SL_OFFSET/SL_ADDRESS	–
SRAM_DATA_ADDRESS + 0x4		
Bits [31:0]	SL_SIZE	–
SRAM_DATA_ADDRESS + 0x8		
Bits [31:0]	SL_ATT	–
SRAM_DATA_ADDRESS + 0xC		
Bits [31:0]	MS_ATT	–

35.4.20 ReadUniqueID

Returns the unique ID of the die from SFlash.

Table 360 Parameters

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x1F	Read Unique ID opcode
Bits [23:0]	–	Not used

Table 361 Return

Address	Return value	Description
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS/program command ongoing in background 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	Error if any or DIE_ID0	In case of fail, error code (see SROM API status codes)
SRAM_SCRATCH_ADDR + 0x4		
Bits [31:0]	DIE_ID1	–
SRAM_SCRATCH_ADDR + 0x8		
Bits [31:0]	DIE_ID2	–

***Note**: ID includes production date of the device as well as other manufacturing information such as lot, wafer, and die serial numbers, which in combination ensures the uniqueness of the ID within the XMC7000 family.*

35.4.21 SetEnforcedApproval

Sets the EnforcedApproval bit in SRAM. Enforced Approval bit is stored in PC1 private SRAM. If this bit is set then API checks for supervised marker.

Table 362 Parameters if DAP is master

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:24]	0x2E	Set enforced approval opcode
Bit [0]	0x01	Indicates all arguments are passed in IPC_DATA0

Table 363 Parameters if CM0+/CM7_0/CM7_1 is master

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x2E	Set Enforced Approval opcode
Bits [23:0]	–	Not used

Table 364 Return if DAP invoked the system call

Address	Return value	Description
IPC_DATA0 Register		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	–	Error code if any.

Table 365 Return if CM0+/CM7_0/CM7_1 invoked the system call

Address	Return value	Description
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	–	Error code if any.

35.4.22 SiliconID

This function returns a 12-bit family ID, 16-bit silicon ID, 8-bit revision ID, and the current protection state.

Note that only 32 bits are available to store the return value in the IPC structure. Therefore, the API takes a parameter ID type based on which it will return family ID and revision ID if the ID type is set to '0'. It will return silicon ID and protection state if the ID type is set to '1'.

If invoked by a CMx core, the API returns zero; the family ID and revision ID must be obtained from the CPUSS_PRODUCT_ID Register.

Table 366 Silicon ID

Infineon IDs	Memory location	Data
Family ID [7:0]	0xF0000FE0	Part number [7:0]
Family ID [11:8]	0xF0000FE4	Part number [3:0]
Major revision	0xF0000FE8	Revision [7:4]
Minor revision	0xF0000FEC	Rev and minor revision field [7:4]
Silicon ID	SFlash	Silicon ID [15:0]
Protection state	MMIO	Protection [3:0]

Table 367 Arguments if IPC_STRUCT.DATA[0] = 1

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:24]	0x00	Silicon ID opcode

(table continues...)

Table 367 (continued) Arguments if IPC_STRUCT.DATA[0] = 1

Address	Value to be written	Description
Bits [15:8]	0 - returns 0. Use the CPUSS_PRODUCT_ID Register to get family ID and revision ID 1 - returns 16-bit silicon ID and protection state 2 - returns SROM firmware version Others - returns invalid argument status	ID type
Bits [7:1]	–	Not used
Bit [0]	0x1	Indicates that all the arguments are passed in IPC_DATA0.

Table 368 Arguments if IPC_STRUCT.DATA[0] = 0

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH		
Bits [31:24]	0x00	Silicon ID opcode
Bits [15:8]	0 - returns 12-bit family ID and revision ID 1 - returns 16-bit silicon ID and protection state 2 - returns SROM firmware version Others - returns invalid argument status	ID type
Bits [7:0]	–	Not used

Return if IPC_STRUCT.DATA[0] = 1

Table 369 If ID type is 0

Address	Return value	Description
IPC_DATA0 Register		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [23:20]	Major revision ID	See the XMC7000 MCU Programming Specifications for these values.
Bits [19:16]	Minor revision ID	–
Bits [15:8]	Family ID byte high	See the device datasheet for silicon ID values for different part numbers.
Bits [7:0]	Family ID byte low	–

Table 370 If ID type is 1

Address	Return value	Description
IPC_DATA0 Register		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [27:24]	–	Not used
Bits [23:20]	0: VIRGIN 1: NORMAL 2: SEC_W_DBG 3: SECURE 4: RMA 5: SORT 6: PROVISIONED 7: NORMAL_PROVISIONED 9: CORRUPTED	Note that devices are in the NORMAL_PROVISIONED stage when shipped. The VIRGIN, NORMAL, SORT, and PROVISIONED life-cycle stages are not applicable for final samples.
Bits [19:16]	0: UNKNOWN 1: VIRGIN 2: NORMAL 3: SECURE 4: DEAD	Protection state
Bits [15:8]	Silicon ID Byte High	See the device datasheet for silicon ID values for different part numbers.
Bits [7:0]	Silicon ID Byte Low	–

Table 371 If ID type is 2

Address	Return value	Description
IPC_DATA0 Register		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [27:24]	Flash boot major version	–
Bits [23:16]	Flash boot minor version	–
Bits [15:8]	SROM firmware major version	–
Bits [7:0]	SROM firmware minor version	–

Return if IPC_STRUCT.DATA[0] = 0

Same values as for DAP but located in the SRAM_SCRATCH location.

35.4.23 SoftReset

Resets the system by setting CM0+ AIRCR system reset bit. This will result in a system-wide reset except for debug logic. This API can also be used to selective reset just CM7_0/CM7_1 core based on 'type' parameter. CM7_0/CM7_1 should be in Deep sleep mode when selectively resetting CM7_0/CM7_1. Soft reset API called with type parameter set to 1 will result in CM7_0/CM7_1 transition to enabled state. Note that this API will return an error status if CM7_0/CM7_1 core reset is requested when CM7_0/CM7_1 is in active mode.

Table 372 Parameters if DAP is master

Address	Value to be written	Description
IPC_DATA0 register		
Bits [31:24]	0x01B	Soft reset opcode.
Bits [23:8]	Value in the range [0,511]	eFuse address
Bits [7:1]	0: System reset 1: Only CM7_0/CM7_1 resets	Type
Bit [0]	0x01	Indicates all arguments are passed in IPC_DATA0

Table 373 Parameters if CM0+/CM7_0/CM7_1 is master

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x01B	Soft Reset opcode
Bits [23:8]	Value in the range [0,511]	eFuse address
Bits [7:1]	0: System reset 1: Only CM7_0/CM7_1 resets	Type
Bit [0]	–	Not used

Table 374 Return if DAP invoked the system call

Address	Return value	Description
IPC_DATA0 Register		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	Error code (if any)	See System call status for details.

Table 375 Return if CM0+/CM7_0/CM7_1 invoked the system call

Address	Return value	Description
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).

(table continues...)

Table 375 (continued) Return if CM0+/CM7_0/CM7_1 invoked the system call

Address	Return value	Description
Bits [23:0]	Error code (if any)	See System call status for details.

35.4.24 Transition to RMA

Converts parts from SECURE or SECURE WITH DEBUG to the RMA life-cycle stage. This API returns the 0xF00000A9 failure code if any active embedded flash operations are going on. Note that TransitiontoRMA will consume an additional 2 KB of SRAM starting at address 0x28000800. For successful execution of the system call, read and write access for this area should be provided for protection context 1 (PC1). Otherwise, the execution will fail and there will be no transition into the RMA life-cycle stage. When using the TransitiontoRMA API, to move a device to the RMA life-cycle stage, parameters such as certificate and digital signature must be placed from [SRAM0 start address + 4 KB].

***Note**: Due to improper initialization of the Crypto memory buffer and internal SRAM0, Crypto and SRAM0 ECC errors may be set after the TransitionToRMA SROM API call. The PERI_GROUP_VIO_2 fault may also get set. To avoid this issue, do not configure the fault structure for Crypto, SRAM0 ECC errors, and PERI_GROUP_VIO_2 fault before triggering TransitionToRMA, or ignore the ECC faults reported during TransitionToRMA execution.*

Table 376 Parameters

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x28	Transition to RMA opcode
Bits [23:0]	–	Not used
SRAM_SCRATCH_ADDR + 0x4		
Bits [31:0]	–	Object size in bytes (including itself). It should always be 20 bytes.
SRAM_SCRATCH_ADDR + 0x8		
Bits [31:0]	0x120028F0	Command ID
SRAM_SCRATCH_ADDR + 0xC		
Bits [31:0]	–	Unique ID word 0
SRAM_SCRATCH_ADDR + 0x10		
Bits [31:0]	–	Unique ID word 1
SRAM_SCRATCH_ADDR + 0x14		
Bits [31:0]	–	Unique ID word 2 (3 bytes)
SRAM_SCRATCH_ADDR + 0x18		
Bits [31:0]	–	SRAM address where signature is stored (4 bytes)

Table 377 Return

Address	Return value	Description
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	Error if any	In case of fail, error code (see SROM API status codes)

35.4.25 Transition to Secure

Validates the FACTORY_HASH and programs SECURE_HASH, secure access restrictions and dead access restrictions into eFuse.

Programs secure or secure with debug fuse to transition to SECURE or SECURE with DEBUG life-cycle stage.

Only allowed in NORMAL_PROVISIONED life-cycle stage

***Note**: See section BlowFuseBit for more details on clock requirements while blowing eFuses.*

Table 378 Parameters

Address	Value to be written	Description
IPC_DATA0 register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x2F	Transition to secure opcode
Bits [15:8]	1: Blow D fuse Other: Blow Sfuse	Debug
Bits [7:0]	–	Not used
SRAM_SCRATCH_ADDR + 0x4		
Bits [31:0]	bit[1:0]AP_CTL_CM0_DISABLE bit[3:2] AP_CTL_CM7_0/CM7_1_DISABLE bit[5:4]AP_CTL_SYS_DISABLE bit[6]SYS_AP_MPU_ENABLE bit[7]DIRECT_EXECUTE_DISABLE bit[10:8]FLASH_ALLOWED bit[13:11]SRAM_ALLOWED bit[15:14]WORK_FLASH_ALLOWED bit[17:16]SFLASH_ALLOWED bit[19:18]MMIO_ALLOWED	SECURE_ACCESS_RESTRICT
SRAM_SCRATCH_ADDR + 0x8		
Bits [31:0]	–	DEAD_ACCESS_RESTRICT

Table 379 Return

Address	Return value	Description
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Birs[23:0]	Error if any	In case of fail, error code (see SROM API status codes)

35.4.26 DirectExecute

Directly executes code located at a configurable address. API is allowed in VIRGIN state. In NORMAL state, API is allowed only if the corresponding DIRECT_EXECUTE_DISABLE bit (in SFlash/eFuse) is '0'.

Table 380 Parameters

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:24]	0x0F	Direct execute opcode.
Bits [23:2]	Address[21:0]	–
Bit [1]	0: SRAM 1: Flash	Memory
Bit [0]	0x1	Indicates that all the arguments are passed in the IPC_DATA0 Register.

Table 381 Parameters if IPC0_DATA0[0] is 0

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x0F	Direct execute opcode
Bits [23:2]	Not used	–
Bits [1:0]	0: (void, void) 1: (void, long32) 2: (long32, void) 3: (long32, long32)	FuncType Value: (return,arg)
SRAM_SCRATCH_ADDR + 4		
Bits [31:0]	–	Argument
SRAM_SCRATCH_ADDR + 8		
Bits [31:0]	–	Address (32-bit system address of the code to execute)

(table continues...)

Table 381 (continued) Parameters if IPC0_DATA0[0] is 0

Address	Value to be written	Description
SRAM_SCRATCH_ADDR + 12		
Bits [31:0]	–	Return
SRAM_SCRATCH_ADDR + 16		
Bits [31:0]	–	FM API status (this field is primarily used by s40flash functions to return status)

Return when arguments are passed only in IPC_DATA.

Table 382 On successful execution

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	0xF = ERROR	Does not return any status on successful execution. The function that is being executed should return a meaningful status.

Table 383 On error

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:28]	0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	Error code	See System call status for details.

Table 384 Return when arguments are passed in SRAM_SCRATCH

Address	Return value	Description
IPC_DATA0 Register		
Bits [31:0]	0xA0000000	Success status on completion of execution

35.4.27 WriteRow

This API is used to program flash. The user needs to provide data to be loaded and flash address to be programmed.

This API can be called only on SFlash.

The API is allowed only in single bank mode. When called in dual bank mode will return STATUS_INVALID_BANK_MODE.

All operations performed are blocking CM0+ & IPC used to invoke the call. This API returns an error status when called during an active embedded operation.

The API returns an invalid address error status if called on wounded flash.

The API returns fail status if the user does not have write access to flash according to SMPU settings.

This API can also be called in 'blocking' mode by setting blocking parameter as 1, in which case the API will return only after all flash operation completes. The API will be polling for each of the timer to expire instead of configuring the flash interrupt and splitting up in phases.

This API does not operate on SFlash in protection states other than VIRGIN and NORMAL.

This API can be used to program all of SFlash rows only in VIRGIN state.

This API can be used to program user SFlash rows (row 4 to 7), NORMAL access restriction row (row13) (refer to Table 387 for the encoding scheme details), public key rows (row 50 to 55), and the TOC2 row (row 62) in NORMAL state. When used to program the allowed SFlash rows the API copies the flash high-voltage parameters into a local array of 512 bytes increasing the stack size accordingly. SFlash programming is always CM0+ blocking. For XMC7000, the application protection settings (row 59) are also considered as user row and can be updated using WriteRow API.

To define flash (program/erase) access restrictions, SWPU objects need to be configured in row 59 of SFlash. This specific row in SFlash is updated using the Write Row API. See the BootROM for more details.

When NORMAL access restrictions are requested to be updated in NORMAL state and if new restrictions are wider than the existing ones, the API will return the STATUS_INVALID_ACCESS_RESTRICTION status.

If WriteRow is used to program the NORMAL access restriction row (row13) of SFlash, first disable CM0+ cache before call to WriteRow. This can be done by writing '0' to the FLASHC_CM0_CA_CTL0.CA_EN bit. After the API is executed successfully, CM0+ cache can be again enabled by writing '1' to the FLASHC_CM0_CA_CTL0.CA_EN bit.

All operations performed are blocking CM0+ and IPC used to invoke the call. This API returns an error status when called during an active embedded operation.

***Note**: The "Inject Public Key", "Write Normal Access Restriction", and "Write TOC2" APIs have been removed from the system calls. The user must use the "Write Row" API to update normal access restriction, public key, and TOC2 to the SFlash.*

Table 385 Parameters

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x05	Write row opcode
Bits [23:16]	–	Not used
Bits [15:8]	0x01 - API blocks CM0+ Other - non-blocking	Blocking mode
Bits [7:0]	–	Not used
SRAM_SCRATCH_ADDR + 0x04		
Bits [31:0]	–	Not used
SRAM_SCRATCH_ADDR + 0x08		
Bits [31:0]	–	Flash address to be programmed. This should be provided in 32-bit system address format.
SRAM_SCRATCH_ADDR + 0x0C		
Bits [31:0]	SRAM_SCRATCH_DATA_ADDR	Address of SRAM where data to be programmed is stored
SRAM_SCRATCH_DATA_ADDR		
Bits [31:24]	–	Data byte 3 to be programmed in flash

(table continues...)

Table 385 (continued) Parameters

Address	Value to be written	Description
Bits [23:16]	–	Data byte 2 to be programmed in flash
Bits [15:8]	–	Data byte 1 to be programmed in flash
Bits [7:0]	–	Data byte 0 to be programmed in flash
SRAM_SCRATCH_DATA_ADDR + (n-3)		
Bits [31:24]	–	Data byte n to be programmed in flash
Bits [23:16]	–	Data byte n-1 to be programmed in flash
Bits [15:8]	–	Data byte n-2 to be programmed in flash
Bits [7:0]	–	Data byte n-3 to be programmed in flash

Table 386 Return

Address	Return value	Description
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS/program command ongoing in background 0xF = ERROR	Status code (see System call status for details).
Bits [27:0]	Error code (if any)	See System call status for details. In case of success: 0x0 indicates successful completion of all phases 0x9 indicates successful completion of first phase, program command is ongoing in the background.

Table 387 Access restrictions encoding

Name	Description
bit[1:0] AP_CTL_M0_DISABLE	00 – Enable M0-AP 01 – Disable M0-AP 1x – Permanently Disable M0-AP
bit[3:2] AP_CTL_CM7_0/CM7_1_DISABLE	00 – Enable CM7_0/CM7_1-AP 01 – Disable CM7_0/CM7_1-AP 1x – Permanently disable CM7_0/CM7_1 AP
bit[5:4] AP_CTL_SYS_DISABLE	00 – Enable SYS-AP 01 – Disable SYS-AP 1x – Permanently Disable SYS AP

(table continues...)

Table 387 (continued) Access restrictions encoding

Name	Description
bit[6] SYS_AP_MPU_ENABLE	Indicates that the MPU on the system debug port must be programmed and locked according to the settings in the next field. ***Note**: When this bit is set, SRAM except SRAM0 cannot be accessed via SYS_AP.*
bit[7] DIRECT_EXECUTE_DISABLE	Disables DirectExecute system call functionality
bit[10:8] FLASH_ ALLOWED	This field indicates what portion of the flash main region is accessible through the system debug port. Only a portion of flash starting at the bottom of the area is exposed. Encoding is as follows: 0: Entire region 1: Seven-eighth 2: Three-fourth 3: One-half 4: One-quarter 5: One-eighth 6: One-sixteenth 7: Nothing
bit[13:11] SRAM0_ ALLOWED	This field indicates what portion of the SRAM0 region is accessible through the system debug port. Only a portion of SRAM starting at the bottom of the area is exposed. Encoding is the same as FLASH_ALLOWED.
bit[15:14] WORK_FLASH_ ALLOWED	This field indicates what portion of work flash is accessible through the system access port. Only a portion of work flash starting at the bottom of the area is exposed. Encoding is as follows: 0: Entire region 1: One-half 2: One-quarter 3: Nothing
bit[17:16] SFLASH_ ALLOWED	This field indicates what portion of the flash supervisory region is accessible through the system debug port. Only a portion of SFlash starting at the bottom of the area is exposed. Encoding is as follows: 0: Entire region 1: One-half 2: One-quarter 3: Nothing

(table continues...)

Table 387 (continued) Access restrictions encoding

Name	Description
bit[19:18] MMIO_ ALLOWED	This field indicates what portion of the MMIO region is accessible through the system debug port. Encoding is as follows: 0: All MMIO Registers 1: Only IPC MMIO Registers accessible (system calls) 2,3: No MMIO access

35.4.28 WriteSWPU

Updates the identified SWPU in SRAM if client has appropriate access. The PU ID is based on the storage of SWPU in SFlash. Only one contiguous SWPU indexing in SFlash even though there are two physically separate storage in SFlash.

The MS_ATT field of selected PU defines who can modify the specific PU structure.

The update is allowed only if the PC that is requesting the update is in the PC_MASK of MS_ATT. The update only modifies the fields SL_ATT, MS_ATT, and SL_SIZE.ENABLE. For a successful update, the other fields SL_ADDR and SL_SIZE.REGION_SIZE should match what is stored in that entry. The safe way to update is to first read the entry, modify, and write it back.

Table 388 Parameters

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x2D	Write SWPU opcode
Bits [23:20]	0: Update SWPU 1: Enable SWPU Other: Disable SWPU	Control
Bits [19:16]	1: eFuse Write 2: eFuse Read Other: Flash Write	PU type
Bits [15:8]	Structure ID to be read. Indexed from 0	PU ID
Bits [7:0]	–	Not used
SRAM_SCRATCH_ADDR + 0x04		
Bits [31:0]	SRAM_DATA_ADDRESS	–

Table 389 Return

Address	Return value	Description
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS/Program command ongoing in background 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	–	In case of fail, error code (see SROM API status codes)
SRAM_DATA_ADDRESS		
Bits [31:0]	SL_OFFSET/SL_ADDRESS	Read only
SRAM_DATA_ADDRESS + 0x4		
Bits [31:0]	SL_SIZE	Read only
SRAM_DATA_ADDRESS + 0x8		
Bits [31:0]	SL_ATT	Used only when control is '0'
SRAM_DATA_ADDRESS + 0xC		
Bits [31:0]	MS_ATT	Used only when control is '0'

35.4.29 Debug power up down

The debug power up down function is used for handling the power transitions of CM7_0/1 power domains to properly connect/disconnect debug probe to/from the device. The system call does not switch off the CM7_0_PWR_CTL. The function first waits until the RegHC power-up is complete. Then, it sets the PWR_MODE for CM7 power domain to ENABLED. When the CM7 power domain is ON (PWR_DONE = 1), the function restores the remembered PWR_MODE. This step is realized for CM7_0 and CM7_1 cores. The function temporarily enables CM7 power because the CM7 power FSM runs on the CM7 clock, which may be OFF.

Table 390 Arguments if IPC_STRUCT.DATA[0] = 1

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:24]	0x12	Opcode
Bit [1]	0: Power down 1: Power up	–
Bit [0]	0x1	Indicates that all the arguments are passed in the IPC_DATA0 Register.

Table 391 Arguments if IPC_STRUCT.DATA[0] = 0

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.

(table continues...)

Table 391 (continued) Arguments if IPC_STRUCT.DATA[0] = 0

Address	Value to be written	Description
SRAM_SCRATCH_ADDR	–	–
Bits [31:24]	0x12	Opcode
Bits [23:2]	Not used	–
Bit [1]	0: Power down 1: Power up	–
Bit [0]	Not used	–

Table 392 Return if IPC_STRUCT.DATA[0] = 1

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	Error code	See System call status for details.

Table 393 Return if IPC_STRUCT.DATA[0] = 0

Address	Value to be written	Description
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	Error code	See System call status for details.

35.4.30 LoadRegulatorTrims

This API is used to adapt the output voltage for internal regulators during handover. This API must be called every time a load transition requires switching between external and internal regulators, except when using the SwitchOverRegulators API with a blocking call.

***Note**: Executing this API can cause a PERI_MS_VIO_0 fault (CM0 + peripheral master interface PPU violation). Ignore the PERI_MS_VIO_0 fault with violation address = 0x40262064 after executing the API. See the specific datasheet for fault details.*

Table 394 Arguments if IPC_STRUCT.DATA[0] = 1

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:24]	0x16	Opcode

(table continues...)

Table 394 (continued) Arguments if IPC_STRUCT.DATA[0] = 1

Address	Value to be written	Description
Bits [3:2]	0 – Force trim setting. The syscall will ignore regulator configuration and will set requested trims 1 – Deep sleep entry use case 2. Deep sleep exit use case 3. Reset recovery use case	Use case
Bit [1]	0 - Internal regulator (LDO) 1 - REGHC	Operating mode
Bit [0]	0x1	Indicates that all the arguments are passed in IPC_DATA0 Register.

Table 395 Arguments if IPC_STRUCT.DATA[0] = 0

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x16	–
Bits [3:2]	0 – Force trim setting. The syscall will ignore regulator configuration and will set requested trims 1 – Deep sleep entry use case 2. Deep sleep exit use case 3. Reset recovery use case	Use case
Bit [1]	0 - Internal regulator (LDO) 1 - REGHC	Operating mode
Bit [0]	Not used	–

Table 396 Return if IPC_STRUCT.DATA[0] = 1

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	Error code	See System call status for details.

Table 397 Return if IPC_STRUCT.DATA[0] = 0

Address	Value to be written	Description
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details).
Bits [23:0]	Error code	See System call status for details.

35.4.31 OpenRMA

This API enables full access to the device in the RMA life-cycle stage upon successful execution. The API returns the 0xF00000A9 failure code if there are any active embedded Flash operations. Users can trigger this API with DAP as Master after transitioning the device to RMA. In the RMA life-cycle stage, before successful OpenRMA execution, DAP will only have access via SYSTEM AP to IPC MMIOs and one-sixteenth of SRAM0. Only OpenRMA system call is allowed before successful OpenRMA execution. When using the OpenRMA API, parameters such as certificate and digital signature must be placed as follows:

- Devices with SRAM0 size larger than 64 KB: the parameters must be placed from [SRAM0 start address + 4 KB] to [SRAM0 start address + 1/16 of SRAM0 size].
- Devices with SRAM0 of 64 KB or less: the parameters must be placed within 600 bytes from [SRAM0 start address + 2 KB]. Certificate and signature address are 24 bytes, and digital signature is 512 bytes (for example, RSA-4K).

Table 398 Parameters

Address	Value to be written	Description
IPC_DATA0 Register		
Bits [31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
SRAM_SCRATCH_ADDR		
Bits [31:24]	0x29	OpenRMA opcode
Bits [23:0]	–	Not used
SRAM_SCRATCH_ADDR + 0x4		
Bits [31:0]	–	Object size in bytes including itself. It should always be 20 bytes.
SRAM_SCRATCH_ADDR + 0x8		
Bits [31:0]	0x120029F0	Command ID
SRAM_SCRATCH_ADDR + 0xC		
Bits [31:0]	–	Unique ID word 0
SRAM_SCRATCH_ADDR + 0x10		
Bits [31:0]	–	Unique ID word 1
SRAM_SCRATCH_ADDR + 0x14		
Bits [31:0]	–	Unique ID word 2 (3 bytes)

(table continues...)

Table 398 (continued) Parameters

Address	Value to be written	Description
SRAM_SCRATCH_ADDR + 0x18		
Bits [31:0]	–	SRAM address where signature is stored (4 bytes)

Table 399 Return

Address	Return value	Description
SRAM_SCRATCH_ADDR		
Bits [31:28]	0xA = SUCCESS 0xF = ERROR	Status code (see System call status for details)
Bits [23:0]	Error if any	In case of fail, error code (see SROM API status codes)

35.5 System call status

At the end of every system call, a status code is written over the arguments in the IPC data structure or the SRAM address pointed by the IPC location. A success status is 0xAXXXXXXX, where X indicates don't care values or return data for system calls that return a value. A failure status is indicated by 0xF00000XX, where XX is the failure code.

If any address of SRAM_SCRATCH is protected, a failure status is indicated by 0xF00000F1.

Table 400 System call status

Status code	Description
0xAXXXXXXX	Success – The X denotes a don't care value, which has a value of ‘0’ returned by the SROM
0xA0000009	Command in progress
0xF0000001	Invalid protection state – This API is not available in current protection state
0xF0000002	Invalid eFuse address
0xF0000004	Wrong or out-of-bound flash address
0xF0000005	FLASH or eFuse bytes are read/write protected via protection units
0xF0000006	Client did not use its reserved IPC structure for invoking system call
0xF0000008	Returned by all APIs when client does not have access to the region it is using to pass argument
0xF0000009	Command in progress. The code begins with “F” from fail. To be replaced by the next code in the future.
0xF000000A	Checksum of flash resulted in non-zero
0xF000000B	The opcode is not a valid API opcode
0xF000000E	Invalid address range
0xF000000F	Invalid arguments passed to the API
0xF0000010	Boot flash authentication failed
0xF0000011	Indicates that TEST_KEY_DFT_EN was set during boot up
0xF0000012	Indicates that TST_KEY_SAFE_MODE was set during boot up

(table continues...)

Table 400 (continued) System call status

Status code	Description
0xF0000013	Invalid arguments location
0xF0000015	Invalid trims length
0xF0000016	Invalid HASH object
0xF0000017	Number of zeros in the HASH computed by ROM boot and number of zeros stored in eFuse do not match
0xF0000018	Invalid table of contents 1's CRC
0xF000001A	Returned during secure boot if SFlash bank 1 authentication check fails
0xF0000020	Invalid table of contents 2's CRC
0xF0000022	Returned when flash embedded operations are invoked during margin mode operation
0xF0000080	Flash trim hunt failed for SONOS. For eCT magic number was not found in TOC1
0xF0000091	Program is called on a sector that is suspended from erase
0xF0000092	EraseResume is called when no sector is suspended from erase
0xF0000095	The requested system call is not approved by TEE
0xF00000A0	FUR download fails with POR_NATIVE = 1
0xF00000A1	FUR download fails due to ECC error
0xF00000A2	IRAM download fails due to ECC error
0xF00000A3	8051 software download fails due to ECC error
0xF00000A4	ProgramRow is invoked on non-erased cells or blank check fails
0xF00000A5	EraseSuspend when called with no ongoing erase operation
0xF00000A6	ProgramRow when active erase operation is going on
0xF00000A7	Embedded operation fails
0xF00000A8	Invalid program width option is provided
0xF00000A9	WriteRow/ProgramRow/ProgramWorkFlash when invoked during an active embedded operation
0xF00000AA	Returned by FLASH program/erase APIs when writes are disabled in safety register. To avoid this error, ensure that 'MainFlashWriteEnable' bit in the FLASHC_MAIN_FLASH_SAFETY register and/or the 'WorkFlashWriteEnable' bit in the FLASHC_WORK_FLASH_SAFETY register is set to '1'.
0xF00000AB	Returned by WriteRow when invoked in dual bank mode
0xF00000B1	Returned when WriteNormalAccessRestrict is called to restrict less
0xF00000B2	Returned when WriteRow is called on invalid SFlash rows in NORMAL state
0xF00000B3	Invalid unique ID is passed during RMA
0xF00000B4	Invalid signature is found during RMA
0xF00000B5	Invalid FACTORY_HASH
0xF00000B8	Returned when more than 15 HASH objects are indicated in TOC1

(table continues...)

Table 400 (continued) System call status

Status code	Description
0xF00000B9	Returned when more than 15 HASH objects are indicated in TOC2
0xF00000BA	Returned by TransitionRMA and OpenRMA when public key structure is invalid
0xF00000BC	Returned during boot when SWPU in SFlash is more than expected
0xF00000BD	Returned during boot when SWPU in SFlash is more than expected
0xF00000BE	Returned during boot when SWPU in SFlash is more than expected
0xF00000BF	Returned during boot when SWPU in SFlash is more than expected
0xF00000C0	Returned during boot when SWPU in SFlash is more than expected
0xF00000C1	Returned during boot when SWPU in SFlash is more than expected
0xF00000C2	Returned by Read or WriteSWPU API when invalid ID is passed
0xF00000C3	Returned by WriteSWPU API when client does not have access to update SWPU
0xF00000C4	Returned by WriteSWPU API when client does not provide matching SL_ADDR and SL_SIZE
0xF00000C5	Returned by ReadSWPU API if ECC error occurred during SRAM read operation
0xF00000C6	Returned by read and WriteSWPU API if the ID'd PU was rejected during boot due to overlap or out-of-order region
0xF00000C7	Returned by read and WriteSWPU APIs if there was a pending ECC error before performing SWPU operations
0xF00000C8	Returned during boot if valid life-cycle fuse combinations are not read from eFuse
0xF00000CB	Returned by BlowFuseBit API when read value from programmed fuse is ‘0’
0xF00000CF	User has provided arguments in protected region
0xF00000D0	Address pointer fetched from TOC/patched syscall table is not in SFlash
0xF00000D1	The bootrow is not zero in VIRGIN
0xF00000D2	SRAM BIHR repair operation fails
0xF00000D3	SRAM repair fuse redundancy check fails
0xF00000D5	Returned when SFlash markers are corrupted during boot
0xF00000D6	Returned by WriteRow when marker overflows by 2^32 times
0xF00000D7	Returned by SoftReset API when CM7 reset is requested with CM7 not being in Deep sleep mode.
0xF00000D8	Invalid life cycle
0xF00000E0	REGHC is configured for Manual mode
0xF00000E1	REGHC is currently in transition
0xF00000E2	REGHC is already enabled
0xF00000E3	Regulator is not configured with ConfigureRegulator()
0xF00000E4	Returned by SwitchOverRegulators() when the syscall is called with a different OpMode parameter than ConfigureRegulator()
0xF00000F0	HardFault occurs during bootup

(table continues...)

Table 400 (continued) System call status

Status code	Description
0xF00000F1	HardFault occurs in context of system calls (SRAM_SCRACH is write protected for PC1, and so on)
0xF4000000	Invalid programmable PPU access
0xF5000000	Invalid fixed PPU access
0xF6000019	Returned when bootrow fuse and MMIO do not match
0xF6000029	Key in the bootrow mismatch
0xF6000039	Returned when trim and trim inverse in bootrow are not equal
0xF6000049	Returned when life-cycle fuses fail its redundancy check
0xF6000059	Returned when invalid life-cycle fuse combinations are blown
0xF1000000	Hash on SFlash trims failed. The computed hash is OR'd with this status
0xF2000000	CRC8 of the eFuse group failed.The computed CRC is OR'd with this status
0xF3000000	Returned during boot in IPC_STRUCT0.DATA1 if fault structure 0 valid bit is set. The LSBs will hold the fault ID information.

35.6 eFuse memory

The eFuse memory consists of a set of eFuse bits. When an eFuse bit is programmed, or “blown”, its value can never be changed. Some of the eFuse bits are used to store various unchanging device parameters, including critical device factory trim settings, device life-cycle stages, DAP security settings, and encryption keys. Other eFuse bits are available for customer use.

35.6.1 Features

eFuses have the following features:

- A total of 1024 eFuse bits.
- The eFuse bits are programmed one at a time, in a manufacturing environment. The eFuse bits cannot be programmed in the field.
- Multiple eFuses can be read at the bit or byte level through an SROM call. An unblown eFuse reads as logic 0 and a blown eFuse reads as logic 1. There are no hardware connections from eFuse bits to elsewhere in the device.
- SROM system calls are available to program and read eFuses.

35.6.2 Customer eFuses

eFuses have bits available for customer use. They can be programmed in the NORMAL life cycle stage via CM0+ and CM7/DAP, and in the SECURE protection state via CM0+ and CM7.

Offset	Width	Name
0x068	32	Customer data

To program customer data, the Blow Fuse Bit system call must be called; the logic for calculation is:

macro Address = AddressOffset% EFUSE_NR

Byte Address = AddressOffset/EFUSE_NR

Where EFUSE_NR = number of eFuse macros (that is, number of columns) available for a product.

36 Flash boot

Flash boot is the firmware that resides in SFlash, runs on the security processor (Arm® Cortex®-M0+), and is executed after ROM boot has completed the basic hardware configuration and trim.

The purposes of flash boot are as follows:

- Initial configuration for a hardware subset
- Security configuration that must be done at programming context (PC) = 0
- Initialization of a debugger pin and the debug access port (DAP) subsystem
- Authentication for secure application
- Launching the application in a boot chain

Flash boot performs the following tasks:

- Configures the hardware that is not part of ROM boot
- Validates TOC2
- Sets up the CM0+ and peripheral clocks based on TOC2_FLAGS
- Enables system calls
- Configures SWD and JTAG pins and enables DAP
- Configures and enables a listen window for DAP
- Validates user applications structure
- Validates an RSA public key structure
- Authenticates secure applications by verifying their digital signature
- Sets a PC value – either PC = 2 or PC = 4
- Launches a bootloader for end-of-line programming, controlled through TOC2_FLAGS
- Launches a user application if there are no errors
- Enters NORMAL_DEAD or DEAD protection mode if an error occurs

36.1 Features

- Secure boot support
  - Digital signature verification by RSASSA-PKCS1-v1.5 with SHA-256 and RSA[8] up to 4096 bits
  - Public key in SFlash for RSA up to 4096
  - Control enabling DAP by access restrictions (AR)
- User configuration through TOC2
  - The next launched application's address and format
  - A listen window to facilitate debugging
  - Boot time and power consumption
  - Authentication options for secure applications
- Embedded CAN bootloader to replace SWD or JTAG for factory programming
  - CAN at 100 or 500 kbps

[8] For RSA2K/3K/4K support, see the device-specific datasheet (under the section Part Number/Ordering Code Nomenclature, Hardware option).

36.2 Using flash boot

36.2.1 Flash boot shared functions

Flash boot contains a few functions that may be executed from user code. Table 401 provides memory locations for the function pointers and a short description for each function.

Table 401 Flash boot functions

Memory location	Function name	Comment
0x1700_2040	Cy_FB_VerifyApplication	Validates the application signature with RSASSA-PKCS1-v1.5 (up to 4096 bits)
0x1700_2044	Cy_FB_IsValidKey	Validates the public key

36.2.1.1 Cy_FB_VerifyApplication

- Function description

  Used in flash boot for authentication of the next application.

  Can be used by the other code to authenticate RSASSA-PKCS1-v1.5 for any data; it need not be a signed application image.
- Parameters

  uint32_t address: The start address of the data image to be authenticated.

  uint32_t length: The length of the data image.

  uint32_t signature: The start address of the signature for the data image.

  cy_stc_crypto_rsa_pub_key_t* publicKey: The pointer to a public key structure.
- Return value

  uint32_t

  0 - digital signature is invalid

  1 - digital signature is valid

36.2.1.2 Cy_FB_IsValidKey

- Function description

  Checks whether the public key structure is valid.

  ***Note**: It may be used only for a public key that is referenced in TOC2.*
- Parameters

  uint32_t address: The address of TOC2.

  cy_stc_crypto_rsa_pub_key_t *publicKey: The pointer to public key structure.
- Return value

  uint32_t:

  0 - public key is invalid

  1 - public key is valid

If any of the following steps results in false, the function returns 0, otherwise it returns 1:

1. Check if the address of a public key in TOC2 points to a valid location in the internal memory
2. Check if the address of the Object Size member of a public key object is a valid location in the internal memory

3. Check if the Object Size value is within the allowed range [MIN, MAX]. MIN and MAX depend on the signature scheme and are implementation details
4. Check if the address of the last word in the public key object points to a valid location within the internal memory
5. Check if the Signature Scheme value of the public key object is valid. For Signature Scheme 0: RSASSA-PKCS1-v1.5 with RSA up to 4096 and SHA2-256
6. Check if the RSA public key exponent size is less than or equal to 32 × 8 bits. Check if the RSA public key module size is less than or equal to 256 × 8 bits
7. Check if the values of RSA public key module and exponent members of the public key structure are inside the memory region for the public key object
8. Validate the RSA optional coefficients (barretCoefPtr, inverseModuloPtr, rBarPtr). Their values should either be zero or the addresses inside the memory range of the public key object

36.2.2 Using a bootloader

36.2.2.1 Bootloader host requirements

- Bootloader packet structure

  Figure 364 shows the structure of communication packets sent from the host to the MCU.

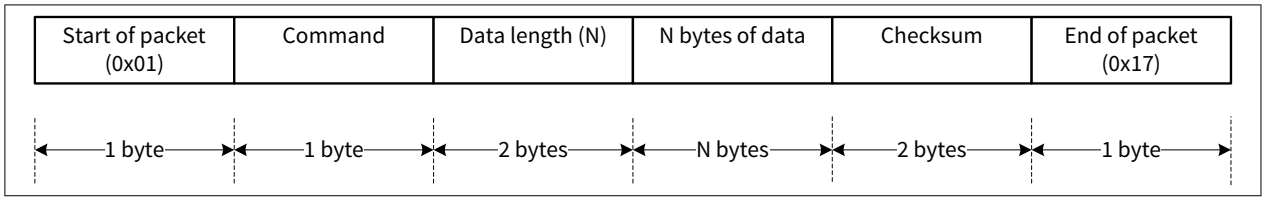


Figure 364 Bootloader command packet structure

  Figure 365 shows the structure of response packets sent from the MCU to the bootloader host.

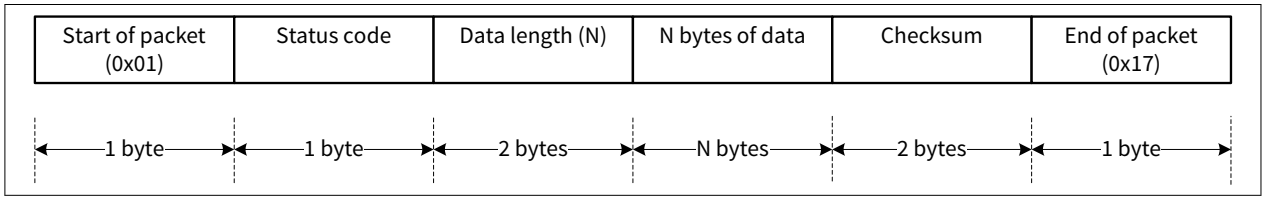


Figure 365 Bootloader response packet structure

All multi-byte fields are little endian.

Bootloader packet length is limited to four CAN messages, each with 8 bytes of data.

Each CAN message may contain up to 8 bytes of user data, which hold bootloader command data.

Bootloader commands

- Enter bootloader

  Begins a bootload operation. All the other commands except Exit Bootloader are ignored until this command is received. Responds with device information and Bootloader SDK version.

  Input

  - Command byte: 0x38
  - Data bytes:

    4 bytes: Product ID. Internal bootloader requires Product ID = 0x01020304

Output

- Status/error codes:
  Success
  Error command
  Error data, used for product ID mismatch
  Error length
  Error checksum
- Data bytes:
  4 bytes: Device JTAG ID
  1 byte: Device revision
  3 bytes: Bootloader SDK version

- Sync bootloader

  Resets the bootloader to a known state, making it ready to accept a new command. Any buffered data is discarded. This command is needed only if the bootloader and the host are out of sync with each other.

  Input

  - Command byte: 0x35
  - Data bytes: N/A

  Output

  N/A

  This command is not acknowledged

- Exit bootloader

  Exits from the bootloader and ends the bootload operation

  After this command is received, the internal bootloader stops reading a bootloading communication, verifies the bootloadable application image, and launches it if it is valid.

  Input

  - Command byte: 0x3B
  - Data bytes: N/A

  Output

  N/A

  This command is not acknowledged

- Send data

  Transfers a block of data to the bootloader. This data is buffered in anticipation of a Program Data command. If a sequence of multiple Send Data commands are sent, the data is appended to the previous block. This command is used to break up large data transfers into smaller pieces, to prevent channel starvation in some communication protocols.

  Input

  - Command byte: 0x37
  - Data bytes:
    n bytes: Data to be appended to the buffer
    CAN allows up to 25 bytes of data

  Output

  - Status/error Codes:
    Success
    Error command
    Error data
    Error length

Error checksum
- Data bytes: N/A

- Send data without response

  Same as the Send Data command, except that no response is generated by the bootloader. This reduces bootloading time.

  Input
  - Command byte: 0x47
  - Data bytes:

    n bytes: Data to be appended to the buffer

  Output

  N/A

  This command is not acknowledged.
- Program data

  Writes data to one row of device internal flash or page of external nonvolatile memory (NVM). May follow a series of send data or send data without response commands.

  Input
  - Command byte: 0x49
  - Data bytes:

    4 bytes: Address. Must be within the correct memory address space, and 32-bit aligned

    4 bytes: CRC-32C of the entire data buffer to be written

    ***Note**: The buffer includes data that is already appended to it with send data or send data without response commands that precede program data.*

    n bytes: Data to write into the flash row or external NVM page.

  Output
  - Status/error codes:

    Success

    Error command

    Error data

    Error length

    Error checksum

    Error flash row

    Error flash row access
  - Data bytes: N/A
- Verify application

  Reports whether the checksum for the bootloadable application image is valid.

  Input
  - Command byte: 0x31
  - Data bytes:

    1 byte: App ID of the application to be verified. Must be the same value as in set application metadata command.

  Output
  - Status/error codes:

    Success - returned when either the application is valid

    Error command

    Error data

Error length
Error checksum
Error flash row access

- Data bytes:
  1 byte: 1/0 for application is valid or not valid

- Set application metadata
  This command is used to set a given application's metadata.
  It must be the second bootloader command, which the bootloader host delivers to the MCU, the first one is Enter Bootloader.

  Input

  - Command byte: 0x4C
  - Data bytes:
    1 byte: App ID
    4 bytes: Bootloadable application start address
    4 bytes: Bootloadable application size in bytes

  Output

  - Status/error codes:
    Success
    Error command
    Error length
    Error data
    Error checksum
    Error flash row access
  - Data bytes: N/A

Data constraints

App ID may have the following values:

Bootloadable application start address must be within a valid RAM memory length - [RAM_START + 3 KB, RAM_END – 6 KB].

Bootloadable application length must be a value for which the bootloadable application image fits into a RAM address range [RAM_START + 3 KB, RAM_END – 6 KB].

36.2.2.2 Using CAN

Bootloader polls for enter bootloader command on CAN pins as follows:

1. Bootloader polls for CAN messages at 100 kbps; if no valid CAN message with enter bootloader command is received during 10 ms, it goes to (2). If a valid command is received, bootloader continues using CAN at 100 kbps for the next bootloader commands.
2. Bootloader polls CAN at 500 kbps for a duration of 10 ms. If no valid enter bootloader command is received it goes to (3).
3. If bootloading has started on CAN, but later communication has stopped, the bootloader uses a timer, which detects that there was no bootloader communication for two seconds and resets the communication configuration. It then goes to (1), (2) depending on the communication channel for which the bootloading has failed.
4. If there are no valid Enter Bootloader command during 300 seconds, the bootloader stops. The device goes into sleep power mode.

The following figures show a few examples of CAN bootloading communication.

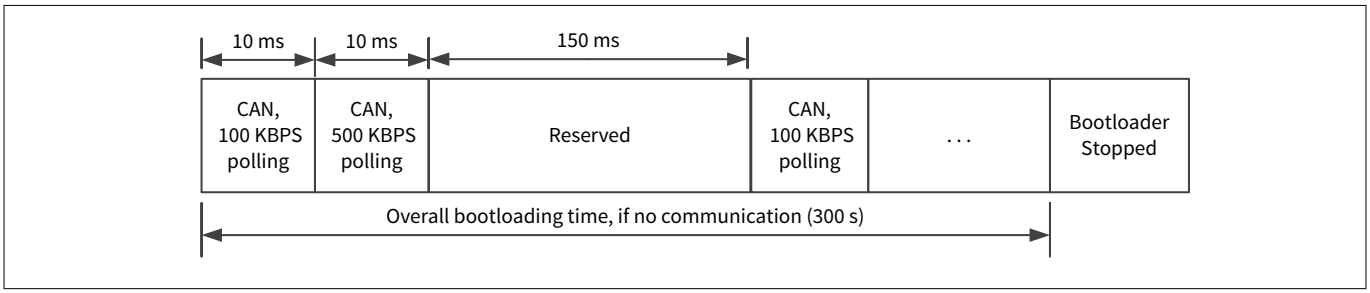


Figure 366 Polling CAN with no bootloader commands

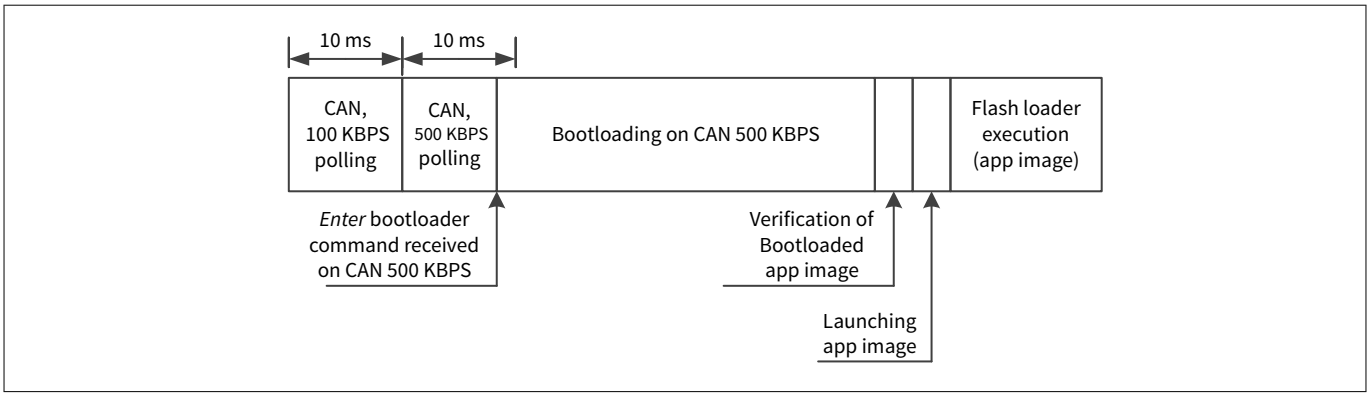


Figure 367 Successful bootloading on CAN 500 kbps

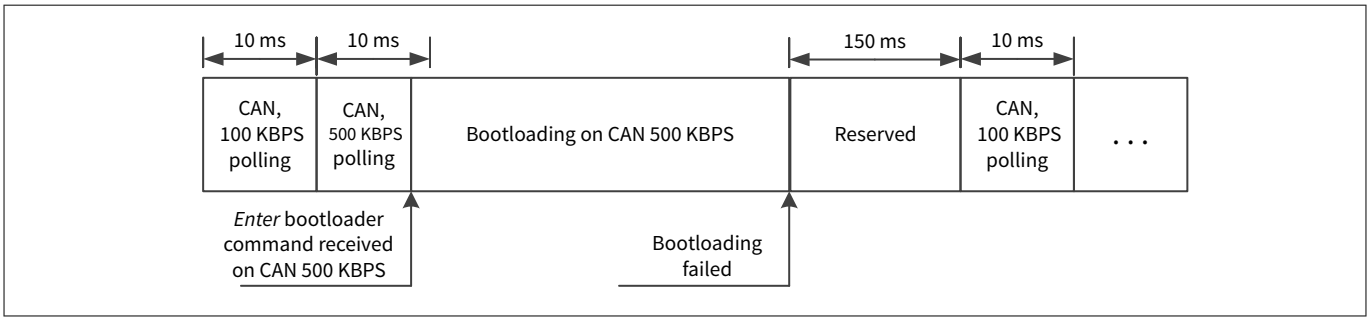


Figure 368 An example of a failed bootloading

36.2.2.3 CAN driver configuration

See the device datasheet for CAN configuration details.

- CAN driver limitations

  The CAN specification states the clock accuracy should be at most ±0.5% at 500 kbps. The internal generator (IMO) for XMC7000 devices does not meet this accuracy and hence cannot be used. However, the CAN block may use SJW (ReSynchronisation jump width) to adjust CAN clock to the baud rate, which allows ±1.0% clock tolerance.

  It is recommended to use a single point-to-point connection and have the wire length within the allowed range for CAN 500 kbps to have a stable communication at ±1.0% clock frequency tolerance.

36.3 Flash boot internals

36.3.1 Definitions

- Firmware image

  A specific format for a firmware module stored in the internal memory. See Application formats for more details.
- ROM boot

  ROM code stored at the device address range that starts at address 0x0000_0000. The first code is executed when the device is powered on. This is the first phase of the boot process.
- Flash boot

  A firmware image stored in SFlash that provides code for the second phase of the boot process.
- “Secure Boot”

  “Secure Boot” is the term used to include the entire secure chain of trust boot process. It includes ROM boot, flash boot, and optionally secure image.
- DAP

  Debug access port
- SFlash

  Supervisory flash is a dedicated flash region used by Infineon to store manufacturing information, hardware trim and wounding information, special user sections, TOC, and code for the second phase of the boot process and flash boot.
- TOC, TOC1, and TOC2

  Table of contents. This table is broken up into two parts. The first part (TOC1) includes addresses of items frozen in the factory, such as items that are included in the FACTORY_HASH calculation and cannot be changed by the user. The second part (TOC2) includes addresses of the user application, public key, and other user configurable items that are used by secure boot. Entities from TOC1 and TOC2 are used to calculate SECURE_HASH.
- Secure application

  An application in Infineon secure application format (CySAF). This application contains digital signature and may be authenticated using RSASSA-PKCS1-v1.5 with RSA up to 4096 and SHA-256.
- FACTORY_HASH

  128 most significant bits of the SHA-256 hash value computed to authenticate objects frozen in the factory.
- Private key

  A private key for RSA up to 4096 to sign the digital signature of the secure application.
- Public key

  A public key for RSA up to 4096 to verify the digital signature of the secure application.
- RSA

  An asymmetric crypto algorithm by Rivest-Shamil-Adleman.
- RSASSA-PKCS1-v1.5

  A digital data authentication algorithm based on RSA and hashing functions.
- SECURE_HASH

  128 most significant bits of the SHA-256 hash value used to authenticate flash boot and public key in SECURE and SECURE_WITH_DEBUG life-cycle stages. When creating SECURE_HASH, factory frozen objects are authenticated using FACTORY_HASH. This makes sure that the flash boot authenticated by

SECURE_HASH later is the same as the one created in the factory. SECURE_HASH is computed just before transition to SECURE, so public key needs to be known only then; the OEM provides the public key.

- SHA2, SHA3, SHA-256, and SHAKE-128

  SHA-based secure hash functions.

36.3.2 SFlash address mapping

The entire flash boot is located in SFlash. It starts at address 0x17001C00 and ends at 0x170063FF. You cannot overwrite or change the flash boot. The flash boot version can be read from address 0x17002018, which has an unsigned 32-bit integer value.

The area from 0x17000800 to 0x17000FFF can be used for user applications, and storing keys and other information. In this area, it is possible to write only in the Normal mode.

A public key is located at address 0x17006400. The maximum length is 3072 bytes.

TOC2 is located near the end of SFlash at 0x17007C00. Both public key and TOC2 are available for write in normal mode only.

Two additional areas are available for the user.

- The area from 0x1700_1A00 to 0x1700_1BFF for configuring normal access restrictions and normal dead access restrictions.
- The area from 0x1700_7600 to 0x1700_77FF for configuring protection unit objects.

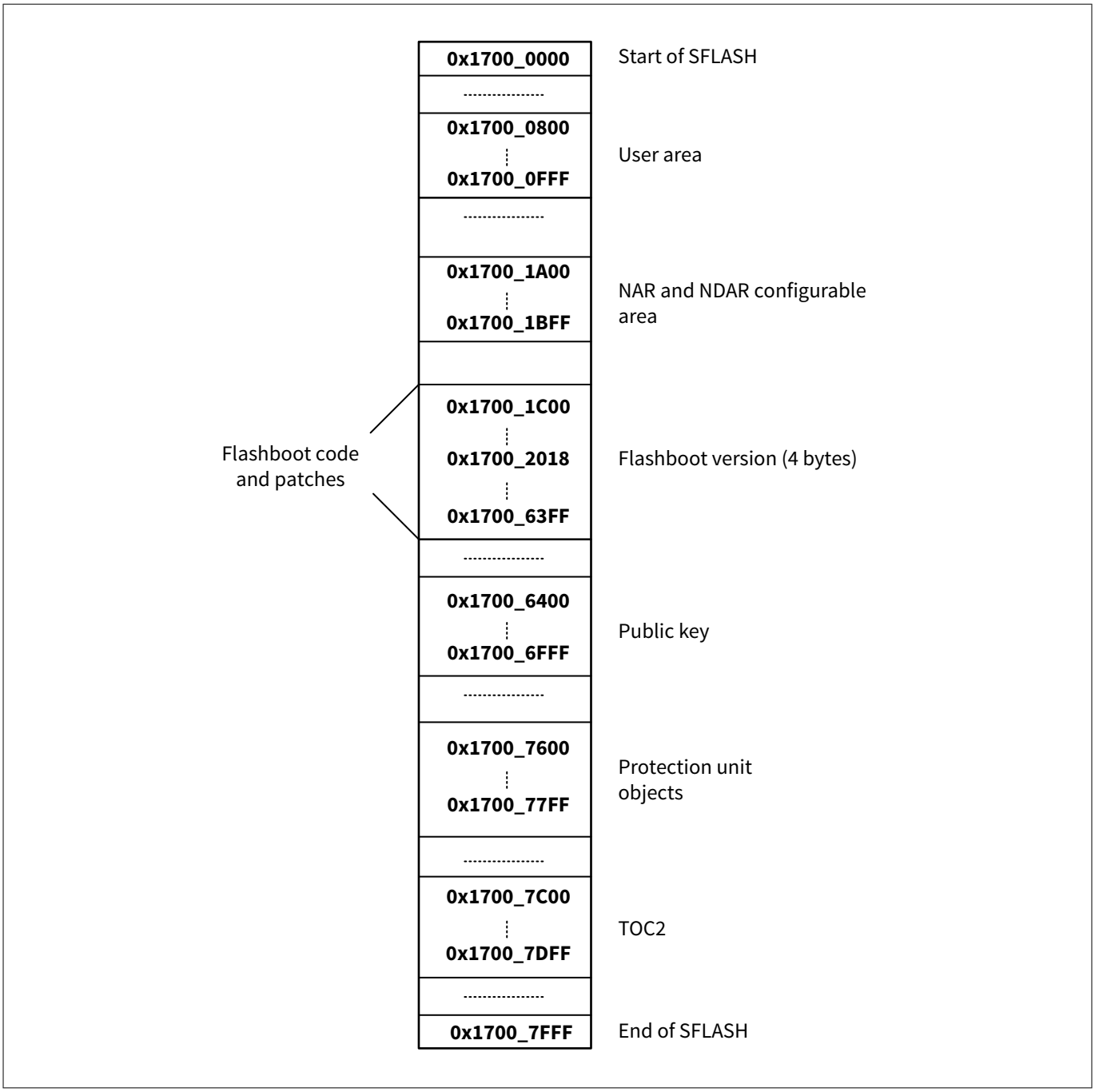
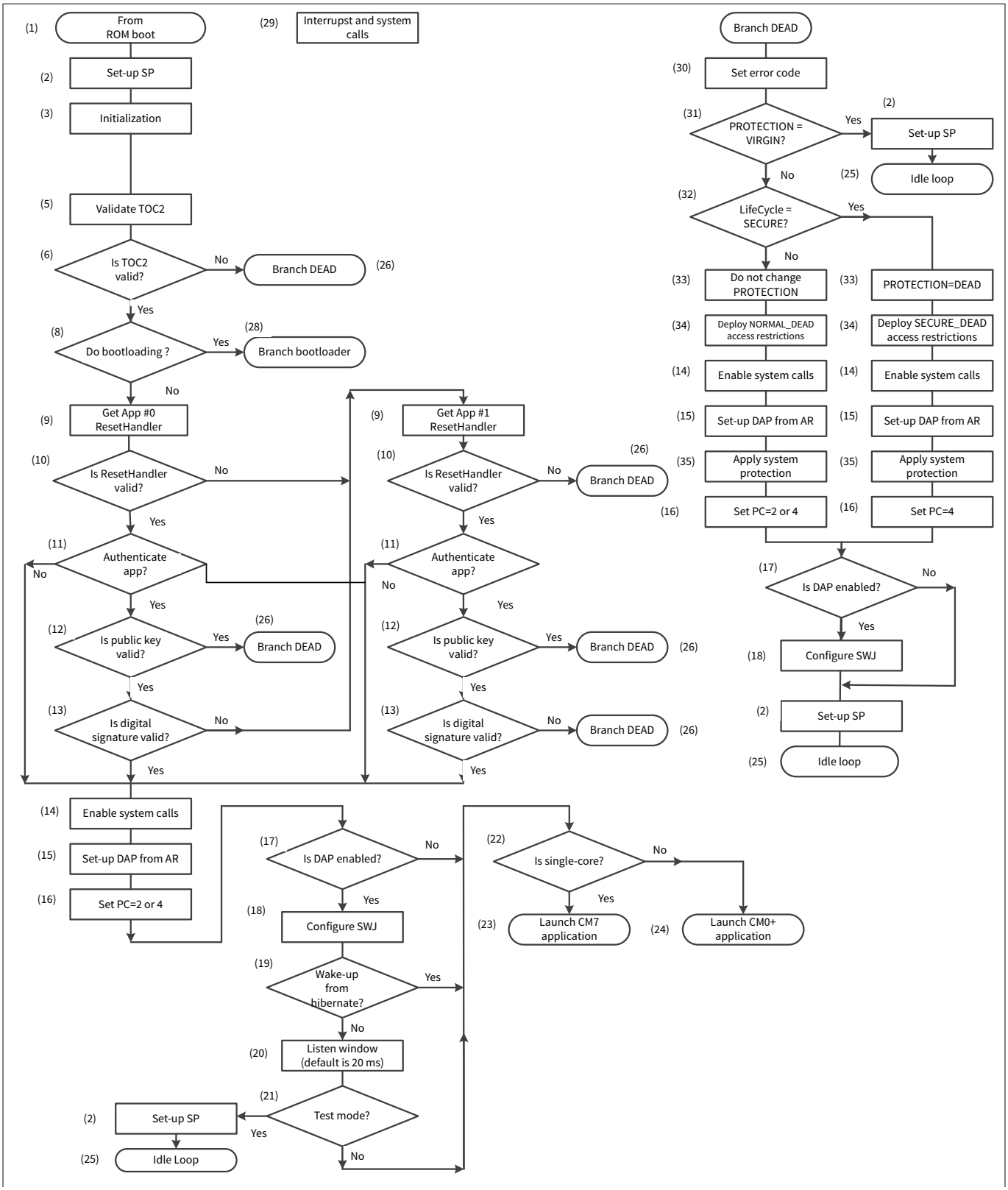


Figure 369 SFlash address mapping

36.3.3 Flash boot flow

Figure 370 shows the flash boot program flow. The entry point to flash boot must be at a fixed offset inside the SFlash block. The ROM boot code will transfer control to flash boot after its tasks are completed and SFlash is validated. Each section of the flow chart is labeled with an index number (n), which is used for reference in the next sections.

36 Flash boot



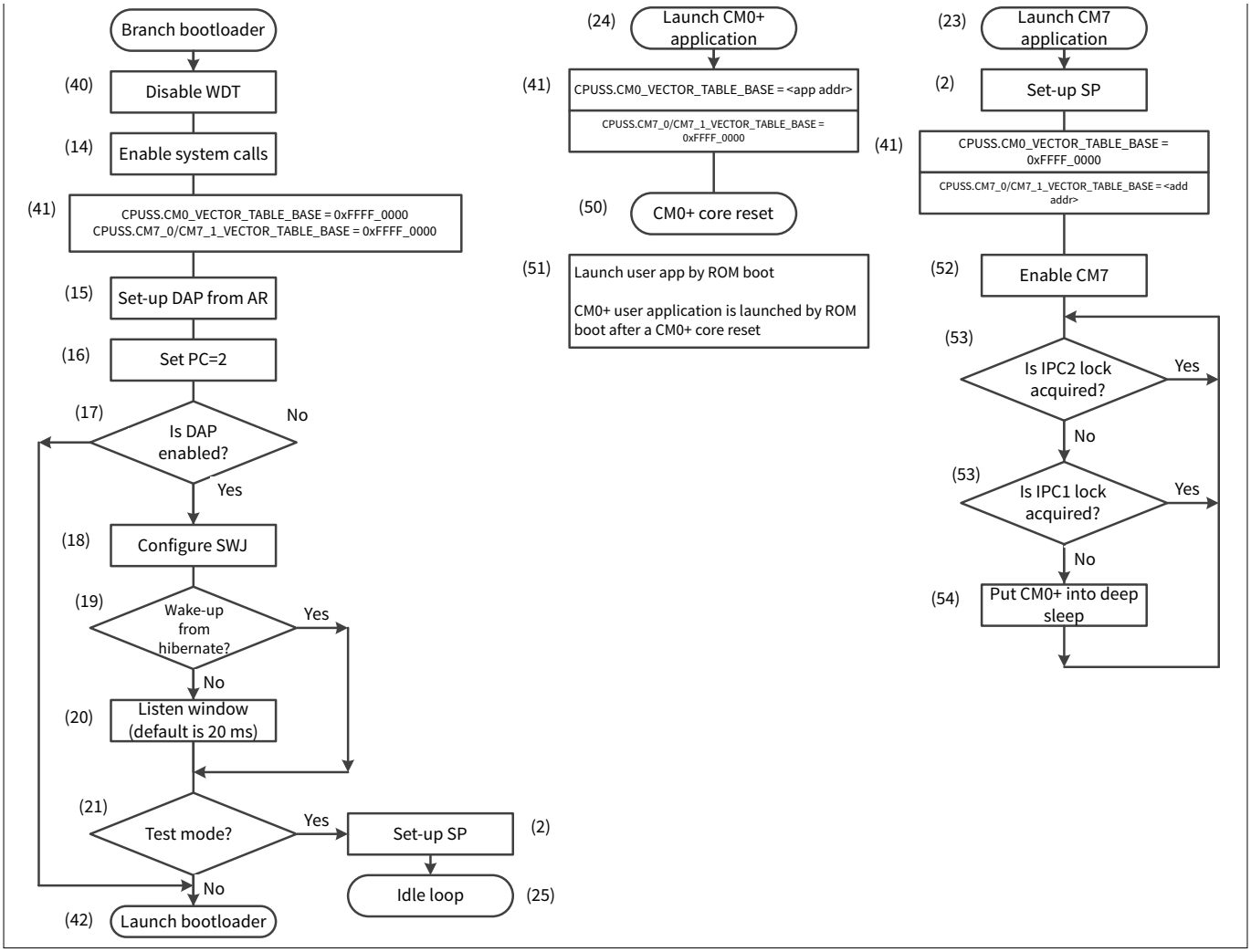


Figure 370 Flash boot flow

36.3.3.1 Entry from ROM boot (1)

At this stage ROM boot has finished its tasks and transfers the execution for CM0+ code to flash boot.

36.3.3.2 Set-up SP (2)

The same flash boot image is programmed in all XMC7000 devices. Within the device family, different devices have different sizes of SRAM. The SP Register value for flash boot must be at the top of user SRAM. Thus, it is impossible to know the SP value at build time.

At the start of flash boot, the SP Register value is ‘0’. Flash boot calculates and sets the value of SP Register at runtime.

36.3.3.3 Initialization (3)

This function executes a hardware-specific initialization code.

All XMC7000 devices support SRAM ECC, and flash boot initializes the SRAM it uses for stack.

Flash boot enables S&H mode for SRSS.PWL_CTL2.BGREF_LPMODE.

During boot, PERI_MS_PPU_FX_PERI_GR2_BOOT fixed PPU is configured as follows: no write access; read access only for all PCs.

36.3.3.4 Validate TOC2 (5)

The current procedure to validate TOC2 is as follows:

- SFLASH_TOC2_OBJECT_SIZE <= 512 for XMC7000
- SFLASH_TOC2_OBJECT_SIZE >= 8
- SFLASH_TOC2_MAGIC_NUMBER == 0x01211220

If all the conditions above are true, the TOC2 state is VALID. For information on the TOC2 structure, refer to TOC2 structure.

36.3.3.5 Is TOC2 valid (6)

TOC2 may be in three states:

- VALID: TOC2 structure and CRC are valid
- ERASED: The first two 32-bit words at the start of TOC2 are equal to the SFlash erase value and protection mode is either VIRGIN or NORMAL.
- For eCT SFlash, the erased value is 0xFFFF_FFFF
- CORRUPTED: When both ERASE and VALID conditions are false

If the TOC2 state is ERASED, then flash boot uses the default values for all the TOC2 elements instead of reading them from TOC2. The following are a list of TOC2 elements for which the default values are used:

- SFLASH_TOC2_FIRST_USER_APP_ADDR is 0x1000_0000 (the start of flash)
- SFLASH_TOC2_FIRST_USER_APP_FORMAT is 0 (basic application format)

The other TOC2 entries are not used when TOC2 state is ERASED.

36.3.3.6 Bootloading (8)

Bootloading triggers in VIRGIN and NORMAL protection modes if the following conditions are met:

- CPUSS.PROTECTION != SECURE
- TOC2 state is either ERASED or VALID and SFLASH.TOC2_FLAGS bit FB_BOOTLOADER_DISABLE is zero.
- The first two 32-bit words at 0x1000_0000 are equal to 0xFFFF_FFFF.

36.3.3.7 Get app #{0, 1} reset handler (9)

This step may be executed when the TOC2 state is either VALID or ERASED (see Bootloading (8)).

If TOC2 state is ERASED and CPUSS.PROTECTION = NORMAL then:

1. Application start address is 0x1000_0000
2. Application format is CyBAF
3. Second application is ignored; thus, if a validation of the first application leads to an error, the second application is not validated and DEAD branch is executed.

Otherwise, TOC2 state is VALID and application parameters are calculated as shown in this section.

Flash boot reads the application start address from the following TOC2 entries:

- SFLASH_TOC2_FIRST_USER_APP_ADDR for App#0
- SFLASH_TOC2_SECOND_USER_APP_ADDR for App#1

The following application formats are stored in TOC2 entries:

- SFLASH_TOC2_FIRST_USER_APP_FORMAT for App #0
- SFLASH_TOCR_SECOND_USER_APP_FORMAT for App #1

The reset handler address inside the application depends on the application format. See Application formats.

36.3.3.8 Valid reset handler (10)

Flash boot checks whether the address of the reset handler for the user application is inside a valid range.

The valid range is the following: SRAM, SFlash, code flash, and work flash.

***Note**: This check is made to prevent the HardFault that would otherwise occur if the reset handler for the user application points to an invalid memory location.*

36.3.3.9 App authentication (11)

Flash boot optionally authenticates a digital signature for the application image based on the value of TOC2_FLAGS bits APP_AUTH_DISABLE.

36.3.3.10 Is public key valid (12)

The public key structure is filled by the user. It must be validated to ensure the correctness of the entries before being used.

The validation is done using Cy_FB_IsValidKey() function described in Cy_FB_IsValidKey.

36.3.3.11 Valid digital signature (13)

The application to be launched by flash boot may be authenticated using a digital signature verification with RSASSA-PKCS1-v1_5 (RSA up to 4096, SHA2-256) see details in RFC3447.

The public key used for this operation may be placed in the User Public Key area of SFlash or in another internal flash; its pointer should be updated in TOC2_SIGNATURE_VERIF_KEY (See Table 406). The format of the key is shown in RSA public key format.

The application to be authenticated should be in the CySAF or Customer Formats. Applications in CyBAF may not be authenticated. The application formats are described in Application formats.

36.3.3.12 Enable system calls (14)

The system calls are enabled in this stage. System calls are functions such as writing to code flash. The function calls are performed via the IPC communication to the CM0+ NMI interrupt. The SROM function EnableSystemCalls() is called to enable these system calls.

36.3.3.13 Set up DAP from AR (15)

Enable or disable DAP based on the current AR.

ROM boot function GetAccessRestrictStruct() is called to determine which APs (access points, one of CM0+ AP, CM7_0/CM7_1, and TC) are enabled. Based on this information the proper values are written to the CPUSS_AP_CTL Register.

***Note**: For a VIRGIN protection mode the DAP setup is performed in the ROM boot; thus Flash boot skips this step.*

36.3.3.14 Set PC (16)

ROM boot and Flash boot are being executed in PC = 0, system calls are executed in PC = 1, all other code must be executed in PC = 2 or higher. Flash boot is responsible for setting PC = 2 before launching a user application or jumping into the idle loop.

Flash boot sets PC = 4 in SECURE_DEAD branch (when the life-cycle stage is SECURE and protection mode is DEAD).

36.3.3.15 Is DAP enabled (17)

For DEAD and Bootloader branches the result of step (17) is TRUE if CPUSS.AP_CTL enables DAP.

For a common branch (which ends by launching a user application), there is an additional check to determine if DAP is enabled. This check reads the TOC2_FLAG bit SWJ_PIN_CTL. If TOC2_FLAGS.SWJ_PINS_CTL is set and CPUSS.AP_CTL enables DAP then the result of step (17) is TRUE.

36.3.3.16 Configure SWJ (18)

Flash boot uses the ConfigureSWJ() ROM boot function if it is implemented for the device family; otherwise, this function is implemented in the Flash boot code base.

This function configures the GPIO pins to work in SWJ mode. All JTAG pins must be configured, except the TRST pin for the devices that have a problem.

36.3.3.17 Wake-up from hibernate (19)

If the reason for a reset was “wake from hibernate”, skip the wait window and test mode check.

36.3.3.18 Listen window (20)

The CPU delays execution for a period of time to allow the debug hardware to acquire the CM0+. The default is 20 ms, but other delay options may be set. If the Listen window is not required, the user may set the Listen window timeout to 0 ms.

This delay allows the debug hardware to acquire the debug interface before any user code is executed; it helps recovering the device in which a user code re-purposes SWJ pins.

36.3.3.19 Test mode (21)

After the listen window delay, the firmware checks if the SRSS_TST_MODE Register has either TEST_MODE or TEST_KEY_DFT_EN bit set. If either bit is set, execution is transferred to an endless loop in SROM. This is done by calling the ROM boot function BusyWaitLoop().

Some programmers use listen window and set a test mode bit to perform either programming or debugging tasks.

36.3.3.20 Is single-core (22)

Detects if a MCU is a single core. A single-core MCU does not allow a user code to be executed on CM0+.

Flash boot determines if MCU is a single-core by reading SFLASH_SINGLE_CORE_WOUND.

36.3.3.21 Launch CM0+ application (24)

The procedure to launch a user application is as follows:

1. Set CPUSS_CM7_VECTOR_TABLE_BASE to 0xFFFF_0000

   ***Note**: For CM7 cores, the CPUSS_CM7_0/CM7_1_VECTOR_TABLE_BASE Register should not be touched by Flash boot.*

2. Set CPUSS_CM0_VECTOR_TABLE_BASE to the start of the user application interrupts vector table

3. Perform a CM0+ core reset
4. After a core-reset is performed a ROM boot is launched (on CM0+)
5. ROM boot checks if CPUSS_PROTECTION != 0, which means ROM boot is launched on CM0+ after a core-reset
6. If (5) is true, ROM boot sets SP and PC Register values from the user interrupt vector table. The address of a user application interrupt vector table is stored at step (1) to CPUSS_CM0_VECTOR_TABLE_BASE
7. When ROM boot sets the PC Register value with the user reset handler address, the user code starts executing

36.3.3.22 Idle loop (25)

The SP Register value is saved to R8 before entering Idle loop. Then SP is updated per step (2) in the Set-up SP (2).

Then CM0+ core is placed into a sleep power mode by calling ROM boot function BusyWaitLoop().

***Note**: Any interrupt (IPC system call or another interrupt source) may wake the device; therefore, the AR and other security settings should be properly configured by the user for each life-cycle stage.*

36.3.3.23 Branch DEAD (8)

Flash boot goes into a DEAD branch if it detects any error. The list of the required errors is provided in Set error code (30).

36.3.3.24 Branch bootloader (28)

If the bootloader feature is enabled for the device family and the bootloader launch condition is triggered, then flash boot launches a bootloader by going into this branch. The implementation may implement this branch either as a function call or as launching an application.

36.3.3.25 Interrupts and system calls (29)

Flash boot should support patching the system calls using the system call patch table. Flash boot may patch a HardFault handler and re-configure the CM0+ interrupts during the startup.

36.3.3.26 Set error code (30)

Flash boot sets an error code into following:

- IPC_STRUCT[3].DATA0 Register for CM7 based devices

Table 402 Error code

Error name	Value	Description
CY_FB_STATUS_SUCCESS	0xA100_0100	Success status value
CY_FB_STATUS_BUSY_WAIT_LOOP	0xA100_0101	Debugger probe acquired the device in test mode. The flash boot to entered a busy wait loop
CY_FB_STATUS_BOOTLOADING	0xA100_0101	Bootloading in progress
CY_FB_STATUS_BTLD_OK	0xA100_0102	Bootloading finished successfully

(table continues...)

Table 402 (continued) Error code

Error name	Value	Description
CY_FB_ERROR_INVALID_APP_SIGN	0xF100_0100	App signature validation failed for the device families where flash boot launches only one application from TOC2. Either app structure or a digital signature is invalid for the device families for which Flash boot may launch either of two apps in TOC2.
CY_FB_ERROR_INVALID_TOC	0xF100_0101	Empty or invalid TOC
CY_FB_ERROR_INVALID_KEY	0xF100_0102	Invalid public key
CY_FB_ERROR_UNREACHABLE	0xF100_0103	Unreachable code
CY_FB_ERROR_TOC_DATA_CLOCK	0xF100_0104	TOC contains invalid CM0+ clock attribute
CY_FB_ERROR_TOC_DATA_DELAY	0xF100_0105	TOC contains invalid listen window delay
CY_FB_ERROR_FLL_CONFIG	0xF100_0106	FLL configuration failed
CY_FB_ERROR_INVALID_APP0_DATA	0xF100_0107	App structure is invalid, for the device families where flash boot may launch only one app from TOC2.
CY_FB_ERROR_CRYPTO	0xF100_0108	Error in Crypto operation
CY_FB_ERROR_INVALID_PARAM	0xF100_0109	Invalid parameter value
CY_FB_ERROR_UNEXPECTED_INTERRUPT	0xF100_010B	Any unexpected interrupt had happened in the Flash boot
CY_FB_ERROR_BOOTLOADER	0xF100_0140	Any bootloader error
CY_FB_ERROR_BOOT_CAN_INIT	0xF100_0143	Bootloader error, CAN initialization failure
CY_FB_ERROR_BOOT_SECURE	0xF100_0144	Bootloader launched while CPUSS.PROTECTION=SECURE

36.3.3.27 PROTECTION = VIRGIN (31)

CPUSS_PROTECTION MMIO Register value is compared to the wished protection mode.

36.3.3.28 LifeCycle = SECURE (32)

A life-cycle stage is stored in eFuse. The life-cycle stage is not the same as protection mode. In this case, SECURE_WITH_DEBUG life-cycle stage is not equal to SECURE life-cycle stage, but for both, the protection mode equals to SECURE.

36.3.3.29 PROTECTION = DEAD (33)

Flash boot sets CPUSS_PROTECTION to DEAD in the DEAD branch only for SECURE life-cycle stage. For SECURE_WITH_DEBUG, NORMAL, and other life-cycle stages flash boot keeps the existing protection mode.

36.3.3.30 Deploy AR (34)

Flash boot deploys the AR that applies to the new protection mode.

NORMAL_DEAD AR are applied when entering DEAD branch from NORMAL, NORMAL_PROVISIONED, or SECURE_WITH_DEBUG life-cycle stages.

SECURE_DEAD AR are applied in the case of entering DEAD branch from SECURE life-cycle stage.

Assess restrictions are applied by calling ROM boot function RestrictAccess().

36.3.3.31 Apply system protection (35)

System protection settings are applied. Usually they are applied by the ROM boot code before entering flash boot. In the case of DEAD branch, the system protection settings may be changed and flash boot needs to call the ROM boot function ApplyProtectionSettings() to reconfigure them.

36.3.3.32 Disable WDT (40)

Bootloader may run for a significantly longer time then WDT timeout. Therefore, WDT must be either periodically reset or disabled at the start of bootloader.

36.3.3.33 Set VECTOR_TABLE_BASE (41)

CPUSS.CM0_VECTOR_TABLE_BASE and CPUSS.CM7_0/CM7_1_VECTOR_TABLE_BASE Registers must be set to 0xFFFF_0000 value to show the debugger that no user application is running.

36.3.3.34 Launch bootloader (42)

A bootloader firmware is launched. This firmware is a part of flash boot and launching it may be as simple as calling a function.

36.3.3.35 CM0+ core reset (50)

Flash boot resets the CM0+ core by writing to the CPUSS_CM0_CTL Register.

After the write, CM0+ must enter a sleep mode and wait until the core is reset.

36.3.3.36 Launch a user app by ROM boot (51)

Flash boot performs the following to switch to the user application on CM0+:

1. Flash boot sets CPUSS.CM0_VECTOR_TABLE_BASE value with an address of the user CM0+ interrupt vector table.
2. Flash boot performs CM0+ core reset.
3. ROM boot starts up, tests if CPUSS.PROTECTION <> 0, which means ROM boot is launched from a software reset.
4. If (3) succeeds, ROM boot sets SP and PC Register values from the users interrupt vector table, which is read out from CPUSS.CM0_VECTOR_TABLE_BASE.
5. When ROM boot sets the PC Register value with the user's reset handler address, a user code starts executing.

36.3.4 Data structures

36.3.4.1 Application formats

Basic application format (CyBAF)

This is the most basic format and requires the least complicated setup and support. TOC2 can be left with default values, or in the ERASED state.

***Note**: CyBAF can be used only in VIRGIN and NORMAL protection modes. SECURE protection mode requires the format of the application, which is to be launched by flash boot, to be CySAF.*

Code flash and SRAM are divided into two parts, one for CM0+, the other for CM7_0/CM7_1 application. The CM0+ vector table usually starts at the beginning of code flash with the application code and the data following it, as shown in Figure 372.

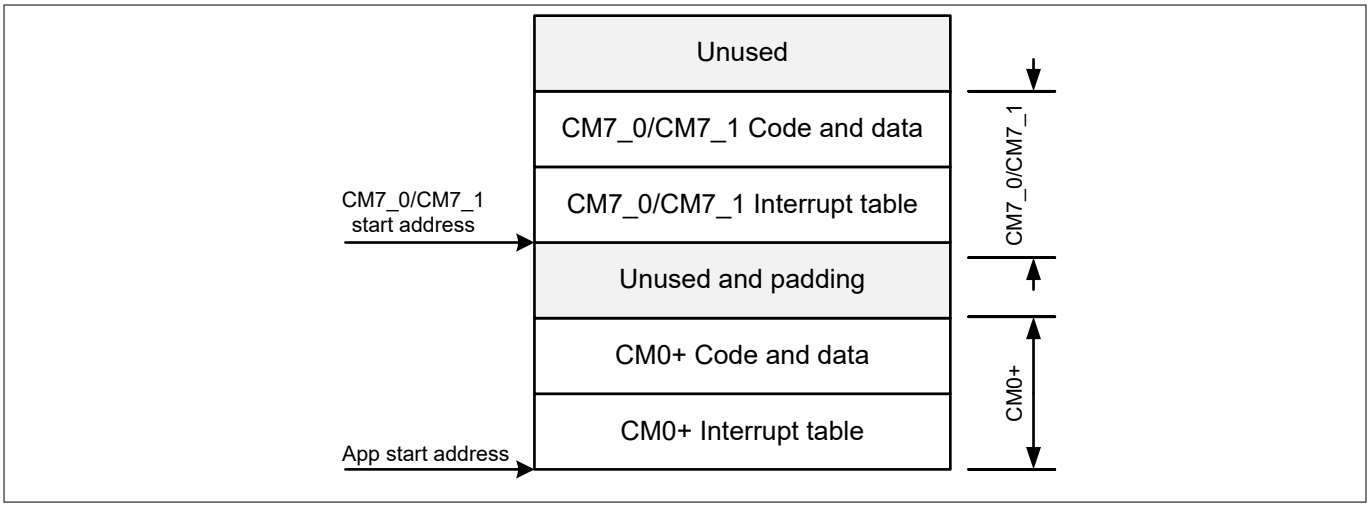


Figure 371 Basic application format structure

The CM7_0/CM7_1 vector table is located at the lowest address location in the CM7_0/CM7_1 part of the code flash. The fraction of code flash and SRAM allocated for CM0+ and CM7_0/CM7_1 are controlled by the user.

The code does not have headers or footers and no predefined location for a digital signature for application validation. The user must validate the code if needed and have the CM0+ start up the CM7_0/CM7_1 CPU when ready.

If the application start address needs to be different from the start of code flash, update the TOC2 member SFLASH_TOC2_FIRST_USER_APP_ADDR with the new application start address.

***Note**: The Arm® specification requires CM0+ interrupt vector table to be 256-byte aligned and CM7_0/CM7_1 interrupt vector table to be 1024-byte aligned.*

Infineon secure application format (CySAF)

This format is used for secure systems where the application code is authenticated using RSASSA-PKCS1-v1.5. Flash boot launches the application in the CySAF if:

1. SFLASH_TOC2_FIRST_USER_APP_ADDR points to a valid memory address
2. SFLASH_TOC2_FIRST_USER_APP_FORMAT value is 1, which means the app format is CySAF
3. The same as (1) and (2) but with SFLASH_TOC2_SECOND_USER_APP_ADDR, and SFLASH_TOC2_SECOND_USER_APP_FORMAT

The structure of CySAF is shown in Figure 372.

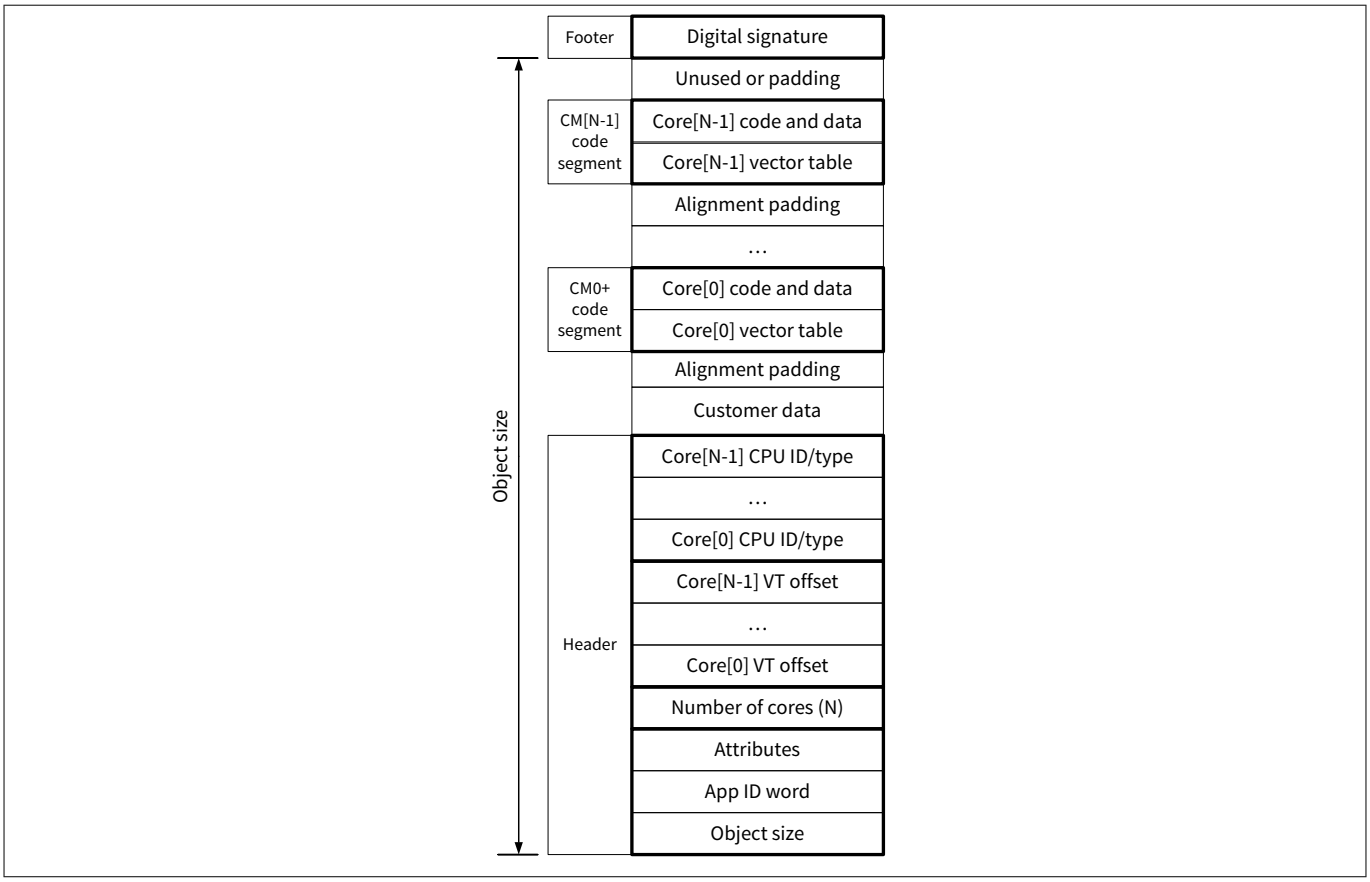


Figure 372 CySAF structure

CySAF consists of the following entries:

- Application header: The header for the flash image is shown in Table 403.

Table 403 Application header

Offset	Size	Item	Description
0x0	4 bytes	Object size	The size in bytes of an area for a digital signature authentication
0x4	4 bytes	Application ID/version	Identifies the type of the app image. Flash boot does not use this value, however, the Infineon applications have the following values: Bit 31 - 16: Application version Bit 15 - 0: Application ID • 0x8001 - Flash boot • 0x8002 - Security image
0x8	4 bytes	Attribute	Reserved for future use
0xC	4 bytes	Number of cores (N)	Number of MCU cores used by the application. Flash boot does not use this value.

(table continues...)

Table 403 (continued) Application header

Offset	Size	Item	Description
0x10 + (4*i)	4 bytes	Core[i] VT offset	Offset to the interrupts vector table for core[i]. An absolute address for core[i] interrupt vector table is calculated as: Application start address + 0x10 + (4*i) + value of core(i) VT offset. Flash boot does not use this information for i>0 and always launches the reset handler for core[0].
0x10 + (4*N) + (4*i)	4 bytes	Core[i] CPU ID and core index	Customer assigned CPU ID and core index. Bit 31 - 20: CPU ID. This is the part number value from the CPUID [15:4] Register in an Arm® device. Bit 7 - 0: Core index The core index is used to distinguish between multiple cores within the system. The system consists of a CM0+ and two CM7 cores. The CM0+ core is identified by a CPUID of '0xC60' and a core index of '0' the first CM7 (CM7_0) core is identified by a CPUID of '0xC24' and a core index of '0' and the second CM7 (CM7_1) core has a CPUID of '0xC24' and a core index of '1'. Flash boot does not use this information.

- MCU code segment

Each flash image in CySAF may contain one or more MCU code segments. At least one MCU code segment is required for the main MCU to be launched, this is usually CM0+. Flash boot requires the application address in SFLASH_TOC2_FIRST_USER_APP_ADDR and SFLASH_TOC2_SECOND_USER_APP_ADDR to have the first MCU code segment for CM0+ application.

Table 404 MCU code segment

Absolute address	Item	Description
App start address + 0x10 + (4*i) + Core(i) VT offset	Interrupts vector table for core[i]	An offset to an Interrupts vector table for core[i].
-	Core[i] code and data	Code and data for the core[i] code segment.

- Application footer

The footer of the application in CySAF contains the signature for authentication. Flash boot authenticates the application; it launches using RSASSA-PKCS1-v1.5 with RSA up to 4096 bits and SHA-256.

36.3.4.2 RSA public key format

The RSA public key may be stored anywhere in the internal flash. For convenience, SFlash contains a region allocated for user data where RSA public key is assumed to be placed by default.

Figure 373 shows the structure of the RSA public key object used for signature authentication. The "Signature Scheme" (specified in TOC) defines the structure of the key. Figure 373 shows the key structure for RSASSA0PKCS1-v1_5.

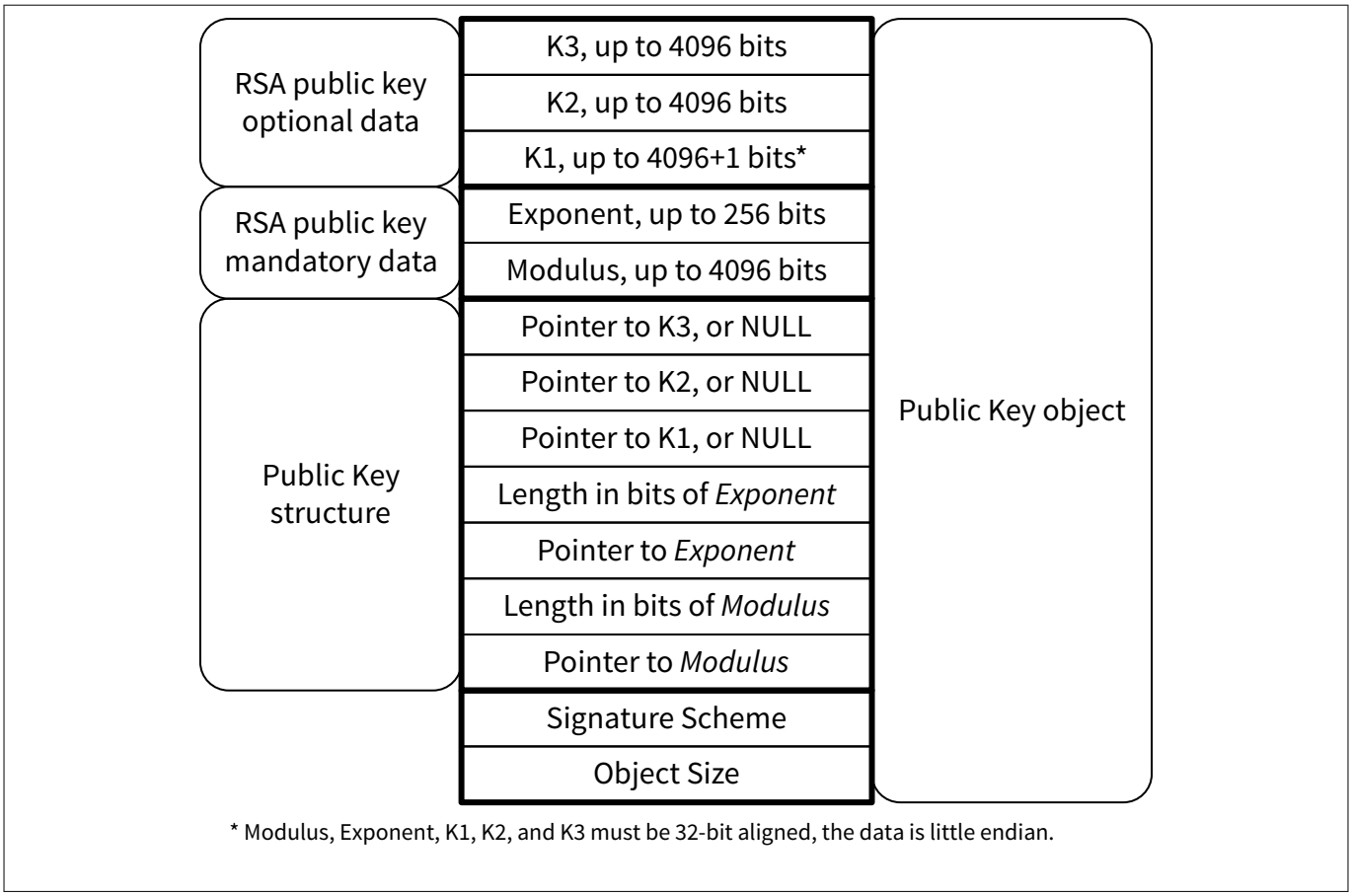


Figure 373 Public key format

Table 405 Public key format

Public key object member name	Description
Object size	A size in bytes used in SECURE_HASH calculation for a public key data protection.
Signature scheme	A signature scheme. 0 - RSASSA-PKCS1-v1.5 with RSA up to 4096 and SHA-256 Other values are reserved.
Pointer to modulus	A pointer to an RSA public key modulus data.
Length in bits of modulus	A length in bits of an RSA public key modulus.
Pointer to exponent	A pointer to an RSA public key exponent data.
Length in bits of exponent	A length in bits of an RSA public key exponent data.
Pointer to K1	A pointer to an optional RSA public key coefficient, named Barrett coefficient. Or NULL if not present.
Pointer to K2	A pointer to an optional RSA public key coefficient, named inverse modulus. Or NULL if not present.
Pointer to K3	A pointer to an optional RSA public key coefficient, named rBarr coefficient. Or NULL if not present.

***Note**: Pointers to K1, K2, and K3 coefficients are optional. When they are set to NULL, flash boot will calculate the value for these coefficients at run time. Providing pre-calculated values for K1, K2, and K3 in the RSA public key object speeds up an RSA calculation up to three times.*

The public key structure format and the public key object data is designed to be compatible with PDL functions for RSA operations (struct name cy_stc_crypto_rsa_pub_key_t).

36.3.4.3 TOC2 structure

TOC2 is a structure stored in SFlash, which is used to configure flash boot and ROM boot firmware. It contains a reference to the SMIF configuration structure used by the programming tools.

Table 406 TOC2 structure

Offset	Name	Purpose
0x00	TOC2_OBJECT_SIZE	Object size in bytes starting from offset 0x00 until the last entry in TOC2.
0x04	TOC2_MAGIC_NUMBER	Magic number (0x01211220)
0x08	TOC2_SMIF_CFG_STRUCT_ADDR	Null terminated table of pointers representing the SMIF configuration structure.
0x0C	TOC2_FIRST_USER_APP_ADDR	Address of CM0+ first user application object (such as HSM in XMC7000)
0x10	TOC2_FIRST_USER_APP_FORMAT	First application object format. 0 - Basic 1 - Infineon standard 2 - Simplified
0x14	TOC2_SECOND_USER_APP_ADDR	Address of CM0+ second user application object (0's if none)
0x18	TOC2_SECOND_USER_APP_FORMAT	Second application object format 0 - Basic 1 - Infineon standard 2 - Simplified
0x1C	TOC2_FIRST_CM7_0_USER_APP_ADDR	Address of CM7 core0 first user application object
0x20	TOC2_SECOND_CM7_0_USER_APP_ADDR	Address of CM7 core0 second user application object
0x24	TOC2_FIRST_CM7_1_USER_APP_ADDR	Address of CM7 core1 first user application object
0x28	TOC2_SECOND_CM7_1_USER_APP_ADDR	Address of CM7 core1 second user application object
0xFC	TOC2_SECURITY_UPDATES_MARKER	The additional PPUs are configured by programming the magic word (0xFEDEEDDF). This field is valid in XMC7000 devices (new flash boot version) only. See BootROM for details of additional PPUs and flash boot versions.
0x100	TOC2_SHASH_OBJECTS	Number of additional objects (not including objects for FACTORY_HASH) starting from offset 0x104 to be verified for SECURE_HASH

(table continues...)

Table 406 (continued) TOC2 structure

Offset	Name	Purpose
0x104	TOC2_SIGNATURE_VERIF_KEY	Address of signature verification key (0 if none). The object is signature scheme specific. It is the public key in case of RSA. The default value is zero.
0x108	TOC2_APP_PROTECTION_ADDR	Address of User SWPU object stored in SFlash. The default value is an address of SFlash row 59.
0x110-0x1F4	-	… (additional objects if needed or 0's if none)
0x1F8	TOC2_FLAGS	TOC2 configuration; see Table 407 for more details. If TOC2 is erased, Flash boot assumes the default TOC2_FLAGS based on the device. Refer to the device-specific Registers TRM for the default values.
0x1FC	Reserved	Unused for XMC

***Note**: If additional objects need to be added to the TOC2 structure, the 32-bit address (in SFlash) and object size should be provided; otherwise, the SECURE_HASH calculation may fail. The maximum number of HASH objects allowed in TOC2 is 10. Three of these objects are already present and fixed: signature verification key, application protection, and TOC2. Therefore, users can add up to seven objects to the SEQURE HASH calculation. If the total number of HASH objects is more than 10, the STATUS_INVALID_TOC2_HASH_OBJECT error will be generated.*

Table 407 SFLASH_TOC2_FLAGS description

Bits	Name	Description
1:0	CLOCK_CONFIG	Indicates clock frequency configuration. The clock should stay the same after flash boot execution. 0 = 8 MHz, IMO, no FLL 1 = 25 MHz, IMO + FLL 2 = 50 MHz, IMO + FLL 3 = Use ROM boot clock configuration
4:2	LISTEN_WINDOW	Determines the Listen window to allow sufficient time to acquire debug port. When CLOCK_CONFIG is set to '3', these window times are calculated assuming ROM boot clock is at 100 MHz. 0 = 20 ms (Default) 1 = 10 ms 2 = 1 ms 3 = 0 ms (No Listen window) 4 = 100 ms

(table continues...)

Table 407 (continued) SFLASH_TOC2_FLAGS description

Bits	Name	Description
6:5	SWJ_PINS_CTL	Determines if SWJ pins are configured in SWJ mode by flash boot. ***Note**: SWJ pins may be enabled later in the user code.* 0 = Do not enable SWJ pins in flash boot. Listen window is skipped 1 = Do not enable SWJ pins in flash boot. Listen window is skipped 2 = Enable SWJ pins in flash boot (default) 3 = Do not enable SWJ pins in flash boot. Listen window is skipped
8:7	APP_AUTH_CTL	Determines if the application image digital signature verification (authentication) is performed: 0 = Authentication is enabled (default) 1 = Authentication is disabled 2 = Authentication is enabled (recommended) 3 = Authentication is enabled
10:9	FB_BOOTLOADER_CTL	Determine if the internal bootloader in flash boot is disabled: 0 = Internal bootloader is disabled 1 = Internal bootloader is launched if the other bootloader conditions are met (default) 2 = Internal bootloader is disabled 3 = Internal bootloader is disabled

36.3.5 Internal bootloader

Bootloaders are a common part of the MCU system design. A bootloader enables product firmware update in the field. In a typical product, firmware is stored in the MCU's flash memory.

At the factory, initial programming of firmware into a product is typically done at PCB assembly time, using the MCU's JTAG or a SWD interface. However, these interfaces are not usually available in the field, and are generally not used for firmware updates.

A better way to update firmware in the field is to use an existing connection between the product and the outside world. The connection may be a standard communication port such as I^2C, USB, or UART, a wireless channel such as Bluetooth low-energy, an industrial protocol such as CAN or a custom protocol.

The flash boot contains an internal bootloader that may be used by OEMs for initial bootloading when the code flash is empty, besides SWD or JTAG initial programming. This may allow OEMs to repurpose SWD or JTAG pins, or completely disable the debugger.

The internal bootloader supports CAN communication interfaces. A bootloader is a separate region of the flash boot that receives data from CAN communication interfaces and places it into the RAM. The intention of the bootloader is to upload and launch the user application flash loader in the RAM. The flash loader programs a user application into code flash during the OEM serial production with no SWD or JTAG connection. After data is transferred, the flash boot validates the data and executes the code to start the second stage of bootloading or run the user application.

***Note**: The flash loader image does not require an encryption because the flash loader is uploaded into the device by the OEM on the factory setup.*

36.3.5.1 Terms and definitions

The product's embedded firmware must be able to use the communication port for two different purposes – normal operation and to update flash. The part of the embedded firmware that knows how to update flash is called a bootloader (Figure 374).

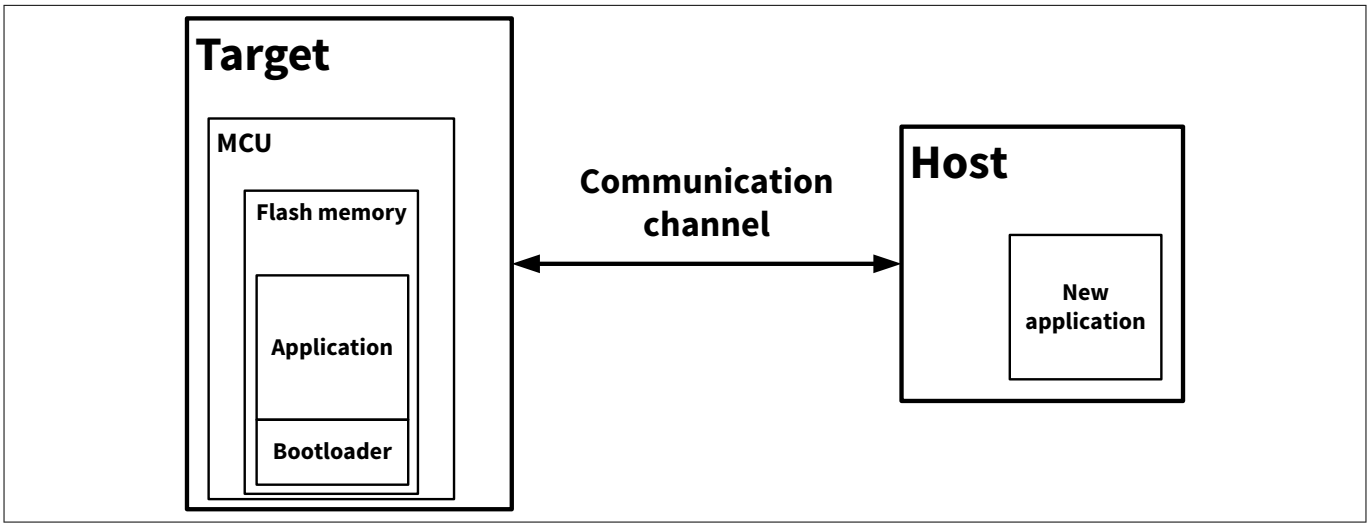


Figure 374 Bootloader system

Typically, the system that provides data to update internal flash is called the host, and the system being updated is called the target. The host can be an external computer or another MCU on the same PCB as the target.

The act of transferring data from the host to the target flash is called bootloading, or bootload operation, or just bootload. Data placed in flash is called the application or firmware image.

36.3.5.2 Using bootloader

The bootloader and the application typically share a communication port. The first step in using a bootloader is to manipulate the product so that the bootloader, and not the application, is executing. This can be done in response to an event such as pressing a button on the product, or by sending a command to the product. The application detects such an event and responds by transferring control to the bootloader.

After the bootloader starts running, the host can send a **Start Bootload** command over the communication channel. If the bootloader sends **OK** in response, bootloading can begin.

36.3.5.3 Bootloader activation conditions

The internal bootloader will activate if all the following conditions are met:

- Two words at the start of flash must be 0xFFFF_FFFF
- TOC2 is valid and TOC2_FLAGS bit FB_BOOTLOADER_DISABLE should be 2'b01 (default). Otherwise, TOC2 is erased
- Protection mode is not SECURE or SECURE_DEAD
- No debugger connection occurs during a 1-second wait window

If any of these conditions are not met, the bootloader will not start.

36.3.5.4 Basic bootloader function flow

During bootloading, the host reads the file for the new application, parses it into flash write commands, and sends those commands to the bootloader. After the entire file is received and installed in the target flash, the bootloader can pass control to the new application.

After a device reset, a bootloader typically executes first. It can then perform the following actions:

- Check the application's validity before transferring control to that application
- Manage the timing to start host communication
- Do the bootload/flash update operation
- Pass control to the application

The flash boot is designed to update the user application in the flash with the algorithm described in Figure 375.

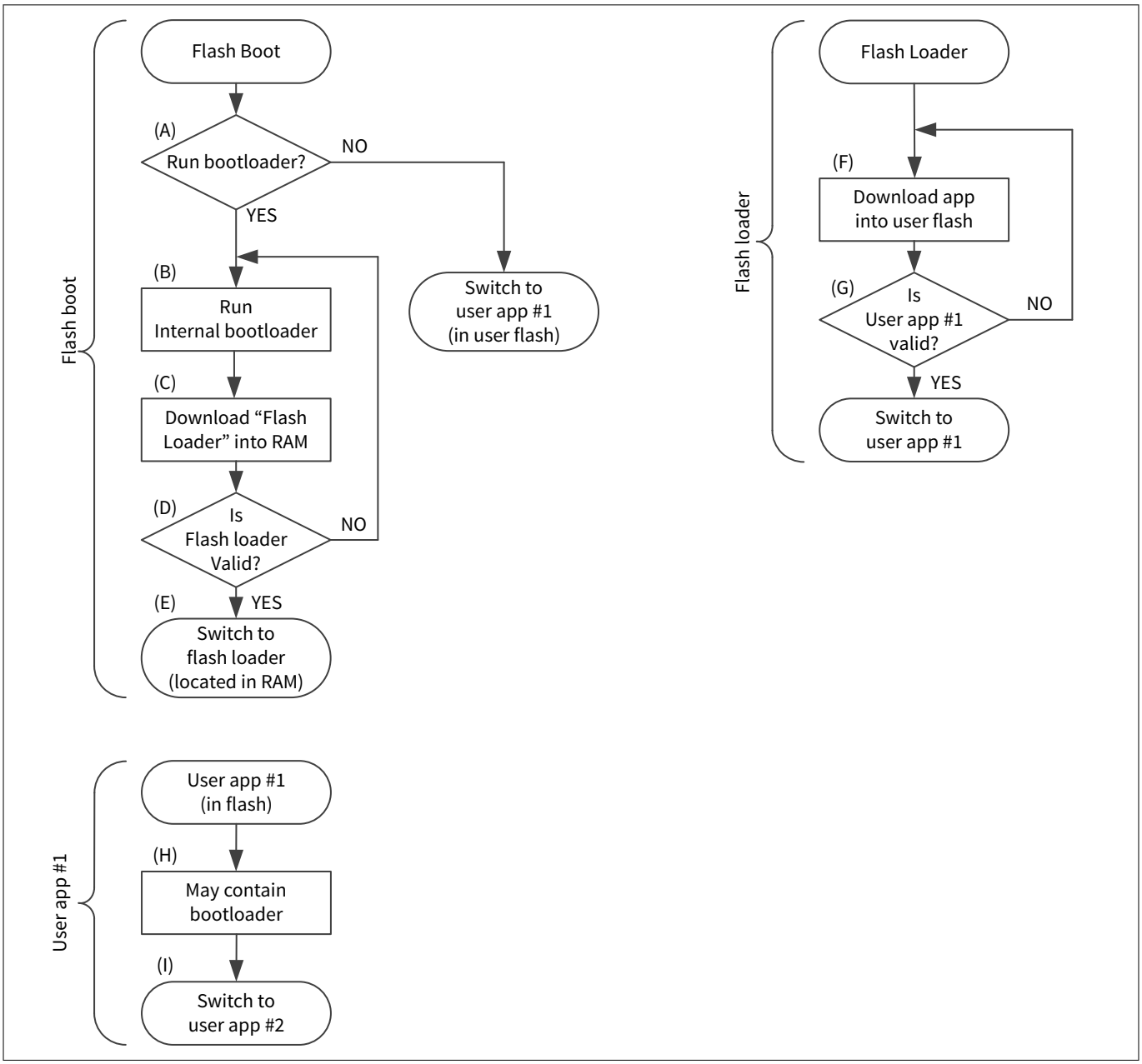


Figure 375 Startup and bootloading sequence

(A) The flash boot checks if the internal bootloader (part of the flash boot) should be run.

(D) The internal bootloader is part of the flash boot firmware that downloads the flash loader into SRAM (C) and launch it (E).

(D) The flash loader requires neither a secure signature nor an encryption because it is uploaded into the device by the OEM on the factory setup.

The flash loader application format is the basic application format with CRC-32C appended to its end, the same format is used by the bootloader SDK non-secure applications.

The CRC-32C hash is used only to check the flash loader image integrity check.

Bootloadable application start address must be within a valid RAM memory length - [RAM_START + 3 KB, RAM_END – 6 KB].

Bootloadable application length must be a value for which the bootloadable application image fits into a RAM address range [RAM_START + 3 KB, RAM_END – 6 KB].

The flash boot bootloader receives the start address and length of the application from the data of the Set App metadata bootloader command, which is the second bootloader command to be sent from the bootloading host to the device.

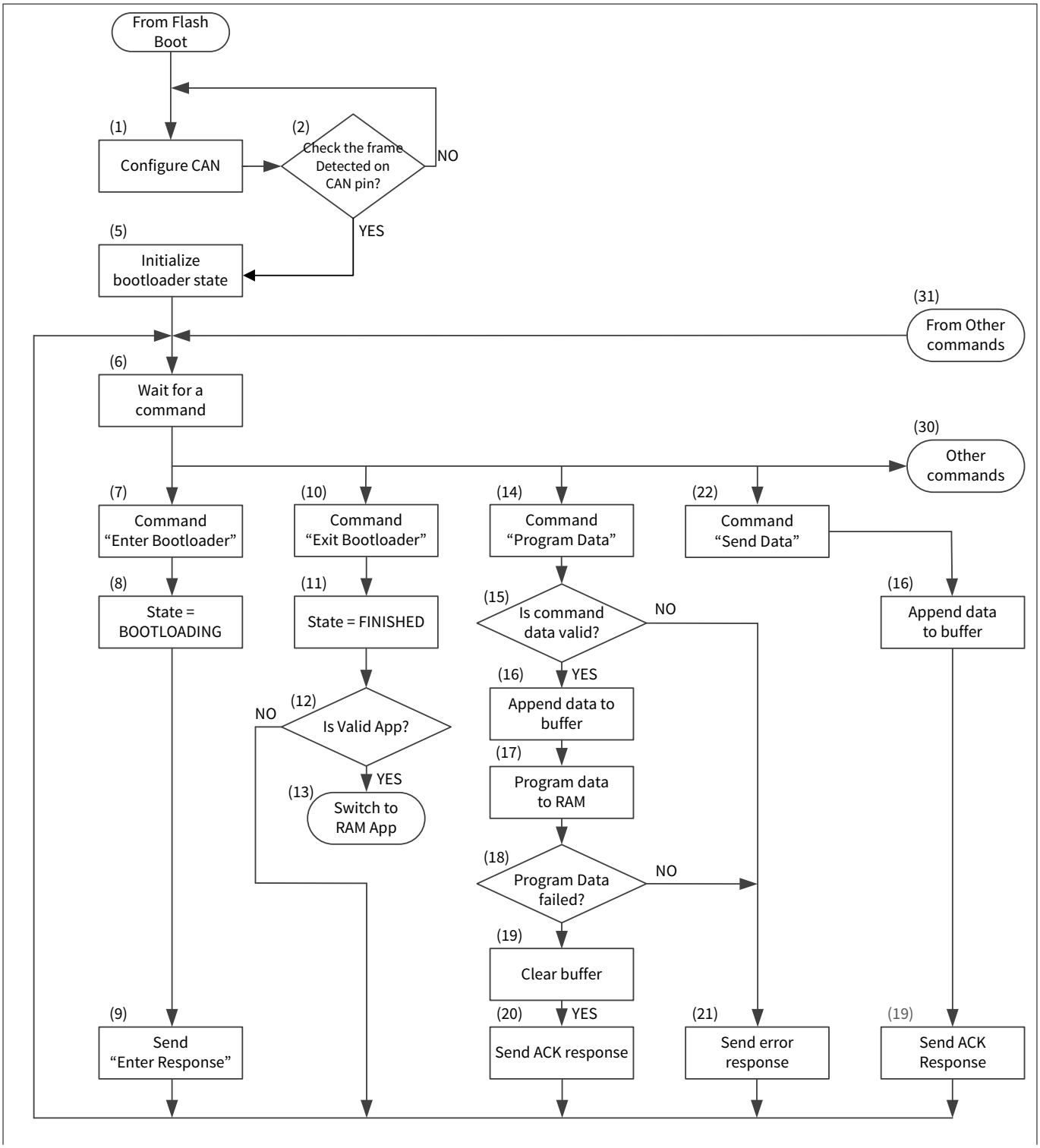
(F) The flash loader downloads a user application through the CAN communication and stores it into the code flash or work flash.

(G) The user application is verified for integrity by the flash loader.

If the user application signature verification fails, the flash loader tries to restart bootloading and receives a new image.

(H) The user application may or may not contain a bootloader. It is up to the user.

Note that only the flash boot part of the bootloading sequence (A) to (E) is developed as the flash boot firmware; the remaining sequence is developed by the user.


From Flash Boot
(1)
Configure CAN
(2)
Check the frame Detected on CAN pin?
NO
YES
(5)
Initialize bootloader state
(31)
From Other commands
(6)
Wait for a command
(30)
Other commands
(7)
Command "Enter Bootloader"
(10)
Command "Exit Bootloader"
(14)
Command "Program Data"
(22)
Command "Send Data"
(8)
State = BOOTLOADING
(11)
State = FINISHED
(15)
Is command data valid?
NO
(16)
Append data to buffer
(16)
YES
Append data to buffer
(12)
NO
Is Valid App?
YES
(13)
Switch to RAM App
(17)
Program data to RAM
(18)
Program Data failed?
NO
(19)
Clear buffer
(9)
Send "Enter Response"
(20)
YES
Send ACK response
(21)
Send error response
(19)
Send ACK Response

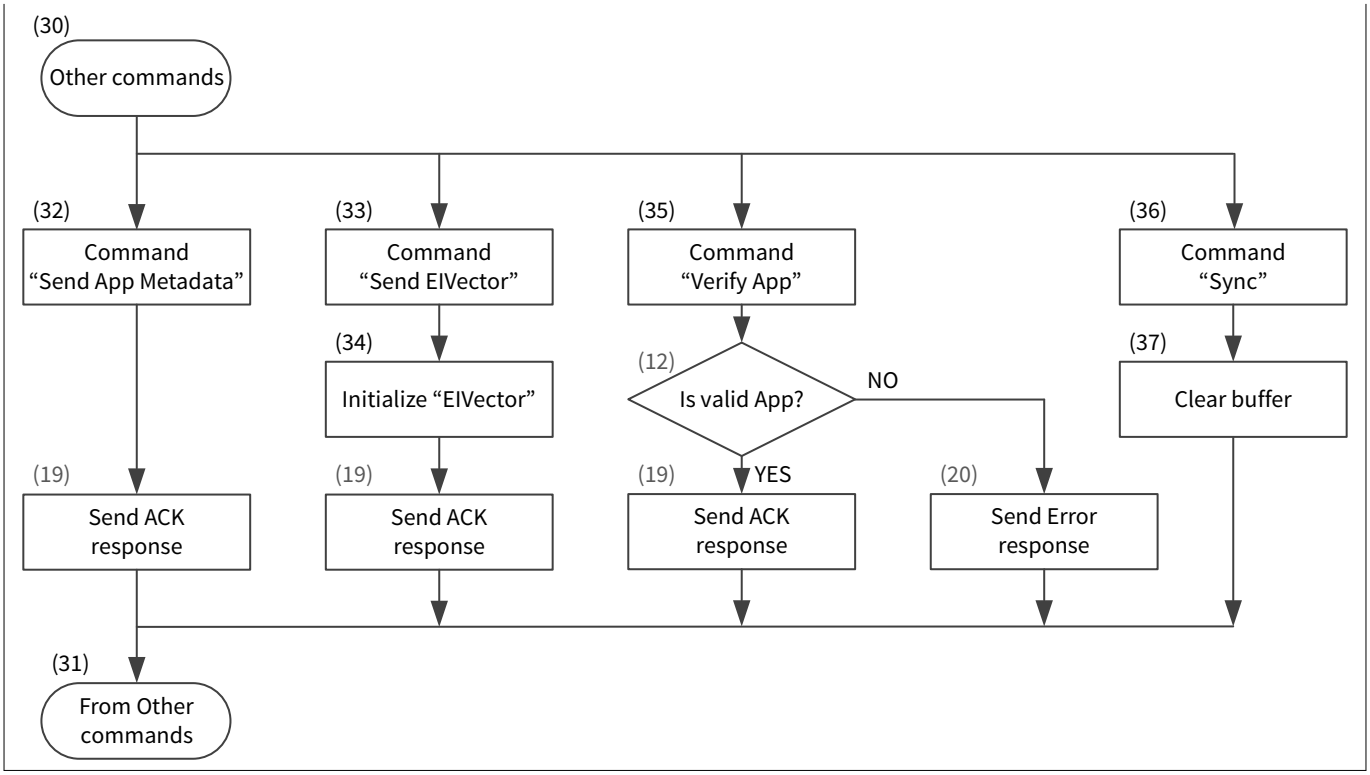


Figure 376 Internal bootloader flow

36.3.5.5 End-of-line programming

The internal bootloader is the part of flash boot firmware that has a goal to download a flash loader into the SRAM and launch it. The flash loader downloads the user application through the CAN communication interface and stores it into the code flash or work flash. The bootloader enables the end-of-line programming using only CAN.

The CAN interfaces are combined on the same device pins to minimize the number of connections for end-of-line programming.

First, the bootloader prepares the channel configuration for CAN and waits for the preconfigured time for the frame from the host. If there is a timeout, the channel is reconfigured for it again waits for the frame. This procedure is cyclically repeated until the frame from the host is received.

The frame receipt completion ensures that the bootloader is attached to the CAN pins, and bootloading continues with the current CAN channel configuration.

Bootloader transport layer implementation details

For the bootloader to transmit commands and responses the 8-byte packet format is chosen because it fits best with the CAN protocols. This approach significantly gains in the performance of big packet transmission between a host and a device. The proof of the selected approach (pack data into 8-byte frames).

Note that except the CAN protocols overhead, there is the bootloader protocol overhead, which consists of service bytes. These bytes enclose the actual data (bootloader commands).

The bootloader command overhead is 8 bytes per command. The bootloader command program data is 32 bytes. It also has the specific overhead of 8 bytes (address of the row for programming that comes in every program data command).

The efficiency coefficients (versus overhead) are:

- For the bootloader program data command:
  - (32 – 8 – 8) / 32 = 0.5, for 8-byte frames
  - (32 – 7 – 8) / 32 = 0.53, for 8-byte frames.

The total time to transmit the program data command:

transmit time = command size / (baud rate × efficiency coefficients)

- Through 8-byte frames: 32 / [(115200 × 103 / 8) × (0.5 × 0.52)] = 8.55 µs
- Through 1-byte frames: 32 / [(115200 × 103 / 8) × (0.53 × 0.12)] = 34.94 µs.

Note that this time will be bigger due to inter frame intervals (4 for 8-byte frames and 32 for 1-byte frames), and the difference will be drastic.

Transmitting 8-byte frames four times more effective than using 1-byte frames.

Sending data as a byte sequence is not supported in the current implementation; however, this can be implemented on request in future.

CAN transport layer implementation details

The classic CAN with the 500 kbps baud rate and 8-byte data size of the RX/TX buffers is used.

The CAN RX FIFO is used to receive long messages from the host. The implementation relies on the approach described above – the host should pack data into 8-byte packets (to have a smaller number of transfers). The bootloader protocol has commands with different sizes. Long commands (more than 8 bytes) are transmitted as series of 8-byte messages. If there is a remainder (less than 8 bytes), after sending full 8-byte messages, it should be sent by the last CAN message with a smaller data field size (DLC) equal to the remainder size (in bytes).

The CAN_Transport_Read() function extracts received data from the CAN RX FIFO (FIFO element is 8-byte wide) and pack it into a complete command. Then the received command is passed to the bootloader's internal byte buffer for future command processing. The bootloader handles data from this buffer as one bootloader command.

The response to the host is also formed as a byte buffer. The CAN_TransportWrite() function sends data from the bootloader's internal byte buffer in 8-byte messages. The remainder (if any) is sent as the last CAN message with a data size less than 8 bytes, with the corresponding DLC size.

Two IDs are used to communicate with the device through CAN:

- 0x1A1 ID is used to send a bootloader command from the host to device.
- 0x1B1 ID is used to send a response from the device to host.

References

Registers reference manual

[1] Infineon Technologies AG: *XMC7100 registers reference manual*; Available online
[2] Infineon Technologies AG: *XMC7200 registers reference manual*; Available online

Architecture reference manual

[3] Arm®: *Arm® v7-M architecture reference manual*; Available online
[4] Arm®: *Arm® v6-M architecture reference manual*; Available online

Revision history

Document revision	Date	Description of changes
**	2021-12-14	Initial release
*A	2022-07-13	• Removed GMII from the section MII • Updated Table 240
*B	2022-11-21	• Updated Figure 1 • Updated Figure 61 • Updated Figure 292, Figure 293 • Removed "DLP" from the Table 258 • Updated CAN driver configuration

Revision history

Document revision	Date	Description of changes
*C	2024-11-29	• Updated the following: - Updated Features - Updated Master with missing access attributes - Updated PPU - Updated ECC parity generation by software - Updated the note in P-DMA channel configuration SRAMs - Updated DST_TRANSFER_SIZE in Table 28 - Updated ECC for P-DMA channel configuration SRAMs - Updated ECC details - Updated RAM retention configuration - Updated SMPU configuration in SFlash - Updated Chapter 11.4.2, Table 92, Table 102 - Updated Spread spectrum clock generation (SSCG) - Updated the footnote for Table 106 and Table 119 - Updated the note for Table 134 - Updated the Table 134 through Table 140 - Updated Transmitter and receiver signals in DMA/DW trigger signals - Updated the equations in Oversampling and bit rate - Updated the section to reflect RX buffer handling in Dedicated RX buffers - Updated the Features - Updated the Block diagram - Updated Trigger outputs - Updated cc0_match (cc1_match) event - Updated Table 195 and Table 195 - Updated Quadrature decoder mode - Updated RAM retention configuration - Updated PWM mode functionalities - Updated Figure 112 and Figure 203 - Updated Figure 293 and Figure 344 - Updated Features overview - Updated Clock and prescaling - Updated Counter status - Updated Trigger multiplexer - Updated Figure 286, Figure 288 - Updated Table 234 - Updated Trigger multiplexing - Updated Table 400 - Updated Table 242: removed ETHx_TX_CLK - Updated RMII and Clock sources for PHY interface • Added the following: - Added a note in Regulators - Added a note in Protection context 0 and 1

Revision history

Document revision	Date	Description of changes
		- Added a note in Usage of high-current regulator controller - Added OFF column in Table 104 - Added Table 140 - Added a footnote for Table 106 - Added footnote for Table 134 - Added a note and updated all the notes in Slew rate control - Added Configuring Counter for PWM Mode for Stepper Motor Control (SMC) - Added Usage Guidelines - Added note in Supported features - Added note in DMA interface • Deleted “Design configuration parameters” on 559
*D	2025-11-21	Updated the template

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2025-11-21

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2025 Infineon Technologies AG

All Rights Reserved.

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-yth1741186358257

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.

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.