PSOC™ Control C3M/P8, C3M/P7 architecture

Reference manual

About this document

Scope and purpose

This document provides comprehensive information about Infineon's PSC3M/P8, PSC3M/P7 series of microcontrollers. These devices are based on Arm® Cortex®-M33 running at up to 180 MHz with DSP and FPU capabilities. Additionally, the devices contain Programmable Power Control Accelerator (PPCA) peripherals, with two Arm® Cortex®-M33 running at up to 200 MHz, 12-bit 25 MSPS ADCs, comparators, hardware filters, a custom logic block, a differential comparator slope generator, DAC, and advanced timers with high-resolution capability for high-performance real-time closed-loop control systems.

This document covers the following subfamilies of PSC3M/P8, PSC3M/P7 MCUs:

- PSOC™ Control C3 P (Power Control) series: PSC3P7xx, PSC3P8xx
- PSOC™ Control C3 M (Motor Control) series: PSC3M7xx, PSC3M8xx

For an overview of the PSC3M/P8, PSC3M/P7 MCU product family, refer to the device datasheet.

Intended audience

This document is intended for hardware engineers, embedded software developers, and system architects who want to understand the architecture of Infineon's PSC3M/P8, PSC3M/P7 series of microcontrollers.

Table of contents

About this document . . . 1
Table of contents . . . 2
1 Introduction . . . 27
1.1 Detailed features . . . 27
1.2 Architectures . . . 33
1.3 Document organization and conventions . . . 34
1.3.1 Major sections . . . 34
1.3.2 Document conventions . . . 34
1.3.2.1 Register conventions . . . 34
1.3.2.2 Numeric naming . . . 34
1.3.2.3 Unit of measure . . . 34
2 CPU subsystem (CPUSS) . . . 36
2.1 Features . . . 36
2.2 Architecture . . . 36
2.2.1 CPUSS . . . 36
2.2.2 Address and memory maps . . . 37
2.2.2.1 Wait state lookup tables . . . 39
2.3 Operating modes and privilege levels . . . 40
2.3.1 Security states . . . 40
2.3.2 Operating modes . . . 40
2.3.3 Operating states . . . 40
2.3.4 Privilege levels . . . 41
2.4 Instruction set . . . 41
2.5 Registers summary . . . 41
3 Bus infrastructure . . . 46
3.1 SYSCPUSS AHB5 interconnect . . . 46
3.1.1 AHB5 bus matrix interconnect . . . 47
3.1.2 SYSCPU buses . . . 48
3.1.2.1 Cortex® -M33 C-AHB bus . . . 48
3.1.2.2 Cortex® -M33 S-AHB bus . . . 48
3.1.2.3 DMA bus . . . 48
3.2 AHB5 MMIO . . . 48
3.3 Peripheral interconnect (PERI) . . . 48
3.3.1 AHB bus error handling . . . 50
3.4 Bus master peripherals . . . 50
3.4.1 Direct memory access (DMA-DW) . . . 50
3.4.2 CryptoLite . . . 50
4 Inter processor communication (IPC) . . . 51

Table of contents

4.1 Features . . . 51
4.2 Architecture . . . 51
4.2.1 IPC channel . . . 51
4.2.2 IPC interrupt . . . 52
4.2.3 IPC channels and interrupts . . . 53
4.3 Implementing locks . . . 54
4.4 Message passing . . . 55
4.5 Typical usage models . . . 57
4.5.1 Full-duplex communication . . . 57
4.5.2 Half-duplex with independent event handling . . . 57
4.5.3 Half-duplex with shared event handling . . . 58

5 Flash . . . 60
5.1 Features . . . 60
5.2 Configuration . . . 60
5.2.1 Block diagram . . . 60
5.2.2 Flash geometry . . . 61
5.2.3 Bank modes and address mapping . . . 63
5.3 Flash controller . . . 63
5.3.1 Power modes . . . 63
5.4 Read While Write support . . . 64
5.5 ECC . . . 65
5.5.1 ECC generation/encoding . . . 65
5.5.2 ECC (error) injection . . . 65
5.6 Security and protection . . . 66
5.7 SROM APIs . . . 66

6 SRAM controller . . . 67
6.1 Features . . . 67
6.2 Architecture . . . 67
6.2.1 AHB5 read transfers . . . 67
6.2.2 32-bit AHB5 write transfers . . . 67
6.2.3 Partial AHB5 write transfers . . . 67
6.2.4 Power partition control . . . 67
6.3 Error Correcting Code (ECC) . . . 68
6.3.1 ECC Error injection . . . 68
6.3.2 ECC error report functionality . . . 68
6.3.2.1 Partial AHB5 write transfers . . . 68
6.3.2.2 AHB5 read transfers . . . 68
6.3.2.3 Correctable ECC error reporting . . . 68
6.3.2.4 Non-correctable ECC error reporting . . . 69
6.4 Wait states . . . 69

7 Interrupts . . . 70

Table of contents

7.1 Features . . . 70
7.2 Architecture . . . 70
7.3 Interrupts and exceptions operation . . . 71
7.3.1 Interrupt/exception handling . . . 71
7.3.2 Level interrupts . . . 71
7.3.3 Exception vector table . . . 72
7.4 Exception sources . . . 74
7.4.1 Reset exception . . . 75
7.4.2 Non-maskable interrupt exception . . . 75
7.4.3 HardFault exception . . . 76
7.4.4 MemManage fault exception . . . 76
7.4.5 BusFault exception . . . 76
7.4.6 UsageFault exception . . . 76
7.4.7 SecureFault exception . . . 77
7.4.8 Supervisor Call (SVCall) exception . . . 77
7.4.9 DebugMonitor exception . . . 77
7.4.10 PendSupervisory (PendSV) exception . . . 77
7.4.11 System Tick (SysTick) exception . . . 77
7.5 Interrupt sources . . . 78
7.6 Interrupt/exception target security state and banking . . . 82
7.6.1 Interrupt target security state . . . 83
7.6.2 GPIO secure and non-secure interrupts . . . 83
7.7 Interrupt/exception priority . . . 84
7.8 Enabling and disabling interrupts . . . 84
7.9 Interrupt/exception states . . . 85
7.9.1 Pending interrupts/exception . . . 86
7.10 Stack usage for interrupts/exceptions . . . 87
7.11 Interrupts and low-power modes . . . 87
7.12 Interrupt/exception - initialization/configuration . . . 88

8 Fault monitoring . . . 89
8.1 Features . . . 89
8.2 Architecture . . . 89
8.2.1 Fault report . . . 90
8.2.2 Signaling interface . . . 90
8.2.3 Monitoring . . . 91
8.2.4 Low-power mode operation . . . 91
8.2.5 Using a fault structure . . . 92
8.2.6 CPU exceptions versus fault monitoring . . . 92
8.3 Fault sources . . . 93
8.4 Fault encodings . . . 96
8.4.1 PERI Master0 interface PPC violation . . . 96
8.4.2 PERI Master1 interface PPC violation . . . 96

8.4.3 PERI peripheral PPC PC mask violation . . . 97
8.4.4 PERI peripheral Group1 AHB timeout . . . 98
8.4.5 PERI peripheral Group2 AHB timeout . . . 98
8.4.6 PERI peripheral Group3 AHB timeout . . . 98
8.4.7 PERI peripheral Group4 AHB timeout . . . 98
8.4.8 PERI peripheral Group5 AHB timeout . . . 99
8.4.9 PERI peripheral Group0 AHB violation . . . 99
8.4.10 PERI peripheral Group1 AHB violation . . . 99
8.4.11 PERI peripheral Group2 AHB violation . . . 100
8.4.12 PERI peripheral Group3 AHB violation . . . 100
8.4.13 PERI peripheral Group4 AHB violation . . . 101
8.4.14 PERI peripheral Group5 AHB violation . . . 101
8.4.15 RAMC-0 MPC violation . . . 102
8.4.16 RAMC-1 MPC violation . . . 103
8.4.17 AHB bridge error fault . . . 103
8.4.18 PROMC MPC violation . . . 104
8.4.19 RAMC-0_C ECC mmio . . . 104
8.4.20 RAMC-0_NC ECC mmio . . . 104
8.4.21 RAMC-1_C ECC mmio . . . 104
8.4.22 RAMC-1_NC ECC mmio . . . 105
8.4.23 FLASHC MPC fault . . . 105
8.4.24 FLASHC bus access fault . . . 105
8.4.25 FLASHC main_c ECC mmio fault . . . 106
8.4.26 FLASHC main_nc ECC mmio fault . . . 106
8.4.27 FLASHC FM CTL fault . . . 106
8.4.28 FLASHC FM r_q fault . . . 107
8.4.29 FLASHC flash CTL fault . . . 107
8.4.30 FLASHC addr hole fault . . . 108
8.4.31 FLASHC lock fault . . . 108
8.4.32 SRSS clock supervisor fault . . . 109
8.4.33 SRSS supply supervisor fault . . . 109
8.4.34 PPCA SRAM0_NC ECC fault . . . 109
8.4.35 PPCA SRAM1_NC ECC fault . . . 109
8.4.36 PPCA SRAM2_NC ECC fault . . . 109
8.4.37 PPCA SRAM3_NC ECC fault . . . 110
8.4.38 PPCA SRAM4_NC ECC fault . . . 110
8.4.39 PPCA hresp error CPU0 fault . . . 110
8.4.40 PPCA SRAM0_C ECC fault . . . 110
8.4.41 PPCA SRAM1_C ECC fault . . . 111
8.4.42 PPCA SRAM2_C ECC fault . . . 111
8.4.43 PPCA SRAM3_C ECC fault . . . 111
8.4.44 PPCA SRAM4_C ECC fault . . . 111

Table of contents

8.4.45 PPCA hresp error CPU1 fault . . . 111
8.4.46 Supply voltage glitch sensor fault . . . 112
8.4.47 Advanced encryption standard fault . . . 112
8.5 Registers summary . . . 113

9 Direct memory access (DMA - DW) . . . 114
9.1 Features . . . 114
9.2 Architecture . . . 115
9.2.1 Standard flow . . . 115
9.3 Channels . . . 116
9.3.1 Channel priority . . . 116
9.3.2 Channel security attributes . . . 116
9.3.3 Channel state . . . 116
9.3.4 Channel interrupt generation . . . 117
9.4 Descriptors . . . 118
9.4.1 Descriptor memory transfer types . . . 118
9.4.1.1 Single transfer . . . 119
9.4.1.2 1D memory transfer . . . 119
9.4.1.3 2D memory transfer . . . 120
9.4.1.4 Cyclic Redundancy Check (CRC) transfer . . . 121
9.4.2 Descriptor size . . . 122
9.4.3 Descriptor configuration . . . 122
9.4.3.1 Descriptor control (DESCR_CTL) . . . 123
9.4.3.2 Source and destination address (DESCR_SRC/DESCR_DST) . . . 124
9.4.3.3 Descriptor X loop control (DESCR_X_CTL) . . . 124
9.4.3.4 Descriptor Y loop control (DESCR_Y_CTL) . . . 124
9.4.3.5 Descriptor next pointer (DESCR_NEXT_PTR) . . . 124
9.4.4 Descriptor transfer Size . . . 124
9.4.5 Descriptors chaining . . . 125
9.5 Triggers . . . 126
9.5.1 Input triggers . . . 126
9.5.2 Pending triggers . . . 127
9.5.3 Output triggers . . . 127
9.6 Performance . . . 128
9.6.1 Single transfer . . . 128
9.6.2 1D transfer . . . 128
9.6.3 2D transfer . . . 129
9.6.4 CRC transfer . . . 129

10 Trigger multiplexer (Trigger MUX) . . . 130
10.1 Features . . . 130
10.2 Architecture . . . 130
10.3 Functional description . . . 131

Table of contents

10.4 Software triggers . . . 133
10.5 Trigger functionality . . . 133
10.6 Trigger multiplexer connections . . . 136
10.6.1 Trigger group inputs . . . 138
10.6.2 Trigger group outputs . . . 148
10.7 Trigger one-to-one connections . . . 149
10.8 Registers summary . . . 149

11 Debug and trace system . . . 151
11.1 Features . . . 151
11.2 Architecture . . . 151
11.2.1 Debug access port (DAP) . . . 153
11.2.1.1 DAP power domain . . . 153
11.2.1.2 DAP security . . . 153
11.2.2 Trace . . . 154
11.2.3 ROM tables . . . 154
11.2.4 Cross trigger functionality . . . 155
11.2.4.1 Freeze peripherals in debug mode . . . 156
11.3 Serial wire debug (SWD) interface . . . 158
11.4 JTAG interface . . . 159
11.5 Programming the MCU . . . 159
11.5.1 SWD port acquisition . . . 159
11.5.2 SWD programming mode entry . . . 160
11.5.3 SWD programming routine executions . . . 160
11.6 PSC3M/P8, PSC3M/P7 MCU SWD debug interface . . . 160
11.6.1 Debug control and configuration registers . . . 160
11.6.2 Break point units (BPU) . . . 160
11.6.3 Data watchpoints (DWT) . . . 161
11.6.4 Debugging the PSC3M/P8, PSC3M/P7 device . . . 161
11.7 Debug handler . . . 161
11.8 Registers summary . . . 162

12 Cryptographic accelerator . . . 163
12.1 Features . . . 163
12.2 Architecture . . . 163
12.2.1 Randomized clock . . . 164
12.2.2 Hardware/Software interface . . . 164
12.3 Block functions . . . 166
12.3.1 AES-128 . . . 166
12.3.2 SHA . . . 167
12.3.2.1 SHA-256/384/512 . . . 167
12.3.3 Vector unit (VU) . . . 169
12.3.4 True random number generator (TRNG) . . . 172

Table of contents

12.3.4.1 Fixed ring oscillators: RO11, RO15, GARO15, FIRO15 . . . 174
12.3.4.2 Ring oscillator: GARO31 . . . 174
12.3.4.3 Ring oscillator: FIRO31 . . . 175
12.3.4.4 TRNG result . . . 176
12.3.4.5 TRNG health monitor . . . 176
12.4 Interrupts . . . 178
12.4.1 TRNG interrupt . . . 178
12.4.2 Error interrupt . . . 178
12.5 Registers summary . . . 179

13 TrustZone . . . 181
13.1 Features . . . 181
13.2 Architecture . . . 181
13.3 Memory regions . . . 183
13.4 Attribution units . . . 184
13.4.1 Implementation Defined Attribution Unit (IDAU) . . . 184
13.4.2 Secure Attribution Unit (SAU) . . . 185
13.5 State transition . . . 187
13.5.1 Secure to non-secure state transition . . . 187
13.5.2 Non-secure to Secure state transition . . . 188
13.5.3 State transition from exception or interrupt . . . 188

14 Protection units . . . 189
14.1 Architecture . . . 190
14.2 Protection context . . . 193
14.2.1 Special protection contexts . . . 194
14.3 Master protection attributes . . . 194
14.3.1 Master Security Controller (MSC) . . . 195
14.3.2 Memory Protection Unit (MPU) . . . 195
14.4 Slave protection attributes . . . 195
14.4.1 Memory Protection Controller (MPC) . . . 195
14.4.1.1 Software view . . . 196
14.4.1.2 MPC MMIO register protection . . . 198
14.4.2 Peripheral Protection Controller (PPC) . . . 198
14.4.3 PPC access control rules . . . 199
14.4.3.1 Security aware peripherals . . . 200
14.4.3.2 PPC regions . . . 201

15 Secure boot . . . 215
15.1 Basic definitions . . . 215
15.2 Features . . . 216
15.3 Architecture . . . 216
15.3.1 Policies and keys . . . 216
15.3.1.1 Policies . . . 217

15.3.1.2 Keys . . . 217
15.3.2 Life cycle stages (LCS) . . . 217
15.3.2.1 Infineon manufacturing . . . 218
15.3.2.2 Development LCS . . . 218
15.3.2.3 Production LCS . . . 218
15.3.2.4 RMA . . . 219
15.3.3 Boot sequence . . . 219
15.3.3.1 Secure boot . . . 221
15.3.3.2 Protection states . . . 222
15.3.4 BootROM serial interface . . . 222
15.3.5 Secured provisioning and re-provisioning . . . 222
15.3.6 Protection context used in secure boot . . . 223

16 Advanced Encryption Standard (AES) . . . 224
16.1 Introduction . . . 224
16.2 Overview . . . 224
16.3 Programming model . . . 225
16.4 Functional description . . . 227
16.4.1 Module reset . . . 227
16.4.2 Endianess . . . 227
16.4.3 BusFault scenarios . . . 227
16.4.4 Handling of key state clearing . . . 227
16.4.5 Fault structure . . . 228
16.4.6 Timing . . . 228
16.4.7 Debug support . . . 228
16.5 AES registers interface . . . 228
16.5.1 Registers overview . . . 229
16.5.2 Access conditions . . . 230
16.5.3 Resets . . . 231
16.5.4 AES configuration register . . . 231
16.5.5 AES command register . . . 232
16.5.6 AES status register . . . 233
16.5.7 AES configuration register 2 . . . 235
16.5.8 AES interrupt cause register . . . 236
16.5.9 AES interrupt set register . . . 237
16.5.10 AES interrupt mask register . . . 238
16.5.11 AES masked interrupt register . . . 238
16.5.12 AES clear register . . . 239
16.5.13 AES bus counter register . . . 240
16.5.14 AES Q-channel response register . . . 241
16.5.15 AES user seeding register . . . 242
16.5.16 AES Key registers . . . 242
16.5.16.1 AES key register (x for index of shares; y for index of word) . . . 243

16.5.17 AES Data registers . . . 243
16.5.17.1 AES data register (x for index of shares; y for index of word) . . . 244
16.5.17.2 AES data reread register (x for index of shares; y for index of word) . . . 245

17 Power supply and monitoring . . . 246
17.1 Features . . . 246
17.2 Architecture . . . 247
17.3 Power supply . . . 248
17.3.1 Active regulator . . . 248
17.3.2 Deep Sleep regulator . . . 249
17.3.3 Retention regulator . . . 249
17.3.4 Backup domain . . . 249
17.4 SRSS reference block . . . 249
17.5 Voltage monitoring . . . 249
17.5.1 Power-on-reset (POR) . . . 249
17.5.2 Brown-out detection (BOD) . . . 249
17.5.3 Low voltage detect (LVD) . . . 250
17.6 Sleep control . . . 250
17.7 Registers summary . . . 250

18 Device power modes . . . 251
18.1 Features . . . 251
18.2 Architecture . . . 251
18.2.1 Power modes . . . 252
18.2.1.1 Active and Sleep . . . 254
18.2.1.2 Deep Sleep . . . 254
18.2.1.3 Deep Sleep RAM . . . 257
18.2.1.4 Deep Sleep OFF . . . 257
18.2.1.5 Hibernate . . . 258
18.2.2 Other operation states . . . 258
18.2.2.1 Reset state (XRES) . . . 258
18.2.2.2 Off state (OFF) . . . 258
18.3 Power mode transitions . . . 259
18.4 Summary . . . 260
18.5 Registers summary . . . 261

19 Clocking system . . . 263
19.1 Features . . . 263
19.2 Architecture . . . 263
19.3 Clock sources . . . 265
19.3.1 Internal main oscillator (IMO) . . . 265
19.3.2 Internal high-frequency oscillator (IHO) . . . 265
19.3.3 Internal low-speed oscillator (ILO) . . . 265
19.3.4 External clock (EXT_CLK) . . . 265

Table of contents

19.3.5 External crystal oscillator (ECO) . . . 265
19.3.6 Watch crystal oscillator (WCO) . . . 266
19.3.7 ECO prescaler . . . 266
19.4 Clock generation . . . 266
19.4.1 Digital phase-locked loop (DPLL) . . . 266
19.4.2 Frequency lock loop (FLL) . . . 269
19.5 Clock trees . . . 271
19.5.1 Clock path . . . 271
19.5.2 High-frequency root clocks (CLK_HF) . . . 272
19.5.3 Low-frequency clock (CLK_LF) . . . 274
19.5.4 Backup clock (CLK_BAK) . . . 274
19.5.5 Timer clock (CLK_TIMER) . . . 274
19.5.6 Power clock (CLK_PWR) . . . 274
19.6 CLK_HF distribution . . . 275
19.7 Peripheral clock dividers . . . 276
19.7.1 Fractional clock dividers . . . 276
19.7.2 Peripheral clock divider configuration . . . 276
19.7.3 Peripheral clock divider groups . . . 278
19.8 Clock calibration counters . . . 280
19.9 Clock supervision (CSV) . . . 280
19.10 Registers summary . . . 283

20 Watchdog timer . . . 286
20.1 Features . . . 286
20.2 Architecture . . . 286
20.3 Basic watchdog timer . . . 287
20.3.1 Overview . . . 287
20.3.2 Watchdog reset . . . 289
20.3.3 Watchdog interrupt . . . 290
20.4 Multi-counter WDT . . . 291
20.4.1 Overview . . . 291
20.4.1.1 MCWDT0_WDT0 and MCWDT0_WDT1 counters operation . . . 291
20.4.1.2 MCWDT0_WDT2 counter operation . . . 294
20.4.2 Enabling and disabling MCWDT . . . 296
20.4.3 Watchdog cascade options . . . 297
20.4.4 MCWDT reset . . . 300
20.4.5 MCWDT interrupt . . . 300
20.5 Reset cause detection . . . 301
20.6 Registers summary . . . 301

21 Backup . . . 303
21.1 Features . . . 303
21.2 Architecture . . . 303

Table of contents

21.3 Power supply . . . 304
21.4 Clocking . . . 304
21.4.1 WCO with external clock/sine wave input . . . 305
21.4.2 Calibration . . . 305
21.5 Reset . . . 305
21.6 Real-time clock . . . 306
21.6.1 Reading RTC User registers . . . 307
21.6.2 Writing to RTC User registers . . . 307
21.7 Alarm feature . . . 307
21.8 Backup registers . . . 309
21.9 Registers summary . . . 309

22 Reset system . . . 311
22.1 Features . . . 311
22.2 Reset sources . . . 311
22.2.1 Brownout reset . . . 313
22.2.2 WDT/MCWDT reset . . . 313
22.2.3 Software reset . . . 313
22.2.4 External reset . . . 314
22.2.5 Debugger reset . . . 314
22.2.6 Fault detection reset . . . 314
22.2.7 **High frequency clock error reset** . . . 314
22.2.8 Reference clock error reset . . . 314
22.2.9 Hibernate wakeup reset . . . 314
22.3 Identifying reset sources . . . 315
22.4 Registers summary . . . 316

23 I/O subsystem . . . 317
23.1 Features . . . 317
23.2 Architecture . . . 317
23.2.1 I/O cell architecture . . . 318
23.2.2 HSIO to map alternate functions on GPIO . . . 319
23.2.3 I/O state on power-up . . . 323
23.2.4 Behavior in low-power modes . . . 324
23.3 Functional Description . . . 324
23.3.1 Digital input buffer . . . 324
23.3.2 Digital output driver . . . 325
23.3.2.1 GPIO-OVT pins . . . 325
23.3.2.2 Drive strength . . . 325
23.3.2.3 Slew rate control . . . 326
23.3.2.4 Drive mode . . . 326
23.3.3 Firmware-controlled GPIO . . . 330
23.3.4 Interrupt . . . 330

23.3.5 Analog I/O . . . 332
23.3.6 Secure GPIO . . . 333
23.4 Registers summary . . . 333

24 Timer, counter, and pulse-width modulator (TCPWM) . . . 335
24.1 Features . . . 335
24.1.1 Supported features of TCPWM block . . . 335

25 Controller area network flexible data-rate (CAN FD) . . . 337
25.1 Overview . . . 337
25.1.1 Features . . . 337
25.1.2 Features not supported . . . 338
25.2 Architecture . . . 338
25.2.1 Block diagram . . . 338
25.2.2 Dual clock sources . . . 338
25.2.3 Interrupt line . . . 338
25.3 Functional description . . . 339
25.3.1 Modes of operation . . . 339
25.3.1.1 Software initialization . . . 339
25.3.1.2 Normal operation . . . 340
25.3.1.3 CAN FD operation . . . 340
25.3.1.4 Transmitter delay compensation . . . 341
25.3.1.4.1 Description . . . 341
25.3.1.4.2 Transmitter delay compensation measurement . . . 342
25.3.1.5 Restricted Operation mode . . . 343
25.3.1.6 Bus Monitoring mode . . . 343
25.3.1.7 Disable automatic retransmission . . . 344
25.3.1.8 Power Down (Sleep mode) . . . 344
25.3.1.9 Test mode . . . 345
25.3.1.9.1 External Loop Back mode . . . 345
25.3.1.9.2 Internal Loop Back mode . . . 345
25.3.1.10 Application watchdog . . . 346
25.3.2 Timestamp generation . . . 346
25.3.3 Timeout counter . . . 347
25.3.4 Receive handling . . . 347
25.3.4.1 Acceptance filtering . . . 347
25.3.4.2 Receive FIFOs . . . 350
25.3.4.2.1 RX FIFO blocking mode . . . 351
25.3.4.2.2 RX FIFO overwrite mode . . . 352
25.3.4.2.3 RX FIFO top pointer . . . 352
25.3.4.3 Dedicated RX buffers . . . 353
25.3.4.4 Debug on CAN support . . . 354
25.3.4.4.1 Filtering debug messages . . . 354

25.3.4.4.2 Debug message handling . . . 354
25.3.5 Transmission handling . . . 355
25.3.5.1 Transmit pause . . . 356
25.3.5.2 Dedicated TX buffers . . . 356
25.3.5.2.1 Addressing dedicated TX buffers . . . 356
25.3.5.3 Transmit FIFO . . . 357
25.3.5.4 TX queue . . . 357
25.3.5.5 Mixed dedicated TX buffers/TX FIFO . . . 358
25.3.5.6 Mixed dedicated TX buffers/TX queue . . . 358
25.3.5.7 Transmit cancellation . . . 359
25.3.5.8 Transmit event handling . . . 359
25.3.6 FIFO acknowledge handling . . . 359
25.3.7 Configuring the CAN bit timing . . . 360
25.3.7.1 CAN bit timing . . . 360
25.3.7.2 CAN bit rate . . . 362
25.4 Message RAM (MRAM) . . . 363
25.4.1 Message RAM configuration . . . 363
25.4.2 RX buffer and FIFO element . . . 364
25.4.3 TX buffer element . . . 367
25.4.4 TX event FIFO element . . . 370
25.4.5 Standard message ID filter element . . . 371
25.4.6 Extended message ID filter element . . . 373
25.4.7 Trigger memory element . . . 375
25.4.8 RAM watchdog (RWD) . . . 377
25.5 Time trigger CAN . . . 378
25.5.1 Reference message . . . 378
25.5.2 TTCAN configuration . . . 379
25.5.2.1 TTCAN timing . . . 379
25.5.2.2 Message scheduling . . . 380
25.5.2.3 Trigger memory . . . 381
25.5.2.3.1 Trigger types . . . 382
25.5.2.3.2 Restrictions for the node's trigger list . . . 383
25.5.2.3.3 Examples of trigger handling . . . 383
25.5.2.3.4 Detection of configuration errors . . . 385
25.5.2.4 TTCAN schedule initialization . . . 385
25.5.2.4.1 Time slaves . . . 385
25.5.2.4.2 Potential time master . . . 385
25.5.3 TTCAN gap control . . . 386
25.5.4 Stop watch . . . 387
25.5.5 Local time, cycle time, global time, and external clock synchronization . . . 387
25.5.6 TTCAN error level . . . 390
25.5.7 TTCAN message handling . . . 391

Table of contents

25.5.7.1 Reference message . . . 391
25.5.7.2 Message reception . . . 392
25.5.7.3 Message transmission . . . 392
25.5.7.3.1 Transmission in exclusive time windows . . . 393
25.5.7.3.2 Transmission in arbitrating time windows . . . 393
25.5.7.3.3 Transmission in merged arbitrating time windows . . . 393
25.5.8 TTCAN interrupt and error handling . . . 394
25.5.9 Level 0 . . . 394
25.5.9.1 Synchronizing . . . 395
25.5.9.2 Handling error levels . . . 395
25.5.9.3 Master slave relation . . . 396
25.5.10 Synchronization to external time schedule . . . 396
25.6 Setup procedures . . . 397
25.6.1 General program flow . . . 397
25.6.2 Clock stop request . . . 398
25.6.3 MRAM OFF operation . . . 399
25.6.4 MRAM ON operation . . . 400
25.6.5 Procedures specific to M_TTCAN channel . . . 400
25.6.6 CAN Bus configuration . . . 402
25.6.7 Message RAM configuration . . . 402
25.6.8 Interrupt configuration . . . 405
25.6.9 Transmit frame configuration . . . 406
25.6.10 Interrupt handling . . . 407
25.7 Registers summary . . . 414

26 Serial Communication Block (SCB) . . . 415
26.1 Features . . . 415
26.2 Architecture . . . 415
26.2.1 Trigger interface . . . 415
26.2.2 Clocking modes . . . 416
26.2.3 Buffer modes . . . 417
26.2.3.1 FIFO mode . . . 417
26.2.3.2 EZ mode . . . 417
26.2.3.3 CMD RESP mode . . . 417
26.2.4 Block operation . . . 418
26.2.4.1 Block enable and recovery . . . 418
26.2.4.2 Initialization flow after reset . . . 418
26.2.5 Power modes . . . 418
26.3 Serial peripheral interface (SPI) . . . 419
26.3.1 Features . . . 419
26.3.2 General description . . . 419
26.3.3 SPI modes of operation . . . 421
26.3.3.1 Motorola SPI . . . 421

26.3.3.2 Texas Instruments SPI . . . 423
26.3.3.3 National semiconductors SPI . . . 425
26.3.4 SPI buffer modes . . . 426
26.3.4.1 FIFO mode . . . 427
26.3.4.2 EZSPI mode . . . 427
26.3.4.3 Command-response mode . . . 430
26.3.5 Clocking and oversampling . . . 432
26.3.5.1 Clock modes . . . 432
26.3.5.2 Using SPI master to clock slave . . . 433
26.3.5.3 Oversampling and bit-rate . . . 433
26.3.6 SPI master SELECT output timing control . . . 435
26.3.7 SPI parity functionality . . . 437
26.3.8 Loop-back . . . 437
26.3.9 Enabling and initializing SPI . . . 437
26.3.10 Fast SPI Master/Slave support . . . 438
26.3.11 I/O pad connection . . . 438
26.3.11.1 SPI master . . . 438
26.3.11.2 SPI slave . . . 439
26.3.12 Registers summary . . . 442
26.4 UART . . . 442
26.4.1 Features . . . 442
26.4.2 General description . . . 443
26.4.3 UART modes of operation . . . 444
26.4.3.1 Standard protocol . . . 444
26.4.3.1.1 Parity . . . 445
26.4.3.1.2 Start skipping . . . 446
26.4.3.1.3 Half-duplex mode . . . 447
26.4.3.1.4 Break detection . . . 448
26.4.3.1.5 UART flow control . . . 449
26.4.3.1.6 Multi-processor mode . . . 449
26.4.3.2 SmartCard (ISO 7816) . . . 451
26.4.3.3 IrDA . . . 452
26.4.3.4 UART local interconnect network (LIN) mode . . . 453
26.4.4 Clocking and oversampling . . . 455
26.4.5 Enabling and initializing UART . . . 456
26.4.6 I/O pad connection . . . 456
26.4.6.1 UART mode . . . 456
26.4.6.1.1 Standard UART mode . . . 456
26.4.6.1.2 Half-duplex mode . . . 458
26.4.6.2 SmartCard mode . . . 459
26.4.6.3 IrDA mode . . . 459
26.4.6.4 LIN mode . . . 460

26.4.7 Registers summary . . . 460
26.5 Inter-Integrated Circuit (I2C) . . . 460
26.5.1 Features . . . 461
26.5.2 General description . . . 461
26.5.3 Terms and definitions . . . 462
26.5.3.1 Clock stretching . . . 462
26.5.3.2 Bus arbitration . . . 462
26.5.4 I2C modes of operation . . . 462
26.5.4.1 Write transfer . . . 463
26.5.4.2 Read transfer . . . 464
26.5.5 I2C buffer modes . . . 465
26.5.5.1 FIFO mode . . . 465
26.5.5.2 EZI2C mode . . . 466
26.5.5.3 Command-response mode . . . 469
26.5.6 Clocking and oversampling . . . 470
26.5.6.1 Glitch filtering . . . 472
26.5.6.2 Oversampling and bit rate . . . 473
26.5.7 Loop-back . . . 474
26.5.8 Enabling and initializing I2C . . . 474
26.5.8.1 Configuring for I2C FIFO mode . . . 474
26.5.8.2 Configuring for EZ and CMD_RESP modes . . . 475
26.5.9 I/O pad connection . . . 475
26.5.10 Registers summary . . . 476
26.6 Timer generation support . . . 476
26.6.1 Key features of TGS functionality . . . 476
26.6.2 Functional description . . . 476
26.6.3 Intended use case . . . 478
26.6.4 Special cases and protocol-specific behavior . . . 478
26.7 Multiple slave address support . . . 480
26.8 SCB interrupts . . . 481
26.8.1 SPI interrupts . . . 481
26.8.2 UART interrupts . . . 482
26.8.3 I2C interrupts . . . 484

27 Programmable Power Control Accelerator (PPCA) . . . 486
27.1 Features . . . 486
27.2 Architecture . . . 486
27.2.1 Block diagram . . . 486
27.2.2 Operating modes . . . 488
27.2.3 Clock . . . 488
27.2.4 Operation . . . 489
27.3 CPU subsystem (CPUSS) . . . 490
27.3.1 Features . . . 490

27.3.2 Architecture . . . 490
27.3.2.1 CPUSS . . . 491
27.3.2.1.1 CPU0 . . . 491
27.3.2.1.2 CPU1 . . . 491
27.3.2.2 CPU0/CPU1 memories . . . 491
27.3.2.3 Address and memory maps . . . 492
27.3.2.3.1 Memory map visible from CPU0 . . . 493
27.3.2.3.2 Memory map visible from CPU1 . . . 495
27.3.2.3.3 Overall Peripherals map table . . . 496
27.3.2.3.4 Addressing External PPCA address space . . . 497
27.3.2.3.5 Remap function . . . 497
27.3.2.3.6 AHB Master ID remapped . . . 498
27.3.3 Operating modes and privilege levels . . . 499
27.3.3.1 Operating modes . . . 499
27.3.3.2 Operating states . . . 499
27.3.3.3 Privilege levels . . . 499
27.3.4 Instruction set . . . 500
27.3.5 Registers summary . . . 500
27.4 Bus infrastructure . . . 504
27.4.1 AHB bus matrix . . . 506
27.4.1.1 AHB bus matrix interconnect . . . 506
27.4.2 AHB slave interconnect . . . 507
27.4.3 Dedicated Data Path . . . 507
27.4.4 CPUSS buses . . . 509
27.4.4.1 Cortex® -M33 C-AHB bus . . . 509
27.4.4.2 Cortex® -M33 S-AHB bus . . . 509
27.5 PPCA Inter processor communication (IPC) . . . 509
27.5.1 Features . . . 509
27.5.2 Architecture . . . 510
27.5.2.1 IPC channel . . . 510
27.5.2.2 IPC interrupt . . . 511
27.5.2.3 IPC channels and interrupts . . . 511
27.5.3 Implementing locks . . . 512
27.5.4 Message passing . . . 512
27.5.5 Typical usage models . . . 515
27.5.5.1 Full-duplex communication . . . 515
27.5.5.2 Half-duplex with independent event handling . . . 516
27.5.5.3 Half-duplex with shared event handling . . . 517
27.6 SRAM controller . . . 518
27.6.1 Features . . . 518
27.6.2 Architecture . . . 519
27.6.2.1 AHB5 read transfers . . . 519

27.6.2.2 32-bit AHB5 write transfers . . . 519
27.6.2.3 Partial AHB5 write transfers . . . 519
27.6.2.4 Power partition control . . . 519
27.6.3 Error correcting code (ECC) . . . 519
27.6.3.1 ECC error injection . . . 519
27.6.3.2 ECC error report functionality . . . 520
27.6.3.2.1 Partial AHB5 write transfers . . . 520
27.6.3.2.2 AHB5 read transfers . . . 520
27.6.3.2.3 Correctable ECC error reporting . . . 520
27.6.3.2.4 Non-correctable ECC error reporting . . . 520
27.6.4 Wait states . . . 520
27.7 Interrupts . . . 521
27.7.1 Features . . . 521
27.7.2 Architecture . . . 521
27.7.3 Interrupts and exceptions operation . . . 521
27.7.3.1 Interrupt/exception handling . . . 522
27.7.3.2 Level interrupts . . . 522
27.7.3.3 Exception vector table . . . 523
27.7.4 Exception sources . . . 524
27.7.4.1 Reset exception . . . 524
27.7.4.2 Non-maskable interrupt exception . . . 524
27.7.4.3 HardFault exception . . . 525
27.7.4.4 MemManage fault exception . . . 525
27.7.4.5 BusFault exception . . . 525
27.7.4.6 UsageFault exception . . . 525
27.7.4.7 Supervisor Call (SVCall) exception . . . 525
27.7.4.8 DebugMonitor exception . . . 526
27.7.4.9 PendSupervisory (PendSV) exception . . . 526
27.7.5 Interrupt sources . . . 526
27.7.6 Interrupt/exception target state . . . 530
27.7.6.1 Interrupt target state . . . 531
27.7.7 Interrupt/exception priority . . . 531
27.7.8 Enabling and disabling interrupts . . . 531
27.7.9 Interrupt/exception states . . . 532
27.7.9.1 Pending interrupts/exception . . . 533
27.7.10 Stack usage for interrupts/exceptions . . . 534
27.7.11 Interrupt/exception - initialization/configuration . . . 534

28 PPCA configuration (PPCA CNFG) . . . 536
28.1 CNFG . . . 536
28.2 PPCA peripheral interconnect . . . 536
28.3 ADC data mirror . . . 537
28.4 PPCA input selector module (PPCAIN_SEL) . . . 537

Table of contents

28.5 PPCA output selector module (PPCAOUT_SEL) . . . 540
28.6 Fault handler . . . 541
28.7 S2IRQ (Signal to interrupt) . . . 542

29 PPCA Event Processing Unit (EPU) . . . 543
29.1 Features . . . 543
29.2 Architecture . . . 543
29.3 EPU . . . 543
29.4 Register blocks . . . 545
29.5 Processing Unit_Type 1 (PU_T1) . . . 546
29.5.1 PU_T1 block . . . 546
29.5.2 PU_T1 . . . 547
29.5.3 Filtering functions . . . 548
29.5.3.1 Pulse delay . . . 549
29.5.3.2 Masking filter . . . 550
29.5.3.3 Blanking filter . . . 551
29.5.3.4 Debouncing filter . . . 554
29.5.3.5 Synchronous bypass . . . 556
29.6 Processing Unit_Type 2 (PU_T2) . . . 557
29.6.1 PU_T2 block . . . 558
29.7 Combiner . . . 559
29.8 Combiner feedback . . . 561
29.9 EPU interrupt generator . . . 561
29.10 EPU trigger inputs . . . 562
29.11 EPU trigger outputs . . . 568
29.12 Use case . . . 574
29.13 Register summary . . . 575

30 PPCA Analog Top subsystem (ATOP subsystem) . . . 576
30.1 Features . . . 576
30.2 Architecture . . . 578
30.2.1 IO connections . . . 579
30.2.2 Clock . . . 583
30.2.3 BUS interface . . . 583
30.3 Analog to Digital Converter (ADC) . . . 583
30.3.1 ADC Group 0 (ADC_0) . . . 584
30.3.1.1 Architecture . . . 584
30.3.1.2 Features . . . 585
30.3.2 ADC Group 1 (ADC_1) . . . 586
30.3.2.1 Architecture . . . 586
30.3.2.2 Features . . . 588
30.3.3 ADC Group 2 (ADC_2) . . . 588
30.3.3.1 Architecture . . . 588

Table of contents

30.3.3.2 Features . . . 589
30.3.4 ADC Group 3 (ADC_3) . . . 590
30.3.4.1 Architecture . . . 590
30.3.4.2 Features . . . 590
30.3.5 ADC core . . . 591
30.3.5.1 AFE . . . 591
30.3.6 Start of conversion (SOC) and end of conversion (EOC) generation . . . 593
30.3.6.1 Slot control for AUX and ALT_AUX channels . . . 594
30.3.6.2 Control scheme . . . 599
30.3.6.3 Arbitrary trigger control scheme . . . 603
30.3.6.4 Group channel control scheme . . . 605
30.3.6.5 Single channel control scheme . . . 608
30.3.7 Offset and gain correction . . . 610
30.3.8 Conversion timing . . . 611
30.3.9 Signed and unsigned translation . . . 611
30.3.10 Single ended or differential setup . . . 612
30.3.11 Handling of debug state . . . 612
30.3.12 ADC mask handling . . . 613
30.3.12.1 Firmware mask operation on group mode . . . 613
30.3.12.2 Hybrid mask operation on group mode with manual trigger . . . 613
30.3.13 Digital comparator (DCMP) . . . 614
30.3.13.1 Invert comparator output configuration . . . 616
30.3.13.2 Mode 1 operation - Accumulate . . . 620
30.3.13.3 Mode 2 operation - Increment . . . 621
30.3.13.4 Mode 3 operation . . . 622
30.3.14 Digital filter . . . 623
30.3.14.1 Median filter . . . 626
30.3.14.2 Linear interpolator filter . . . 627
30.3.14.3 Low pass filter . . . 632
30.3.14.4 CIC3 filter . . . 634
30.3.14.5 CIC comparator . . . 637
30.3.14.6 Average/Integrator filter . . . 637
30.3.14.7 Min/Max detection . . . 641
30.3.14.8 Filter module dedicated data path . . . 643
30.3.15 ADC result . . . 646
30.3.15.1 Filter result format . . . 647
30.3.16 ADC interrupt error condition . . . 649
30.4 IDAC . . . 649
30.4.1 Using the IDAC for PMBus detection . . . 651
30.4.2 Using the IDAC for input pin float detection . . . 652
30.5 Analog Reference (AREF) . . . 653
30.6 General Purpose Input (GPI) . . . 654

30.6.1 Using GPI . . . 655
30.7 Temperature sensor (TSNS) . . . 656
30.8 Differential Comparator Slope Generation (DCSG) . . . 656
30.8.1 Features . . . 656
30.8.2 Architecture . . . 656
30.8.2.1 Block diagram . . . 656
30.8.3 Functionality description . . . 657
30.8.3.1 DCSG group . . . 658
30.8.3.1.1 Differential mode handling for DCSG_TH . . . 660
30.8.3.2 Slope generator function . . . 660
30.8.3.2.1 Slope Generator Handling of Glitch from the CMP_O . . . 665
30.8.3.2.2 Handling of debug state . . . 666
30.8.3.2.3 DCSG_TH timing requirement . . . 666
30.8.3.2.4 AUTO_RESET feature . . . 670
30.8.3.2.5 SLGEN_MAP feature . . . 672
30.8.3.3 DCSG interleave blanking feature . . . 672
30.8.3.3.1 CFG_REFRESH_RATE consideration . . . 674
30.8.3.3.2 Interleave blanking enable procedure . . . 674
30.8.3.4 DCSG Control . . . 674
30.8.3.4.1 Slope generator control behavior . . . 674
30.8.3.4.2 A vs B Control Behavior . . . 675
30.8.3.4.3 Firmware control of DCSG . . . 678
30.8.3.5 R2R DAC . . . 679
30.8.3.5.1 Using the R2R DAC . . . 679
30.8.3.5.2 Buffer offset correction . . . 681
30.9 Registers summary . . . 682

31 PPCA Custom Logic Block (CLB) . . . 685
31.1 Features . . . 685
31.2 Architecture . . . 686
31.3 Digital current observer (DICO) subsystem . . . 687
31.3.1 Phase current observer (PCO) . . . 687
31.3.1.1 Current estimator . . . 690
31.3.1.2 Modulator . . . 691
31.3.1.3 Error observer . . . 692
31.3.2 Multi-phase adapter (MPA) . . . 692
31.3.2.1 Phase enabler (PE) . . . 693
31.3.2.2 Phase synchronizer (PS) . . . 698
31.3.3 Mask and kill event . . . 704
31.3.3.1 ADC Filter data selection . . . 704
31.3.3.2 Incorrect setting handling . . . 704
31.4 Matrix commutation (MACO) . . . 704
31.5 Spread pulse startup (SPS) . . . 706

31.6 Debug handling . . . 707
31.7 Safe state . . . 708
31.8 Monitor . . . 708
31.9 Interrupts . . . 709
31.10 Registers summary . . . 710

32 PPCA Hardware filter (HWFILT) . . . 713
32.1 Overview . . . 713
32.2 Trigger filter processing . . . 715
32.3 Data path connection . . . 716
32.4 Registers summary . . . 717

33 PPCA Timer, counter, and pulse-width modulator (TCPWM) . . . 718
33.1 Features . . . 718
33.1.1 Supported features of PPCA TCPWM blocks . . . 719
33.2 Architecture . . . 722
33.2.1 Enabling and disabling counters in a TCPWM block . . . 723
33.2.2 Clocking . . . 723
33.2.2.1 Clock pre-scaling . . . 723
33.2.2.2 Counter operation clock . . . 723
33.2.3 Trigger inputs . . . 724
33.2.3.1 Hardware trigger input . . . 724
33.2.3.2 Software trigger command . . . 727
33.2.4 Internal events . . . 729
33.2.4.1 Underflow event . . . 729
33.2.4.2 Overflow event . . . 729
33.2.4.3 Terminal count event . . . 730
33.2.4.4 cc0_match (cc1_match) event . . . 730
33.2.5 Trigger outputs . . . 731
33.2.6 PWM outputs . . . 732
33.2.7 Interrupts . . . 735
33.2.8 Debug mode . . . 737
33.2.9 Power modes . . . 737
33.3 Counter function modes . . . 738
33.3.1 Timer mode . . . 739
33.3.1.1 Overview . . . 740
33.3.1.2 Up/down count modes . . . 743
33.3.1.3 One shot mode . . . 744
33.3.1.4 Clock pre-scaling . . . 745
33.3.1.5 CC auto reload . . . 745
33.3.1.6 External input event . . . 746
33.3.1.7 External direction control . . . 748
33.3.1.8 Event generation on defined threshold . . . 749

33.3.1.9 How to configure counter for timer mode . . . 749
33.3.2 Capture mode . . . 750
33.3.2.1 Overview . . . 751
33.3.2.2 Capture COUNTER value into CC0/CC0_BUFF . . . 752
33.3.2.3 Up/down count modes . . . 754
33.3.2.4 One shot mode . . . 754
33.3.2.5 Clock pre-scaling . . . 754
33.3.2.6 How to configure counter for capture mode . . . 754
33.3.3 Pulse-width modulation (PWM) mode . . . 755
33.3.3.1 Overview . . . 755
33.3.3.2 Up/down count modes . . . 762
33.3.3.3 One shot mode . . . 764
33.3.3.4 Clock pre-scaling . . . 764
33.3.3.5 CC and PERIOD auto reload with switch event . . . 764
33.3.3.6 Left/Right/Center align PWM with CC0/CC0_BUFF auto reload . . . 766
33.3.3.7 Asymmetric PWM – Using a single compare function . . . 769
33.3.3.8 Asymmetric PWM – Using two compare functions . . . 771
33.3.3.9 Kill mode . . . 772
33.3.3.10 PWM output selection for stepper motor control . . . 780
33.3.3.11 Shifting PWM output using counter value . . . 782
33.3.3.12 External event based mode . . . 783
33.3.3.13 Masking of the PWM output . . . 786
33.3.3.14 How to configure counter for PWM mode . . . 789
33.3.4 PWM with dead time (PWM-DT) mode . . . 789
33.3.4.1 Functionality details . . . 790
33.3.4.2 How to configure counter for PWM_DT mode . . . 794
33.4 High-resolution PWM (HRPWM) . . . 795
33.4.1 Overview . . . 795
33.4.1.1 Features . . . 795
33.4.1.2 Block diagram . . . 796
33.4.1.3 Clocks . . . 797
33.4.2 PWM . . . 797
33.4.2.1 Computation concept . . . 799
33.4.2.2 PWM output mode . . . 801
33.4.2.3 PWM output mode - Left align . . . 802
33.4.2.4 PWM output mode - Right align . . . 804
33.4.2.5 PWM output mode – Asymmetric with cc0 & cc1 . . . 806
33.4.2.6 PWM output mode – Center align . . . 807
33.4.2.7 PWM output mode – Center align asymmetrical with cc0 . . . 813
33.4.2.8 PWM output mode – Center align asymmetric with cc0 and cc1 . . . 813
33.4.3 PWM_DT . . . 813
33.4.3.1 Computation concept . . . 814

33.4.3.2 PWM_DT output mode – left align . . . 815
33.4.3.3 PWM_DT output mode – right align . . . 815
33.4.3.4 PWM_DT output mode – center align . . . 816
33.5 Motion interface (MOTIF) . . . 818
33.5.1 Overview . . . 818
33.5.1.1 Features . . . 818
33.5.1.2 Block diagram . . . 818
33.5.2 MOTIF functional description . . . 819
33.5.2.1 MOTIF MMIO registers . . . 820
33.5.2.2 MOTIF Pin mapping . . . 821
33.5.2.3 MOTIF clocks . . . 822
33.5.2.4 MOTIF reset . . . 823
33.5.2.5 Input trigger mapping . . . 823
33.5.2.6 Output trigger mapping . . . 823
33.5.2.7 Function selector connection scheme . . . 823
33.5.3 Hall sensor control . . . 825
33.5.3.1 Hall Sensor standard operation . . . 825
33.5.3.2 Hall Sensor operation with LUT . . . 831
33.5.4 Quadrature decoder control . . . 836
33.5.4.1 Quadrature clock and direction decoding . . . 841
33.5.4.2 Index control . . . 843
33.5.5 Stand-alone multi-channel mode . . . 844
33.5.6 Modulation interface from MOTIF . . . 846
33.5.6.1 Modulation inside counter logic . . . 850
33.5.7 Synchronous start . . . 850
33.5.8 Output selector . . . 851
33.5.9 Using the MOTIF . . . 852
33.5.10 Service request generation . . . 860
33.5.10.1 Hall sensor mode flags . . . 860
33.5.10.2 Quadrature decoder flags . . . 862
33.5.11 MOTIF debug behavior . . . 864
33.5.12 Initialization and System dependencies . . . 865
33.5.12.1 Initialization . . . 865
33.5.12.2 System dependencies . . . 866
33.6 Recovery . . . 866
33.7 Initialize . . . 866
33.8 Pin status . . . 866
33.9 Registers summary . . . 866
34 PPCA Coordinate Rotation Digital Computer (CORDIC) . . . 870
34.1 Features . . . 870
34.2 Architecture . . . 871
34.2.1 Block Diagram . . . 871

34.2.1.1 MMIO . . . 872
34.2.1.2 CORDIC_KERNEL . . . 872
34.3 Functional Description . . . 872
34.3.1 Operation . . . 872
34.3.2 Normalized Result Data . . . 873
34.3.3 Operating Modes . . . 874
34.3.4 Domains of Convergence . . . 875
34.3.5 Overflow Considerations . . . 876
34.3.6 Data Format . . . 876
34.3.7 Performance . . . 877
34.3.8 Look-Up Tables . . . 877
34.3.9 Interrupt Generation . . . 879
34.3.10 Notes on MMIO Access . . . 880
34.4 Power, Clock, and Reset . . . 880
34.4.1 Power Supply . . . 880
34.4.2 Clock . . . 880
34.4.3 Reset . . . 880
34.5 Registers summary . . . 880

35 Low-power comparator (LPCOMP) . . . 882
35.1 Features . . . 882
35.2 Architecture . . . 882
35.2.1 Input configuration . . . 883
35.2.2 Output and interrupt configuration . . . 883
35.2.3 Power mode and speed configuration . . . 885
35.2.4 Hysteresis . . . 886
35.2.5 Wake up from low-power modes . . . 886
35.2.6 Comparator clock . . . 886
35.2.7 Using low-power comparator . . . 886
35.3 Registers summary . . . 887

36 Supply Voltage Glitch Sensor (SVGS) . . . 888
36.1 Introduction . . . 888
36.2 Features . . . 888
36.3 Architecture . . . 888
36.4 Functional description . . . 888
36.5 Registers summary . . . 893

Glossary . . . 894
Revision history . . . 899
Trademarks . . . 900
Disclaimer . . . 901

1 Introduction

This document provides an overview of the key features and specifications of Infineon's PSOC™ Control C3 (P/M) (8/7) series of microcontrollers, designed for industrial applications. These microcontrollers feature an ARM® Cortex® - M33 CPU and integrated programmable peripherals. Optimized for real-time applications, and are manufactured using an advanced 40-nm process. The microcontrollers offer a range of features that support efficient and secure operation, including low-power modes, advanced security features, and a flexible clocking system. Additionally, they provide rich communication peripherals, advanced digital and analog peripherals, and are compatible with industry-standard development tools.

1.1 Detailed features

This device supports the following features:

- **CPU subsystem**:
  - Arm® Cortex®-M33 processor running at up to 180 MHz
  - Digital signal processor (DSP), floating-point unit (FPU), memory protection unit (MPU), and 16 KB instruction cache (I-cache)
  - One direct memory access (DMA) controller with 16 channels
- **Security**:
  - Platform security architecture level 3 (PSA L3) ready
  - Features secured boot, secured firmware update, secured debug, and secured RMA mode for field failure
  - Hardware and software isolation with TrustZone®
  - Infineon-proprietary protection using protection contexts, a memory protection controller (MPC), and a peripheral protection controller (PPC)
  - Cryptography accelerator
    - Hardware acceleration for symmetric-key cryptography using AES-128 and AES-256
    - Vector unit (VU) for accelerating asymmetric-key cryptography, such as RSA and ECC
    - Hash functions support SHA-256,SHA-384, and SHA-512
    - True random number generator (TRNG) function
    - ECC signature verification: ECDSA NIST P-256, ECDSA NIST P-384, and ECDSA NIST P-521
  - Supply voltage glitch sensor (SVGS) designed to detect supply-voltage manipulation intended to disrupt device operation
  - Support for device identifier composition engine (DICE) mechanisms
- **Memory**:
  - On-chip flash memory:
    - Up to 512 KB flash with error correction code (ECC) support, RWW capability, 64 KB ROM for boot code, and bootloader functions, 32 KB reserved for the supervisory region
    - Built-in device firmware upgrade (DFU) support in the boot ROM via a serial interface (UART)
  - Static random access memory (SRAM):
    - 128 KB SRAM with ECC support
    - Organized as two independent 64 KB blocks (SRAM0 and SRAM1)
    - Full 64 KB + 64 KB SRAM is available in Deep Sleep mode
    - Protected SRAM data path using hardware ECC for soft error detection and correction
- **Clocking subsystem**:
  - 8 MHz IMO with Deep Sleep operation, offering ±2% accuracy

- 48 MHz internal high-frequency oscillator (IHO) with ±1% accuracy
- 4 to 35 MHz external crystal oscillator (ECO) with a phase-locked loop (PLL) for CPU and subsystem clocking
- 32 kHz external watch crystal oscillator (WCO), usable for real-time clock (RTC) applications
- 32 kHz internal low-frequency oscillator (ILO) with ±10% accuracy
- External clock (EXTCLK): Maximum frequency 100 MHz
- One frequency lock loop (FLL) with 25 MHz to 100 MHz output range
- Two digital phase-locked loops, DPLL#0 and DPLL#1, with 50 MHz to 200 MHz output range

- **Low-power operation**:
  - Operates at low power with a voltage range of 1.71 V to 3.63 V
  - Supports six power modes for fine-grained power management:
    - Active
    - Sleep
    - Deep Sleep
    - Deep Sleep-RAM
    - Deep Sleep-OFF
    - Hibernate
  - Four power profiles, Ultra-Low Power, Medium Frequency, Low Power, and Overdrive), to balance performance and power consumption in Active mode
- **Communication peripherals**:
  - **Serial communication blocks (SCBs)**:
    - Up to five independent run-time-reconfigurable SCBs:
      - Each SCB is software-configurable as I^2C, SPI, or UART in master or slave mode
    - Two additional run-time-reconfigurable SCBs with PMBus® protocol support:
      - Software-configurable as master or slave for I^2C, SPI, or UART protocols
      - PMBus® protocol features:
        - Baud rates of up to 1 Mbps
        - Timeout event generation (TGS) for wrong or stuck commands
        - Hardware address screening with support for a minimum of five slave address masks (MSAs) to reject or NACK incoming address commands
        - Compliance with PMBus® and SMBus specifications
      - One SCB instance supports wakeup from Deep Sleep mode with PMBus® extensions
    - **Additional features**:
      - One SCB supports operation in Deep Sleep mode and wakeup from Deep Sleep in I^2C slave and SPI slave modes
      - One SCB supports fast SPI operation with speeds of up to 50 MHz
      - SCBs support single-wire, half-duplex mode for UART communication
  - **CAN FD**:
    - Supports up to two CAN FD channels
    - Each channel operates at speeds of up to 8 Mbps

- **I/O subsystem**:
  - **Programmable GPIO pins**:
    - Supports up to 86 functional pins, including 60 digital GPIOs, 24 dedicated analog-only inputs (14 of the 60 GPIOs are analog inputs capable), and two dedicated analog pins configurable as inputs or outputs only
    - Provides programmable drive modes, strengths, and slew rates
    - Six of the 60 GPIOs as *GPIO_OVT* to support PMBus®/SMBUS compliance with input leakage specifications
    - Up to four *GPIO_OVT* pins can be routed to analog inputs of ATOP and used by the ADC or comparator
- **Low-power comparator (LPComp)**:
  - Supports a pair of low-power comparators
  - Provides comparator thresholds for comparison, which can be sourced either from an external pin or an internal fixed reference of approximately 0.6 V ± 0.2 V
- **Timer/counter pulse-width modulator (TCPWM)**:
  - One group provides two 32-bit TCPWM channels
  - Another group provides two 16-bit TCPWM channels
  - Both TCPWM outputs and inputs are exclusively connected to the event processing unit (EPU) and trigger MUX
  - One instance is clocked using a stable clock source and can be used for monitoring the main CPU
  - The second instance is clocked using a robust clock source to ensure reliable heart-beat monitoring of the PPCA and can be used for monitoring the PPCA CPUs
- **Programmable power control accelerator subsystem (PPCA)**:
  - PPCA CPU subsystem:
    - **CPUs**:
      - Two Arm® Cortex®-M33 processors (CPU0 and CPU1), each operating at up to 200 MHz
      - Digital signal processor (DSP), floating-point unit (FPU)
    - **CPUs memories**:
      - Five instances of SRAM memory modules are available
      - SRAM supports ECC, allowing them to detect two errors and correct one error
      - Four local memory blocks with the following sizes: 32 KB, 16 KB, 32 KB, and 16 KB. The first two blocks are assigned to the first PPCA CPU (CPU0) for code and data. The other two blocks are allocated to the second PPCA CPU (CPU1) for code and data. If CPU1 is not used, these blocks serve as code extension and data extension for CPU
      - A 16 KB shared memory block is accessible by both PPCA CPU0 and PPCA CPU1. This memory can also be used as a data extension for either CPU0 or CPU1
    - **IPC block**:
      - An IPC block facilitates communication between PPCA CPU0 and PPCA CPU1. This block operates through the shared 16 KB SRAM
      - The IPC block supports one group, four channels, and four interrupts
    - **AMBA bus infrastructure**:
      - Includes a bus matrix, five AHB slave inputs, and 12 AHB master outputs

- **Analog TOP subsystem (ATOP SS)**:
  - **Analog-to-digital converter (ADC)**:
    - Four 12-bit (greater than 10-bit ENOB) SAR ADCs (ADC0 - ADC3) with a conversion rate of up to 25 Msps
    - Supports up to 12 dedicated differential analog input channels or up to 24 dedicated single-ended input channels
    - An option to connect 16 multiplexed additional analog inputs to two analog input channels
    - Input voltage range: –0.4 V to 3.3 V
    - Enables simultaneous sampling, arbitrary sampling, or continuous programmable sampling sequences
    - A selectable internal or external 1.2 V ADC voltage reference
    - An analog front end (AFE) with selectable input gain values of ×8.25, ×16.5, ×33, and ×66
    - Initiates the start of conversion (SOC) via a dedicated hardware signal per channel, per group of channels, or by writing into the ADC trigger register
    - Aligns the end of sampling (EOS) with the provided SOC or by using the control ADC conversion FSM
    - Provides a configurable ADC trigger source from the event processing unit (EPU)
  - **Digital comparators (DCMP)**:
    - Features digital comparator outputs directed to the EPU
    - Up to 12 digital comparators with configurable digital thresholds
    - Generates an event when the input signal crosses the programmed threshold
    - Supports configurable comparison methods including immediate comparison, integration of immediate comparison, accumulation of immediate comparisons, and averaged comparisons
  - **ADC filters (ADC FILTs)**:
    - The ADC embeds a configurable set of digital filters, including a median filter, linear interpolation/extrapolation filter, low-pass filter, cascaded integrator third-order comb filter, configurable average filter, detection of minimum values in the input data stream, and detection of maximum values in the input data stream
    - Supports up to 64× oversampling, providing 15-bit effective resolution on up to six ADC channels
    - Filters can be chained together and optionally sourced from the output of another ADC core
    - The MIN/MAX module can be sourced either by the input or the output of the median filter
  - **Differential analog comparator with slope generator (DCSG)**:
    - Features 9 active, configurable differential or single-ended comparators with a worst-case comparator delay of 5 ns
    - A differential analog comparator with an embedded DAC for programmable thresholds or references
    - Employs a dynamic latch comparator architecture, supporting a maximum clock frequency of 200 MHz
    - Three comparators with selectable digital slope generator
    - Supports a selectable input voltage range of 0 V to 3.3 V or ±200 mV
    - Ensures a maximum jitter of one clock cycle and a propagation delay of less than two clock cycles
    - Offset matches DAC resolution, providing ±0.7 mV for a 3.3 V rail-to-rail range and a 12-bit DAC
    - Supports 12-bit DAC threshold settings with differential non-linearity (DNL) less than four LSB

    - Ensures 10-bit monotonicity and a maximum DAC refresh rate of 50 Msps (minimum 20 Msps)
    - Guarantees less than 10 clock cycles settling time for 400 code jumps with an accuracy of ±8 LSB
    - Integrates a DAC refresh trigger function with a blanking time of less than 12 clock cycles
    - Supports A versus B comparisons
    - Provides comparator trigger inputs and outputs connected to the event processing unit (EPU) module
  - **Digital-to-analog converter (DAC) R2R**:
    - Two 12-bit single-ended DACs with a refresh rate of 2 MHz
    - Ensures 10-bit monotonicity
    - Selectable DAC output ranges, including 0 V to VREF (1.2 V), 0 V to VDDA/2, ±0.2 V, or ±VDDA
    - Provides a selectable buffered DAC output full-scale range of 0.1 V to VDDA/2 or 0.1 V to VREF
    - Features a 50 pF load driver
  - **Additional features**:
    - Temperature sensor (TSNS) with an accuracy of ±5°C
    - Provides a current reference (AREF) for ATOP subsystem resources
    - Voltage reference (VREF) for ATOP SS resources
    - Features a current digital-to-analog converter (IDAC) module to generate a selectable, known current for sensing external resistance
    - General-Purpose Input (GPI) module to convert an analog input into a digital output by using fixed analog thresholds
- **Event processing unit (EPU)**:
  - Facilitates flexible management of events generated by peripherals to create triggers
  - Supports trigger generation through a linear combination of processed input events and allows routing of input triggers as interrupts to the PPCA CPUs
  - Configurable and programmable event processing system, comprising 32 Type-1 processing units and 16 Type-2 processing units
  - Processes selected inputs connected to events generated by CPUs and peripheral resources, with outputs connected to triggers of CPUs and peripheral resources
  - Supports various functionalities, including asynchronous operation, synchronous bypass, input polarity inversion, and rising-edge extraction
  - Integrates filtering capabilities such as masking filters, blanking filters, de-bouncing filters (through integration), and pulse delays of up to 128 clock cycles
- **Hardware filter (HWFILT3P3Z)**:
  - Supports two instances of the hardware filter (HWFILT3P3Z), each of which can be allocated to one of the PPCA CPUs
  - The first instance supports four 3Poles3Zeros (3P3Z) hardware filters
  - The second instance supports two 3Poles3Zeros (3P3Z) hardware filters
  - Features a 16-bit input and a 24-bit output
  - A 40-bit multiplier and a 44-bit internal accumulator
  - Ensures execution latency of up to eight CPU clock cycles
  - Provides support for arithmetic 2's complement and saturation operators
  - Programmable dedicated data path for enhanced performance

- **Custom logic block (CLB)**:
  - The CLB module provides dedicated hardware support for specific functions or control algorithms that require high computational resources
  - Examples of supported functions include digital observers used in Totem Pole PFC controllers and matrix converter commutation
  - Additionally, the CLB module supports trajectory control and spread-pulse start-up functionalities
- **Coordinate Rotation Digital Computer (CORDIC)**:
  - Supports all CORDIC operating modes for solving circular (trigonometric) and hyperbolic functions
  - Integrates independent lookup tables to accelerate calculations for enhanced performance
- **Timer/counter pulse-width modulation (TCPWM)**:
  - Four instances of TCPWM (TCPWM0 - TCPWM3) are available within the PPCA
  - All counters (PERIOD, DUTY, CC0, CC1, DT) support greater than or equal to (≥) match as well as equal to (=) match conditions
  - Center-aligned, edge modes
  - Shadow update of duty, period, dead-time, output signal polarity
  - Comparator-based triggering of kill signals
  - Trigger inputs for all instances are connected to the event processing unit (EPU) outputs, while trigger outputs for all instances are connected to the EPU inputs
  - All four instances operate synchronously with the PPCA CPUs and run on the PPCA clock, with a maximum frequency of 200 MHz
  - HRPWM feature for period, duty, and dead-time insertion with a typical resolution of less than 100 ps
  - **First TCPWM instance (TCPWM0)**:
    - Eight 32-bit counters with high-resolution PWM (HRPWM) functionality
    - Parallel input data path for CC0 and CC1 registers enabled for four 32-bit counters
    - Four 16-bit counters, with the 16-bit counter group supporting MOTIF
    - PWM outputs for all 16-bit and 32-bit counters are connected to HSIOM
  - **Second TCPWM instance (TCPWM1)**:
    - Four 32-bit counters with HRPWM functionality
    - Parallel data path for CC0 and CC1 enabled for two 32-bit counters
    - Four 16-bit counters, with the 16-bit counter group supporting MOTIF
  - **Third TCPWM instance (TCPWM2)**:
    - Four 16-bit counters
    - Trigger inputs and outputs are connected to the EPU
    - Trigger outputs for PWM outputs are also connected to the trigger MUX
  - **Fourth TCPWM instance (TCPWM3)**:
    - Two 16-bit counters
    - Trigger inputs and outputs are connected to the EPU
    - Trigger outputs for PWM outputs are also connected to the trigger MUX

1.2 Architectures

A detailed block diagram of the MCU is shown in Figure 1.

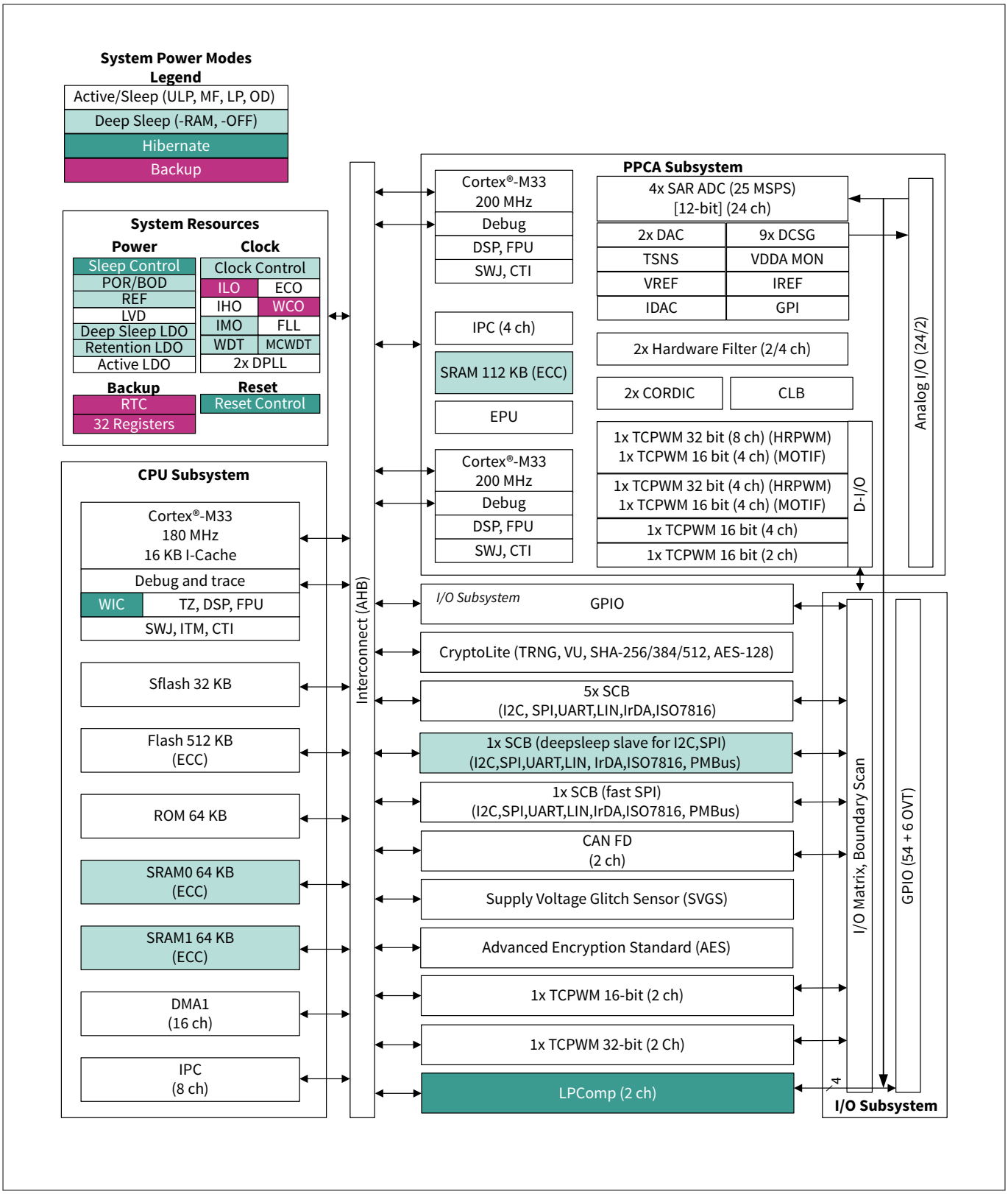


Figure 1 Functional block diagram

Device identification and revisions

Family ID = 0x119 (12-bit); Si ID range = EE80 - EE96; Major-minor rev ID = 0x1, 0x1

1.3 Document organization and conventions

1.3.1 Major sections

The information in this reference manual is organized as follows:

- Chapter – Presents the chapters specific to an individual aspect of the device. These are the detailed implementation and use information for some aspect of the integrated circuit
- Registers summary – Summary of registers associated with the chapter
- Registers Reference Manual - Supplies details of all register device summarized in the reference manual. This is an additional document

1.3.2 Document conventions

1.3.2.1 Register conventions

Register conventions are detailed in the Registers Reference Manual.

1.3.2.2 Numeric naming

Hexadecimal numbers are represented in uppercase with an appended lowercase ‘h’ (for example, ‘14h’ or 3Ah) and hexadecimal numbers may also be represented by a ‘0x’ prefix, as per 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.

1.3.2.3 Unit of measure

Units of measure lists the units of measure used in this document.

Table 1 Units of measure

Abbreviation	Unit of measure
bps	bits per second
°C	degrees Celsius
dB	decibels
dBm	decibels-milliwatts
fF	femto farads
G	Giga
Hz	Hertz
k	kilo, 1000
K	kilo, 2^10
KB	1024 bytes, or approximately 1000 bytes

(table continues...)

Table 1 (continued) Units of measure

Abbreviation	Unit of measure
Kbit	1024 bits
kHz	kilo Hertz (32.000)
kΩ	kilo Ohms
MHz	mega Hertz
MΩ	mega Ohms
µA	micro Amperes
µF	micro Farads
µs	micro seconds
µV	micro volts
µVrms	micro volts root-mean-square
mA	milli amperes
ms	milli seconds
mV	milli Volts
nA	nano Amperes
ns	nano seconds
nV	nano Volts
Ω	ohms
pF	pico Farads
pp	peak-to-peak
ppm	parts per million
SPS	samples per second
σ	sigma: one standard deviation
V	Volts

2 CPU subsystem (CPUSS)

The CPU subsystem is based on the 32-bit Arm® Cortex®-M33 (CM33) CPU. It is designed for security, control, communication, short interrupt response time, high code density, and high 32-bit throughput while maintaining a strict cost and power consumption budget.

This section provides only an overview of the Arm® Cortex® CPU in PSC3M/P8, PSC3M/P7 MCU. For details, see the Arm® documentation sets for cortex®-M33.

2.1 Features

The PSC3M/P8, PSC3M/P7 MCU Arm® Cortex®-M33 (CM33) CPU has the following features:

- Maximum clock frequency of 180 MHz @ over drive (OD) mode
- Single-precision floating-point unit (FPU) and DSP extension
- Memory protection unit (MPU) with up to 8 regions per security state (secure and non-secure)
- Nested vectored interrupt controller (NVIC) integrated with the CPU core yielding low interrupt latency
- Wakeup interrupt controller (WIC) that allows the CPU to wake up from the Deep Sleep power mode through Deep Sleep interrupts
- 16 KB instruction cache (I-cache) on Cortex®-M33 C-AHB bus with four-way set associativity
- Supports TrustZone-M memory mapping for secure/non-secure aliasing based on programmable secure attribution unit (SAU) and a fixed implementation defined attribution unit (IDAU)

2.2 Architecture

CM33 is a 32-bit processor with a 32-bit data path, 32-bit registers, and a 32-bit memory interface. The CPU supports a wide variety of instructions in the thumb instruction set. The CM33 CPU supports thread and handler operating modes (refer to the Operating modes and privilege levels section). Additionally, the CM33 CPU supports secure and non-secure security states based on the security extension.

The CM33 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 2 to 11 cycles (depending on values)
- DSP instructions
- Complex memory load and store access
- Complex bit manipulation

The CM33 processor with FPU supports single precision arithmetic as defined by the FPv5 architecture. This is compliant with the ANSI/IEEE Std 754-2008, IEEE standard for binary floating-point arithmetic.

2.2.1 CPUSS

The PSC3M/P8, PSC3M/P7 CPUSS is a security, control, and communication CPU that is built on a Cortex®-M33 (CM33) core and runs at up to 180 MHz.

The CM33 CPU is a 3-stage pipeline (3PIP) from Arm® based on the Arm® v8-M architecture having a low-power domain with rapid clock switching. The CPUSS implements Arm® TrustZone-M compatible security and protection infrastructure. It provides the initial boot for security and protection schemes.

- The CM33 CPU is a TrustZone-M CPU that implements an inbuilt security extension. As part of the security extension, the CM33 implements a secure attribution unit (SAU) inside the CPU. The TrustZone-M

architecture also defines an external interface to the CPU, which can be connected to an implementation defined attribution unit (IDAU)

- For non-CPU masters in CPUSS, a master security controller (MSC) is instantiated along with an implementation defined attribution unit (IDAU) to distinguish secure/non-secure transfers. For more details, refer to the TrustZone section

The CPUSS has a 16 KB 4-way set associative instruction cache on the C-AHB bus. CPUSS supports on-chip SRAM controller and memories. It provides C-AHB bus access from SRAM for code execution. The CPUSS includes a 128-bit AHB5 bus interface for instruction fetch and data access from FLASH (NVM), as well as internal SRAMs, which are accessible on their aliased addresses. Also it supports a 32-bit AHB5 bus interface to access the memory-mapped Flash, internal SRAM present in the system memory map.

The CPUSS includes two DMA controllers supporting 16 channels with Arm® Q-channel support. These channels are primarily used to service the SCB, CORDIC, and TCPWM peripherals. The CPUSS includes four inter-processor communication (IPC) hardware for secure and non-secure applications to communicate and synchronize their activities.

The CPUSS has clk_hf0 clock domain for the majority of hardware modules.

- clk_hf0: Up to 180 MHz @ 1.2 V. Used by majorities of the CPUSS including I-Cache, AHB bus infrastructures, DMA, MSC/IDAU/MPC, and all on-chip memory controllers running at a single synchronous frequency

The CPUSS implements a centralized fault reporting infrastructure (only a single fault interrupt handler is required), which captures faults related to (not an exclusive listing) MPC/PPC violations, peripheral-specific errors, memory controller-specific errors such as ECC errors, and time-out errors.

In CPUSS, the active power domain becomes active or initialized first (by default, following the power-on reset (POR)). Refer the device power modes section for more details.

2.2.2 Address and memory maps

The CPU has a fixed address map. The default memory map of the CPU covers the range 0x00000000 - 0xFFFFFFFF. For the default memory map, see the memory model section of Arm® Cortex®-M33 Processor Technical Reference Manual. This 32-bit (4 GB) address space is further partitioned into non-secure and secure memory regions as shown in Table 2. Based on Arm® TrustZone, bit [28] of the address defines if a memory is secure or non-secure, resulting in the following memory map.

Table 2 PSC3M/P8, PSC3M/P7 MCU address and memory map

Access type	Address range	Subregion name	Bank mode	Size	Region name - interface
Non-secure	0x22000000 – 0x2207FFFF	FLASH	Single Bank	512 KB	S-bus
Secure	0x32000000 - 0x3207FFFF			512 KB	
Non-secure	0x02000000 - 0x0207FFFF		Single Bank	512 KB	C-bus
Secure	0x12000000 - 0x1207FFFF			512 KB	
Non-secure	0x22000000 - 0x2203FFFF		Dual bank (Bank 0)	256 KB	S-Bus

(table continues...)

Table 2 (continued) PSC3M/P8, PSC3M/P7 MCU address and memory map

Access type	Address range	Subregion name	Bank mode	Size	Region name - interface
	0x22800000 - 0x2283FFFF		Dual bank (Bank 1)	256 KB	
Secure	0x32000000 - 0x3203FFFF		Dual bank (Bank 0)	256 KB	
	0x32800000 - 0x3283FFFF		Dual bank (Bank 1)	256 KB	
Non-secure	0x02000000 - 0x0203FFFF		Dual bank (Bank 0)	256 KB	C-Bus
	0x02800000 - 0x0283FFFF		Dual bank (Bank 1)	256 KB	
Secure	0x12000000 - 0x1203FFFF		Dual bank (Bank 0)	256 KB	
	0x12800000 - 0x1283FFFF		Dual bank (Bank 1)	256 KB	
Non-secure	0x04000000 – 0x0400FFFF	SRAM0	N/A	64 KB	C-Bus
Secure	0x14000000 – 0x1400FFFF		N/A	64 KB	
Non-secure	0x24000000 - 0x2400FFFF		N/A	64 KB	S-Bus
Secure	0x34000000 - 0x3400FFFF		N/A	64 KB	
Non-secure	0x04010000 – 0x0401FFFF	SRAM1	N/A	64 KB	C-Bus
Secure	0x14010000 – 0x1401FFFF		N/A	64 KB	
Non-secure	0x24010000 - 0x2401FFFF		N/A	64 KB	S-Bus
Secure	0x34010000 - 0x3401FFFF		N/A	64 KB	
Non-secure	0x23400000 - 0x23407FFF	SFLASH	N/A	32 KB	S-Bus
Secure	0x33400000 - 0x33407FFF		N/A	32 KB	S-Bus

(table continues...)

Table 2 (continued) PSC3M/P8, PSC3M/P7 MCU address and memory map

Access type	Address range	Subregion name	Bank mode	Size	Region name - interface
Non-secure	0x03400000 - 0x03407FFF		N/A	32 KB	C-Bus
Secure	0x13400000 - 0x13407FFF		N/A	32 KB	C-Bus
Non-secure	0x00800000 – 0x0080FFFF	Reserved	N/A	64 KB	ROM region
Secure	0x10800000 – 0x1080FFFF		N/A	64 KB	
Non-secure	0x42000000 - 0x43FFFFFF	SYSCPUSS peripheral (SYS MMIO)	N/A	32 MB	Peripheral region
Secure	0x52000000 - 0x53FFFFFF		N/A	32 MB	

***Note**: Secure and non-secure memory pairs are the same physical memory but mapped differently depending on whether they are configured as secure or non-secure.*

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. Coherency of caches during PC (protection context) switches are all maintained by the software. For CM33, all accessible address spaces in FLASH are cacheable.

All peripheral address spaces specified in 0x4000_0000/0x5000_0000 are NOT cacheable. The XIP execution instruction from this address space causes MemFault. Address aliasing only applies to SRAM when accessed by the CM33 processor via code bus to facilitate code execution out of SRAM memory.

***Note**: The CM33_S_VECTOR_TABLE_BASE, CM33_NS_VECTOR_TABLE_BASE registers determine the location of the secure and non-secure vector table for the CM33 CPU.*

2.2.2.1 Wait state lookup tables

The wait state lookup tables show the wait states for different memory modules.

Table 3 Memory wait state

Parameter	Symbol	Values			Note or test condition
		Min	Typ	Max	
Memory wait states					
Wait states	$M33_{cache}$	-	-	0	
Wait states from CM33	$M33_{SRAM}$	-	0	3	Max is 3 with ECC enabled

(table continues...)

Table 3 (continued) Memory wait state

Parameter	Symbol	Values			Note or test condition
		Min	Typ	Max	
Wait states from flash via C-bus (cached)	$M33_{Flash}$	3*	-	9	Depending on cache hit, pre-fetch and opcode
Wait states from flash via S-bus	$M33_{Flash}$	-	-	9	

***Note**: The wait state lookup tables are applicable for all power modes (ULP/MF/LP/OD).*

***Note**: [*] $M33_{Flash}$Min is 0 on cache hit*

2.3 Operating modes and privilege levels

The CM33 CPU supports both secure and non-secure states. The CPU has thread and handler operating modes and can run in either thumb or debug operating states. In addition, the processor can limit or exclude access to some resources by executing code in privileged or unprivileged mode.

See the Arm® v8-M Architecture Reference Manual for more information about the modes of operation and execution.

2.3.1 Security states

When the Arm® v8-M Security Extension is included in the processor, the programmer model includes two orthogonal security states: secure and non-secure. The CM33 CPU has a security extension enabled it always resets into the secure state.

Registers in the system control space are banked across secure and non-secure states and the non-secure register view is available at an aliased address to the secure state.

2.3.2 Operating modes

For each security state, the processor can operate in thread or handler mode.

- **Thread mode**: Used to execute application software. The processor enters the thread mode on reset, or as a result of an exception return to the thread mode. The thread mode includes privileged and unprivileged codes
- **Handler mode**: Used to handle exceptions. The processor enters the handler mode as a result of an exception. The handler mode includes only the privileged code

2.3.3 Operating states

The processor can operate in the thumb or debug state:

- The thumb state is the state of normal execution running 16-bit and 32-bit half-word aligned thumb instructions
- The debug state is the state when the processor is in halting debug

2.3.4 Privilege levels

- **Unprivileged:** The software has limited access to the move from the general-purpose register to the special register (MSR) and move from special register to general-purpose register (MRS) instructions, and cannot use the change processor state/disable interrupts (CPSID) and change processor state/enable interrupts (CPSIE) instructions. It cannot access the system timer, NVIC, or system control block. Restricted access to memory or peripherals
- **Privileged:** The software can use all the instructions and has access to all resources

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

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

In the 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.

2.4 Instruction set

The CM33 CPU supports all the 16-bit thumb instructions defined by the Arm®v8-M architecture. In addition to that, the processor implements the T32 instruction set based on the thumb-2 technology, ensuring high code density and reduced program memory requirements.

The CM33 CPU implements the following instruction from Arm®v8-M:

- All base instructions
- All instructions in the main extension
- All instructions in the security extension when security extension is enabled
- All instructions in the DSP extension (optional)
- All single-precision instructions and double-precision load and store instructions in the floating-point extension when FPU is enabled

For more information about these instructions, see section 3 - The Cortex®-M33 Instruction Set of the Arm® Cortex®- M33 Devices Generic User Guide.

The processor also implements custom datapath extension (CDE) instructions. The CDE introduces 2×3 classes of instructions in the coprocessor instruction space:

- Three classes operate on the general-purpose register file
- Three classes operate on the floating-point register file

For specific information on the CDE instructions implemented in the processor, see section 9. Arm® custom instructions in Arm® Cortex®-M33 processor RM.

2.5 Registers summary

The CM33 CPU has the following 32-bit core registers.

When the security extension is included, some of the registers are banked. The secure view of these registers is available when the CPU is in the secure state and the non-secure view when the CPU is in non-secure state.

Table 4 CM33 core register set summary

Name	Type[1]	Required privilege[2]	Reset value	Description
R0-R12	RW	Either	UNKNOWN	R0-R12 are general-purpose registers for data operations.
MSP (R13)	RW	Either	-[3]	The stack pointer (SP) is register R13. In Thread mode, the CONTROL register indicates the stack pointer to use, Main Stack Pointer (MSP) or Process Stack Pointer (PSP). When the Armv8-M Security Extension is included, there are two MSP registers in the Cortex® -M33 processor: • MSP_NS for the non-secure state • MSP_S for the secure state When the Armv8-M Security Extension is included, there are two PSP registers in the Cortex® -M33 processor: • PSP_NS for the non-secure state • PSP_S for the secure state When the Armv8-M Security Extension is included, there are two PSP registers in the Cortex® -M33 processor: • PSP_NS for the non-secure state • PSP_S for the secure state
PSP (R13)	RW	Either	UNKNOWN	
MSPLIM	RW	Privileged	0x00000000	The stack limit registers limit the extent to which the MSP and PSP registers can descend respectively. When the Armv8-M Security Extension is included, there are two MSPLIM registers in the Cortex®-M33 processor: • MSPLIM_NS for the non-secure state • MSPLIM_S for the secure state When the Armv8-M Security Extension is included, there are two PSPLIM registers in the Cortex®-M33 processor: • PSPLIM_NS for the non-secure state • PSPLIM_S for the secure state
PSPLIM	RW	Privileged	0x00000000	
LR (R14)	RW	Either	UNKNOWN	The Link Register (LR) is register R14. It stores the return information for subroutines, function calls, and exceptions.
PC (R15)	RW	Either	-[3]	The Program Counter (PC) is register R15. It contains the current program address.
xPSR	RW	Either	-[4]	The Program Status Register (PSR) combines: • Application Program Status Register (APSR) • Interrupt Program Status Register (IPSR) • Execution Program Status Register (EPSR) These registers provide different views of the PSR.
APSR	RW	Either	UNKNOWN	The APSR contains the current state of the condition flags from previous instruction executions.

(table continues...)

Table 4 (continued) CM33 core register set summary

Name	Type[1]	Required privilege[2]	Reset value	Description
IPSR	RO	Privileged	0x00000000	The IPSR contains the exception number of the current ISR.
EPSR	RO	Privileged	-[4]	The EPSR contains the thumb state bit and the execution state bits for the If-then (IT) instruction, and interruptible-continuable instruction (ICI) field for an interrupted load multiple or store multiple instruction.
PRIMASK	RW	Privileged	0x00000000	The PRIMASK register prevents activation of exceptions with configurable priority. When the Armv8-M Security Extension is included, there are two PRIMASK registers in the Cortex®-M33 processor: • PRIMASK_NS for the non-secure state • PRIMASK_S for the secure state
BASEPRI	RW	Privileged	0x00000000	The BASEPRI register defines the minimum priority for exception processing. When the Armv8-M Security Extension is included, there are two BASEPRI registers in the Cortex®-M33 processor: • BASEPRI_NS for the non-secure state • BASEPRI_S for the secure state
FAULTMASK	RW	Privileged	0x00000000	The FAULTMASK register prevents activation of all exceptions except for NON-MASKABLE INTERRUPT (NMI) and optionally secure HardFault. When the Armv8-M Security Extension is included, there are two FAULTMASK registers in the Cortex®-M33 processor: • FAULTMASK_NS for the non-secure state • FAULTMASK_S for the secure state
CONTROL	RW	Privileged	0x00000000	The CONTROL register controls the stack used, and optionally the privilege level, when the processor is in the thread mode. When the Armv8-M Security Extension is included, there are two CONTROL registers in the Cortex®-M33 processor: • CONTROL_NS for the non-secure state • CONTROL_S for the secure state

1) Describes the access type during program execution in the thread mode and the handler mode. Debug access can differ.
2) An entry of either means privileged and unprivileged software can access the register.
3) Soft reset to the value retrieved by the reset handler.
4) Bit[24] is the T-bit and is loaded from bit[0] of the reset vector. All other bits are reset to 0.

The CM33 floating-point unit (FPU) has the following registers:

- Thirty-two 32-bit single-precision registers, S0 to S31. These registers can also be addressed as sixteen 64-bit double-precision registers, D0 to D15
- The four FPU control and status registers are as follows:
  - FPCCR – Floating-point Context Control Register
  - FPCAR – Floating-point Context Address Register
  - FPSCR – Floating-point Status Control Register
  - FPDSCR – Floating-point Default Status Control Register

For more information on how these registers are used, see the Arm® Cortex®-M33 reference manual.

Use the MSR and MRS instructions to access the PSR, PRIMASK, CONTROL, FAULTMASK, and BASEPRI registers. The following tables show how the PSR bits are assigned. See the Combined Program Status Register section in the Arm® Cortex®-M33 reference manual for detailed information.

Table 5 CM33 APSR bit assignments

Bit	Name	Usage
31	N	Negative flag
30	Z	Zero flag
29	C	Carry or borrow a flag
28	V	Overflow flag
27	Q	DSP overflow and saturation flag
26:20	-	Reserved
19:16	GE[3:0]	Greater than or equal flags
15:0	-	Reserved

Table 6 CM33 IPSR bit assignments

Bit	Name	Usage
31:9	-	Reserved

(table continues...)

Table 6 (continued) CM33 IPSR bit assignments

Bit	Name	Usage
8:0	Exception number	This is the number of the current exceptions: • 0 = Thread mode • 1 = Reset • 2 = NMI • 3 = HardFault • 4 = MemManage • 5 = BusFault • 6 = UsageFault • 7 = SecureFault • 8-10 = Reserved • 7-10 = Reserved • 11 = SVCall • 12 = DebugMonitor • 13 = Reserved • 14 = PendSV • 15 =SysTick • 16 = IRQ0 • … 495 = IRQ479

Table 7 CM33 EPSR bit assignments

Bit	Name	Usage
31:27	-	Reserved, RES0
26:25, 15:10	ICI	Interruptible-continuable instruction bits
26:25, 15:10	IT	Indicates the execution state bits of the IT instruction
26:25, 11:10, 15:12	ECI	Exception continuation flags for beat-wise vector instructions. This field encodes which beats of the in-flight instructions have completed.
24	T	Thumb state bit
23:16	-	

3 Bus infrastructure

The bus infrastructure in a system-on-chip (SoC) forms the backbone of on-chip communication. It is a bus network that connects IP blocks such as processors, memory, peripherals, and accelerators. The bus infrastructure provides a standardized interface for data and control signals, enabling seamless interaction and data exchange among these components. It typically includes different bus types, such as system bus, code bus, peripheral bus, and interconnects, each serving a specific purpose. System buses and interconnects, such as the Advanced High-Performance Bus (AHB), support high-bandwidth communication between processors, memory, and other critical components.

The PSC3M/P8, PSC3M/P7 MCU family features SYSCPUSS, which uses an AHB5-based bus infrastructure that interconnects multiple bus masters and slaves. The bus interconnect uses bridges, upsizers, and downsizers to connect to other interconnects with varying bus widths, allowing multiple masters to access slaves and enabling concurrent access and efficient operation even when several high-speed peripherals operate simultaneously. This bus infrastructure for PSOC™ Control C3 devices is shown in Figure 2.

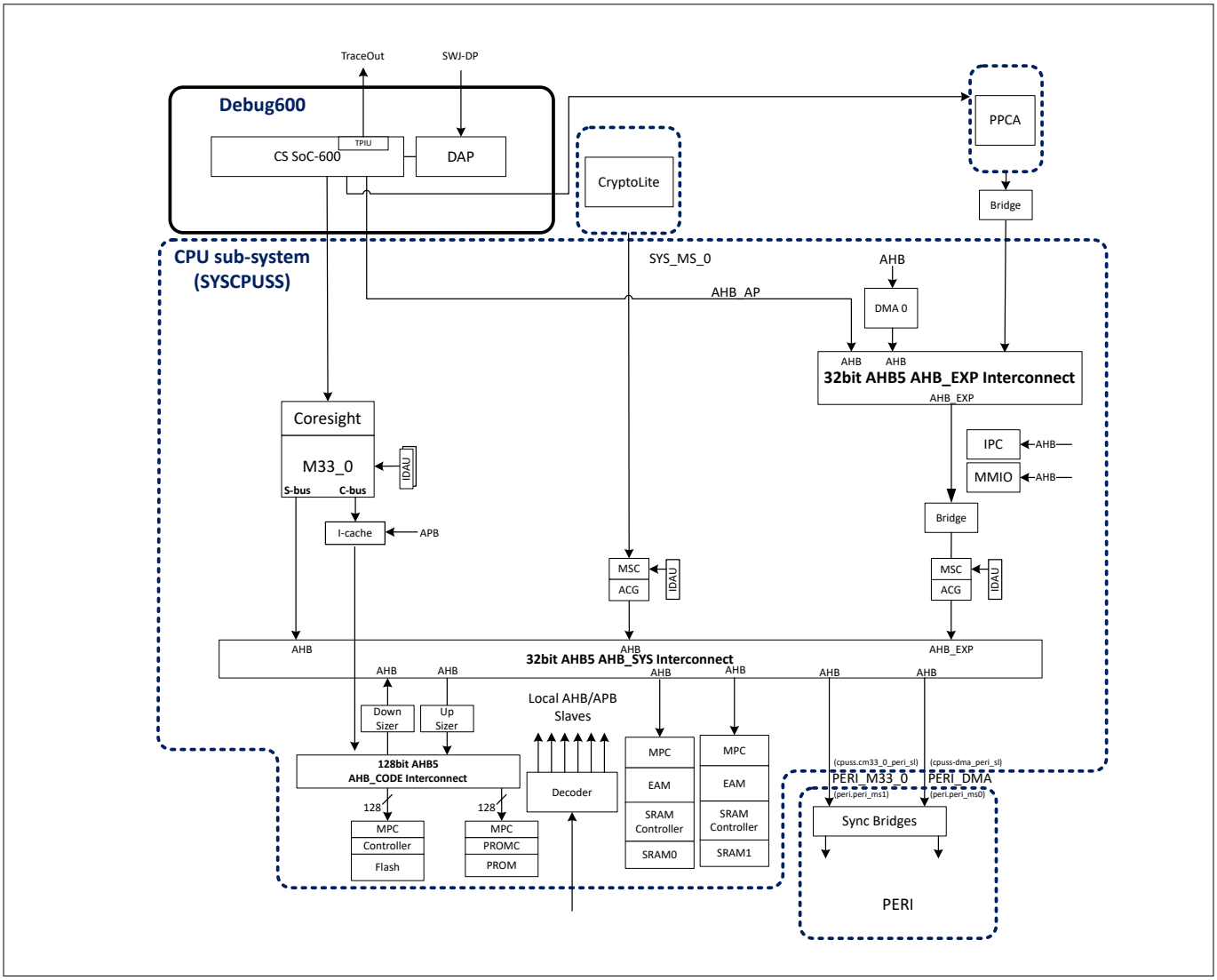


Figure 2 Bus infrastructure for PSC3M/P8, PSC3M/P7 devices

3.1 SYSCPUSS AHB5 interconnect

The SYSCPUSS (CM33 CPU) uses a 128-bit and 32-bit multilayer AHB5 bus matrix interconnect. The AHB5 bus masters may reside in the SYSCPUSS or outside of the SYSCPUSS (the SYSCPUSS external bus masters). Similarly, the AHB5 bus slaves may reside inside the SYSCPUSS (for example, the on-chip memory controllers)

or outside of the SYSCPUSS (for example, slaves connected to the peripheral interconnect (PERI) or to the external slave interfaces).

The SYSCPUSS AHB5 bus matrix interconnects the following masters and slaves:

Masters:

- Cortex®-M33 code AHB (C-AHB) bus via instruction cache (I-cache)
- Cortex®-M33 Slave AHB (S-AHB) bus
- One cryptographic accelerator (CryptoLite)
- One direct AHB_EXP (AHB expansion interconnect) on which memory access – datawire (DMA0), AHB_AP, and PCCA (via bridge) are connected

Slaves:

- Flash controller inside SYSCPUSS
- Two SRAM controllers inside SYSCPUSS (SRAM0, SRAM1)
- One ROM inside SYSCPUSS
- CM33 to peripheral interconnect (PERI_CM33)
- One AHB DMA to peripheral interconnect (PERI_DMA)

3.1.1 AHB5 bus matrix interconnect

The SYSCPUSS uses a round-robin-with-latency scheme to implement bus arbitration in the AHB5 interconnect. Table 8 outlines the SYSCPUSS AHB5 bus matrix interconnect.

Table 8 AHB5 bus matrix interconnect

Bus masters	Bus slaves					
	Flash	**ROM**	**SRAM**	**SRAM**	**PERI_CM33**	**PERI_DMA**
CM33_C-AHB	**X**	**X**	x	x	-	
CM33_S-AHB	**x**	-	X	X	X	-
DMA0	**X**	-	X	X	-	X
PPCA bridge	**X**	-	X	X	X	-
CryptoLite	**X**	-	X	X	-	X

Where:

- X indicates a connection
- x indicates a connection with an aliased address path
- **X** indicates a connection, with bold font type denoting a 128-bit bus
- **x** indicates a connection with an aliased address path, with bold font type denoting a 128-bit bus
- - indicates no connection

***Notes**:*

1. *Memory aliasing allows the CM33 C-AHB bus (I-cache) to access system SRAM. The CM33 S-AHB bus utilizes the aliased address path to write into flash memory (located in the C-AHB bus address space). Data can be read from the aliased address of flash memory, but this is not preferred because it is not cached by the CM33 I-cache*
2. *All CM33 C-AHB bus accesses are cacheable*
3. *DMA writes to aliased addresses occur using the S-AHB bus address space (above 0x20000000)*
4. *Writes to aliased addresses by the DMA or CM33 S-AHB bus must be followed by a cache-invalidation sequence to ensure that stale data does not remain in the cache*
5. *The PERI_CM33 and PERI_DMA interfaces provide access to peripherals connected via the peripheral interconnect (PERI)*

3.1.2 SYSCPU buses

3.1.2.1 Cortex®-M33 C-AHB bus

This bus connects the C-AHB bus of the Cortex®-M33 core to the bus matrix through the instruction cache. It is used for instruction fetches and data access to internal and external memories mapped in the code region.

The targets of this bus include flash, internal SRAMs, and ROM.

3.1.2.2 Cortex®-M33 S-AHB bus

This bus connects the S-AHB bus of the Cortex®-M33 core to the bus matrix. It is used by the core to access data located in a peripheral or SRAM area.

The targets of this bus include the aliased flash, internal SRAM, and peripherals connected via PERI_CM33.

3.1.2.3 DMA bus

This bus connects the bus matrix to the 32-bit AHB master interface of the DMA, which shares a clock with the CM33 CPU. The targets of this bus include the flash, internal SRAM, and peripherals connected via PERI_DMA.

3.2 AHB5 MMIO

The memory-mapped I/O registers for CM33, CryptoLite, DMA0, CPUSS, and all other peripheral configurations are accessible through the AHB5 MMIO bus interface.

3.3 Peripheral interconnect (PERI)

The peripheral interconnect (PERI) implements the AHB5 bus matrix infrastructure, connecting AHB masters to groups of peripheral devices. It enables internal and external masters of the SYSCPUSS to interface with peripherals operating in the low-power domain.

These peripheral devices include all memory-mapped registers (MMIO registers) of PERI, PCLK, and external peripheral devices, as well as synchronous and asynchronous peripherals mapped to the peripheral address space:

- `0x4200_0000 ~ 0x43FF_FFFF` (non-secure address map)
- `0x5200_0000 ~ 0x53FF_FFFF` (secure address map)

In the PSC3M/P8, PSC3M/P7 MCU family, there is one PERI instance. As detailed in Table 9, PERI consists of six peripheral groups. These groups connect slave peripherals within their respective group to the SYSCPUSS and other AHB masters.

Table 9 Peripheral groups and slave peripherals

Peripheral group	Slave peripheral	Peripheral type	Description
PERI GROUP 0	Peripheral interconnect MMIO registers	Internal	SYSCPUSS platform component without I/O subsystem
	Peripheral protection control (PPC)		
	PERI_PCLK registers		
	APB Slave group-0, SRAM0, SRAM1, PROM, Flash, MXCM33-0, DMA-0, SYSCPUSS, IPC, fault structure MMIO registers		
	SRSS core	Peripheral	
	SRSS power mode		
	SRSS backup		
	Cryptographic accelerator (CryptoLite)		
	Debug600		
	Supply Voltage Glitch Sensor (SVGS)		
	Advanced Encryption Standard (AES)		
PERI GROUP 1	High-Speed I/O Matrix (HSIOM)	Peripheral	Async peripherals
	GPIO		
	Low-power comparator (LPComp)		
	Timer/Counter/PWM (TCPWM)		
PERI GROUP 2[1)]	DFT	Peripheral	
	eFuse		
PERI GROUP 3	Controller area network flexible data-rate (CAN FD)	Peripheral	Async peripherals
	Serial communication block (SPI/UART/I^{2}C) (SCB[0])		
	Serial communication block (SPI/UART/I^{2}C) (SCB[1:4])		
PERI GROUP 4	Programmable power control accelerator (PPCA)	Peripheral	Async peripherals
PERI GROUP 5	Serial communication block (SPI/UART/I^{2}C) (SCB[5])	Peripheral	Async peripherals
	Serial communication block (SPI/UART/I^{2}C) (SCB[6])		

1) Slave peripherals DFT and eFuse are meant for internal use.

The AHB bus interfaces of the peripheral slaves within a peripheral group all operate at the same frequency, referred to as `clk_group[i]`. To reduce power consumption, slaves can be disabled using `SL_CTL[i].ENABLED[j]`.

The trigger component handles the multiplexing of trigger signals. For more information, see Trigger multiplexer (Trigger MUX).

3.3.1 AHB bus error handling

The following bus errors can occur in the bus infrastructure:

- AHB error response detected at the master interface
- AHB error response from the peripheral decoder. The peripheral decoder generates an error response in any of the following scenarios:
  - Access to the respective slave is disabled or reset using SL_* registers (refer to the PSOC™ Control C3M/P8, C3M/P7 MCU registers reference manual for details on these registers)
  - Access to an unmapped slave address
  - Error response from the respective peripheral itself
- No response for the AHB transaction from the peripheral for a greater number of cycles than expected

PERI uses Fault monitoring to report these errors.

3.4 Bus master peripherals

3.4.1 Direct memory access (DMA-DW)

The DMA0 can function as an AHB master within the CPU subsystem. The DMA-DW is ideal for small data sizes and transactional DMA operations. It is typically used to transfer data between peripherals, such as from the ADC to SRAM.

The DMA-DW operates using a 32-bit AHB bus.

3.4.2 CryptoLite

CryptoLite provides an AHB-Lite master interface that supports 8, 16, and 32-bit transfers. The master interface allows the subsystem to access operand data for operations directly from system memories, such as flash or SRAM.

4 Inter processor communication (IPC)

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

- IPC Channel: Communication and synchronization between processors are 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

The channel and interrupt structures are independent and have no correlation to each other, as shown in Figure 3. This allows for building varying models of interfaces, as shown in the Typical usage models.

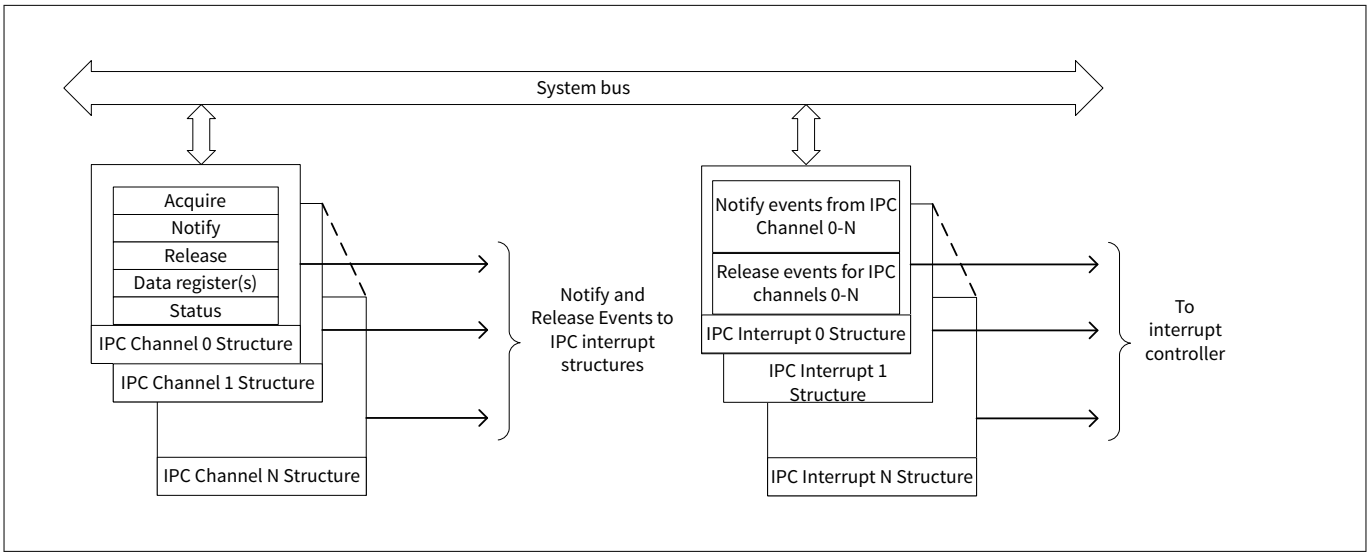


Figure 3 IPC register architecture

4.1 Features

The features of IPC are as follows:

- Implements locks for mutual exclusion and synchronization between secure and non-secure applications
- Allows sending messages between secure and non-secure applications
- Supports up to 8 channels for communication
- Supports up to 4 interrupts, which can be triggered using notify or release events from the channels

4.2 Architecture

4.2.1 IPC channel

An IPC channel is implemented as six hardware registers, as shown in Figure 4. The IPC channel registers are accessible to both secure and non-secure applications 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 value of ‘1’, it means that the read acquired the lock. If the SUCCESS field returns a value of '0', it means that the read operation failed to acquire the lock. The P, NS, PC, and MS fields represent the access attributes of the transaction that previously obtained the lock successfully, and the current access does not affect these fields.

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 CPU with multiple tasks that can preempt each other. When acquired, this register is released by writing any value into the IPC_STRUCTx_RELEASE register. If the register is already in an acquired state, another attempt to read it 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 any or multiple interrupt structures
- IPC_STRUCTx_RELEASE: This field unlocks and triggers release events for the IPC interrupt structures, but only when the lock is obtained (LOCK_STATUS.ACQUIRED is '1'). The release event generated by an IPC channel can trigger any or multiple interrupt structures. To only release the IPC channel and not generate an interrupt, you can write a zero into the IPC release register. Masking of the interrupt can also be achieved by the IPC_INTR_STRUCTx_INTR_MASK register
- IPC_STRUCTx_DATA0 and IPC_STRUCTx_DATA1: These registers are designed to hold data, and they are both 32-bit in size. These registers can be considered as the shared data memory for the channel. Typically, these registers will hold messages that need to be communicated between secure and non-secure applications. If the messages are larger than the combined 64-bit size, place pointers in one or both of these registers
- IPC_STRUCTx_LOCK_STATUS: This register provides the instantaneous lock status for the IPC channel. If the channel is acquired (LOCK_STATUS.ACQUIRED is '1'), it shows the protection context, security, and other information in the respective bit fields. However, if the lock is not acquired (LOCK_STATUS.ACQUIRED is '0'), the values in fields P, NS, PC, and MS are not valid. The lock status reading only gives an immediate status, which can change in the next cycle

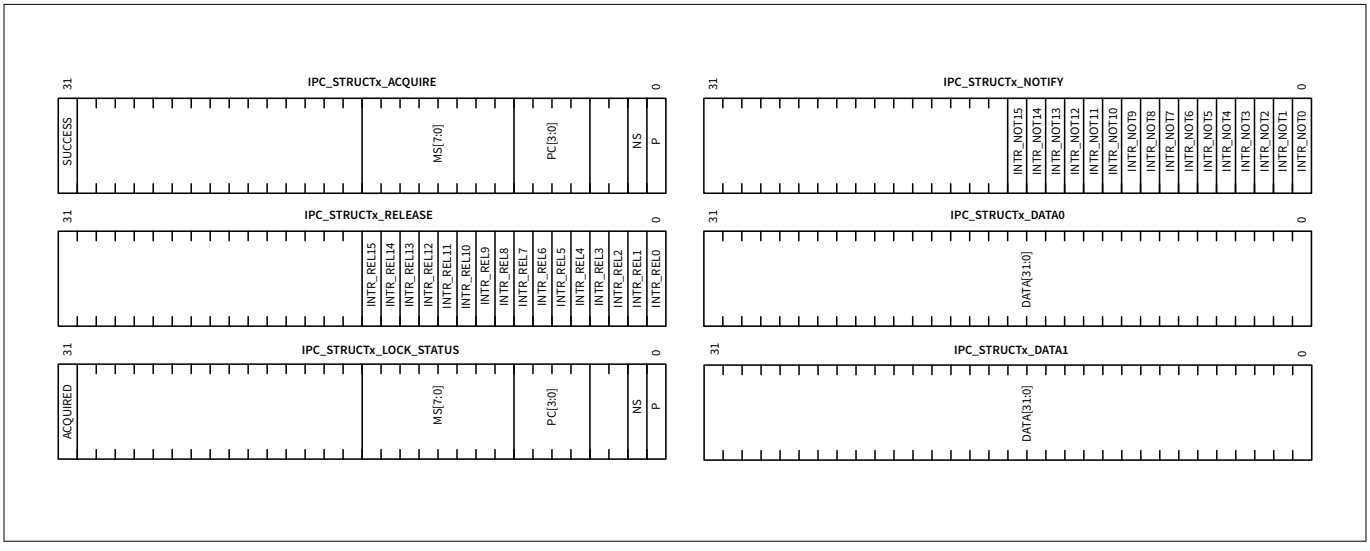


Figure 4 IPC channel structure

4.2.2 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. You 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 register. These are the notify and release events

corresponding to the 16 IPC channels. When a notify event is triggered in IPC channel 0, the corresponding Notify 0 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 sets the interrupt
- IPC_INTR_STRUCTx_INTR_MASKED: This register provides the instantaneous value of the interrupts after they are masked. The values in this register are IPC_INTR_STRUCTx_INTR and IPC_INTR_STRUCTx_INTR_MASK

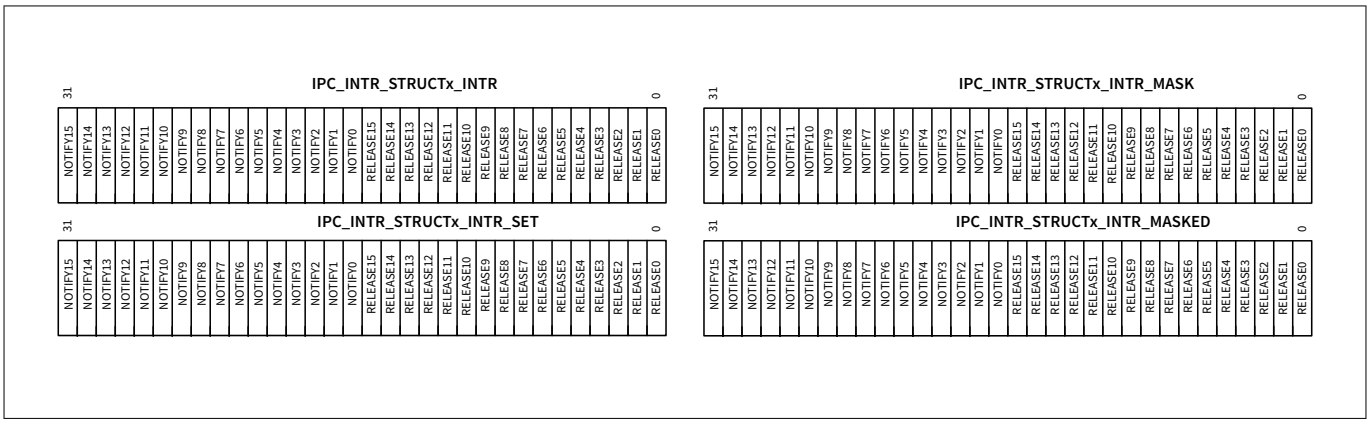


Figure 5 IPC interrupt structure

4.2.3 IPC channels and interrupts

The IPC block has a set of associated IPC interrupts. 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 is dependent on the device architecture.

Each IPC channel has a release and notify register, which can drive events to any of the IPC interrupts. An illustration of this relationship between the IPC channels and the IPC interrupt structure is shown in Figure 6.

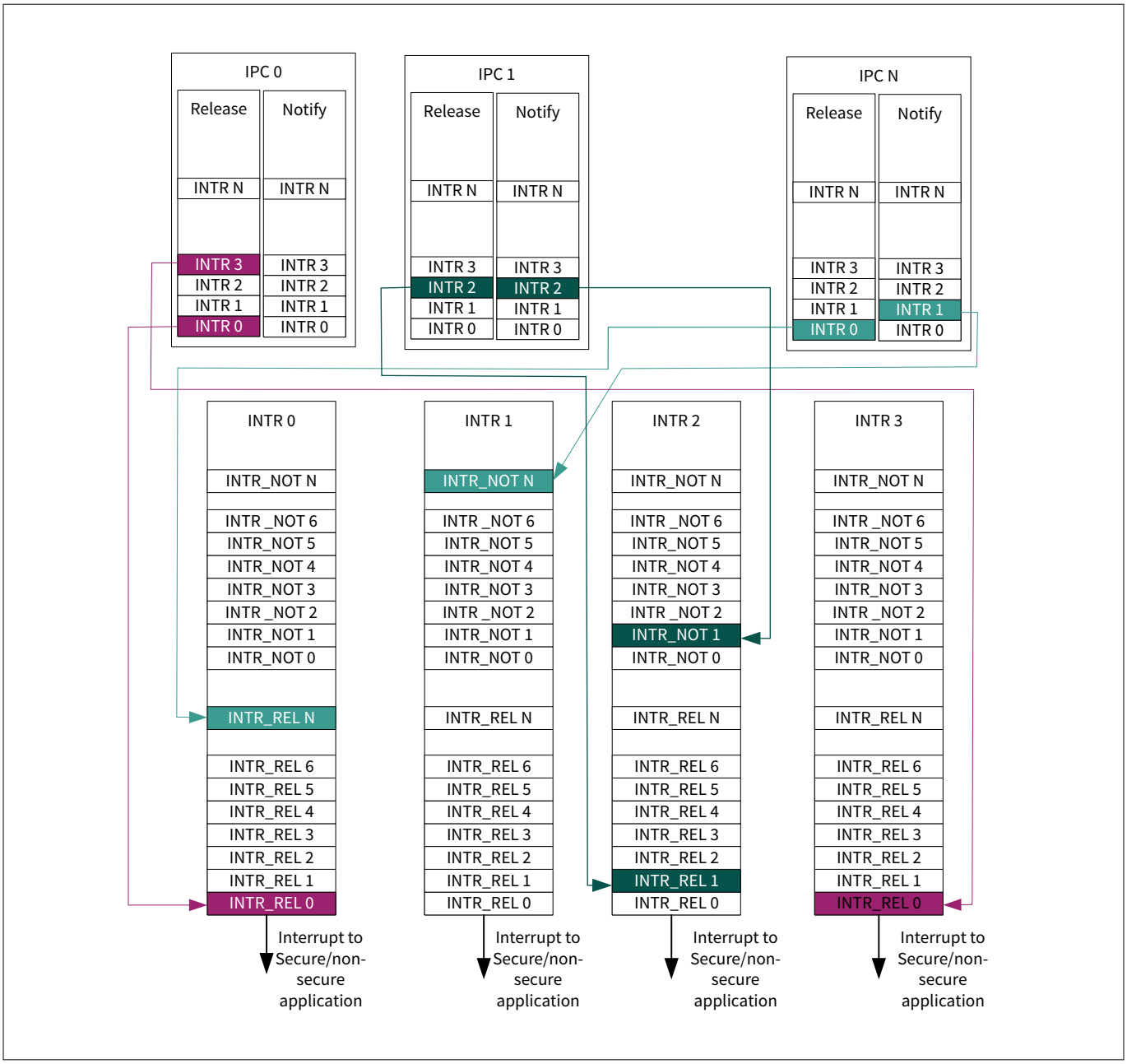


Figure 6 IPC channels and interrupts

4.3 Implementing locks

The IPC channels can be used to implement locks, which are typically used between secure and non-secure applications to enable some form of mutually exclusive access to a shared resource. When secure and non-secure applications share a resource, the secure or non-secure application can acquire and release the IPC channel. So secure and non-secure applications can assume an IPC channel as a lock. The semantics of this code are that access to the shared resource is gated by the secure and non-secure application ownership of the channel. So the secure or non-secure application must acquire the IPC channel before they can access the shared resource.

A failure to acquire the IPC channel by the secure or non-secure application signifies a lock on the shared resource. Note that the IPC channel does not enforce which application acquires or releases the channel. Secure or non-secure applications can both acquire or release the IPC channel.

4.4 Message passing

IPC channels can be used to communicate messages between secure and non-secure applications. In this use case, the channel is used in conjunction with the interrupt structures. The IPC channel is used to lock access to the data registers. 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 transmitting data into the channel and the receiver reading it, the channel is locked for all other task access. 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 registers. After the reception is complete, the receiver releases the channel and can also generate a release event for the sender's IPC interrupt. Note that the action of locking the channel does not restrict access to the data registers in hardware. This is a semantic that should be enforced by software.

Figure 7 shows an example of a sender (a non-secure application) sending data to a receiver (a secure application). IPC interrupt A is configured to interrupt non-secure application. IPC interrupt B is configured to interrupt secure application.

1. The sender will attempt to acquire the IPC channel by reading the IPC_STRUCTx_ACQUIRE register. If the channel is acquired, the sender has ownership of the channel for data transmission. This also changes the status of the channel and its corresponding IPC_STRUCTx_LOCK_STATUS register. If the channel is not acquired, the sender should wait until the channel is free for acquisition. This can be done by polling the IPC channel's IPC_STRUCTx_LOCK_STATUS register
2. After the IPC channel is acquired, the sender has control of the channel for communication and places the data 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 achieves 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 (IPC_INTR_STRUCTx_INTR). If the IPC channel's notify event is enabled by setting the mask bit (IPC_INTR_STRUCTx_INTR_MASK [31:16]) in the IPC interrupt B, this will generate an interrupt in the receiver
4. When it receives IPC interrupt B, the receiver can poll the IPC_INTR_STRUCTx_INTR_MASKED register to understand which IPC channel 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 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 generates a release event interrupt on the sender if the corresponding channel release event is masked

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

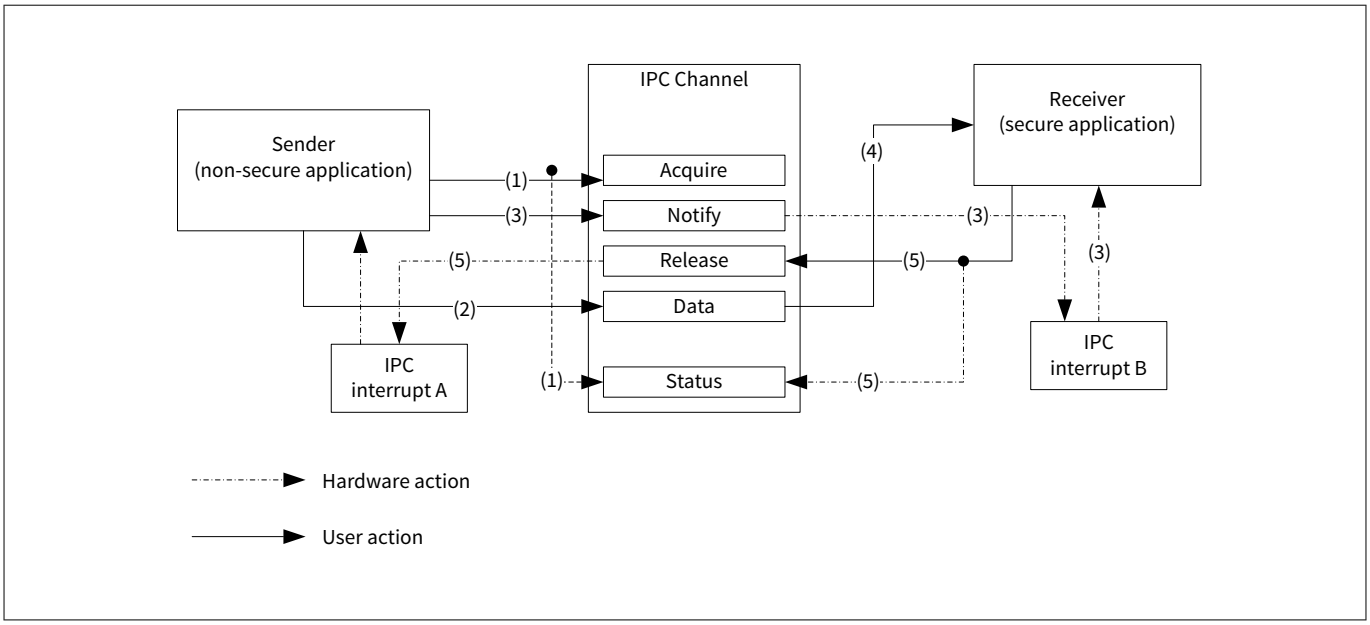


Figure 7 Sending messages using IPC

In the previous example, the size of the data being transmitted was only 64 bits. Larger messages can be sent as pointers. The sender can allocate a larger message structure in memory and pass the pointers to the data registers, see Figure 8 for usage. Note that the user code must implement the synchronization of the message read process.

- The implementation can stall the channel until the receiver has used all the data in the message packet and the message packet can be rewritten. This is worthless because it will stall other inter-process 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 the receiver clears on read complete

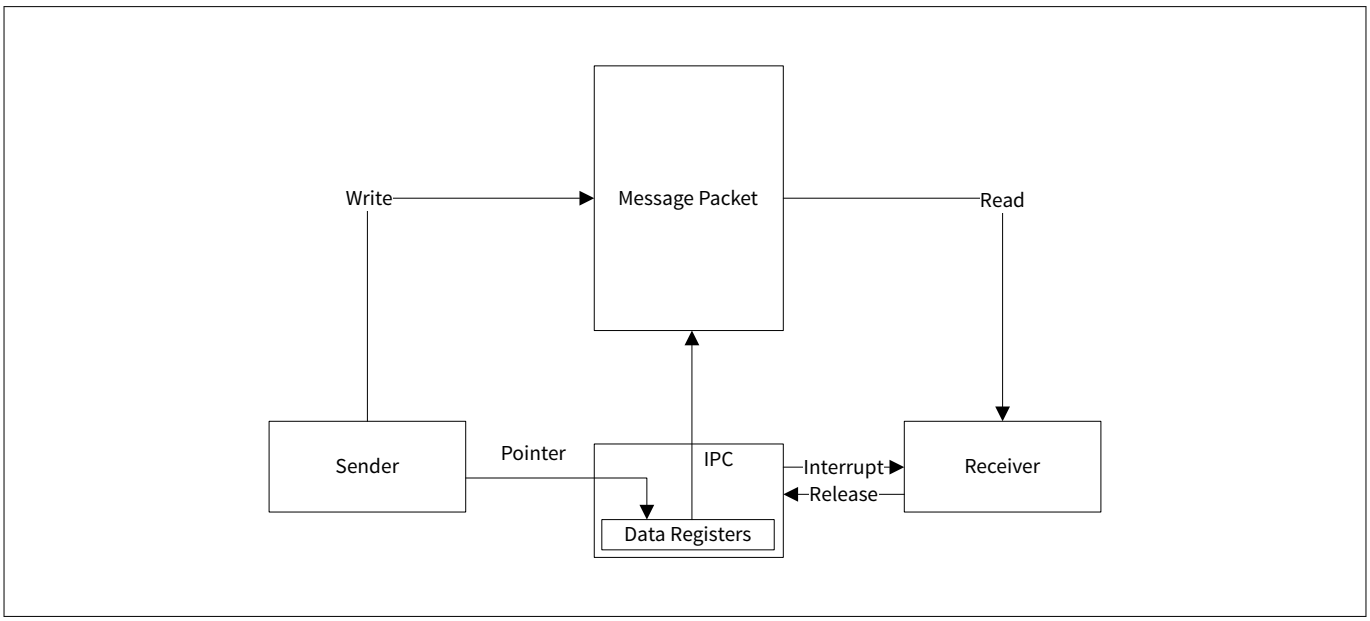


Figure 8 Communicating larger messages

4.5 Typical usage models

The unique channel and interrupt architecture of the PSC3M/P8, PSC3M/P7 IPC allows for a range of usage models for communication between secure and non-secure applications. Some of these are listed here as an example. Note that the communication models possible, based on the IPC architecture, are not restricted to the ones listed in this document. Also note that this document only provides a high-level usage model and does not provide details of data management in the communication. This must be determined based on the specific application use case.

4.5.1 Full-duplex communication

In this usage model, an IPC channel is used according to the direction of communication between secure and non-secure applications. In this example, IPC channel X is dedicated to data communication from non-secure application to secure application and IPC channel Y is for data communication from secure application to non-secure application. The IPC interrupt X will signal events on secure application. Therefore, its interrupt output is connected to the secure application's system interrupts. The events are triggered by writing into the IPC interrupt register structure over the system bus. Similarly, IPC interrupt Y is dedicated to non-secure application.

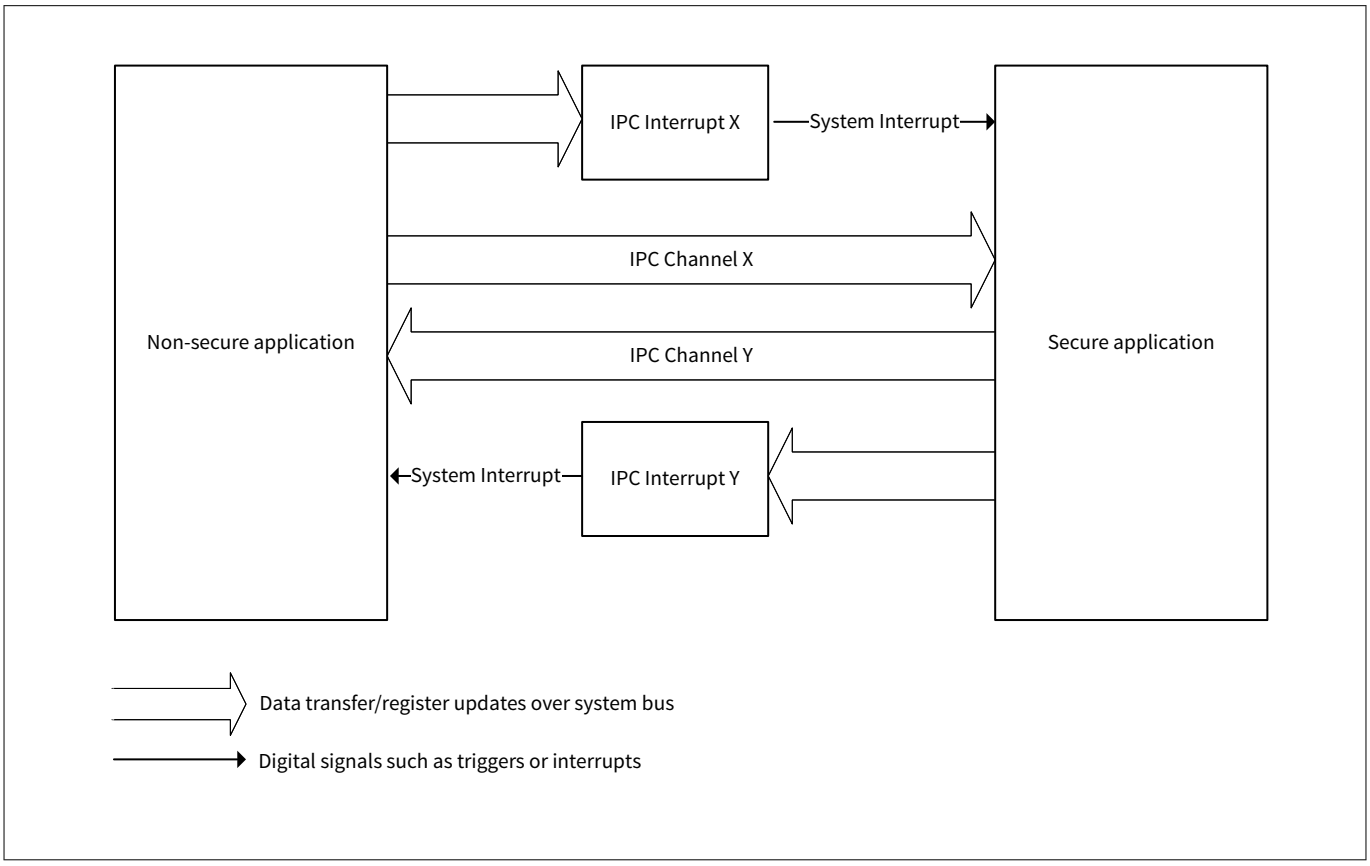


Figure 9 Full-duplex IPC between secure and non-secure application

4.5.2 Half-duplex with independent event handling

In this case, only one IPC channel is used to set up the transfer between the secure and non-secure applications. This means that only one side controls data transfer at a time. The channel's lock register must be used to avoid contention of the one shared IPC channel. Two independent events are supported since two IPC interrupt structures are used. This model is shown in Figure 10.

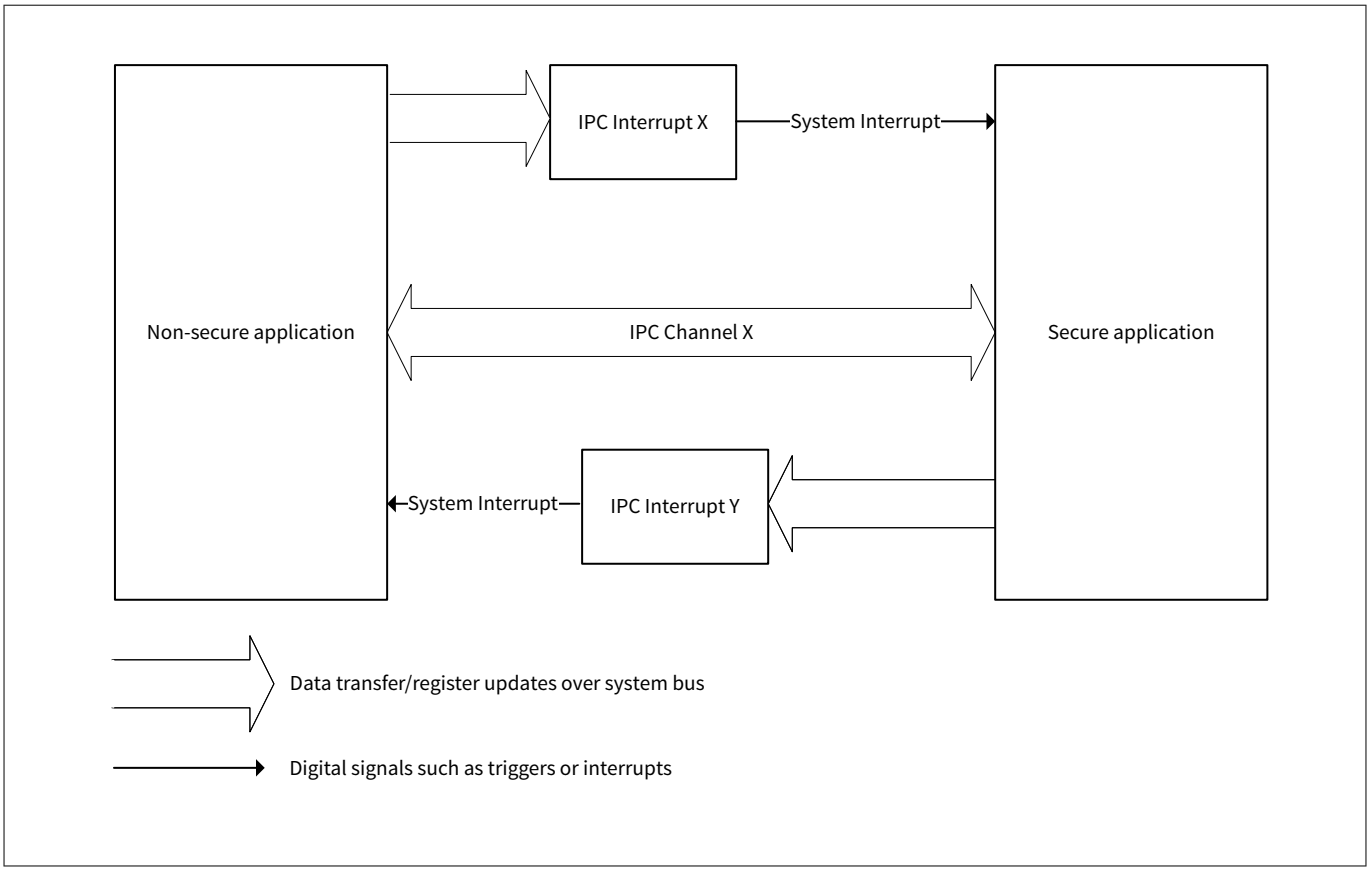


Figure 10 Half-duplex with independent event handling

4.5.3 Half-duplex with shared event handling

In this model, both the IPC channel and interrupt are shared between the secure and non-secure applications. Since the interrupt is also shared, access to the interrupt registers must be managed using the IPC lock of the channel. As shown in Figure 11, the IPC interrupt is set up to trigger interrupts in both secure and non-secure applications. Therefore, secure or non-secure application interrupts should have logic in their ISR to check which application is in control of the IPC and determine if the message and event were for that application.

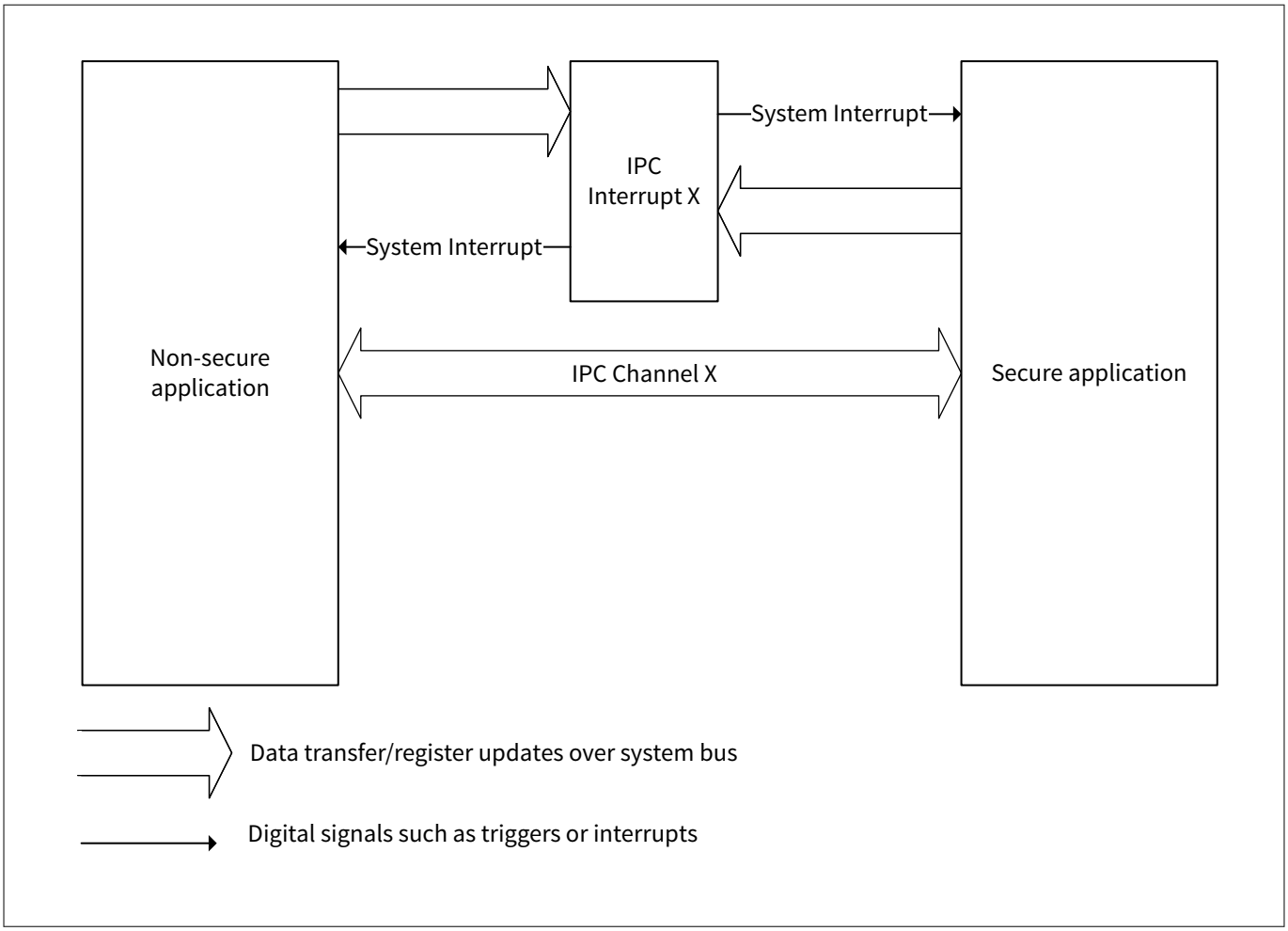


Figure 11 Half-duplex with shared event handling

5 Flash

The PSC3M/P8, PSC3M/P7 flash offers high bulk program performance and supports ultra-low-power operation. Flash is typically used to store CPU instructions and data when the device's power is off. Flash may be written, but the process is much slower and more restrictive than for SRAM. This chapter explains the geometry and capability of flash memory.

5.1 Features

Flash has the following features:

- Flash memory size of up to 512 KB
- 512-byte row size; minimum programmable unit, containing 2 sectors and each sector with 512 rows
- 9-bit ECC for single error correction and dual error detection (SECDED)
- Supports the Read While Write feature
- Flash memory protection using the memory protection controller (MPC)
- Supports a programmable number of wait states
- Arm® Q-Channel LPI for low-power mode management
- 10 - 20 year retention
- Endurance of 100k program cycles

5.2 Configuration

5.2.1 Block diagram

This section explains the major components within the Flash controller (FLASHC).

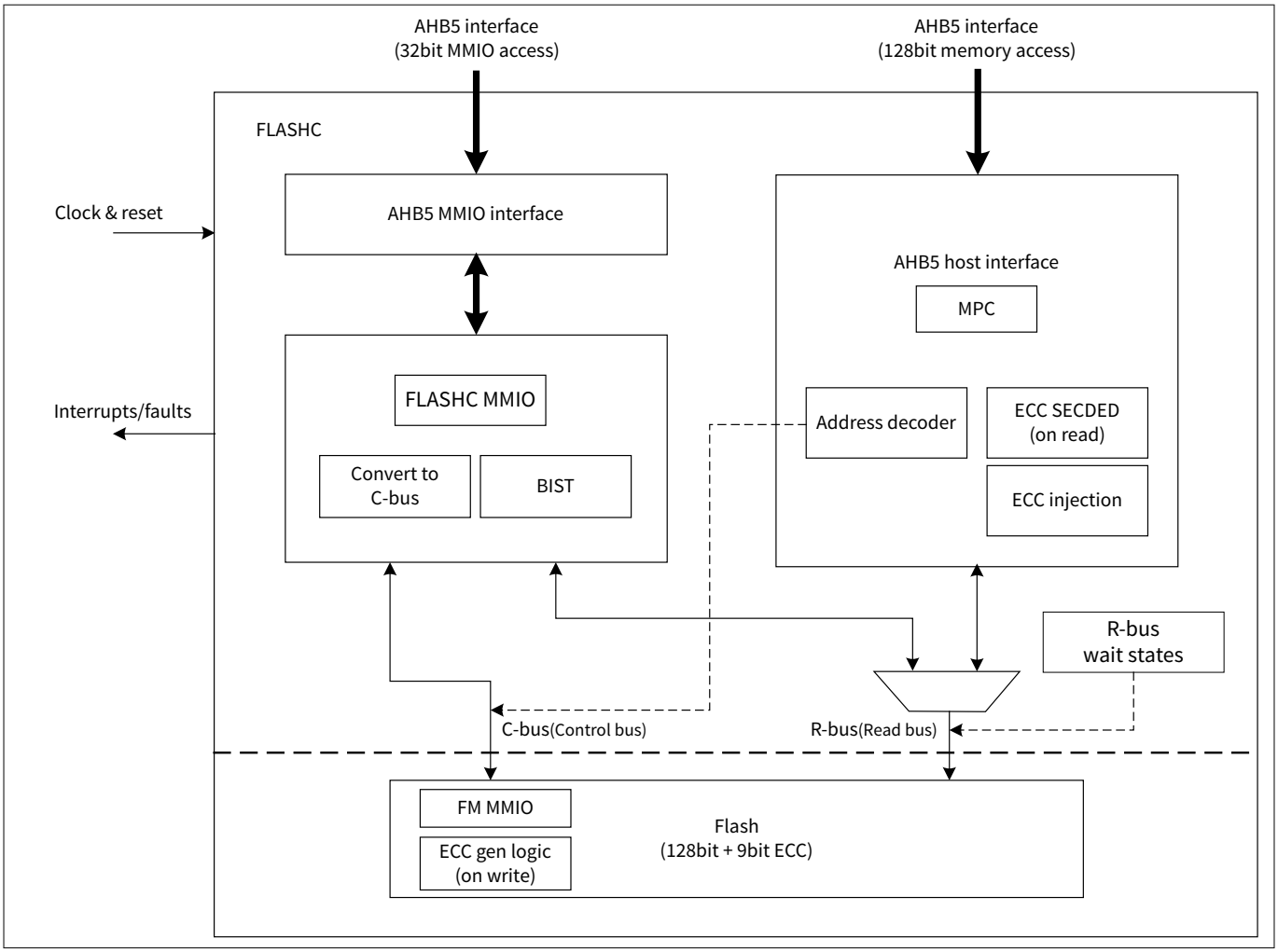


Figure 12 FLASHC high-level block diagram

5.2.2 Flash geometry

The following figure shows the geometry of Flash Macro (FM).

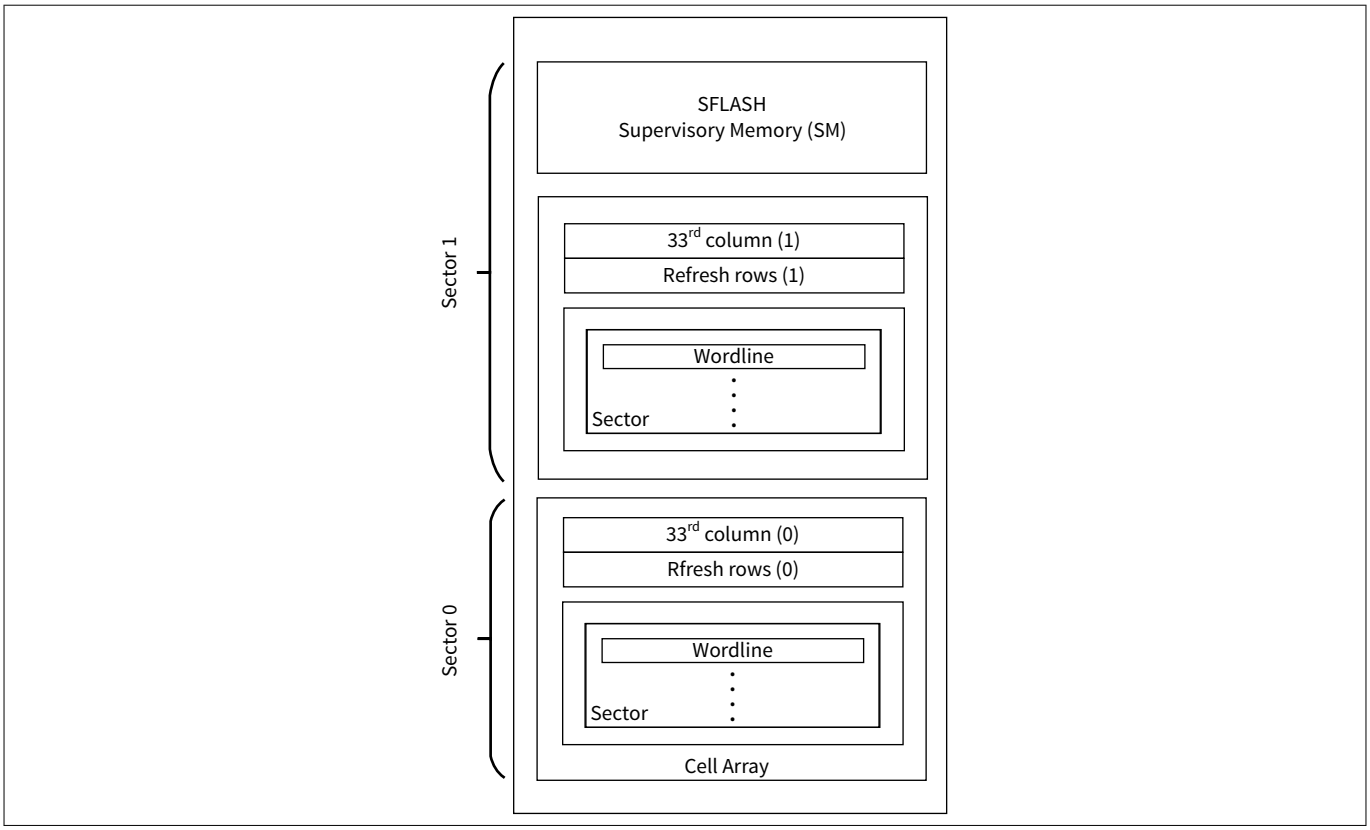


Figure 13 Flash geometry

- Read Word: Flash Macros (FMs) have a Read Word size that is read and returned on each read command. Each word is 128 data bits with 9 bits of ECC
- Rows: Rows are associated with a wordline in the FM and can only be programmed/erased in their entirety. This is composed of 32 units of Read Word. Each wordline size is 512 bytes
- Sector: A sector is composed of 512 rows. Each sector size is 256 KB
- 33rd column: In the FM, when erase/program cycles occur to the other rows in the same sector, the unselected rows see a program disturb, which degrades the endurance/retention of memory cells. The 33rd column is used as a counter to track the oldest row and the youngest row in the sector through these count values. The data from the 33rd column is used to initiate a refresh operation
- Refresh rows: These are additional pages in each sector only to support refresh operations. A refresh operation involves erasing and destroying original data in a row before reprogramming it. Any interruptions during this sequence may leave critical rows of data in a corrupted or erased state. Therefore, data from the pages to be refreshed is copied to refresh rows. Then the rows to be refreshed are erased and reprogrammed with the same data from the refresh rows with a new counter value. Refresh is not allowed in sector 1 because SFLASH is shared with this sector. The access to the refresh rows must be configured to a Protection Context (PC) of 0, 1, or 2 to prevent corruption of the recovery process. It is the responsibility of the OEM code to test for refresh and to initiate the flash refresh routine when required

A refresh is required after more than 100,000 flash write cycles. Writing to part of a flash sector degrades data in other areas of that sector. The refresh function restores unaffected areas of the flash sector after repeated writes. This function is required only when flash is used for data logging or EEPROM emulation. It is not required if flash is used only for program storage.

***Note**: Any EEPROM emulation memory must be allocated in physical sector 0.*

5.2.3 Bank modes and address mapping

This flash memory controller has a dual-bank mode feature. When using dual-bank mode, the flash memory region is split into two half-banks. One is called Bank 0, and the other is called the Bank 1.

FM supports a Read While Write operation on the same flash. This means that a write can be done on one logical bank and a read can be done from the other bank. If the write and read is done from the same logical bank, it will result in an error.

The main flash region supports dual-bank mode. The user can select the mode through FLASH_CTL.BANK_MODE.

This is to support Firmware updates of the software image in flash memory.

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

- When configuring single bank mode, the entire main flash regions are mapped as a single contiguous address region
- When configuring dual bank mode, the main flash 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
  - Choosing mapping A will present the Bank 0 in the lower region and the Bank 1 in the upper region.
  - Choosing mapping B will present the Bank 0 in the upper region and the Bank 1 in the lower region.

When switching flash banks, the following rules must be followed:

- The BANK_MODE LSB must be changed for the banks to switch correctly. Modifying BANK_MAPPING without modifying the BANK_MODE LSB may result in errors

Refer to Table 1Table 2 for flash address mapping.

5.3 Flash controller

The flash controller provides the following FM platform controller functionality:

- Read from FM on the 128-bit AHB5 host interface with MPC protection connected to the 128-bit CODE interconnect
- Write sequence initiation to FM on the same 128-bit AHB5 host interface or on the 32-bit AHB5 MMIO interface. The continuous sequence proceeding for control and status check is achieved through the 32-bit AHB5 MMIO interface connected to the peripheral interconnect with PPC protection
- There can be three modes of operation for FM: full R-bus, full C-bus, and RWW (Read While Write) modes
  - The FLASHC converts the AHB5 transaction into an R-bus interface for FM data read
  - The wait states (FLASH_CTL.RBUS_WS) on the 128-bit AHB5 host interface for FM data read
- 32-bit AHB5 MMIO interface for the FLASHC MMIO and FM MMIO access for FM read, write, and control/ configuration:
  - 32-bit word-aligned access only
  - Peripheral protection using the Peripheral Protection Controller (PPC) within the PERI
  - The FLASHC converts the AHB5 MMIO transaction into a C-bus interface for FM sequence proceeding
- The fault interface supports connections to the CPUSS central fault reporting infrastructure
- Configurable single or dual bank modes of the FM

5.3.1 Power modes

The PSC3M/P8, PSC3M/P7 supports system power modes including OFF, HIBERNATE, DEEPSLEEP_OFF, DEEPSLEEP_RAM, DEEPSLEEP, and ACTIVE. The following table shows system and flash power mode mapping.

Table 10 Flash power mode mapping

System power modes	FM power modes	FM time to wake	Condition
Hibernate	Hibernate	< 10 us	All analog circuits Off
DEEPSLEEP_OFF	Hibernate	< 10 us	All analog circuits Off
DEEPSLEEP_RAM	Hibernate	< 10 us	All analog circuits Off
DEEPSLEEP	Hibernate	< 10 us	All analog circuits Off
SLEEP	Standby	-	All analog circuits On, No clock
ACTIVE	Active	-	All analog circuits On, clock present

5.4 Read While Write support

The PSC3M/P8, PSC3M/P7 MCU supports read operations on one sector while programming or erasing in another area. The FM contains two sectors, each 256 KB in size. The RWW feature is available between sectors. The application can read or execute from one sector while there is an ongoing write or erase operation in another sector. However, when the code execution or read is in the last 16 bytes of a given sector (say sector 0) and the flash write/erase operation is in the next sector (sector 1), an RWW violation may occur if cache pre-fetch is enabled. This is because pre-fetch will fetch the next 16 bytes of data, which is part of sector 1, while a write operation is underway in the same sector. This will result in a fault and should be considered during firmware design. Firmware can be designed to place dead code or constant data in the last 16 bytes of every sector, making sure the last 16 bytes of a sector never accessed, or it can disable cache pre-fetch during a flash write or erase operation.

The RWW FMs can be used as a traditional one that is Read Operations through Read-bus, Read or Write Operations through Control-bus, or in RWW mode, for which Read Operations occur through Read-bus while the part is written through Control-bus. The Read-bus and Control-bus have separate sector and row address registers and pre-decoders, their outputs being multiplexed at sector level. During the write operation, the concurrent read cannot happen in the same sector.

During RWW, the following operations can be performed on the Write Sector:

- Row erase or program
- Sector erase or pre-program

During RWW, the following are not available:

- Data read in the Write Sector
- Bulk Erase or Pre-Program
- Test Modes

***Note**: When executing application code from one flash sector during a Read-While-Write (RWW) operation on another sector, the device may trigger a bus fault if an interrupt occurs during the concurrent erase or program activity. This condition affects both single-bank and dual-bank modes and is most likely to occur with periodic or high-frequency ISRs (for example, SysTick). Applications that perform in-application programming or background flash maintenance/logging activities should avoid relying on time-critical interrupt execution during RWW operations to prevent fault exceptions or unintended system resets.*

5.5 ECC

The flash controller supports Error Correcting Code (ECC). The FM uses an additional 9 bits for every word of 128 bits to perform Single Error Correction (SEC) and Dual Error Detection (DED). Due to ECC, the internal read width is extended from 128 bits to 137 bits.

The ECC block can be tested by injecting incorrect ECC values. See ECC (error) injection for details.

5.5.1 ECC generation/encoding

The ECC Encoder is located in the FM, while the ECC Decoder is located in the flash controller within CPUSS. While in full C-bus or RWW mode, it is software configuration that either FM hardware-generated ECC or software-specified ECC written into FM follow FM C-bus ECC generation/encoding on the write sequence.

- When FM ANA_CTL0.ECC_ENC_DIS == 1'b0 implies the ECC encoder is enabled for the FM with the ECC feature. The PL ECC bits are loaded automatically
- When FM ANA_CTL0. ECC_ENC_DIS == 1'b1 implies the ECC encoder is disabled. FM will use the ECC parity from FM_PL_ECC.DATA9

5.5.2 ECC (error) injection

ECC injection can be enabled or disabled based on the register ECC_INJ_EN.ECC_INJ_ENABLE. In full R-bus or RWW mode, ECC parity can be injected based on ECC_INJ_EN.ECC_INJ_ENABLE to replace ECC parity delivered by Flash Macro. The ECC_CTL.WORD_ADDR and ECC_CTL.PARITY registers specify the address and the parity to be injected. ECC_CTL.PARITY is injected when ECC_INJ_EN.ECC_INJ_PC and ECC_CTL.WORD_ADDR both match the AHB5 read transaction over the 128-bit interface.

While in full R-bus or RWW mode, ECC parity (to replace ECC parity delivered by FM) can be injected prior to the ECC correction or decoding logic during the read. The flash MMIO ECC_INJ_EN/ECC_INJ_CTL can be enabled and configured to inject 9-bit ECC based on the Program Counter and address of the 128-bit long-word.

When ECC_INJ_EN.ECC_INJ_ENABLE is LOW, ECC_INJ_EN.ECC_INJ_PC inherits Program Counter (PC) from the master who enables/configures ECC_INJ_ENABLE HIGH.

- When ECC_INJ_ENABLE is HIGH, only the PC specified by ECC_INJ_PC can access (read/write) ECC_INJ_EN (ENABLE, ECC_ERROR, and ECC_INJ_PC bits)
- PC violation results in a bus error, and there is no additional fault/interrupt triggered. The only exception is PC0, which can access (read/write) ECC_INJ_EN at any time to break the potential deadlock of ECC_INJ_EN

ECC_INJ_CTL.WORD_ADDR specifies address (based on module-internal offset address) and ECC_INJ_CTL.PARITY, parity, to be injected.

- When ECC_INJ_ENABLE is HIGH, only the PC specified by ECC_INJ_PC can access (read/write) ECC_INJ_CTL
- PC violation results in a bus error, and there is no additional fault/interrupt triggered

PARITY is injected when ECC_INJ_PC and WORD_ADDR both match the AHB read transaction over the 128-bit interface.

- ECC_INJ_CTL.PARITY replaces ECC parity from FM and is used by the ECC correction/decoding logic
- ECC_INJ_EN.ECC_ERROR:
  - LOW (1'b0): If the injected ECC does not trigger any non-recoverable error (ECC errors <= 1)
  - HIGH (1'b1): If the injected ECC triggers a non-recoverable error (ECC errors >= 2). The AHB read transaction results in a bus error. There is no additional fault/interrupt triggered

ECC injection is subject to MPC protection. Each PC is allowed only to inject ECC parity into its allocated address blocks/ranges.

To disable ECC injection, the PC specified by ECC_INJ_PC should clear ECC_INJ_EN (reset ECC_INJ_ENABLE and ECC_ERROR bits). PC0 can also clear ECC_INJ_EN at any time.

FLASHC hardware clears ECC_INJ_CTL autonomously while ECC_INJ_EN is being cleared.

5.6 Security and protection

The FLASHC 128-bit AHB5 host interface is protected using MPC. The MPC implements Secure/Non-Secure block-based protection similar to the Arm® SIE-200 MPC. However, the functionality is extended to provide block-based protection for each protection context (PC) as well.

Refer to Memory Protection Controller (MPC) for details on MPC.

***Note**: Flash write can only be done from a secure application.*

***Note**: FLASHC APIs can be called only from a secure application.*

5.7 SROM APIs

SROM APIs provide blocking operations for writing, programming, and erasing flash rows. Data is flashed in 512-byte rows, with one row processed per operation. Larger data sets requires iterating across multiple rows.

The main SROM APIs:

- Performs write operation

```
uint32_t cyboot_flash_write_row(uint32_t address, uint8_t* data, flash_context_t* ctx);
```

- Performs program operation

```
uint32_t cyboot_flash_program_row(uint32_t address, uint8_t* data, flash_context_t* ctx);
```

- Performs erase operation

```
uint32_t cyboot_flash_erase_row(uint32_t address, flash_context_t* ctx);
```

Context and parameters of SROM APIs:

- *** ctx**: Points to a RAM memory where flash context (flash_context_t) stored. For blocking mode, all fields must be set to zero
- **address**: Flash address to erase/program/write
- **data**: Pointer to RAM data corresponding to a single 512-byte row that will be written or programmed

6 SRAM controller

This section explains the PSC3M/P8, PSC3M/P7 MCU SRAM controller, its features, architecture, and wait states. The SRAM controller enables the CPU to read and write the PSC3M/P8, PSC3M/P7 MCU's SRAM.

In the PSC3M/P8, PSC3M/P7 MCU, the size of the SRAM is 128 KB. To know more about SRAM instances and their start addresses, see the Address and memory maps section of the CPU subsystem.

6.1 Features

The SRAM controller has the following features:

- Supports 8, 16, and 32-bit access
- Provides exclusive access support using Exclusive Access Monitor (EAM) from Arm® SIE-200
- SRAM memory protection using the MPC (Memory Protection Controller)
- SRAM power mode control using Arm® PPU (Power Policy Unit)
- The SRAM controller supports the Error Correction Code (ECC) for the SRAM

See the device datasheet for electrical specifications.

6.2 Architecture

The SRAM controller has an AHB5 interface that connects to the AHB5 infrastructure. The AHB5 transfers are the origin for all SRAM accesses. The SRAM controller differentiates between the following three types of AHB5 transfers:

- AHB5 read transfers
- 32-bit AHB5 write transfers
- 8-bit and 16-bit AHB5 write transfers (also referred to as partial AHB5 write transfers)

6.2.1 AHB5 read transfers

AHB5 reads transfers, and a read transfer is translated into SRAM read access.

6.2.2 32-bit AHB5 write transfers

A 32-bit AHB5 write transfer is translated into SRAM write access.

6.2.3 Partial AHB5 write transfers

Partial writes (8-bits and 16-bits) are supported through read, modify, and write by applying the associated mask during write.

6.2.4 Power partition control

The PWR_MACRO_CTL register controls the power state of each SRAM power partition independently. Each bit of PWR_MACRO_CTL.OFF[31:0] represents the individual state of each SRAM power partition. Enabling the bit corresponding to the particular power partition turns OFF the partitions, and disabling the bit turns ON the respective partitions. It is mandatory to ensure STATUS.PWR_DONE becomes 1 after these PWR_MACRO_CTL register contents are changed to ensure the SRAM mode transition is completed successfully. Each power partition contains a (RAM[ ]_MACRO_NR/RAM[ ]_PWR_GROUP_NR) number of base SRAM units. In PSC3M/P8, PSC3M/P7 MCU, RAMC0/1_MACRO_NR = 1 and RAMC0/1_PWR_GROUP_NR = 1 create 1 power partitions with 1 macro in each power partition.

The PWR_MACRO_CTL_LOCK register controls the accidental writes into the PWR_MACRO_CTL register by prohibiting read/write access when this (PWR_MACRO_CTL_LOCK[1:0]) field is not equal to zero.

6.3 Error Correcting Code (ECC)

The SRAM controller supports the Error Correcting Code (ECC) for the SRAM.

The 7-bit SECDED parity covers the 32-bit data word as a single entity, there is no dedicated parity for the four 8-bit bytes within the data. Therefore, any partial 8-bit or 16-bit AHB5 write bus transfer requires a SRAM read access and a SRAM write access.

- 32-bit AHB5 write bus transfers require only a single SRAM write access
- 8-bit and 16-bit AHB5 write bus transfers require an additional SRAM read access that precedes the SRAM write access to retrieve the missing data bytes

When a non-recoverable error that is error of more than 1 bit is detected, the data word that is used as the result for an AHB5 bus transfer is incorrect, but NO AHB5 bus error is generated. Instead, the error is communicated through the CPUSS fault reporting infrastructure.

6.3.1 ECC Error injection

The Error Injection mechanism is supported by 32-bit data word write into an address defined in ECC_MATCH. Reading from the address defined in ECC_MATCH.WORD_ADDR triggers a correctable or uncorrectable fault indication, if the value in ECC_CTL.PARITY has 1, 2, or more errors compared to the expected ECC value.

6.3.2 ECC error report functionality

The ECC syndrome logic reports recoverable and non-recoverable errors to the CPUSS fault reporting infrastructure. The ECC syndrome logic corrects recoverable errors. Errors up to one bit are recoverable.

6.3.2.1 Partial AHB5 write transfers

Error reported during partial write transfer involving SRAM read are listed as below.

- **Correctable Error (1-bit error):** This corrected 32-bit data word is then merged with the write data with appropriate mask and stored in the write buffer. A future write-buffer request results in a SRAM write access with the merged write data
- **Multiple-bit error:** the error is communicated through the CPUSS fault reporting infrastructure

6.3.2.2 AHB5 read transfers

During the AHB read transfers below are possible scenarios.

- **Correctable Error (1-bit error):** The corrected data is returned to the AHB5 interface. If ECC auto-correct functionality is enabled, then the corrected data is also stored in write-buffer. A future write-buffer request results in a SRAM write access with the corrected data
- **Multiple-bit error:** Corrupted-data (multiple-bit error data) is returned to the AHB master on hrdata[31:0]

6.3.2.3 Correctable ECC error reporting

When the auto-correct functionality is enabled for 1-bit correctable ECC errors, the corrected data and merged data during partial write along with the corresponding AHB address is stored in the WRITE-BUFFER. Until the data for this AHB address is written back from the WRITE_BUFFER to the SRAM macro, all future accesses to this AHB address result in the data being read from the WRITE_BUFFER.

6.3.2.4 Non-correctable ECC error reporting

When a non-recoverable error is detected, this is reported only via the fault reporting structure and no AHB bus error is generated. Further, the data word that is used as the result for an AHB5 bus transfer is incorrect.

6.4 Wait states

Under standard operating conditions, no wait states are introduced, as only AHB5 requests are issued. This behavior remains consistent even when ECC is enabled. However, a single wait state is inserted under the following conditions:

- Read-after-Write Dependency: A read operation immediately follows a write operation
- ECC Correction Overhead: An ECC correction was triggered during the preceding memory access

7 Interrupts

7.1 Features

The PSC3M/P8, PSC3M/P7 supports the following interrupt features:

- Supports 143 system interrupts
  - Four interrupts can be mapped to the CPU's non-maskable interrupt (NMI)
- All interrupt sources are capable of waking the CPU from CPU Sleep power mode
  - 32 interrupt sources are capable of waking the Cortex®-M33 from CPU DeepSleep power mode
- Nested vectored interrupt controller (NVIC) integrated with CPU core, yielding low interrupt latency
- Wakeup interrupt controller (WIC) enabling interrupt detection (CPU wakeup) in CPU DeepSleep power mode
- A vector table may be placed in either volatile or non-volatile memory
- Separate vector tables for Cortex®-M33 secure and non-secure states
- Configurable priority levels for each interrupt, eight levels on the Cortex®-M33
- Level-triggered interrupt signals

7.2 Architecture

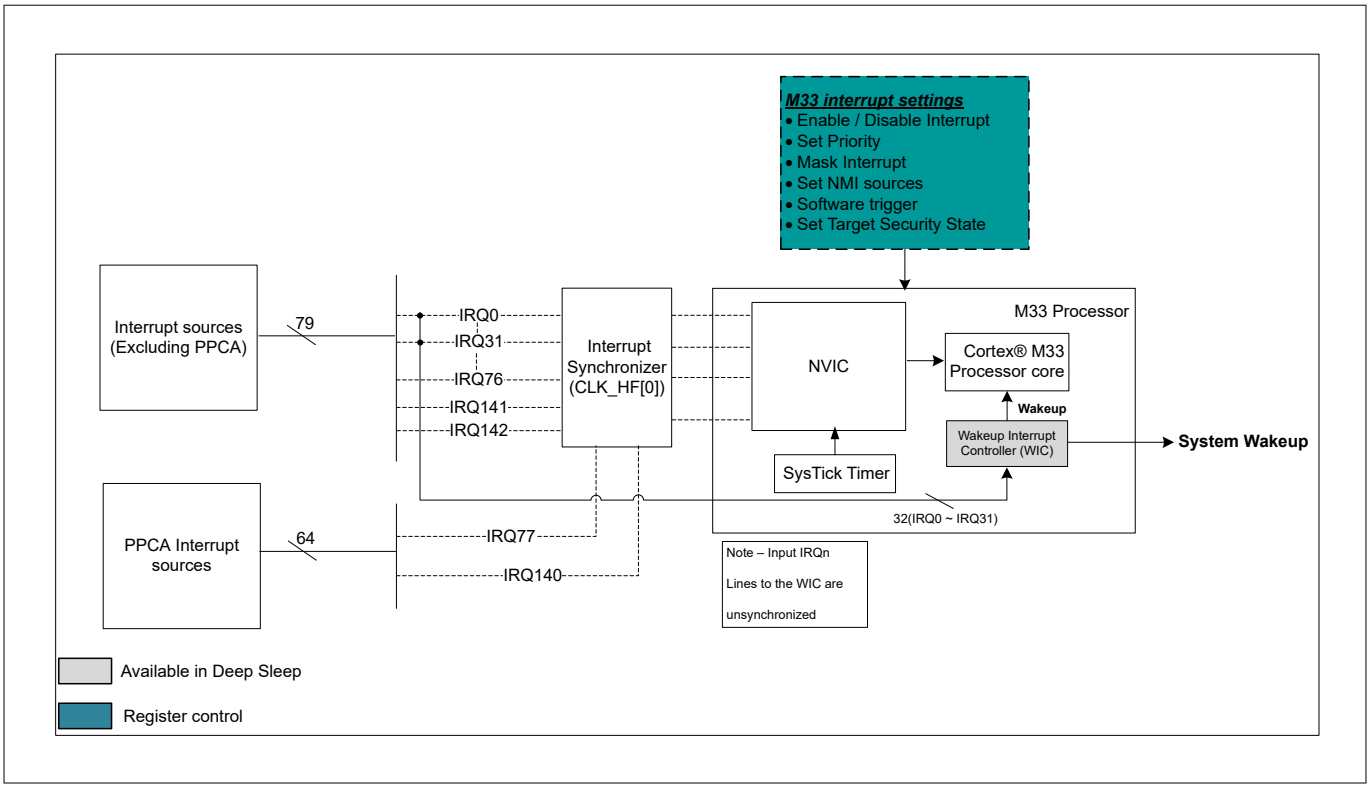


Figure 14 PSC3M/P8, PSC3M/P7 interrupt architecture block diagram

Figure 14 shows the PSC3M/P8, PSC3M/P7 interrupt architecture. The Cortex®-M33 has system interrupt sources directly connected to the IRQn of the NVIC. The NVIC handles enabling/disabling individual interrupt IRQs, priority resolution, and communication with the CPU core. The other exceptions, such as NMI and hard faults, are not shown in Figure 14 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, the PSC3M/P8, PSC3M/P7 supports wakeup interrupt controllers (WIC) for the CPU and interrupt synchronization blocks that synchronize the interrupts to the CPU's CLK_HF0 domain (adds two CLK_HF0 cycles of delay for synchronization). The WIC provides detection of interrupts in the CPU DeepSleep power mode.

7.3 Interrupts and exceptions operation

7.3.1 Interrupt/exception handling

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

- The assumption is that all the interrupt and exception signals are initially low (idle or inactive state) and the processor executes the main code. Any interrupt signal that is held asserted 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
- On detecting the signal from the NVIC, the CPU stores its current context by pushing the contents of the CPU registers onto the stack
- The CPU also receives the exception number of the triggered interrupt from the NVIC. All interrupts and exceptions have a unique exception number. By using this exception number, the CPU fetches the address of the specific exception handler from the vector table. Each interrupt can be configured to target the secure or non-secure vector table
- The CPU then branches to this address and executes the exception handler that follows
- After completion of the exception handler, the CPU registers are restored to their original state using stack pop operations, and the CPU resumes the main code execution

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. The Armv8-M CPU also supports tail-chaining. This means that on completion of an exception handler, if the next pending exception has a higher priority than all stacked exceptions, the stack pop operation is skipped and control is transferred to the new exception handler. Exceptions are handled in the same way that interrupts are handled. Each exception event has a unique exception number, which is used by the CPU to execute the appropriate exception handler.

Security is of importance; it is advised to mask non-secure interrupts during secure code execution. This is because a non-secure interrupt can inherit the protection context of the secure code if a non-secure interrupt were to trigger during secure code execution.

7.3.2 Level interrupts

PSC3M/P8, PSC3M/P7 only supports level signals on the interrupt lines (IRQn).

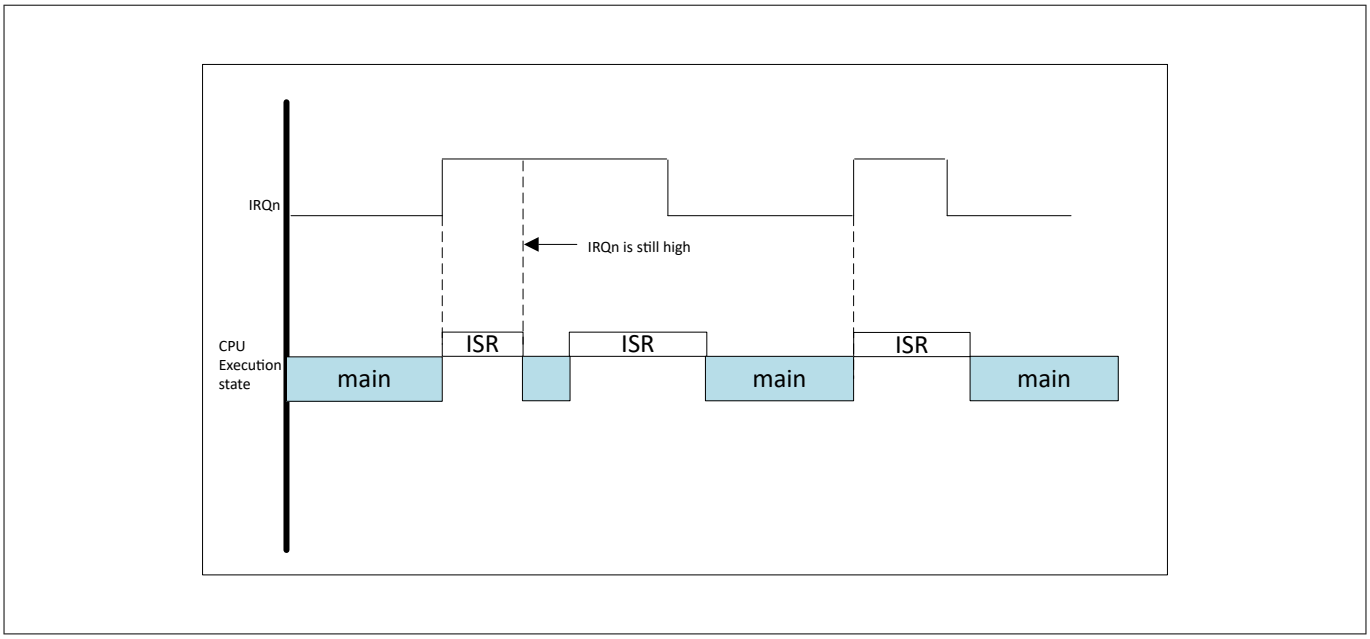


Figure 15 Level interrupts

***Note**: Pulse interrupts are not supported by this device.*

Figure 15 shows how a level interrupt is handled. Assuming the interrupt signal is initially inactive (logic low), the following sequence of events explains the handling of the level:

- For level interrupts, the interrupt signal remains asserted until the NVIC registers the interrupt request. The interrupt is now in the pending state, which means the CPU has not serviced the interrupt requests
- 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 remains high after completing the ISR, it will be pending, and the ISR will be executed again. Figure 15 shows this for level-triggered interrupts, where the ISR is executed as long as the interrupt signal is high

Interrupt behavior

Interrupts may be retriggered after being cleared under certain conditions. If the interrupt register is written immediately before returning from an interrupt service routine (ISR), an additional interrupt can occur even when no interrupt sources are active.

This behavior results from the CPU not waiting for peripheral write operations to complete before executing the next instruction. For details on buffering and write completion, refer to the CPUSS_BUFF_CTL register description in the Register TRM.

7.3.3 Exception vector table

PSC3M/P8, PSC3M/P7 has two exception vector tables that store the entry point addresses for all exception handlers for the Cortex®-M33 secure state and the Cortex®-M33 non-secure state. The CPU fetches the appropriate address based on the exception number. The 143 interrupts (IRQ0 to IRQ142) sources are listed in Table 13.

Table 11 Cortex®-M33 secure exception vector

Exception number	Secure exceptions	Exception priority	Vector address
-	Initial stack pointer value	-	Start_Address =CM33_S_VECTOR_TABLE _BASE or CM33's VTOR_S
1	Reset	-4	Start_Address + 0x0004
2	Secure non-maskable interrupt (NMI_S)	-2	Start_Address + 0x0008
3	Secure HardFault	(-1 or -3)	Start_Address + 0x000C
4	Secure Memory Management	Configurable (0-7)	Start_Address + 0x0010
5	Secure Bus Fault	Configurable (0-7)	Start_Address + 0x0014
6	Secure Usage fault	Configurable (0-7)	Start_Address + 0x0018
7	Secure Fault	Configurable (0-7)	Start_Address + 0x001C
8-10	Reserved	-	...
11	Secure Supervisory Call (SVCall_S)	Configurable (0-7)	Start_Address + 0x002C
12	Secure Debug Monitor	Configurable (0-7)	Start_Address + 0x0030
13	Reserved	-	...
14	Secure Pending Supervisory (PendSV_S)	Configurable (0-7)	Start_Address + 0x0038
15	Secure Sys Tick Timer (SysTick_S)	Configurable (0-7)	Start_Address + 0x003C
16	Secure IRQ(0)	Configurable (0-7)	Start_Address + 0x0040
...	...	...	...
157	Secure IRQ(141)	Configurable (0-7)	Start_Address + 0x055C
158	Secure IRQ(142)	Configurable (0-7)	Start_Address + 0x0560

***Note**: Start address = CM33_S_VECTOR_TABLE_BASE on reset and can be later modified by firmware by updating the CM33's Armv8-M VTOR_S register.*

Secure HardFault can change its priority based on the BFHFNMINS bit in the Armv8-M AIRCR register.

Table 12 Cortex®-M33 non-secure exception vector

Exception number	Secure exceptions	Exception priority	Vector address
-	Initial stack pointer value	-	Start_Address =CM33_S_VECTOR_TABLE _BASE or CM33's VTOR_NS
1	Reset	-4	Start_Address + 0x0004

(table continues...)

Table 12 (continued) Cortex®-M33 non-secure exception vector

Exception number	Secure exceptions	Exception priority	Vector address
2	non-maskable interrupt (NMI_NS)	-2	Start_Address + 0x0008
3	HardFault	(-1 or -3)	Start_Address + 0x000C
4	Memory Management	Configurable (0-7)	Start_Address + 0x0010
5	Bus Fault	Configurable (0-7)	Start_Address + 0x0014
6	Usage Fault	Configurable (0-7)	Start_Address + 0x0018
7	Secure Fault	Configurable (0-7)	Start_Address + 0x001C
8-10	Reserved	-	...
11	Supervisory Call (SVCall_NS)	Configurable (0-7)	Start_Address + 0x002C
12	Debug Monitor	Configurable (0-7)	Start_Address + 0x0030
13	Reserved	-	...
14	Pending Supervisory (PendSV_NS)	Configurable (0-7)	Start_Address + 0x0038
15	Sys Tick Timer (SysTick_NS)	Configurable (0-7)	Start_Address + 0x003C
16	IRQ(0)	Configurable (0-7)	Start_Address + 0x0040
...	...	...	...
157	IRQ(141)	Configurable (0-7)	Start_Address + 0x055C
158	IRQ(142)	Configurable (0-7)	Start_Address + 0x0560

***Note**: Start address = CM33_NS_VECTOR_TABLE_BASE on reset and can be later modified by firmware by updating the CM33's Armv8-M VTOR_NS register.*

The first word (4 bytes) in each vector table 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. The vector table offset register (VTOR) present as part of the Armv8-M System Control Block registers (SCB) configures the vector table offset from the base address (0x0000). The VTOR value determines the location of the vector table, which can be placed in volatile and non-volatile memory. Note that the VTOR registers can be updated only in privilege CPU mode. The advantage of moving the vector table to volatile memory, such as SRAM, is that the exception handler addresses can be dynamically changed by modifying the vector table contents. However, a non-volatile memory vector table must be modified by a non-volatile memory write. Note that the exception table must be 1024 byte-aligned for the Cortex®-M33. The exception sources (exception numbers 1 to 15) are explained in the Exception sources section. The exceptions marked as reserved are not used, although they have addresses reserved for them in the vector table. The interrupt sources (exception numbers 16 and higher) are explained in the Interrupt sources section.

7.4 Exception sources

This section explains the different exception sources listed in Table 11 and Table 12 (exception numbers 1 to 15). This section also discusses a TrustZone concept called "banking". See the Interrupt/exception target security state and banking section for more information about banking.

7.4.1 Reset exception

A device reset is treated as an exception. The reset exception is always enabled with a fixed priority of –4, the highest priority exception.

A device reset can occur for multiple reasons, such as power-on-reset (POR), an external reset signal on the XRES pin, or a watchdog reset. For PSC3M/P8, PSC3M/P7, after a device reset, the Cortex®-M33 core is available, which begins the secure boot process. A Cortex®-M33 secure reset exception is triggered. The Cortex®-M33 reset handler then begins the Cortex®-M33 boot code. Once the Cortex®-M33 boot code is completed successfully, the Cortex®-M33 enters the user application code.

7.4.2 Non-maskable interrupt exception

Non-maskable interrupt (NMI) is always enabled with a fixed priority of –2.

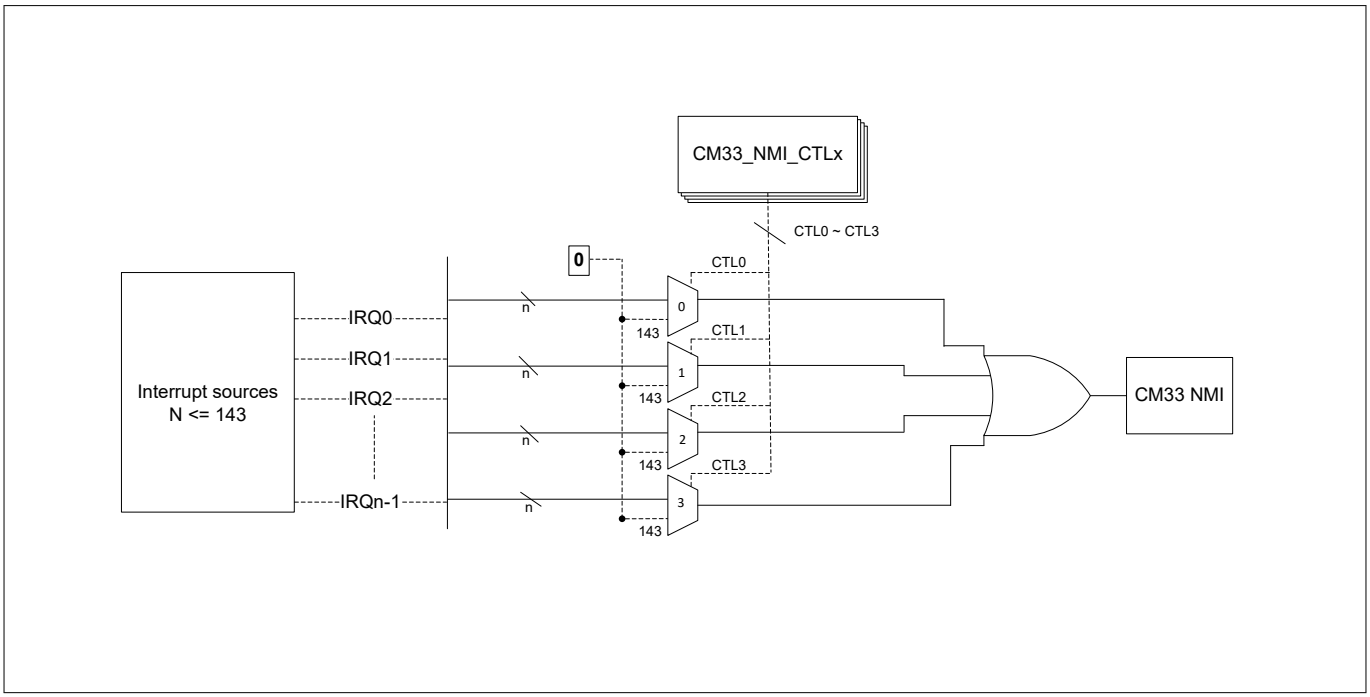


Figure 16 NMI trigger

The Cortex®-M33 provides an option to trigger an NMI exception from up to four of the interrupts the CPU has access to (IRQ0 — IRQn). The NMI exception triggered by the interrupt executes the NMI handler pointed to by the active exception vector table. The CM33_NMI_CTLx registers select the interrupt source that triggers the NMI from hardware. The registers have a default value of 1023; that is, if the register is set to 1023, then that particular register does not map any interrupt source to the NMI. There are four such registers, and each can map one interrupt vector to the NMI. NMI is triggered when any of the four interrupts are triggered; that is, the interrupts are logically ORed.

For the CPU, the exception is not banked between security states.

- If the BFHFNMINS bit in the Armv8-M Application Interrupt and Reset Control Register (AIRCR) is set to 0, the NMI exception targets the Secure NMI handler
- If the BFHFNMINS bit in the Armv8-M AIRCR register is set to 1, the NMI exception targets the non-Secure NMI handler

7.4.3 HardFault exception

PSC3M/P8, PSC3M/P7 Armv8-M CPU supports the HardFault exception. HardFault is an always-enabled exception that occurs because of an error during normal or exception processing. HardFault has a priority of at least –1, meaning it has a higher priority than any exception with configurable priority. A HardFault exception is a catch-all exception for different types of fault conditions, which include executing an undefined instruction and accessing an invalid memory address. 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 can recover from the fault situation.

For CPU, the exception can be banked or not, depending on the BFHFNMINS in the AIRCR register.

- If the BFHFNMINS bit in the Armv8-M AIRCR register is set to 0, only Secure HardFault is enabled with a priority of –1. The Secure HardFault then handles all faults that are unable to preempt the current execution. In this scenario, the exception is not banked
- If the BFHFNMINS bit in the Armv8-M AIRCR register is set to 1, secure and non-secure HardFaults are enabled. Secure HardFault has a priority of –3, and non-secure HardFaults will have a priority of –1. The HardFault handler used depends on whether the fault targets a secure or non-secure state. In this scenario, the exception is banked

7.4.4 MemManage fault exception

A MemManage 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 access to Execute Never (XN) memory regions. The priority of the exception is configurable from 0 (highest) to 7 (lowest).

The exception is banked between security states.

7.4.5 BusFault exception

A BusFault is an exception that occurs because of a memory-related fault for an instruction or data memory transaction. This may be from an error detected on a bus in the memory system. The priority of the exception is configurable from 0 (highest) to 7 (lowest).

For the CPU, the exception is not banked between security states.

- If the BFHFNMINS bit in the Armv8-M AIRCR register is set to 0, the BusFault exception targets the Secure BusFault handler
- If the BFHFNMINS bit in the Armv8-M AIRCR register is set to 1, the BusFault exception targets the Non-Secure BusFault handler

7.4.6 UsageFault exception

A UsageFault 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 half-word memory access
- Division by zero

The priority of the exception is configurable from 0 (highest) to 7 (lowest).

The exception is banked between security states.

7.4.7 SecureFault exception

The SecureFault is an extra exception on Armv8-M CPUs with the security extension, otherwise known as TrustZone. The SecureFault can be triggered by the various security checks that are performed. For example, jumping from a non-secure code to an address in the secure code that is not marked as a valid entry point will trigger the SecureFault. Use caution when handling the SecureFault and make sure that the handler does not inadvertently introduce a security vulnerability. The SecureFault will always target the secure state. The priority of the exception is configurable from 0 (highest) to 7 (lowest).

7.4.8 Supervisor Call (SVCall) exception

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 an SVCall that requires privileged access to the system. The priority of the exception is configurable from 0 (highest) to 7 (lowest).

The exception is banked between security states.

7.4.9 DebugMonitor exception

Armv8-M CPUs support the DebugMonitor exception. DebugMonitor is only available if halting debug is disabled and debug monitoring is enabled. This can be done by setting the C_DEBUGEN bit to disable in the Armv8-M Debug Halting Control and Status Register (DHCSR) and the MON_EN bit to enable in the Armv8-M Debug Exception and Monitor Control Register (DEMCR). The DebugMonitor exception is handled by dedicated debug monitoring software running on the same core. A debug event causes a DebugMonitor exception only when the priority of the DebugMonitor exception is greater than the current exception priority. The priority of the exception is configurable from 0 (highest) to 7 (lowest).

The exception is not banked between security states. DebugMonitor will target the secure handler if the Secure Debug Monitor Enable (SDME) bit in the Armv8-M DEMCR register is set to 1. If SDME is set to 0, the DebugMonitor exception targets the non-secure handler.

7.4.10 PendSupervisory (PendSV) exception

Armv8-M CPUs support the 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 priority of the exception is configurable from 0 (highest) to 7 (lowest).

The exception is banked between security states.

7.4.11 System Tick (SysTick) exception

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 a SysTick exception. The priority of the exception is configurable from 0 (highest) to 7 (lowest).

The exception is banked between security states. This means that there are two SysTick timers, one for each security state.

7.5 Interrupt sources

The PSC3M/P8, PSC3M/P7 supports 143 interrupts from peripherals. The source of each interrupt is listed in Table 13. These system interrupts are mapped directly to the Cortex®-M33 NVIC.

The interrupts include standard interrupts from the on-chip peripherals such as PPCA, SCB, 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 level interrupts. The appropriate interrupt registers should be cleared in the ISR to deassert the interrupt. Usually, a write '1' is required to clear the registers. If the interrupt register is not cleared in the ISR, the interrupt remains asserted, and the ISR is executed continuously.

For the Cortex®-M33 CPU, all interrupts are not banked between security states.

As seen from Table 13, 32 interrupts (IRQ0 to IRQ31) can wake up the Cortex®-M33 from CPU DeepSleep power mode. All interrupts can wake up the CPU from CPU Sleep power mode.

Table 13 List of PSC3M/P8, PSC3M/P7 Cortex®-M33 interrupt sources

Cortex®-M33 interrupt	Cortex®-M33 exception number	CPU DeepSleep capable	Interrupt source
NMI	-2	-	Logical OR of four of the following 143 interrupt sources
IRQ0	16	Yes	GPIO Port Non-Secure #0
IRQ1	17	Yes	GPIO Port Non-Secure #1
IRQ2	18	Yes	GPIO Port Non-Secure #2
IRQ3	19	Yes	GPIO Port Non-Secure #3
IRQ4	20	Yes	GPIO Port Non-Secure #4
IRQ5	21	Yes	GPIO Port Non-Secure #5
IRQ6	22	Yes	GPIO Port Non-Secure #6
IRQ7	23	Yes	GPIO Port Non-Secure #7
IRQ8	24	Yes	GPIO Port Non-Secure #8
IRQ9	25	Yes	GPIO Port Non-Secure #9
IRQ10	26	Yes	GPIO Port Secure #0
IRQ11	27	Yes	GPIO Port Secure #1
IRQ12	28	Yes	GPIO Port Secure #2
IRQ13	29	Yes	GPIO Port Secure #3
IRQ14	30	Yes	GPIO Port Secure #4
IRQ15	31	Yes	GPIO Port Secure #5
IRQ16	32	Yes	GPIO Port Secure #6
IRQ17	33	Yes	GPIO Port Secure #7
IRQ18	34	Yes	GPIO Port Secure #8
IRQ19	35	Yes	GPIO Port Secure #9
IRQ20	36	Yes	GPIO Supply Detect

(table continues...)

Table 13 (continued) List of PSC3M/P8, PSC3M/P7 Cortex®-M33 interrupt sources

Cortex®-M33 interrupt	Cortex®-M33 exception number	CPU DeepSleep capable	Interrupt source
IRQ21	37	Yes	GPIO Non-Secure - All ports
IRQ22	38	Yes	GPIO Secure – All ports
IRQ23	39	Yes	SCB#0
IRQ24	40	Yes	Multi Counter Watchdog Timer
IRQ25	41	Yes	Backup domain
IRQ26	42	Yes	Inter Process Communication #0
IRQ27	43	Yes	Inter Process Communication #1
IRQ28	44	Yes	Inter Process Communication #2
IRQ29	45	Yes	Inter Process Communication #3
IRQ30	46	Yes	WDT
IRQ31	47	Yes	LPCOMP
IRQ32	48	-	SRSS (LVD, CLKCAL)
IRQ33	49	-	SCB#1
IRQ34	50	-	SCB#2
IRQ35	51	-	SCB#3
IRQ36	52	-	SCB#4
IRQ37	53	-	SCB#5
IRQ38	54	-	SCB#6
IRQ39	55	-	FLASH Macro
IRQ40	56	-	CPUSS DMA#0, Channel#0
IRQ41	57	-	CPUSS DMA#0, Channel#1
IRQ42	58	-	CPUSS DMA#0, Channel#2
IRQ43	59	-	CPUSS DMA#0, Channel#3
IRQ44	60	-	CPUSS DMA#0, Channel#4
IRQ45	61	-	CPUSS DMA#0, Channel#5
IRQ46	62	-	CPUSS DMA#0, Channel#6
IRQ47	63	-	CPUSS DMA#0, Channel#7
IRQ48	64	-	CPUSS DMA#0, Channel#8
IRQ49	65	-	CPUSS DMA#0, Channel#9
IRQ50	66	-	CPUSS DMA#0, Channel#10
IRQ51	67	-	CPUSS DMA#0, Channel#11
IRQ52	68	-	CPUSS DMA#0, Channel#12
IRQ53	69	-	CPUSS DMA#0, Channel#13

(table continues...)

Table 13 (continued) List of PSC3M/P8, PSC3M/P7 Cortex®-M33 interrupt sources

Cortex®-M33 interrupt	Cortex®-M33 exception number	CPU DeepSleep capable	Interrupt source
IRQ54	70	-	CPUSS DMA#0, Channel#14
IRQ55	71	-	CPUSS DMA#0, Channel#15
IRQ56	72	-	SRAM#0 PPU
IRQ57	73	-	SRAM#1 PPU
IRQ58	74	-	Cortex®-M33 Floating Point Unit
IRQ59	75	-	Cortex®-M33 CTI #0
IRQ60	76	-	Cortex®-M33 CTI #1
IRQ61	77	-	CPUSS Faults
IRQ62	78	-	CPUSS PPU
IRQ63	79	-	CPUSS Master Security Controller
IRQ64	80	-	TCPWM #0, Group 0, Counter #0
IRQ65	81	-	TCPWM #0, Group 0, Counter #1
IRQ66	82	-	TCPWM #0, Group 1, Counter #0
IRQ67	83	-	TCPWM #0, Group 1, Counter #1
IRQ68	84	-	SRSS Main PPU
IRQ69	85	-	Crypto
IRQ70	86	-	TRNG
IRQ71	87	-	CAN#0, Interrupt#0, Channel#0
IRQ72	88	-	CAN#0, Interrupt#1, Channel#0
IRQ73	89	-	CAN#0, Interrupt#0, Channel#1
IRQ74	90	-	CAN#0, Interrupt#1, Channel#1
IRQ75	91	-	CAN#0, Consolidated interrupt #0
IRQ76	92	-	CAN#0, Consolidated interrupt #1
IRQ77	93	-	PPCA interrupt #0
IRQ78	94	-	PPCA interrupt #1
IRQ79	95	-	PPCA interrupt #2
IRQ80	96	-	PPCA interrupt #3
IRQ81	97	-	PPCA interrupt #4
IRQ82	98	-	PPCA interrupt #5
IRQ83	99	-	PPCA interrupt #6
IRQ84	100	-	PPCA interrupt #7
IRQ85	101	-	PPCA interrupt #8
IRQ86	102	-	PPCA interrupt #9

(table continues...)

Table 13 (continued) List of PSC3M/P8, PSC3M/P7 Cortex®-M33 interrupt sources

Cortex®-M33 interrupt	Cortex®-M33 exception number	CPU DeepSleep capable	Interrupt source
IRQ87	103	-	PPCA interrupt #10
IRQ88	104	-	PPCA interrupt #11
IRQ89	105	-	PPCA interrupt #12
IRQ90	106	-	PPCA interrupt #13
IRQ91	107	-	PPCA interrupt #14
IRQ92	108	-	PPCA interrupt #15
IRQ93	109	-	PPCA interrupt #16
IRQ94	110	-	PPCA interrupt #17
IRQ95	111	-	PPCA interrupt #18
IRQ96	112	-	PPCA interrupt #19
IRQ97	113	-	PPCA interrupt #20
IRQ98	114	-	PPCA interrupt #21
IRQ99	115	-	PPCA interrupt #22
IRQ100	116	-	PPCA interrupt #23
IRQ101	117	-	PPCA interrupt #24
IRQ102	118	-	PPCA interrupt #25
IRQ103	119	-	PPCA interrupt #26
IRQ104	120	-	PPCA interrupt #27
IRQ105	121	-	PPCA interrupt #28
IRQ106	122	-	PPCA interrupt #29
IRQ107	123	-	PPCA interrupt #30
IRQ108	124	-	PPCA interrupt #31
IRQ109	125	-	PPCA interrupt #32
IRQ110	126	-	PPCA interrupt #33
IRQ111	127	-	PPCA interrupt #34
IRQ112	128	-	PPCA interrupt #35
IRQ113	129	-	PPCA interrupt #36
IRQ114	130	-	PPCA interrupt #37
IRQ115	131	-	PPCA interrupt #38
IRQ116	132	-	PPCA interrupt #39
IRQ117	133	-	PPCA interrupt #40
IRQ118	134	-	PPCA interrupt #41
IRQ119	135	-	PPCA interrupt #42

(table continues...)

Table 13 (continued) List of PSC3M/P8, PSC3M/P7 Cortex®-M33 interrupt sources

Cortex®-M33 interrupt	Cortex®-M33 exception number	CPU DeepSleep capable	Interrupt source
IRQ120	136	-	PPCA interrupt #43
IRQ121	137	-	PPCA interrupt #44
IRQ122	138	-	PPCA interrupt #45
IRQ123	139	-	PPCA interrupt #46
IRQ124	140	-	PPCA interrupt #47
IRQ125	141	-	PPCA interrupt #48
IRQ126	142	-	PPCA interrupt #49
IRQ127	143	-	PPCA interrupt #50
IRQ128	144	-	PPCA interrupt #51
IRQ129	145	-	PPCA interrupt #52
IRQ130	146	-	PPCA interrupt #53
IRQ131	147	-	PPCA interrupt #54
IRQ132	148	-	PPCA interrupt #55
IRQ133	149	-	PPCA interrupt #56
IRQ134	150	-	PPCA interrupt #57
IRQ135	151	-	PPCA interrupt #58
IRQ136	152	-	PPCA interrupt #59
IRQ137	153	-	PPCA interrupt #60
IRQ138	154	-	PPCA interrupt #61
IRQ139	155	-	PPCA interrupt #62
IRQ140	156	-	PPCA interrupt #63
IRQ141	157	-	Supply voltage glitch sensor (SVGS)
IRQ142	158	-	Advanced encryption standard (AES)

7.6 Interrupt/exception target security state and banking

For the Cortex®-M33 CPU, exceptions and interrupts can target the secure or non-secure ISR handler. If an exception is said to be banked, this means that the resources for the exception are duplicated and both have a secure and non-secure version. If an exception is said not to be banked, this means that the exception can be configured only to target the secure or non-secure handler.

The following are included in the banked exceptions:

- Banked enabled, pending, and active bits
- A banked SHPRn.PRI field
- A banked exception vector
- A handler for each security state

Table 14 Banked exception

Exception	Banked
Reset	No
HardFault	Yes
NMI	No
MemManagment Fault	Yes
BusFault	No
UsageFault	Yes
SecureFault	No
SVCall	Yes
DebugMonitor	No
PendSV	Yes
SysTick	Yes
IRQ(0)	No
…	...
IRQ(n)	No

***Note**: HardFault is a banked exception when the BFHFNMINS bit is set to 1. When the BFHFNMINS bit is set to 0, all HardFaults go to the secure handler.*

BusFault, NMI, and HardFault can be configured using the BFHFNMINS bit in the Armv8-M Application Interrupt and Reset Control Register (AIRCR). When the BFHFNMINS is set to 0, BusFault, NMI, and HardFault target their respective secure handlers. When the BFHFNMIS is set to 1, BusFault and NMI target their respective non-secure handlers, while HardFault becomes a banked exception with both secure and non-secure handlers.

SecureFault always targets the secure handler.

DebugMonitor targets the secure handler if the Secure Debug Monitor Enable (SDME) bit in the Armv8-M DEMCR register is set to 1. If SDME is set to 0, the DebugMonitor exception targets the non-secure handler.

7.6.1 Interrupt target security state

On the Cortex®-M33 CPU, each interrupt can be configured to target a secure or non-secure state. As a result, an interrupt can be configured to service either the Interrupt Service Routine (ISR) in the secure vector table or the ISR in the non-secure vector table. In the Armv8-M architecture, there are 16 Interrupt Target non-secure registers (ITNSn), which are used to configure the specific vector table that an interrupt uses. Each ITNSn register has 32 bits. The interrupt number is represented as 32n + m, where m is the bit number in the register and n is the register instance. Setting bit m to 0 configures the interrupt target for the secure state, while setting bit m to 1 configures the interrupt target for the non-secure state.

7.6.2 GPIO secure and non-secure interrupts

For the CPU, each GPIO pin can be configured to be secure or non-secure. A single GPIO port can have a combination of secure and non-secure pins. As a result, there are two interrupt sources going into the NVIC for each GPIO port, one for secure and one for non-secure GPIO interrupts. These two interrupt sources are present in Interrupt sources. Configuring a pin's security state also configures the GPIO port interrupt source that will be

triggered. As a result, a secure GPIO pin can only trigger the secure GPIO port interrupt, and a non-secure GPIO pin can only trigger the non-secure GPIO port interrupt.

7.7 Interrupt/exception priority

Exception priority is useful for exception arbitration when there are multiple exceptions that need to be serviced by the CPU. All exceptions other than Reset, NMI, and HardFault can be assigned a configurable priority level. The Reset and NMI have a fixed priority of –4 and –2, respectively. HardFaults on the CPU can have priority of –1 and –3. For more information on HardFault configurations with TrustZone, see the HardFault exception section. 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 7.

The Armv8-M CPU supports nested exceptions, in which a higher-priority exception can obstruct (interrupt) the currently active exception handler. This pre-emption does not happen if the incoming exception priority is the same as or lower than the active exception. The CPU resumes execution of the lower-priority exception handler after servicing the higher-priority exception. With eight priority levels supported, this allows up to eight nested exceptions. When the CPU receives two or more exception requests of the same priority, the lowest exception number is serviced first.

The priority of the exception numbers 4 to 15 can be configured by writing to the three System Handler Priority Registers (SHPR1, SHPR2, SHPR3). Each SHPRn stores the priority of four exceptions, PRI_N. Each priority number corresponds to an exception number. SHPR1 stores the priority for exceptions PRI_4 to PRI_7. SHPR2 stores the priority for exceptions PRI_8 to PRI_11. SHPR3 stores the priority for exceptions PRI_12 to PRI_15.

The priority of the interrupts can be configured by writing to the respective Interrupt Priority registers (NVIC_IPRn). This is a group of 124 32-bit registers, with each register storing the priority values of four interrupts, as specified in Table 15. The interrupt number corresponding to each field is represented as 4n + m, where m is the field number in the register and n is the register instance.

Table 15 Interrupt priority register bit definitions for NVIC_IPRn

Bits	Name	Description
7:5	PRI_N0	Priority of interrupt number 4N + 0
15:13	PRI_N1	Priority of interrupt number 4N + 1
23:21	PRI_N2	Priority of interrupt number 4N + 2
31:29	PRI_N3	Priority of interrupt number 4N + 3

7.8 Enabling and disabling interrupts

The NVICs of the Armv8-M CPUs provide registers individually to enable and disable the interrupts in software. If an interrupt is not enabled, the NVIC does not process the interrupt requests on that interrupt line. The Armv8-M Interrupt Set Enable Register (ISERn) and the Armv8-M Interrupt Clear-Enable Register (ICERn) are used to enable and disable the interrupts, respectively. There are 16 instances of the 32-bit ISER/ICER registers in the Armv8-M architecture. The interrupt number corresponding to each bit is represented as 32n+m, where m is the bit number in the register and n is the register instance. These registers can also be read in software to get the enabled status of the interrupts. Enabling and disabling interrupts/exceptions shows the register access properties for these two registers. Note that writing zero to these registers has no effect. For CPU, ISER and ICER are not banked between security states.

Table 16 Interrupt enable/disable registers

Register	Operation	Bit value	Comment
Interrupt Set Enable Register (ISERn)	Write	1	To enable the interrupt
		0	No effect
	Read	1	Interrupt is enabled
		0	Interrupt is disabled
	Write	1	To disable the interrupt
		0	No Effect
	Read	1	Interrupt is enabled
		0	Interrupt is disabled

The ISER and ICER registers are applicable only for interrupts. These registers cannot be used to enable or disable the exception numbers 1 to 15.

The Armv8-M SHCSR register can be configured to enable and disable SecureFault, UsageFault, BusFault, and MemFault. The SHCSR register has the SECUREFAULTENA, USGFAULTENA, BUSFAULTENA, and MEMFAULTENA bits, where setting them enables the exception while clearing it disables the exception. For the CPU, the register is banked between security states on a bit-by-bit basis. The SECUREFAULTENA and BUSFAULTENA bits are not banked between security states. The USGFAULTENA and MEMFAULTENA bits are banked between security states.

SysTick can be configured with the ENABLE bit in the Armv8-M SysTick Control and Status Register (SYST_CSR). For the CPU, this register is banked between security states.

DebugMonitor can be enabled with the MON_EN bit in the Armv8-M Debug Exception and Monitor Control Register (DEMCR). Note that the C_DEBUGEN bit needs to be disabled in the Armv8-M Debug Halting Control and Status Register (DHCSR) for the DebugMonitor to function. For the CPU, the registers are not banked between security states.

***Note**: The HardFault, NMI, SVCall, and PendSV are always enabled and therefore do not have an exception enable bit.*

Interrupts and exceptions can also be disabled using the Armv8-M special purpose mask register (PRIMASK, FAULTMASK, and BASEPRI). The PRIMASK register 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 13. The FAULTMASK register can be used to disable all faults and interrupts except for the NMI exception. The BASEPRI register can be used to disable interrupts based on priority level. Only interrupts with a priority value lower (higher priority) than what is in the BASEPRI register can trigger. When PRIMASK, FAULTMASK, or BASEPRI are set, none of the masked exceptions can be serviced by the CPU. The masked exceptions remain in a pending state until the mask is cleared. For the CPU, the registers are banked between security states.

7.9 Interrupt/exception states

Each exception can be in one of the following states:

Table 17 Interrupt/exception states

Exception state	Meaning
Inactive	The exception is not active or 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 request has been serviced by the CPU but whose exception handler execution is not yet complete. A high-priority exception can interrupt the execution of a 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 Armv8-M Interrupt Control State Register (ICSR) contains status bits describing the various exception states.

- The VECTACTIVE bits ([8:0]) in the ICSR store the exception number for the current executing exception. This value is zero if the CPU does not execute any exception handler (the CPU is in thread mode). Note that the value in the VECTACTIVE bitfields 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 VECTPENDING bits ([20:12]) in the ICSR store the exception number of the highest priority pending exception. This value is zero if there are no pending exceptions
- The ISRPENDING bit (bit 22) in the ICSR indicates if a NVIC-generated interrupt is in a pending state

7.9.1 Pending interrupts/exception

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 all interrupt lines, by providing separate register bits, for setting and clearing the pending states of the interrupts. The Interrupt Set-Pending register (ISPRn) and the Interrupt Clear-Pending register (ICPRn) are used to set and clear the pending status of the interrupt lines. There are 16 instances of the 32-bit ISPR/ICPR registers, where each bit corresponds to an interrupt. The interrupt number corresponding to each bit is represented as 32n + m, where m is the bit number in the register and n is the register instance. Table 18 shows the register access properties for these two registers. Note that writing zero to these registers has no effect. For the CPU, ISPR and ICPR are not banked between security states.

Table 18 Interrupt set pending/clear pending registers

Register	Operation	Bit value	Comment
Interrupt Set-Pending Register (ISPRn)	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 (ICPRn)	Write	1	To clear a pending interrupt
		0	No effect

(table continues...)

Table 18 (continued) Interrupt set pending/clear pending registers

Register	Operation	Bit value	Comment
	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 does not move to the pending state until it is enabled by writing to the ISER register. Note that the ISPR and ICPR registers are used only for peripheral interrupts. These registers cannot be used for pending exception numbers 1 to 15.

The pending bits for exceptions 1 to 15 can be found in the Armv8-M SHCSR, ICSR, and DEMCR registers.

The SHCSR has the HARDFAULTPENDED, SECUREFAULTPENDED, SVCALLPENDED, BUSFAULTPENDED, MEMFAULTPENDED, and USGFAULTPENDED bits, which are the pending bits for the exceptions with the same name. There are no clear pending registers for these exceptions. Writing and clearing these bits sets and clears the pending state of the exception.

The ICSR has the PENDNMISET, PENDSVSET, and PENDSTSET bits, which set the NMI, PendSV, and SysTick exceptions to the pending state when writing one. The ICSR also has the PENDNMICLR, PENDSVCLR, and PENDSTCLR bits, which clear the NMI, PendSV, and SysTick exceptions when writing one.

The DEMCR register has the MON_PEND bits, which are used to set or clear the pending state of the DebugMonitor.

7.10 Stack usage for interrupts/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. Armv8-M CPUs without TrustZone have two stack pointers: the Main Stack Pointer (MSP) and the Process Stack Pointer (PSP). The Armv8-M CPUs with TrustZone enabled have four stack pointers to separate between secure and non-secure states: MSP_NS, MSP_S, PSP_NS, and PSP_S. 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 downward 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 CPU subsystem (CPUSS) for details.

7.11 Interrupts and low-power modes

The CPU wake up from low-power modes when certain peripheral interrupt requests are generated. The Wakeup Interrupt Controller (WIC) block generates a wakeup signal that causes the CPU to enter the 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) instruction, executed by the CPU, triggers the transition into the CPU Sleep and CPU DeepSleep modes. Both the WFI and WFE instructions can wake up on interrupts. However, the WFE requires the interrupts to be unmasked in the CPU's Exception Mask register (PRIMASK). In addition, 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. WFI does not require PRIMASK unmasking and can wake up the CPU from any pending interrupt masked to the NVIC or WIC. However, WFI cannot wake up the CPU from event signals from other CPUs. The sequence of entering the different low-power modes is detailed in the Device power modes.

Chip low-power modes have two categories of interrupt sources:

- Interrupt sources that are available in the CPU Active, CPU Sleep, and CPU DeepSleep modes
- Interrupt sources that are available only in the CPU Active and CPU Sleep modes

Refer to the Interrupt sources for interrupts that support DeepSleep.

When using the WFE instruction, make sure to call the WFE instruction twice to enter and exit Sleep/DeepSleep modes. This behavior comes from the event register implementation in the Armv8-M architecture.

According to the Armv8-M architecture reference manual (Section B2.1.1):

- A reset clears the event register
- Any WFE wakeup event, or the execution of an exception return instruction, sets the event register
- A WFE instruction clears the event register
- Software cannot read or write the value of the event register directly

Therefore, the first WFE instruction puts the CPU to sleep, and the second WFE clears the event register after a WFE wakeup, which sets the event register. So the next WFE will put the core to sleep.

7.12 Interrupt/exception - initialization/configuration

This section covers the different steps involved in initializing and configuring exceptions in the PSC3M/P8, PSC3M/P7 family.

1. Configuring the Exception Vector Table Location: The first step in using exceptions is to configure the vector table location as required, either in volatile or non-volatile memory. This configuration is done as described in the Exception vector table section. The vector table should be placed in volatile memory if the application must change the vector addresses dynamically. If the table is located in non-volatile memory, such as flash, then a flash write operation is required to modify the vector table contents

***Note**: Only secure applications can do the flash write operation*

2. Configuring individual exceptions: The next step is to configure individual exceptions required in an application, as explained in the 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 section for more information about the interrupt configuration supported by them
- **b.** For the Cortex®-M33 CPU, configure the NVIC to use the secure or non-secure vector table when an interrupt source is triggered, as described in the interrupt target security state
- **c.** Define the exception handler function and write the address of the function to the exception vector table. Table 13 gives the exception vector table format; write the exception handler address to the appropriate exception number entry in the table
- **d.** Set up the exception priority, as explained in the Interrupt/exception priority section
- **e.** Enable the exception, as explained in Enabling and disabling interrupts/exceptions

8 Fault monitoring

Fault monitoring allows you to monitor various faults generated within the device and take actions based on the reported faults. In addition to reporting faults, the fault structures in PSC3M/P8, PSC3M/P7 MCU provide a mechanism to log data from the fault sources and optionally perform a soft reset.

The PSC3M/P8, PSC3M/P7 MCU family supports one centralized fault reporting/monitoring structure that monitors faults generated within the device. The fault report structure can monitor and report faults from up to 47 sources. PSC3M/P8, PSC3M/P7 MCU supports 47 fault sources.

8.1 Features

The PSC3M/P8, PSC3M/P7 MCU fault report structure supports the following features:

- Monitoring protection unit violations, peripheral-specific errors, memory-controller-specific errors, and similar issues
- Four 32-bit data registers for recording fault information
- Soft reset-on-fault detection while retaining fault information
- Interrupt generation on fault detection
- Trigger output to the DW for fault data transfer
- Fault-detected output to a pin for external fault handling

8.2 Architecture

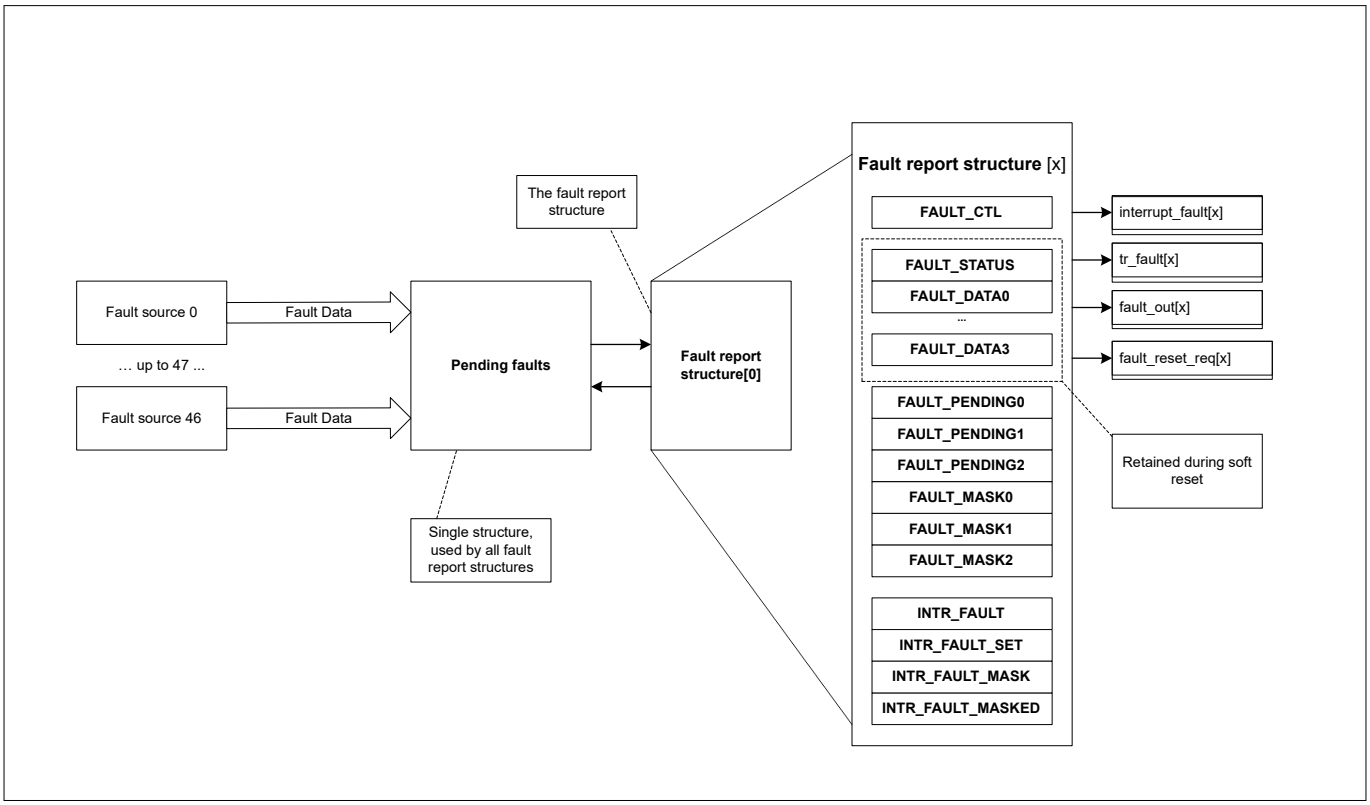


Figure 17 Fault report architecture

The PSC3M/P8, PSC3M/P7 MCU family employs a centralized fault report structure to enable system-wide, consistent fault handling and simplify software development. The fault reporting hardware can be configured so that a single fault interrupt handler services all faults. This structure provides the fault source and additional

fault-specific information through a single set of MMIO registers, eliminating the need for an iterative search. All pending faults can be accessed from the same set of MMIO registers.

The fault structure captures faults associated with the following:

- MPC/PPC protection violations
- Peripheral-specific errors
- Memory controller-specific errors
- ECC errors
- Time-out errors

The fault structure can be configured to capture one or more specific faults as shown in Table 19. When the fault structure is configured to capture a fault, the occurrence is recorded as a pending fault. If all other faults are processed or if no other faults are pending, the fault data is stored in the fault structure registers. A successful capture can optionally trigger an interrupt for further processing by the Cortex®-M33, depending on the application requirement.

In addition to capturing fault data, the fault structure can perform a soft reset while retaining fault information. This reset generates the RESET_ACT_FAULT reset cause in the SRSS_RES_CAUSE register.

8.2.1 Fault report

The PSC3M/P8, PSC3M/P7 MCU family supports a single fault report structure, which includes a dedicated set of control and status registers. This fault report structure captures one fault at a time. The captured fault information includes the following:

- A fault validity bit (FAULT_STRUCT0_STATUS.VALID) that indicates whether a fault has been captured. This bit is set whenever a fault is captured and must be cleared after processing the fault information. New faults are captured only when this bit is set to '0'
- A 7-bit index Table 19 that identifies the fault source (FAULT_STRUCT0_STATUS.IDX)
- Additional fault information (FAULT_STRUCT0_DATA0 through FAULT_STRUCT0_DATA3) that describes fault-specific details. This information is fault-source-specific, and the data field encodings are provided in the Fault encodings section
- A writable fault register (FAULT_STRUCT0_STATUS) that allows software to trigger a fault

8.2.2 Signaling interface

In addition to capturing fault information, the fault report structure includes a signaling interface to notify the rest of the system about the captured fault. The interface supports the following functionalities:

- **Fault interrupt (`interrupt_fault[0]`)**:
  - Use the FAULT_STRUCT0_INTR, FAULT_STRUCT0_INTR_SET, FAULT_STRUCT0_INTR_MASK, and FAULT_STRUCT0_INTR_MASKED registers to monitor, set, and mask the fault structure's interrupt
  - When the interrupt is configured, an interrupt request is sent to the NVIC when FAULT_STRUCT0_STATUS.VALID is set
  - Only one cause exists for the interrupt, indicating that a fault has been detected
  - The fault report structure can be read within the interrupt handler to determine the fault source
- **DMA trigger (`tr_fault[0]`)**:
  - Enable the trigger by setting FAULT_STRUCT0_CTL.TR_EN
  - When the trigger is enabled, it activates the DMA trigger when FAULT_STRUCT0_STATUS.VALID is set
  - The trigger can connect to a DMA controller, which transfers the captured fault information from the fault report structure to memory while clearing the FAULT_STRUCT0_STATUS.VALID field
  - For failure analysis, use a memory location that is retained during a warm or soft reset

- **Chip output signal (`fault_out[0]`)**:
  - Enable the output signal by setting `FAULT_STRUCT0_CTL.OUT_EN`
  - When enabled, the fault structure outputs a signal when `FAULT_STRUCT0_STATUS.VALID` is set
  - This signal can be routed out of the device using the HSIOM to communicate non-recoverable faults to off-chip components, potentially resulting in a chip reset
- **Fault reset request (`fault_reset_req[0]`)**:
  - Enable the reset capability by setting `FAULT_STRUCT0_CTL.RESET_REQ_EN`
  - When enabled, `FAULT_STRUCT0_STATUS.VALID` triggers a soft reset, which is recorded as `RESET_ACT_FAULT` in the `SRSS_RES_CAUSE` register
  - The `FAULT_STRUCT0_STATUS` and `FAULT_STRUCT0_DATA` registers are retained through this reset

The fault report structure offers four distinct signaling interfaces, each with its own enabling functionality. Any enabled interface is activated when `FAULT_STRUCT0_STATUS.VALID` is set.

Since the device has a single `fault_reset_req` signal, the individual `fault_reset_req[x]` signals from multiple fault report structures are combined into a central `fault_reset_req` signal, as shown in Figure 18.

***Note**: 'x' represents different fault report structures sharing the same centralized structure.*

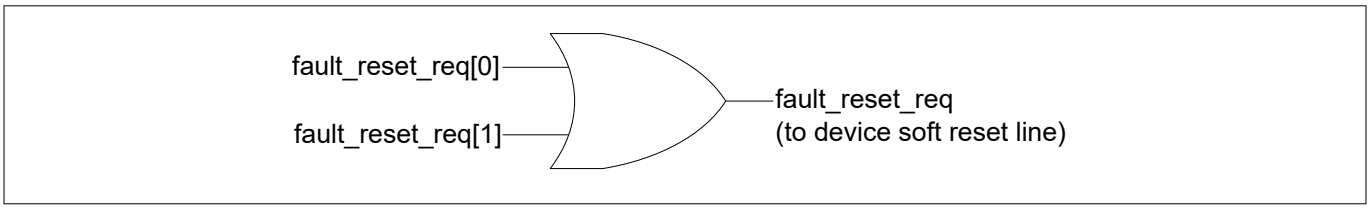


Figure 18 Fault reset

8.2.3 Monitoring

The fault report structure keeps track of all pending faults in the system. The `FAULT_STRUCT0_PENDING0`, `FAULT_STRUCT0_PENDING1`, and `FAULT_STRUCT0_PENDING2` registers show the fault sources that are currently pending. Each register provides a single pending bit for up to 47 fault sources. These `FAULT_STRUCT0_PENDINGx` registers are mirrored in the fault report structure. The fault source numbering scheme matches the numbering scheme of `FAULT_STRUCT0_STATUS.IDX` (see Table 19 for details).

Pending fault sources are not automatically captured by the fault structure. When a pending fault is captured by the fault structure, the associated pending bit is cleared to `0`.

The fault report structure can selectively capture faults. The `FAULT_STRUCT0_MASK0`, `FAULT_STRUCT0_MASK1`, and `FAULT_STRUCT0_MASK2` registers indicate which pending fault sources are enabled for capture by the fault structure. These faults are referred to as "enabled" faults. The fault structure includes its own set of `FAULT_STRUCT0_MASK` registers.

The fault structure only captures "enabled" faults when `FAULT_STRUCT0_STATUS.VALID` is `0`. Once a fault is captured, hardware sets `FAULT_STRUCT0_STATUS.VALID` to `1` and clears the corresponding pending bit to `0`. After the fault structure is processed, firmware or a DMA transfer should clear the `FAULT_STRUCT0_STATUS.VALID` bit to `0`.

Note that fault capturing is independent of the `FAULT_STRUCT0_INTR.FAULT` bit. If an interrupt is enabled, firmware should clear this bit after servicing the interrupt.

8.2.4 Low-power mode operation

The fault report structure functionality is available only in CPU active and CPU sleep power modes. The interfaces between the fault sources and the fault report structures are reset during the CPU Deep Sleep power

mode. Since the fault report structure is an active functionality, pending faults (stored in the FAULT_STRUCT0_PENDING registers) are not retained when transitioning to the CPU Deep Sleep power mode.

The fault structure's registers can be categorized based on their reset domain and retention capabilities, as follows:

- **Active reset domain**:
  - Includes the FAULT_STRUCT0_PENDING, FAULT_STRUCT0_INTR, FAULT_STRUCT0_INTR_SET, and FAULT_STRUCT0_INTR_MASKED registers
  - These registers are not retained in CPU Deep Sleep power mode
- **Deep Sleep reset domain**:
  - Includes the FAULT_STRUCT0_CTL, FAULT_STRUCT0_MASK, and FAULT_STRUCT0_INTR_MASK registers
  - These registers are retained during CPU Deep Sleep power mode
  - Any system reset will reset these registers to their default state
- **Soft reset domain**:
  - Includes the FAULT_STRUCT0_STATUS and FAULT_STRUCT0_DATA registers
  - These registers are retained through soft resets (detectable in the SRSS_RES_CAUSE register)
  - Hard resets, such as XRES, POR, or BOD, will reset these registers

8.2.5 Using a fault structure

Follow these steps to configure and use a fault structure:

1. Identify the faults from Table 19 to be monitored in the system
2. For firmware-driven fault handling through interrupts:
   a. Set the FAULT_STRUCT0_INTR_MASK.FAULT bit to enable the fault interrupt
   b. Clear any pending interrupt by setting the FAULT_STRUCT0_INTR.FAULT
   c. Enable the fault interrupt on the CPU by configuring the appropriate ISER register. See the interrupts section for details
3. For fault handling using the DW:
   a. Set the FAULT_STRUCT0_CTL.TR_EN bit to enable the DMA trigger
   b. Route the tr_fault[0] signal to the trigger input of the DMA controller
   c. Configure and enable the DMA controller to transfer the FAULT_STRUCT0_STATUS and FAULT_STRUCT0_DATA registers to memory. After the transfer is complete, write back 0 to the FAULT_STRUCT0_STATUS register to clear its status
4. For fault handling outside the device:
   a. Set the FAULT_STRUCT0_CTL.OUT_EN bit to enable the chip output signal
   b. Route the fault_out[0] signal to a pin using the HSIOM
   c. Use the signal externally to process the fault. This may include generating an external reset, initiating a power cycle, or logging fault information
5. If a soft reset is required for any detected faults:
   a. Set the FAULT_STRUCT0_CTL.RESET_REQ_EN bit to enable reset-on-fault capability
   b. Clear the FAULT_STRUCT0_STATUS.VALID register to remove any captured fault
6. To define specific faults for capture by the fault structure: Set the appropriate fault index bits in the FAULT_STRUCT0_MASK registers to enable those faults for monitoring

8.2.6 CPU exceptions versus fault monitoring

Some faults captured in Table 19 may also result in bus errors or CPU exceptions, such as Bus, Usage, Memory, or Hard faults. These faults can be communicated in two ways:

- **As a bus error**:
  - The fault is communicated to the master performing the faulting bus transfer
  - This causes exceptions in the CPU, such as Bus, Usage, Memory, or Hard faults
- **As a fault in a fault report structure**:
  - The fault is recorded in the fault report structure and can be communicated as a fault interrupt to any processor in the system
  - This enables fault handling on a processor other than the master of the faulting bus transfer
  - This method is particularly useful for faults that render the master of the faulting transfer unresponsive or unreliable

8.3 Fault sources

The fault sources can vary between device families. Table 19 provides the list of fault sources available in PSC3M/P8, PSC3M/P7 MCU.

Table 19 Fault sources

Fault index	Source	Descriptions	Fault encodings
0	peri.peri_ms0_ppc_vio	Fault interface in the master interface. It is used to report PPC violations (Secure/Non-secure access mismatch, Privilege/User access mismatch, and PC access mismatch) and bus error.	PERI Master0 interface PPC violation
1	peri.peri_ms1_ppc_vio	Fault interface in the Master1 interface. It is used to report PPC violations (Secure/Non-secure access mismatch, Privilege/User access mismatch, and PC access mismatch) and bus error.	PERI Master1 interface PPC violation
2	peri.peri_ppc_pc_mask_vio	Fault interface for PPC MMIO access violation. This fault interface is used to report PPC violation when a locked PC tries to write to the PC_MASK register.	PERI peripheral PPC PC mask violation
3	peri.peri_gp1_timeout_vio	Fault interface for AHB timeout in peripheral Group1	PERI peripheral Group1 AHB timeout
4	peri.peri_gp2_timeout_vio	Fault interface for AHB timeout in peripheral Group2	PERI peripheral Group2 AHB timeout

(table continues...)

Table 19 (continued) Fault sources

Fault index	Source	Descriptions	Fault encodings
5	peri.peri_gp3_timeout_vio	Fault interface for AHB timeout in peripheral Group3	PERI peripheral Group3 AHB timeout
6	peri.peri_gp4_timeout_vio	Fault interface for AHB timeout in peripheral Group4	PERI peripheral Group4 AHB timeout
7	peri.peri_gp5_timeout_vio	Fault interface for AHB timeout in peripheral Group5	PERI peripheral Group5 AHB timeout
8	peri.peri_gp0_ahb_vio	Fault interface for AHB ERROR in peripheral Group0	PERI peripheral Group0 AHB violation
9	peri.peri_gp1_ahb_vio	Fault interface for AHB ERROR in peripheral Group1	PERI peripheral Group1 AHB violation
10	peri.peri_gp2_ahb_vio	Fault interface for AHB ERROR in peripheral Group2	PERI peripheral Group2 AHB violation
11	peri.peri_gp3_ahb_vio	Fault interface for AHB ERROR in peripheral Group3	PERI peripheral Group3 AHB violation
12	peri.peri_gp4_ahb_vio	Fault interface for AHB ERROR in peripheral Group4	PERI peripheral Group4 AHB violation
13	peri.peri_gp5_ahb_vio	Fault interface for AHB ERROR in peripheral Group5	PERI peripheral Group5 AHB violation
14	cpuss.ramc0_mpc_fault_mmio	RAMC-0 MPC fault	RAMC-0 MPC violation
15	cpuss.ramc1_mpc_fault_mmio	RAMC-1 MPC fault	RAMC-1 MPC violation
16	cpuss.exp_ms_ahb_error_mmio	Fault generated due to an AHB error on the EXP bridge	AHB bridge error fault
17	cpuss.promc_mpc_fault_mmio	PROMC MPC fault	PROMC MPC violation
18	cpuss.ramc0_c_ecc_mmio	RAMC-0 correctable ECC Fault	RAMC-0_C ECC mmio
19	cpuss.ramc0_nc_ecc_mmio	RAMC-0 non-correctable ECC fault	RAMC-0_NC ECC mmio
20	cpuss.ramc1_c_ecc_mmio	RAMC-1 correctable ECC fault	RAMC-1_C ECC mmio
21	cpuss.ramc1_nc_ecc_mmio	RAMC-1 non-correctable ECC fault	RAMC-1_NC ECC mmio
22	cpuss.flashc_mpc_fault	FLASHC MPC fault	FLASHC MPC fault
23	cpuss.flashc_bus_access_fault	FLASHC Bus error fault	FLASHC bus access fault

(table continues...)

Table 19 (continued) Fault sources

Fault index	Source	Descriptions	Fault encodings
24	cpuss.flashc_main_c_ecc_mmio	FLASHC ECC correctable Fault	FLASHC main_c ECC mmio fault
25	cpuss.flashc_main_nc_ecc_mmio	FLASHC ECC non-correctable fault	FLASHC main_nc ECC mmio fault
26	cpuss.flashc_fm_ctl_fault	FLASHC FM CTRL fault	FLASHC FM CTL fault
27	cpuss.flashc_fm_r_q_fault	FLASHC FM r_q fault	FLASHC FM r_q fault
28	cpuss.flashc_flash_ctl_fault	FLASHC FLASH CTL fault	FLASHC flash CTL fault
29	cpuss.flashc_addr_hole_fault	FLASHC ADDR HOLE fault	FLASHC addr hole fault
30	cpuss.flashc_lock_fault	FLASHC Lock fault	FLASHC lock fault
31	srss.fault_csv	Clock supervision fault interface	SRSS clock supervisor fault
32	srss.fault_ssv	Supply supervision fault interface	SRSS supply supervisor fault
33	ppca.ppca_ramc0_nc_ecc_mmio	PPCA SRAM0 non-correctable ECC fault	PPCA SRAM0_NC ECC fault
34	ppca.ppca_ramc1_nc_ecc_mmio	PPCA SRAM1 non-correctable ECC fault	PPCA SRAM1_NC ECC fault
35	ppca.ppca_ramc2_nc_ecc_mmio	PPCA SRAM2 non-correctable ECC fault	PPCA SRAM2_NC ECC fault
36	ppca.ppca_ramc3_nc_ecc_mmio	PPCA SRAM3 non-correctable ECC fault	PPCA SRAM3_NC ECC fault
37	ppca.ppca_ramc4_nc_ecc_mmio	PPCA SRAM4 non-correctable ECC fault	PPCA SRAM4_NC ECC fault
38	ppca.ppca_c0_hresp_error_mmio	PPCA CPU0 hresp error fault	PPCA hresp error CPU0 fault
39	ppca.ppca_ramc0_c_ecc_mmio	PPCA SRAM0 correctable ECC fault	PPCA SRAM0_C ECC fault
40	ppca.ppca_ramc1_c_ecc_mmio	PPCA SRAM1 correctable ECC fault	PPCA SRAM1_C ECC fault
41	ppca.ppca_ramc2_c_ecc_mmio	PPCA SRAM2 correctable ECC fault	PPCA SRAM2_C ECC fault
42	ppca.ppca_ramc3_c_ecc_mmio	PPCA SRAM3 correctable ECC fault	PPCA SRAM3_C ECC fault
43	ppca.ppca_ramc4_c_ecc_mmio	PPCA SRAM4 correctable ECC fault	PPCA SRAM4_C ECC fault
44	ppca.ppca_c1_hresp_error_mmio	PPCA CPU1 hresp error fault	PPCA hresp error CPU1 fault
45	svgs.fault	Supply voltage glitch sensor fault	Supply voltage glitch sensor fault

(table continues...)

Table 19 (continued) Fault sources

Fault index	Source	Descriptions	Fault encodings
46	mxaes.fault	Advanced encryption standard fault	Advanced encryption standard fault

8.4 Fault encodings

This section provides the complete encodings of the data fields for each fault. For data fields with an "H" prefix such as HMASTER, HWRITE, and HPROT, see the Arm® AHB protocol specification for more details.

8.4.1 PERI Master0 interface PPC violation

This fault interface is used to report PPC violations and bus errors. The data field 'type of violation' logs only the highest-priority violation if multiple violations occur simultaneously. The peripheral region index value is not valid when the 'type of violation' indicates an AHB error.

Table 20 PERI Master0 interface PPU violation data fields

DATA0	DATA1
[31:0]: Violating address	[9:0]: Peripheral region index [15:12]: Protection context of master interface transaction [18:16]: Type of violation • 0: No violation • 1: Secure/Non-secure access mismatch (second priority) • 2: Privileged/User access mismatch (third priority) • 3: PC access mismatch (first priority) • 4: Reserved • 5: AHB Error (fourth priority) • 6, 7: Reserved [31:24]: HMASTER • 0: main CPU • 0x8: DW0 • 0xe: CryptoLite • 0x12: PPCA expansion bus

8.4.2 PERI Master1 interface PPC violation

This fault interface reports PPC violations and bus errors. The data field 'type of violation' logs only the highest-priority violation when multiple violations occur simultaneously. The peripheral region index value is invalid when the 'type of violation' indicates an AHB error.

Table 21 PERI Master1 interface PPU violation data fields

DATA0	DATA1
[31:0]: Violating address	[9:0]: Peripheral region index [15:12]: Protection context of master interface transaction [18:16]: Type of violation • 0: No violation • 1: Secure/Non-Secure access mismatch (second priority) • 2: Privileged/User access mismatch (third priority) • 3: PC access mismatch (first priority) • 4: Reserved • 5: AHB Error (fourth priority) • 6, 7: Reserved [31:24]: HMASTER • 0: main CPU • 0x8: DW0 • 0xe: CryptoLite • 0x12: PPCA expansion bus

8.4.3 PERI peripheral PPC PC mask violation

This fault interface is used to report a PPC violation that occurs when a locked PC attempts to write to a PC_MASK register.

Table 22 Peripheral PPC PC mask violation data fields

DATA0	DATA1
[31:0]: Violating address	[9:0]: Peripheral region index [11:10]: Reserved [15:12]: Protection context of master interface transaction [18:16]: Type of violation • 4: write to PC_MASK registers from a locked PC • 0, 1, 2, 3, 5, 6, 7: Reserved [23:19]: Reserved [31:24]: HMASTER • 0: CPU0 • 0x8: DW0 • 0xe: CryptoLite • 0x12: PPCA expansion bus

8.4.4 PERI peripheral Group1 AHB timeout

This fault interface reports timeout errors in peripheral Group1. The peripheral group has its own fault interface.

Table 23 Peripheral Group1 AHB timeout data fields

DATA0
[4:0]: Peripheral group slave number [31:5]: Reserved

***Note**: There are two sets, one within the main CPU and another within PPCA.*

8.4.5 PERI peripheral Group2 AHB timeout

This fault interface reports timeout errors in peripheral Group2. The peripheral group has its own fault interface.

Table 24 Peripheral Group2 AHB timeout data fields

DATA0
[4:0]: Peripheral group slave number [31:5]: Reserved

***Note**: There are two sets, one within the main CPU and another within PPCA.*

8.4.6 PERI peripheral Group3 AHB timeout

This fault interface reports timeout errors in peripheral Group3. The peripheral group has its own fault interface.

Table 25 Peripheral Group3 AHB timeout data fields

DATA0
[4:0]: Peripheral group slave number [31:5]: Reserved

***Note**: There are two sets, one within the main CPU and another within PPCA.*

8.4.7 PERI peripheral Group4 AHB timeout

This fault interface reports timeout errors in peripheral Group4. The peripheral group has its own fault interface.

Table 26 Peripheral Group4 AHB timeout data fields

DATA0
[4:0]: Peripheral group slave number [31:5]: Reserved

***Note**: There are two sets, one within the main CPU and another within PPCA.*

8.4.8 PERI peripheral Group5 AHB timeout

This fault interface reports timeout errors in peripheral Group5. The peripheral group has its own fault interface.

Table 27 Peripheral Group5 AHB timeout data fields

DATA0
[4:0]: Peripheral group slave number [31:5]: Reserved

***Note**: There are two sets, one within the main CPU and another within PPCA.*

8.4.9 PERI peripheral Group0 AHB violation

This fault interface reports bus errors on the peripheral group decoder (side facing the bridge). Each peripheral group has its own fault interface.

Table 28 Peripheral Group0 AHB violation data fields

DATA0	DATA1
[31:0]: Violating address	[0]: Privilege mode • 0: User mode • 1: Privilege mode [1]: HNONSEC [2]: HWRITE [3]: Reserved [7:4]: Protection context of master interface transaction [23:8]: HMASTER [31:30]: Error type • 0: AHB error • 1, 2, 3: Reserved

8.4.10 PERI peripheral Group1 AHB violation

This fault interface reports bus errors on the peripheral group decoder (side facing the bridge). Each peripheral group has its own fault interface.

Table 29 Peripheral Group1 AHB violation data fields

DATA0	DATA1
[31:0]: Violating address	[0]: Privilege mode • 0: User mode • 1: Privilege mode [1]: HNONSEC [2]: HWRITE [3]: Reserved [7:4]: Protection context of master interface transaction [23:8]: HMASTER [31:30]: Error type • 0: AHB error • 1, 2, 3: Reserved

8.4.11 PERI peripheral Group2 AHB violation

This fault interface reports bus errors on the peripheral group decoder (side facing the bridge). Each peripheral group has its own fault interface.

Table 30 Peripheral Group2 AHB violation data fields

DATA0	DATA1
[31:0]: Violating address	[0]: Privilege mode • 0: User mode • 1: Privilege mode [1]: HNONSEC [2]: HWRITE [3]: Reserved [7:4]: Protection context of master interface transaction [23:8]: HMASTER [31:30]: Error type • 0: AHB error • 1, 2, 3: Reserved

8.4.12 PERI peripheral Group3 AHB violation

This fault interface reports bus errors on the peripheral group decoder (side facing the bridge). Each peripheral group has its own fault interface.

Table 31 Peripheral Group3 AHB violation data fields

DATA0	DATA1
[31:0]: Violating address	[0]: Privilege mode • 0: User mode • 1: Privilege mode [1]: HNONSEC [2]: HWRITE [3]: Reserved [7:4]: Protection context of master interface transaction [23:8]: HMASTER [31:30]: Error type • 0: AHB error • 1, 2, 3: Reserved

8.4.13 PERI peripheral Group4 AHB violation

This fault interface reports bus errors on the peripheral group decoder (side facing the bridge). Each peripheral group has its own fault interface.

Table 32 Peripheral Group4 AHB violation data fields

DATA0	DATA1
[31:0]: Violating address	[0]: Privilege mode • 0: User mode • 1: Privilege mode [1]: HNONSEC [2]: HWRITE [3]: Reserved [7:4]: Protection context of master interface transaction [23:8]: HMASTER [31:30]: Error type • 0: AHB error • 1, 2, 3: Reserved

8.4.14 PERI peripheral Group5 AHB violation

This fault interface reports bus errors on the peripheral group decoder (side facing the bridge). Each peripheral group has its own fault interface.

Table 33 Peripheral Group5 AHB violation data fields

DATA0	DATA1
[31:0]: Violating address	[0]: Privilege mode • 0: User mode • 1: Privilege mode [1]: HNONSEC [2]: HWRITE [3]: Reserved [7:4]: Protection context of master interface transaction [23:8]: HMASTER [31:30]: Error type • 0: AHB error • 1, 2, 3: Reserved

8.4.15 RAMC-0 MPC violation

This fault is triggered when an access violation occurs, and `MPC_CFG.RESPONSE` is set. Access violations include:

- Reads from non-readable memory
- Writes to non-writable memory
- Non-secure access to secure memory

If `MPC_CFG.RESPONSE` is not set, no fault is generated when an access violation occurs. Instead:

- Read operations return zero
- Write operations are ignored

Table 34 MPC violation data fields

DATA0	DATA1
[31:0]: Violating address	[5:0]: HMASTER [15:6]: Reserved [16]: HNONSEC [17]: Secure or non-secure configuration of a violated block • 0: Secure • 1: Non-secure [18]: HWRITE [23:19]: Reserved [27:24]: Protection context of master interface transaction [29:28]: Unused [30]: Security access violation [31]: PC Access violation

8.4.16 RAMC-1 MPC violation

This fault is triggered when an access violation occurs, and MPC_CFG.RESPONSE is set. Access violations include:

- Reads from non-readable memory
- Writes to non-writable memory
- Non-secure access to secure memory

If MPC_CFG.RESPONSE is not set, no fault is generated when an access violation occurs. Instead:

- Read operations return zero
- Write operations are ignored

Table 35 MPC violation data fields

DATA0	DATA1
[31:0]: Violating address	[5:0]: HMASTER [15:6]: Reserved [16]: HNONSEC [17]: Secure or non-secure configuration of a violated block • 0: Secure • 1: Non-secure [18]: HWRITE [23:19]: Reserved [27:24]: Protection context of master interface transaction [29:28]: Unused [30]: Security access violation [31]: PC Access violation

8.4.17 AHB bridge error fault

This fault interface reports an AHB error on an AHB bridge.

Table 36 AHB bridge error data fields

DATA0	DATA1
[31:0]: Violation address	[0]: HPROT [1]: HNONSEC [2]: HWRITE [3]: Reserved [7:4]: Protection context of master interface transaction [15:8]: HMASTER [31:16]: Reserved

8.4.18 PROMC MPC violation

Table 37 PROMC MPC violation data fields

DATA0	DATA1
[31:0]: Violating address	[5:0]: MASTER [15:6]: Reserved [16]: NONSEC [17]: Reserved [18]: WRITE [23:19]: Reserved [27:24]: Protection context of master interface transaction [29:28]: Unused [30]: Security access violation [31]: PC Access violation

8.4.19 RAMC-0_C ECC mmio

This fault interface reports correctable ECC error detection faults in RAMC-0. The fault is masked if it occurs alongside an MPC access fault.

Table 38 RAMC-0_C ECC mmio data fields

DATA0	DATA1
[31:0]: Violating address	[6:0]: SYNDROME

8.4.20 RAMC-0_NC ECC mmio

This fault interface reports non-correctable ECC error detection faults in RAMC. The fault is masked if it occurs alongside an MPC access fault.

Table 39 RAMC-0_NC ECC mmio data fields

DATA0	DATA1
[31:0]: Violating address	[6:0]: SYNDROME

8.4.21 RAMC-1_C ECC mmio

This fault interface reports correctable ECC error detection faults in RAMC-1. The fault is masked if it occurs alongside an MPC access fault.

Table 40 RAMC-1_C ECC mmio data fields

DATA0	DATA1
[31:0]: Violating address	[6:0]: SYNDROME

8.4.22 RAMC-1_NC ECC mmio

This fault interface reports non-correctable ECC error detection faults in RAMC. The fault is masked if it occurs alongside an MPC access fault.

Table 41 RAMC-1_NC ECC mmio data fields

DATA0	DATA1
[31:0]: Violating address	[6:0]: SYNDROME

8.4.23 FLASHC MPC fault

This is the FLASHC MPC violation fault interface.

Table 42 FLASHC MPC violation data fields

DATA0	DATA1
[31:0]: Violating address	[15:0] MASTER [16] Non-secure of transaction [17] Secure or non-secure configuration of a violated block [18] WR • 1: Write • 0: Read [23:19] Reserved [27:24] Transaction PC of access violation [29:28] Reserved [30] Security access violation [31] PC access violation

8.4.24 FLASHC bus access fault

This fault interface reports the following types of faults:

- C-bus access to FM registers `FM_ADDR` or `FM_REFRESH_ADDR` while in Full R-bus mode
- R-bus access while in Full C-bus mode
- AHB5 write transaction on the 128-bit AHB5 host interface while in Full C-bus/RWW mode with `ENFORCE_PC_LOCK = 0`
- Write to the `FM_ADDR` register in Full R-bus mode through a 128-bit AHB5 bus

Table 43 FLASHC bus access data fields

DATA0	DATA1
[31:0]: Violating address	[0] Full R-bus mode [1] Full C-bus mode [2] RWW mode [3] ENFORCE_PC_LOCK [4] Indicates transaction is from which AHB5 interface [6:5] Type of fault: • 00: C-bus access/write from the 128-bit AHB5 host interface while in full R-bus mode • 01: R-bus access while in full C-bus mode • 10: AHB5 write transaction on the 128-bit AHB5 host interface while in full C-bus/RWW mode, and ENFORCE_PC_LOCK = 0 • 11: Reserved [31:7] Reserved

8.4.25 FLASHC main_c ECC mmio fault

This fault interface reports a fault when the ECC_PRESENT parameter is high, and the ECC recoverable error count exceeds the threshold defined by ECC_LOGIR.ECCTHRESHOLD. This fault is not caused by ECC error injection.

Table 44 flashc_main_c_ecc_mmio data fields

DATA0	DATA1
[31:0] Violating address	[8:0] Syndrome [31:9] Reserved

8.4.26 FLASHC main_nc ECC mmio fault

This fault interface reports a fault when any non-recoverable ECC error occurs. This fault is not caused by ECC error injection.

Table 45 flashc_main_nc_ecc_mmio data fields

DATA0	DATA1
[31:0] Violating address	[8:0] Syndrome [31:9] Reserved

8.4.27 FLASHC FM CTL fault

This fault interface reports the following types of faults:

- Writing to the FLASH_MACRO_CTL register with an invalid mode (IF_SEL = 1 and WR_EN = 1)
- Updating both FLASH_MACRO_CTL.IF_SEL and FLASH_MACRO_CTL.WR_EN simultaneously
- Writing a non-zero value to the TEST_CTL register when FLASH_CTL.ENFORCE_PC_LOCK is high

Table 46 FLASHC FM CTL data fields

DATA0	DATA1
[31:0] Violating address	[0] Value to be written to FLASH_MACRO_CTL.IF_SEL [1] Value to be written to FLASH_MACRO_CTL.WR_EN [2] ENFORCE_PC_LOCK [4:3] Type of fault: • 00: Write to FLASH_MACRO_CTL register with FLASH_MACRO_CTL.IF_SEL and FLASH_MACRO_CTL.WR_EN 1 • 01: Updating FLASH_MACRO_CTL.IF_SEL and FLASH_MACRO_CTL.WR_EN at the same time • 10: Writing non-zero values in the TEST_CTL register when FLASH_CTL.ENFORCE_PC_LOCK is high • 11: Reserved [31:5] Reserved

8.4.28 FLASHC FM r_q fault

This fault interface reports faults caused by R-bus address conflicts (r_q_error from FM). The fault occurs when the read address on the R-bus conflicts with the address on the C-bus.

Table 47 FLASHC FM r_q data fields

DATA0	DATA1
[4:0] Word address [log2(FLASHC_ROWS_PER_SECTOR-1)+4:5] Row address [FM_ADDR_WIDTH-4:FM_ADDR_WIDTH-4-log2(FLASHC_SECTOR_M-1)+1] Sector address [FM_ADDR_WIDTH-3] axa (01 is for SFLASH) [FM_ADDR_WIDTH-2] bxa (01 is for Refresh row) [FM_ADDR_WIDTH-1] cxa (01 is for column 33) [32-FM_ADDR_WIDTH] Reserved	[7:0] Reserved [MASTER_WIDTH+8-1:8] Master ID [31:MASTER_WIDTH+8] Reserved

8.4.29 FLASHC flash CTL fault

This fault interface is designed to report the following faults related to invalid configurations of FLASH_MACRO_CTL.FM_MODE[3:0] based on the status of specific control parameters:

- Setting FLASH_MACRO_CTL.FM_MODE[3:0] to modes 9, 12, and 13 when FLASH_CTL.BLOCK_SECTOR_OPERATIONS is high
- Setting FLASH_MACRO_CTL.FM_MODE[3:0] to modes 6 and 8 when FLASH_CTL.BLOCK_SUBSECTOR_OPERATIONS is high
- Setting FLASH_MACRO_CTL.FM_MODE[3:0] to modes 10 and 15 when FLASH_CTL.BLOCK_BULK_OPERATIONS is high
- Setting FLASH_MACRO_CTL.FM_MODE[3:0] to mode 3 when FLASH_CTL.ENFORCE_PC_LOCK is high

Table 48 FLASHC flash CTL data fields

DATA0
[3:0] FM_MODE[3:0]
[4] ENFORCE_PC_LOCK
[5] BLOCK_SECTOR_OPERATIONS
[6] BLOCK_SUBSECTOR_OPERATIONS
[7] BLOCK_BULK_OPERATIONS
[9:8] Type of fault: • 00: FLASH_CTL.BLOCK_SECTOR_OPERATIONS high forbids software setting FLASH_MACRO_CTL.FM_MODE[3:0] to modes 9, 12, and 13 • 01: FLASH_CTL.BLOCK_SUBSECTOR_OPERATIONS high forbids software setting FLASH_MACRO_CTL.FM_MODE[3:0] to modes 6 and 8 • 10: FLASH_CTL.BLOCK_BULK_OPERATIONS high forbids software setting FLASH_MACRO_CTL.FM_MODE[3:0] to modes 10 and 15 • 11: Setting FLASH_MACRO_CTL.FM_MODE[3:0] to mode 3 when FLASH_CTL.ENFORCE_PC_LOCK is high
[31:10] Reserved

8.4.30 FLASHC addr hole fault

This fault interface reports faults caused by transactions to an invalid address via the 128-bit AHB5 interface. Such transactions may occur due to:

- Wounding
- Targeting FM address holes

Table 49 FLASHC address hole data fields

DATA0
[31:0] Violating address

8.4.31 FLASHC lock fault

This fault interface detects faults related to attempts to write or release FLASH_LOCK.LOCKED under the following conditions:

- By a non-acquired PC
- By the acquired PC with a non-acquired HMASTER_ID

Table 50 FLASHC lock data fields

DATA0
[0] FLASH_LOCK.LOCKED
[4:1] Transaction PC
[8:5] Acquired PC
[MASTER_WIDTH+8:9] Master ID of the violating transfer
[2*MASTER_WIDTH+8:MASTER_WIDTH+9] Acquired Master ID
[31:2*MASTER_WIDTH+9] Reserved

8.4.32 SRSS clock supervisor fault

This fault is generated when the SRSS detects an error in one of the clocks.

Table 51 SRSS clock supervisor (CSV) data fields

DATA0
[6:0]: CSV_HF
[24]: CSV_REF
[25]: CSV_LF
[26]: CSV_ILO

***Note**: For details on the CLK_HF list, see CLK_HF distribution.*

8.4.33 SRSS supply supervisor fault

This fault is generated when the SRSS detects an error in one of the power supplies.

Table 52 SRSS supply supervision (SSV) data field

DATA0
[0]: BOD_VDDA
[16]: HVLVD1
[17]: HVLVD2

8.4.34 PPCA SRAM0_NC ECC fault

This fault interface is used to report uncorrectable ECC error detection faults in PPCA SRAM0, along with the corresponding indication structure.

Table 53 PPCA_SRAM0_NC ECC mmio data fields

DATA0	DATA1
[31:0]: Violating address	[6:0]: SYNDROME

8.4.35 PPCA SRAM1_NC ECC fault

This fault interface is used to report uncorrectable ECC error detection faults in PPCA SRAM1, along with the corresponding indication structure.

Table 54 PPCA_SRAM1_NC ECC mmio data fields

DATA0	DATA1
[31:0]: Violating address	[6:0]: SYNDROME

8.4.36 PPCA SRAM2_NC ECC fault

This fault interface is used to report uncorrectable ECC error detection faults in PPCA SRAM2, along with the corresponding indication structure.

Table 55 PPCA_SRAM2_NC ECC mmio data fields

DATA0	DATA1
[31:0]: Violating address	[6:0]: SYNDROME

8.4.37 PPCA SRAM3_NC ECC fault

This fault interface is used to report uncorrectable ECC error detection faults in PPCA SRAM3, along with the corresponding indication structure.

Table 56 PPCA_SRAM3_NC ECC mmio data fields

DATA0	DATA1
[31:0]: Violating address	[6:0]: SYNDROME

8.4.38 PPCA SRAM4_NC ECC fault

This fault interface is used to report uncorrectable ECC error detection faults in PPCA SRAM4, along with the corresponding indication structure.

Table 57 PPCA_SRAM4_NC ECC mmio data fields

DATA0	DATA1
[31:0]: Violating address	[6:0]: SYNDROME

8.4.39 PPCA hresp error CPU0 fault

This fault interface is used to report PPCA hresp error faults for CPU0, along with the corresponding fault indication structure.

Table 58 PPCA hresp error CPU0 fault data fields

DATA0	DATA1
[31:0]: Violating address	[0] HPROT [1] HNONSEC [2] HWRITE [3] Zero [7:4] HAUSER [15:8] HMASTER

***Note**: All signals are part of the Arm® AHB5 definitions. For more details, see the Arm® AMBA AHB Protocol Specification.*

8.4.40 PPCA SRAM0_C ECC fault

This fault interface is used to report correctable ECC error detection faults in PPCA SRAM0 and provides the corresponding indication structure.

Table 59 PPCA_SRAM0_C ECC mmio data fields

DATA0	DATA1
[31:0]: Violating address	[6:0]: SYNDROME

8.4.41 PPCA SRAM1_C ECC fault

This fault interface is used to report correctable ECC error detection faults in PPCA SRAM1 and provides the corresponding indication structure.

Table 60 PPCA_SRAM1_C ECC mmio data fields

DATA0	DATA1
[31:0]: Violating address	[6:0]: SYNDROME

8.4.42 PPCA SRAM2_C ECC fault

This fault interface is used to report correctable ECC error detection faults in PPCA SRAM2 and provides the corresponding indication structure.

Table 61 PPCA_SRAM2_C ECC mmio data fields

DATA0	DATA1
[31:0]: Violating address	[6:0]: SYNDROME

8.4.43 PPCA SRAM3_C ECC fault

This fault interface is used to report correctable ECC error detection faults in PPCA SRAM3 and provides the corresponding indication structure.

Table 62 PPCA_SRAM3_C ECC mmio data fields

DATA0	DATA1
[31:0]: Violating address	[6:0]: SYNDROME

8.4.44 PPCA SRAM4_C ECC fault

This fault interface is used to report correctable ECC error detection faults in PPCA SRAM4 and provides the corresponding indication structure.

Table 63 PPCA_SRAM4_C ECC mmio data fields

DATA0	DATA1
[31:0]: Violating address	[6:0]: SYNDROME

8.4.45 PPCA hresp error CPU1 fault

This fault interface is used to report PPCA hresp error faults for CPU1 and provides the corresponding fault indication structure.

Table 64 PPCA hresp error CPU1 fault data fields

DATA0	DATA1
[31:0]: Violating address	[0] HPROT [1] HNONSEC [2] HWRITE [3] Zero [7:4] HAUSER [15:8] HMASTER

***Note**: All signals are part of the Arm® AHB5 definitions. For more details, see the Arm® AMBA AHB Protocol Specification.*

8.4.46 Supply voltage glitch sensor fault

This fault interface is used to report SVGS glitch-detection faults.

Table 65 Supply voltage glitch sensor (SVGS) fault data fields

DATA0
[0] `UVC_GLITCH`: UVC Glitch detected
[1] `OVC_GLITCH`: OVC Glitch detected
[2] `UVFC_GLITCH`: UVFC Glitch detected
[3] `NVC_GLITCH`: NVC Glitch detected
[4] `HVC_GLITCH`: HVC Glitch detected

8.4.47 Advanced encryption standard fault

This fault interface is used to report faults in the advanced encryption standard (AES) module.

- When `alarm = 1` in the fault structure, it indicates that the MXAES module has been attacked, and the module has detected this alarm
- The alarm is reported to the SOC using the fault structure field `data[0]`
- Once in this state, the MXAES module becomes inaccessible until a reset (`rst_act_n = 0`) is executed by the SOC
- Internally, the module enters an alarm state and clears its entire content
- During the alarm state, any AHB access to the MXAES module results in a bus-fault response

Table 66 Advanced encryption standard (AES) data fields

DATA0
[0]: alarm

8.5 Registers summary

Table 67 Fault registers summary

Registers	Function
FAULT_STRUCT0_CTL	Fault control registers
FAULT_STRUCT0_STATUS	Fault status registers
FAULT_STRUCT0_DATA	Fault data registers capture fault information
FAULT_STRUCT0_PENDING0	Fault pending0 registers specify pending (not captured) fault sources
FAULT_STRUCT0_PENDING1	Fault pending1 registers specify pending (not captured) fault sources
FAULT_STRUCT0_PENDING2	Fault pending2 registers specify pending (not captured) fault sources
FAULT_STRUCT0_MASK0	Fault mask0 registers specify "enables" for fault sources
FAULT_STRUCT0_MASK1	Fault mask1 registers specify "enables" for fault sources
FAULT_STRUCT0_MASK2	Fault mask2 registers specify "enables" for fault sources
FAULT_STRUCT0_INT	Fault interrupt cause registers
FAULT_STRUCT0_INT_SET	Fault interrupt set registers
FAULT_STRUCT0_INT_MASK	Fault interrupt mask registers
FAULT_STRUCT0_INT_MASKED	Fault interrupt masked registers

9 Direct memory access (DMA - DW)

Direct Memory Access (DMA) transfers data to and from memory, peripherals, and registers using the 32-bit Advanced High-Performance Bus (AHB). These transfers occur independently from the CPU. The DMA can be configured to perform multiple independent data transfers. All data transfers are managed by a channel. There are 16 channels in the DMA. Refer to the MCU datasheet for more device details.

The CPU includes a DMA controller that can be used to transfer data to and from memory, peripherals, and registers. This allows for autonomous, deterministic control of peripherals such as the PWM or enables large data transfers from a peripheral such as the ADC to memory.

The DMA controllers are bus masters in the M33 domain:

- Two DMAs with 16 channels. It has a single transfer engine for all channels that arbitrates for bus master access. This DMA uses the 32-bit AHB bus that shares a clock with the M33. The DMA is optimal for small data transfers; transactional DMA is typically used to transfer data from memory to peripheral or peripheral to memory

***Note**: The DMA is referenced as the Data Wire (DW) in both register names and for triggers in the Trigger multiplexer (Trigger MUX) chapter.*

9.1 Features

Each DMA controller has the following features:

- Supports 16 data channels
- Four levels of priority for each channel
- Descriptors are defined in memory and referenced to the respective channels
- Supports single, 1D, or 2D transfer modes for a descriptor
- Configurable source and destination address increments
- Supports 8-bit, 16-bit, and 32-bit data widths at both source and destination
- Configurable input trigger behavior for each channel
- Configurable interrupt generation in each channel
- Configurable output trigger generation for each channel
- Descriptors can be chained to other descriptors for more complex transfers
- Configurable cyclical redundancy check (CRC)

9.2 Architecture

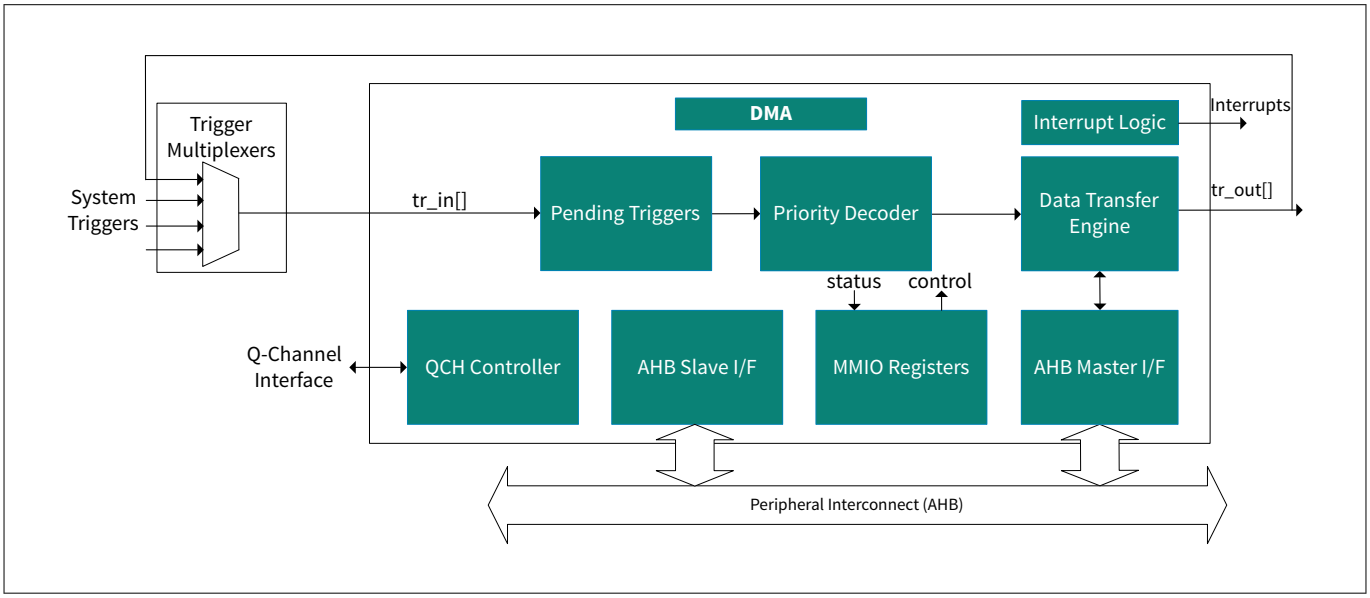


Figure 19 DMA architecture

A data transfer is initiated by an input trigger. This trigger may originate from the source of the transfer, the destination of the transfer, the CPU SW, or from another SoC component. Triggers provide Active/Sleep functionality and are not available in CPU/System DeepSleep and Hibernate power modes.

Once a transfer is triggered, the DMA completes the transfer as configured by the channel descriptor. Channels can be configured to perform a 1D or 2D memory copy. At the end of a transfer, the descriptor can initiate a new transfer by pointing to a new descriptor or generate a completion interrupt.

The DMA is broken up into the following blocks:

- Pending Triggers: Keeps track of activated triggers by locally storing them in pending bits. This is essential because multiple channel triggers may be activated simultaneously, whereas only one channel can be served by the data transfer engine at a time
- 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: Transfers data 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
- AHB Slave I/F: AHB bus slave, which allows the main CPU to access controller MMIO control/status registers
- AHB Master I/F: 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
- Interrupt Logic: Includes interrupt status for each of the channels
- MMIO Registers: Stores each channel's descriptor
- QCH (Q-Channel) Controller: Enables the Power Policy Unit (PPU) to check the current state of the DMA before changing the power state of the device

9.2.1 Standard flow

A typical DMA data transfer follows these steps:

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 MMIO register, which 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 transfer(s)
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 at 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 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. DW can generate an error interrupt if it encounters an error. A channel gets disabled if there is an error

9.3 Channels

The 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 DMA controller.

9.3.1 Channel priority

Only one DMA channel may be moving data at a time. Each channel has a corresponding priority to determine which channel gets bus access if multiple channels are triggered.

Each channel is assigned a priority (CHi_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, round-robin arbitration is applied. Round-robin arbitration (within a priority group) gives the highest priority to the lower channel indices (within the priority group)

Each channel can be marked as pre-emptable by setting CHi_CTL.PREEMPTABLE to '1'. When a higher-priority channel goes into a pending state, it will interrupt a channel marked pre-emptable in between a 1D transfer. After the completion of a higher-priority channel transfer, the channel marked pre-emptable is re-activated by the controller state machine.

9.3.2 Channel security attributes

Each channel has three access control attributes that are used by the memory protection controller (MPC) and peripheral protection controller (PPC) for access control. The following three fields are inherited from the access attributes of the MMIO write transaction that configures the CHi_CTL register.

- "Privileged Mode" (CHi_CTL.P) attribute can be inherited as privileged or user
- "Non-secure" (CHi_CTL.NS) attribute can be inherited as secure or non-secure
- "PC" (CHi_CTL.PC) inherits the protection context from the write transaction

9.3.3 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 the four channel states described in the following table.

Table 68 Channel states

Channel state	Registers	Description
Disabled	CHi_CTL.ENABLED = 0	The channel has been disabled by setting DW_CHi_CTL.ENABLED to 0. The channel trigger is ignored in this state. If an active channel is disabled by software, there should be no assumptions made about the state of the channel (the current "position" of the transfer as reflected by the MMIO registers or descriptor(s)). A software channel re-enable should prepare the new descriptor(s) and reconfigure the channel.
Blocked	CHi_CTL.ENABLED = 1 CHi_STATUS.PENDING = 0	The channel is enabled and is waiting for a trigger to initiate a data transfer.
Pending	CHi_CTL.ENABLED = 1 CHi_STATUS.PENDING = 1	The channel is enabled and has received an active trigger. In this state, the channel is ready to initiate a data transfer but is waiting for it to be scheduled.
Active	CHi_CTL.ENABLED = 1 CHi_STATUS.PENDING = 1 DW_STATUS.CH_IDX = Active Channel DW_STATUS.ACTIVE = 1	The channel is enabled, has received an active trigger, and has been scheduled. It is actively performing data transfer(s). If there are multiple channels pending, the highest-priority pending channel gets scheduled.

Once a channel is in an active state, more information can be found using DW_STATUS.STATE:

- Val 0: Default/inactive state
- Val 1: Loading the channel descriptor from memory
- Val 2: Loading the data element from the source location
- Val 3: Storing the data element at the destination location
- Val 4: CRC functionality (only used when DESCR_CTL.DESCR_TYPE is set to CRC)
- Val 5: Update active control information (source and destination addresses) and wait for trigger deactivation
- Val 6: Error

9.3.4 Channel interrupt generation

Each DMA channel has an interrupt line associated with it. The DMA allows interrupt generation on the completion of a transfer as well as on error conditions. These interrupts can then wake the CPU up at the end of a transfer or signal that the CPU needs to fix a DMA error. Each DMA channel has six registers to control/interact with interrupts. For more information on transfer types, see the Descriptor memory transfer types section.

DESCR_CTL.INTR_TYPE selects what completion interrupt (CHi_STATUS.INTR_CAUSE is set to COMPLETION) is generated by the DMA channel:

- Set 0: An interrupt is generated after a single transfer

- Set 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
- Set 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
- Set 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'

CHi_STATUS.INTR_CAUSE specifies the source of the interrupt cause when CHi_INTR.CH is set to '1'. The following interrupts can be generated in CHi_STATUS.INTR_CAUSE:

- Value 0: No interrupt detected (NO_INTR)
- Value 1: Interrupt on transfer completion configured based on the DESCR_CTL.INTR_TYPE field in the descriptor (COMPLETION)
- Value 2: Source bus error (SRC_BUS_ERROR)
- Value 3: Destination bus error (DST_BUS_ERROR)
- Value 4: Misaligned source address (SRC_MISAL)
- Value 5: Misaligned destination address (DST_MISAL)
- Value 6: Current descriptor pointer is null (CURR_PTR_NULL)
- Value 7: Active channel is in a disabled state (ACTIVE_CH_DISABLED)
- Value 8: Descriptor bus error (DESCR_BUS_ERROR)

CHi_INTR is set to '1' by hardware when an interrupt (as specified by CHi_STATUS.INTR_CAUSE) is detected. The register is cleared by writing a '1'.

CHi_INTR_SET is set to '1' to set the corresponding CHi_INTR field.

CHi_INTR_MASK is used to set the interrupt mask for the specified DMA channel.

CHi_INTR_MASKED when read, gives the bitwise and between CHi_INTR and CHi_INTR_MASK.

For error-related interrupt causes (CHi_STATUS.INTR_CAUSE 2 through 8), the hardware disables the channel (HW sets CHi_CTL.ENABLED to '0'). The interrupt should be cleared before the channel is re-enabled.

The DMA controller is a CPU Active power mode peripheral. Therefore, INTR and INTR_SET are not retained in CPU/System DeepSleep power modes. The INTR_MASK is retained.

9.4 Descriptors

The data transfer between a source and destination in a channel is configured using a collection of 32-bit words called a descriptor. A descriptor is a set of up to six 32-bit words that can be configured to support four different memory transfer types.

The descriptor pointer is specified in the DMA channel registers. CHi_CURR_PTR stores the address of the current descriptor and is updated to DESCR_NEXT_PTR after the execution of the current descriptor. The DMA controller does not modify the descriptor and treats it as read-only.

Each channel includes a CHi_IDX register, which includes the X_IDX and Y_IDX bit fields. These two fields contain the current descriptor X and Y indices as the transfer completes. These are set to zero when CHi_CURR_PTR is updated and are incremented after each X and Y loop.

9.4.1 Descriptor memory transfer types

The DMA supports four different types of memory transfers: single, 1D, 2D, and CRC (cyclic redundancy check). The memory transfer type is configured in DESCR_CTL.DESCR_TYPE but can require up to six configured 32-bit words to support the specified transfer type. The following sections describe the three memory transfer types.

9.4.1.1 Single transfer

The descriptor configuration performs a single data element. The data element size can be configured with DESCR_CTL.DATA_SIZE as 8-bit, 16-bit, or 32-bit. A single transfer requires the configuration of four 32-bit words. The following pseudocode describes the functionality for a single transfer.

Required configurations: DESCR_CTL, DESCR_SRC, DESCR_DST, and DESCR_NEXT_PTR.

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

The following figure illustrates how to move 32 bits in memory using a single transfer.

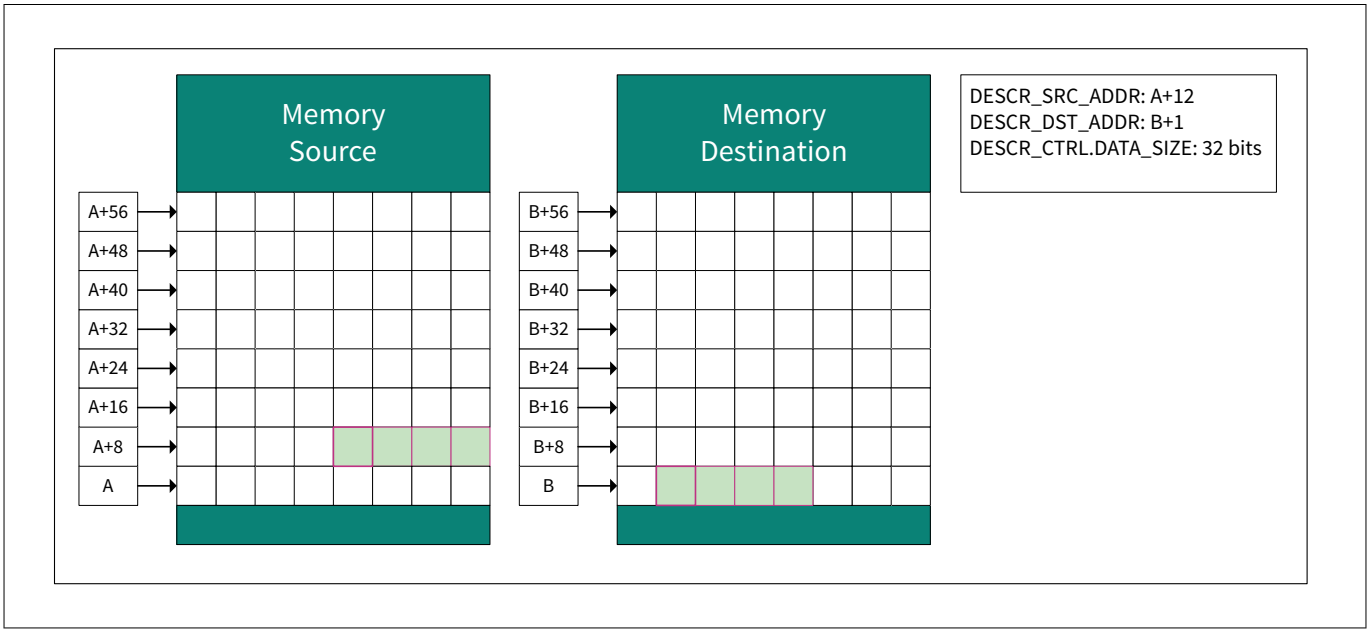


Figure 20 Single transfer

9.4.1.2 1D memory transfer

The descriptor configuration performs a one-dimensional “for loop”, where a channel copies X_COUNT+1 bytes from DESCR_SRC to DESCR_DST. A 1D memory transfer requires the configuration of five 32-bit words. The following pseudocode describes the functionality for a 1D memory transfer.

Required configurations: DESCR_CTL, DESCR_SRC, DESCR_DST, DESCR_X_CTL, and DESCR_NEXT_PTR.

```
// DST_ADDR is a pointer to an object of type defined by DESCR_DST.DST_ADDR
            // DST_X_INCR specifies the source increment between single transfer defined by
DESCR_X_CTL.DST_X_INCR
            // SRC_ADDR is a pointer to an object of type defined by DESCR_SRC.SRC_ADDR
            // SRC_X_INCR specifies the source increment between single transfer defined by
DESCR_X_CTL.SRC_X_INCR
            // t_DATA_SIZE is the type associated with the DESCR_CTL.DATA_SIZE
            // X_COUNT is the number of single transfers defined by DESCR_X_CTL.X_COUNT
            for (X_IDX = 0; X_IDX <= X_COUNT; X_ID++){
            DST_ADDR[X_IDX * DST_X_INCR] = (t_DATA_SIZE) SRC_ADDR[X_IDX * SRC_X_INCR};
            }
```

The following figure illustrates how to move three 32-bit data elements in memory using a 1D memory transfer.

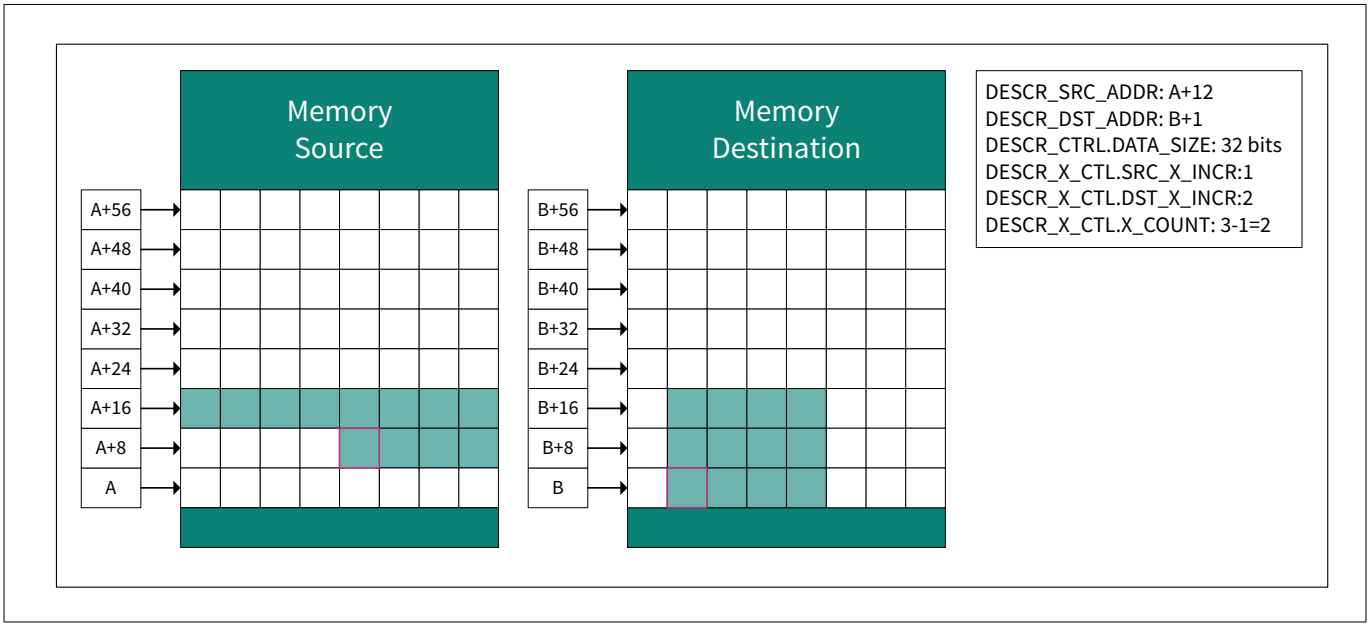


Figure 21 1D memory transfer

9.4.1.3 2D memory transfer

The descriptor configuration performs a two-dimensional “for loop”, where a channel copies (Y_COUNT+1)*(X_COUNT+1) bytes from DESCR_SRC to DESCR_DST. A 2D memory transfer requires the configuration of six 32-bit words. The following pseudocode describes the functionality for a 2D memory transfer.

Required configurations: DESCR_CTL, DESCR_SRC, DESCR_DST, DESCR_X_CTL, DESCR_Y_CTL, and DESCR_NEXT_PTR.

```
// DST_ADDR is a pointer to an object of type defined by DESCR_DST.DST_ADDR
            // SRC_ADDR is a pointer to an object of type defined by DESCR_SRC.SRC_ADDR
            // SRC_X_INCR/DST_X_INCR specifies the source and destination increment between
single transfers defined by DESCR_X_CTL
            // SRC_Y_INCR/DST_Y_INCR specifies the source and destination increment between 1D
transfers defined by DESCR_Y_CTL
            // Y_COUNT is the number of 1D transfers defined by DESCR_Y_CTL.Y_COUNT
            // t_DATA_SIZE is the type associated with the DESCR_CTL.DATA_SIZE
            // X_COUNT is the number of single transfers defined by DESCR_X_CTL.X_COUNT
            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];
            }
            }
```

The following figure illustrates how to move six 32-bit data elements in memory using a 2D memory transfer.

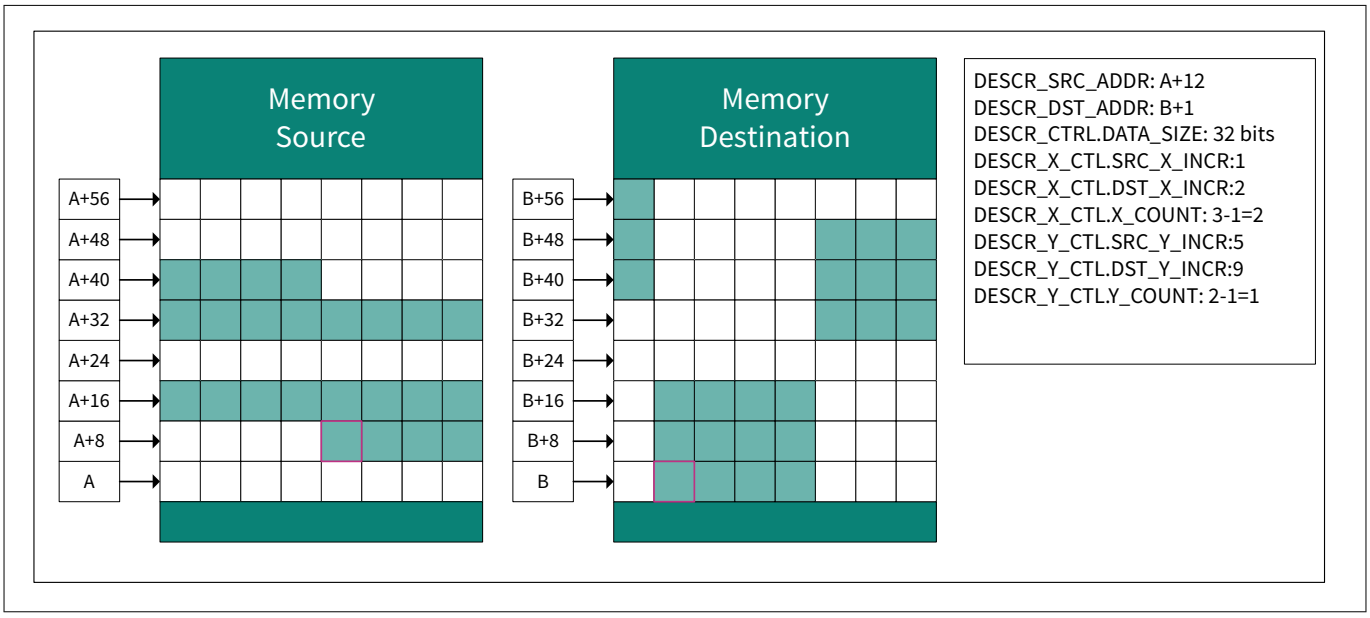


Figure 22 2D memory transfer

9.4.1.4 Cyclic Redundancy Check (CRC) transfer

The descriptor configuration performs a one-dimensional “for loop” similar to the 1D memory transfer. However, the source data is not transferred to a destination. Instead, a CRC checksum is calculated over the source data. A CRC transfer requires the configuration of five 32-bit words. The CRC configuration is provided through a set of registers that is shared by all DW channels, and the assumption is that the DW channels use the CRC functionality, which is mutually exclusive in time. The following pseudocode describes the functionality of a CRC transfer.

Required configurations: DESCR_CTL, DESCR_SRC, DESCR_DST, DESCR_X_CTL, and DESRC_NEXT_PTR.

Required registers: CRC_CTL, CRC_DATA_CTL, CRC_POL_CTL, CRC_LFSR_CTL, CRC_REM_CTL, and CRC_REM_RESULT.

```
// 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 DESCR_SRC
            // 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;
```

9.4.2 Descriptor size

The size of a descriptor depends on its type. Only relevant parameters are stored. For example, a single transfer descriptor does not contain the parameters DESCR_X_CTL and DESCR_Y_CTL. When a channel gets triggered, the descriptor is pulled from memory into the channel's active descriptor registers: CHi_ACT_DESCR_CTL, CHi_ACT_DESCR_SRC, CHi_ACT_DESCR_DST, CHi_ACT_DESCR_X_CTL, CHi_ACT_DESCR_Y_CTL, and CHi_ACT_DESCR_NEXT_PTR. The DMA only pulls the required 32-bit descriptor words for the specified DESCR_TYPE.

9.4.3 Descriptor configuration

There are six 32-bit words that make up a channel descriptor. Not all configurations require all six 32-bit words. When reading the descriptor from memory, only the required descriptor size will be read. This means that for a single transfer, only four 32-bit words are read from memory. The descriptor layout is shown in Figure 19.

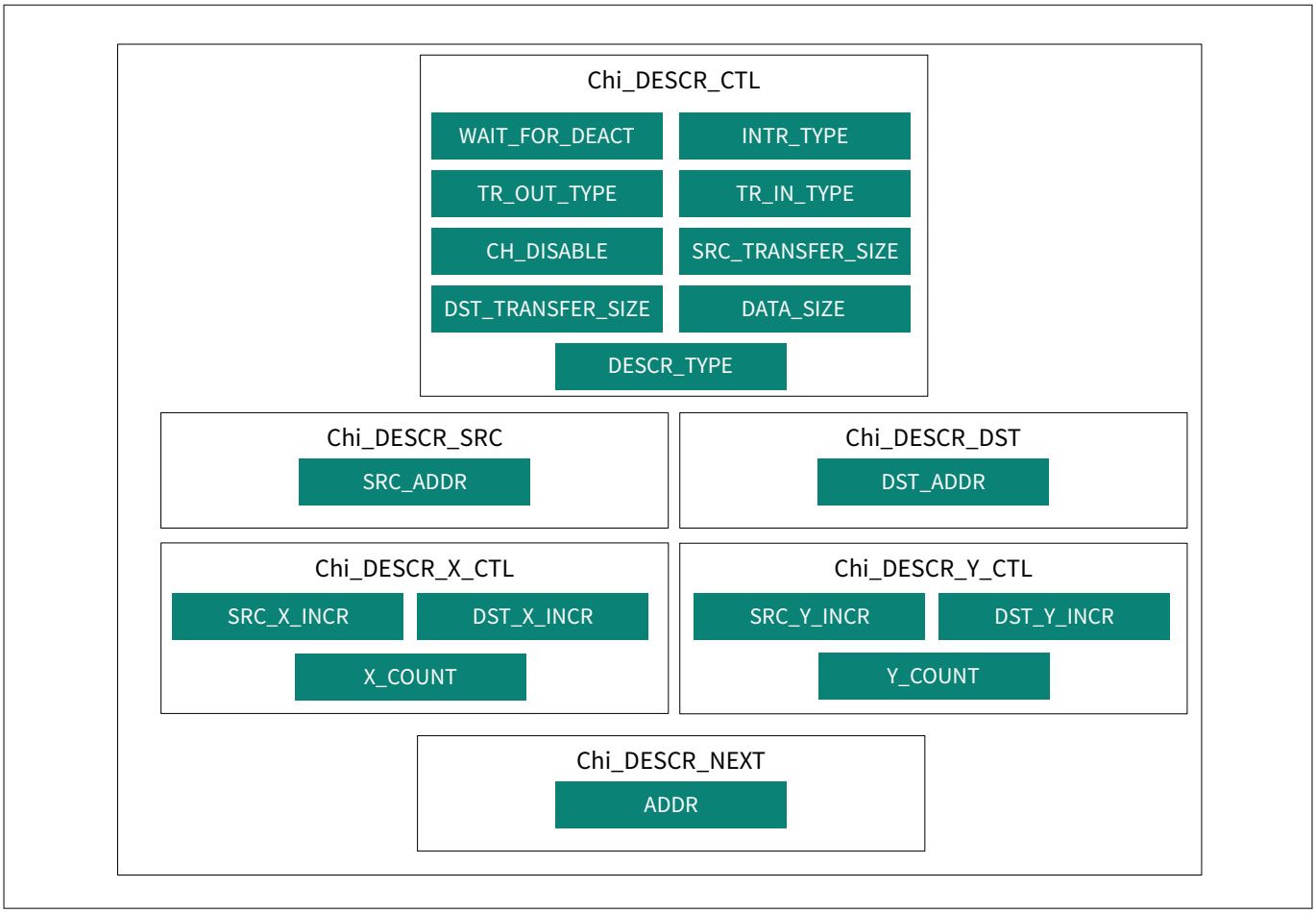


Figure 23 Descriptor layout

9.4.3.1 Descriptor control (DESCR_CTL)

This descriptor control register is broken into seven different controls required by all transfer types.

- WAIT_FOR_DEACT: Specifies whether the 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 no wait (pulse-sensitive triggers), up to 4 cycles, up to 16 cycles, or indefinite. For more information, see the Trigger multiplexer (Trigger MUX) chapter
- INTR_TYPE: This defines when a completion interrupt is generated. The interrupt is generated in CHi_INTR. For more information, see the Channel interrupt generation section
- TR_OUT_TYPE: This defines when an output trigger is generated
- TR_IN_TYPE: An input trigger initiates a data transfer, and the TR_IN_TYPE defines the action on the trigger
- CH_DISABLE: Specifies whether the channel is disabled or not after completion of the current descriptor; this is independent of the value of the DESCR_NEXT_PTR value
- SRC_TRANSFER_SIZE: Specifies the bus transfer size to the source location. This can be set as the data size specified by DATA_SIZE or set to 32 bits
- DST_TRANSFER_SIZE: Specifies the bus transfer size to the destination location. This can be set as the data size specified by DATA_SIZE or set to 32 bits

- DATA_SIZE: Specifies the data element size if SRC_TRANSFER_SIZE or DST_TRANSFER_SIZE uses DATA_SIZE. This can set the data element size to 8-bit, 16-bit, or 32-bit
- DESCR_TYPE: Specifies the transfer type of the descriptor. DMA can be configured for single, 1D, 2D, or CRC transfers. See the Descriptor memory transfer types section for more information

9.4.3.2 Source and destination address (DESCR_SRC/DESCR_DST)

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

When the transfer type is set to CRC, this field should be programmed with the address of the DW_CRC_LSFR_CTL register. The calculated CRC LFSR state is written to this address with the AHB interface.

9.4.3.3 Descriptor X loop control (DESCR_X_CTL)

Used if DESCR_CTL.DESCR_TYPE is configured for 1D or 2D transfers.

- SRC_X_INCR: Specifies the increment of the 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
- DST_X_INCR: Specifies the increment of the 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. This field is not used if the DESCR_TYPE is set to CRC
- 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

9.4.3.4 Descriptor Y loop control (DESCR_Y_CTL)

Used if DESCR_CTL.DESCR_TYPE is configured for 2D transfers.

- SRC_Y_INCR: Specifies the increment of the source address for each Y loop iteration (in multiples of SRC_TRANSFER_SIZE). This field is a signed number in the range [-2048, 2047]
- DST_Y_INCR: Specifies the increment of the destination address for each Y loop iteration (in multiples of DST_TRANSFER_SIZE). This field is a signed number in the range [-2048, 2047]
- 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

9.4.3.5 Descriptor next pointer (DESCR_NEXT_PTR)

Address of the next descriptor in the descriptor list. When this field is "0", this is the last descriptor in the descriptor list.

9.4.4 Descriptor transfer Size

The word width for a transfer can be configured using the transfer/data size parameter in the descriptor. The settings are diversified into source transfer size, destination transfer size, and data size. The data size parameter (DESCR_CTL.DATA_SIZE) sets the width of the bus for the transfer. The source and destination transfer sizes set

by DESCR_CTL.SCR_TRANSFER_SIZE and DESCR_CTL.DST_TRANSFER_SIZE can have a value of either DATA_SIZE or 32 bits. DESCR_CTL.DATA_SIZE can have a 32-bit, 16-bit, or 8-bit setting.

The source and destination transfer size for the DMA must match the addressable width of the source and destination, regardless of the width of data that must be moved. The DATA_SIZE parameter corresponds to the width of the actual data. For example, if a 16-bit PWM is used as a destination for DMA data, the DST_TRANSFER_SIZE must be set to 32 bits to match the width of the PWM register, because the peripheral register width for the TCPWM block (and most peripherals) is always 32 bits wide. However, in this example, the DATA_SIZE for the destination may still be set to 16-bit because the 16-bit PWM only uses two bytes of data.

Table 69 summarizes the possible combinations of the transfer size.

Table 69 Data transfer width

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

9.4.5 Descriptors chaining

When descriptors are chained together, it is known as a descriptor list. The DESCR_NEXT_PTR field contains a pointer to the next descriptor in the chain. Align this pointer to a multiple of 8 bytes. A channel executes the next descriptor in the chain when it completes executing the current descriptor. In the last descriptor in the chain, DESCR_NEXT_PTR is set to “0” (a NULL pointer). 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 SW. A circular list includes descriptors that point to itself.

9.5 Triggers

Each DMA channel has one input trigger and one output trigger. These triggers are used to start a data transfer or to signal another peripheral at the end of the transfer. Both input and output triggers are configured per descriptor.

9.5.1 Input triggers

An input trigger initiates a data transfer. This trigger may originate from the source of the transfer, the destination of the transfer, the CPU SW, or from another peripheral. Triggers provide CPU Active/Sleep functionality and are not available in CPU/System DeepSleep and Hibernate power modes.

Each channel connects to a specific system trigger through a trigger multiplexer that is outside the DMA controller. The trigger multiplexer block routes trigger signals to the specified destinations. The DMA is one such destination. Refer to the Trigger multiplexer (Trigger MUX) chapter for details on the trigger multiplexer block.

The DMA controls what happens on trigger inputs using the TR_IN_TYPE and WAIT_FOR_DEACT fields. These fields are both part of the DESCR_CTL register.

TR_IN_TYPE: Each DMA channel's input triggers are configured using DESCR_CTL.TR_IN_TYPE, which defines the action of each channel when triggered. Table 70 describes the different trigger types.

Table 70 Input trigger types

Trigger type	Description
Type 0	A trigger results in the execution of a single transfer. The descriptor type can be single, 1D, or 2D.
Type 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.
Type 2	A trigger results in the execution of the current descriptor.
Type 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.

WAIT_FOR_DEACT: The DMA has controls for input trigger deactivation. When the DMA transfer based on the TR_IN_TYPE is completed, the data transfer engine checks the state of trigger deactivation. The data transfer on the second trigger is initiated only after the deactivation of the first. The DESCR_CTL.WAIT_FOR_DEACT parameter will determine when the trigger signal is considered deactivated. The first DMA transfer is activated when the trigger is activated, but the transfer is not considered complete until the trigger is deactivated. This field is used to synchronize the controller's data transfers with the agent that generated the trigger. The different trigger deactivations are listed in Table 71.

Table 71 Trigger deactivation types

Deactivation types	Description
Type 0	Pulse trigger: Does not wait for deactivation. When a trigger is detected, the transfer is initiated. After completing the transfer, if the trigger is still active, it is considered another trigger, and the subsequent transfer is initiated immediately.

(table continues...)

Table 71 (continued) Trigger deactivation types

Deactivation types	Description
Type 1	Level-sensitive: Waits for four slow clock cycles after the transfer to be considered as a deactivation. When a trigger is detected, the transfer is initiated. After completing the transfer, if the trigger is still active, then it is considered another trigger after waiting for four cycles. Then, a subsequent transfer is initiated. The transfer corresponding to the trigger is considered complete only at the end of four additional cycles. Even trigger output events will be affected based on this delay. This parameter adds up to a 4-cycle delay in each trigger transaction and hence affects throughput.
Type 2	Level-sensitive: Waits for 16 slow clock cycles after the transfer to be considered as a deactivation. When a trigger is detected, the transfer is initiated. After completing the transfer, if the trigger is still active, then it is considered another trigger after waiting for 16 cycles. Then, a subsequent transfer is initiated. The transfer corresponding to the trigger is considered complete only at the end of the 16 additional cycles. Even trigger output events will be affected based on this delay. This parameter adds up to a 16-cycle delay in each trigger transaction and hence affects throughput.
Type 3	Pulse trigger: Waits indefinitely for deactivation. The DMA transfer is initiated after the trigger signal deactivates. The next transfer is initiated only if the trigger goes low and then high again. A trigger signal that remains active or does not transition to zero between two transactions will simply stall the DMA channel.

A DMA transfer can also be triggered by software. Writing a '1' to CHi_TR_CMD generates a trigger, which sets a trigger pending on the channel. The channel must be enabled.

9.5.2 Pending triggers

A channel is considered pending when an enabled channel is triggered (CHi_STATUS.PEDNING set to 1). Once triggered, the DMA controller initiates a transfer if no other channel is in a pending state.

If there are multiple channels with a pending state, then the DMA will use priority decoding to determine the highest-priority pending channel. This channel is determined as follows:

1. First, the highest-priority group with pending channels is identified
2. Second, within this priority group, round-robin arbitration is applied, which gives the highest priority to the lower channel indices (within the priority group). The channel priority can be set by using CHi_CTL.PRIO

9.5.3 Output triggers

Each channel has an output trigger, which is high for at least two slow clock cycles. This trigger is generated on the completion of a data transfer. At the system level, these output triggers can be connected to the trigger multiplexer component. The trigger multiplexer block is outside the DMA block (refer to the Trigger Multiplexer Block section). This connection allows a DMA controller output trigger to be connected to a DMA controller input trigger. In other words, the completion of a transfer in one channel can activate another channel or even reactivate the same channel.

TR_OUT_TYPE: The DMA channel output triggers are configured using CHi_DESCR_CTL.TR_OUT_TYPE defines when an output trigger is generated. The different trigger types are listed in the following table.

Table 72 Output trigger types

Trigger type	Description
Type 0	An output trigger is generated after a single transfer.
Type 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.
Type 2	An output trigger is generated after the execution of the current descriptor.
Type 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”.

9.6 Performance

A clock cycle estimate can be calculated for a DMA transfer. This estimate depends on the transfer type specified in DESCR_CTL.DESCR_TYPE.

9.6.1 Single transfer

The DMA controller takes 14 clock cycles (of “clk”) for a single data transfer if there are no wait states on the bus infrastructures for loading descriptors or for loading/storing the data elements. The following is a breakdown of the clock cycles:

- Two cycles for trigger synchronization
- One cycle for detecting, priority decoding, and making the channel pending
- One cycle for the controller state machine to detect the pending channel and start
- Two cycles for descriptor channel control information to be loaded from DW channel configuration memory to DW control flip-flops
- Four cycles to load the descriptors (DESCR_CTL, DESCR_SRC, and DESCR_DST)
- One cycle to load the next pointer (DESCT_NEXT_PTR)
- Three cycles for loading the data from the source and for storing the data at the destination

9.6.2 1D transfer

The following is the formula for calculating the number of cycles for completing a 1D transfer of “n” data elements:

Number of cycles = 12 + n*3 + m, where “m” is the total number of wait states seen by the DW controller for loading/storing descriptors/data.

The following is the split of the 12 fixed cycles in the above expression:

- Two cycles for trigger synchronization
- One cycle for detecting, priority decoding, and making the channel pending
- One cycle for the controller state machine to detect the pending channel and start
- Two cycles for descriptor channel control information to be loaded from the DW channel configuration memory to the DW control flip-flops
- Four cycles to load the descriptors (DESCR_CTL, DESCR_SRC, and DESCR_DST)
- One cycle to load descriptor X control (DESCR_X_CTL)
- One cycle to load the next pointer (DESCT_NEXT_PTR)

Each data element transfer from source to destination takes three cycles, which is represented by n*3 in the formula.

9.6.3 2D transfer

The following is the formula for calculating the number of cycles for completing a 2D transfer of “n” data elements:

Number of cycles = 13 + n*3 + m, where “m” is the total number of wait states seen by the DW controller for loading/storing descriptors/data.

The following is the split of the 13 fixed cycles in the above expression:

- Two cycles for trigger synchronization
- One cycle for detecting, priority decoding, and making the channel pending
- One cycle for the controller state machine to detect the pending channel and start
- Two cycles for descriptor channel control information to be loaded from DW channel configuration memory to DW control flip-flops
- Four cycles to load the descriptors (DESCR_CTL, DESCR_SRC, and DESCR_DST)
- One cycle to load descriptor X control (DESCR_X_CTL)
- One cycle to load descriptor Y control (DESCR_Y_CTL)
- One cycle to load the next pointer (DESCT_NEXT_PTR)

Each data element transfer from source to destination takes three cycles, which is represented by n*3 in the formula.

9.6.4 CRC transfer

The following is the formula for calculating the number of cycles for completing the CRC computation of “n” data words.

Number of cycles = 17 + n*9 + m, where “m” is the total number of wait states seen by the DW controller for loading/storing descriptors/data.

The CRC computation for each data word takes ‘9’ clock cycles, which is represented by n*9 in the formula (assuming the CRC descriptor configuration uses word transfers). The extra 17 cycles include trigger synchronization, detection, loading of the descriptor, and storing the computed CRC back at the destination.

10 Trigger multiplexer (Trigger MUX)

Every peripheral in this MCU device is interconnected using trigger signals. Trigger signals are used by peripherals to indicate the occurrence of an event or transition to a state.

These triggers are used to effect or initiate an action in other peripherals. The trigger multiplexers help the user to route triggers from a source peripheral to a destination. Triggers are produced by one peripheral and consumed by another. Unlike interrupts, triggers are used to synchronize between peripherals rather than between a peripheral and the Arm® CPU core.

10.1 Features

Trigger multiplexer has the following features:

- Ability to connect any trigger signal from one peripheral to another
- Provides multiplexer-based trigger groups and 1-to-1 trigger groups
- Supports a software trigger that 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
- Ability to block triggers in debug mode

10.2 Architecture

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 the completion of an action. These trigger signals typically serve as initiators of other actions in other peripheral blocks. An example is an ADC peripheral block sampling three 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 24.

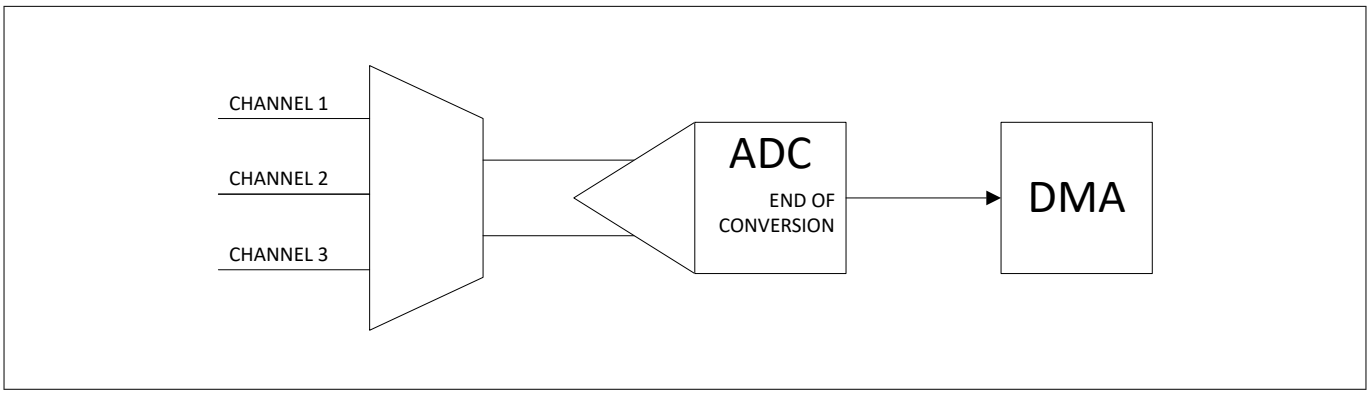


Figure 24 Trigger signal example

The PSC3M/P8, PSC3M/P7 MCU has multiple peripheral blocks; each of these blocks can be connected to other blocks through trigger signals, based on the system implementation. To support this, there is 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 MCU 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 the 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 the occurrence of the trigger.

At a high-level of 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.

In general, a trigger input signal indicates the completion of a peripheral action or a peripheral event. In general, a trigger output signal initiates a peripheral action.

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. The selection of a trigger input signal for a specific trigger output signal (peripheral input signal) is the same for all components.
- 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 on 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 1-to-1 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_GR_TR_CTL.TR_SEL

The trigger component for MCU provides 8 multiplexer-based trigger groups and 1 1-to-1 trigger groups.

10.3 Functional description

The trigger component multiplexes trigger signals. The trigger input source signals are typically peripheral output signals. The trigger output destination signals are typically peripheral input signals.

Examples of trigger input source signals are:

- TCPWM output signals; for example, a counter reaches a pre-programmed limit
- ADC output signals; for example, an ADC conversion has completed
- I/O input signals
- DMA controller output signals that indicate the completion of a transfer

Examples of trigger output destination signals are:

- TCPWM input signals; for example, “start a counter”
- ADC input signals; for example, “start an ADC conversion”
- I/O output signals
- DMA controller input signals to start a transfer

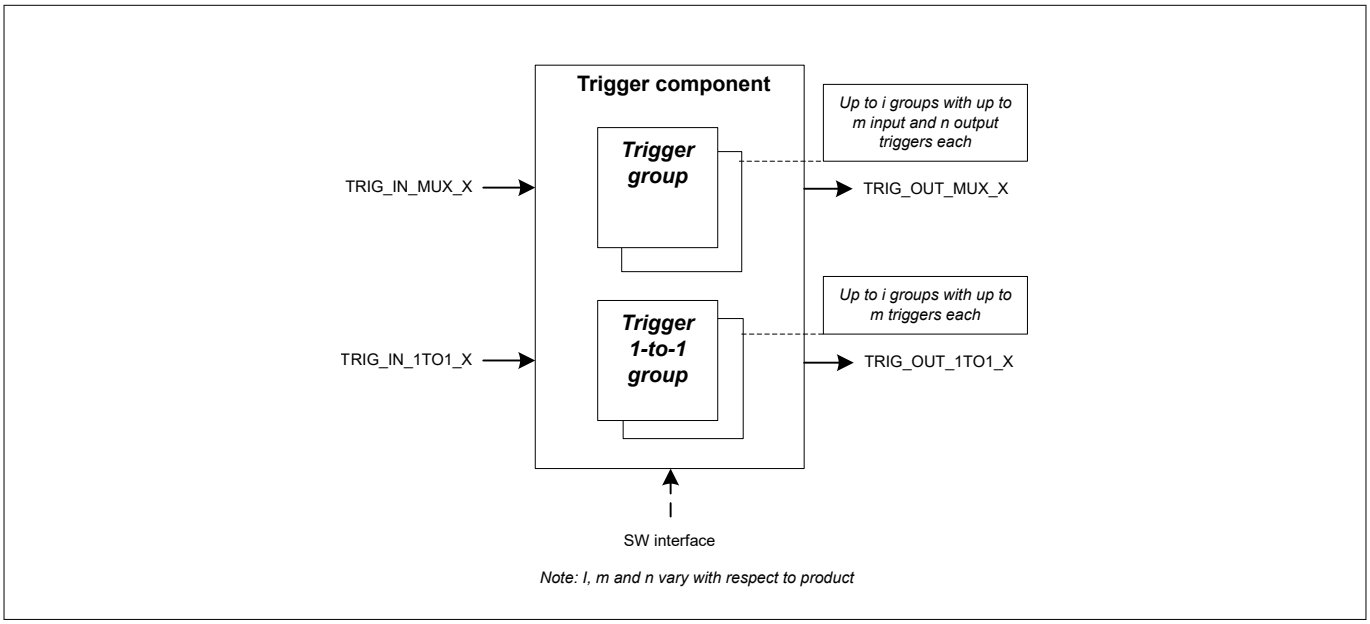


Figure 25 Trigger configuration parameters

Each group is associated with the trigger inputs of a specific peripheral. Figure 25 gives an overview.

Trigger destination peripherals are connected to trigger source peripheral components TRIG_IN_MUX_x and TRIG_IN_1TO1_x. Trigger source peripherals are connected to trigger destination peripheral components TRIG_OUT_MUX_x and TRIG_OUT_1TO1_x. It is important to distinguish between 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 1-to-1 based group, an input trigger TRIG_IN_1TO1_x is connected to an output trigger TRIG_OUT_1TO1_x
- Software control is provided for trigger activation. This control allows for the 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 during a debug halt. The debug mode is indicated by the level trigger input x_DEBUG_FREEZE_TR_IN1, which is connected to a CPUSS CTI trigger output
- 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

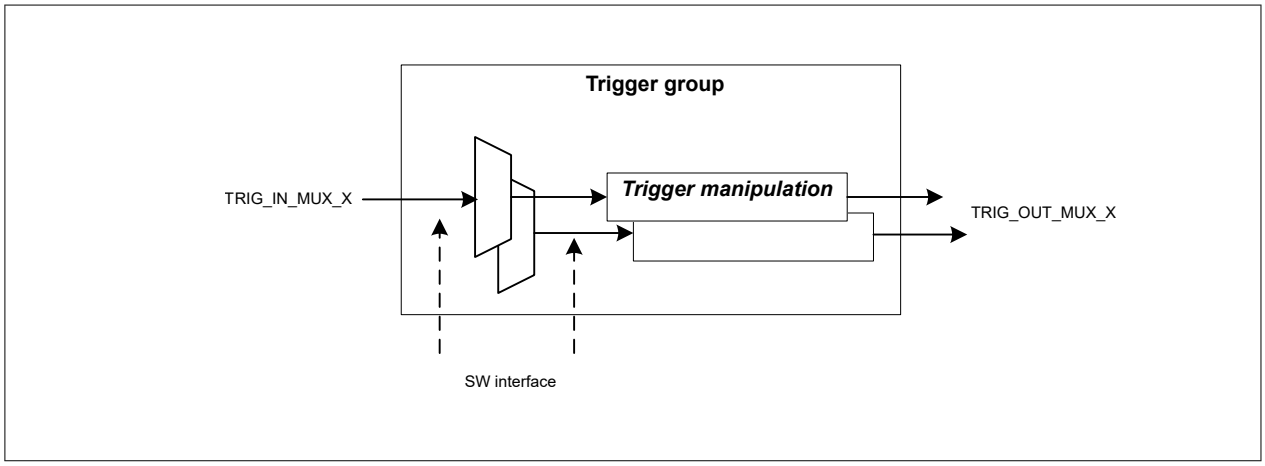


Figure 26 Trigger group

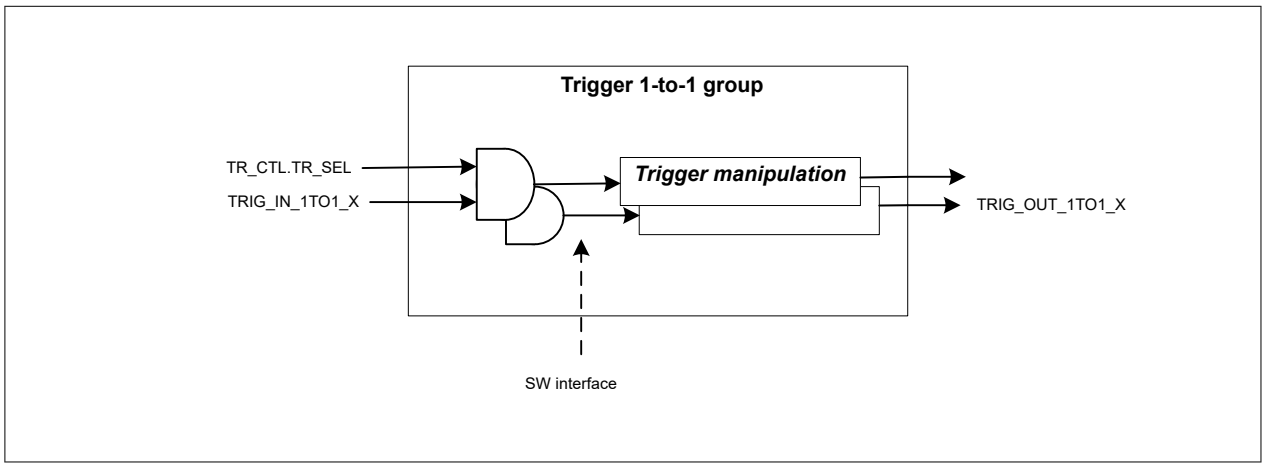


Figure 27 Trigger 1-to-1 group

***Note**: A 1-to-1 group has AND-gate functionality to disable an input trigger*

10.4 Software triggers

All input and output signals to a trigger multiplexer can be triggered by software. This is accomplished by writing into the PERI_TR_CMD register. This register allows you to trigger the corresponding signal for a number of peripheral clock cycles.

10.5 Trigger functionality

Figure 28 gives an overview of a multiplexer trigger group.

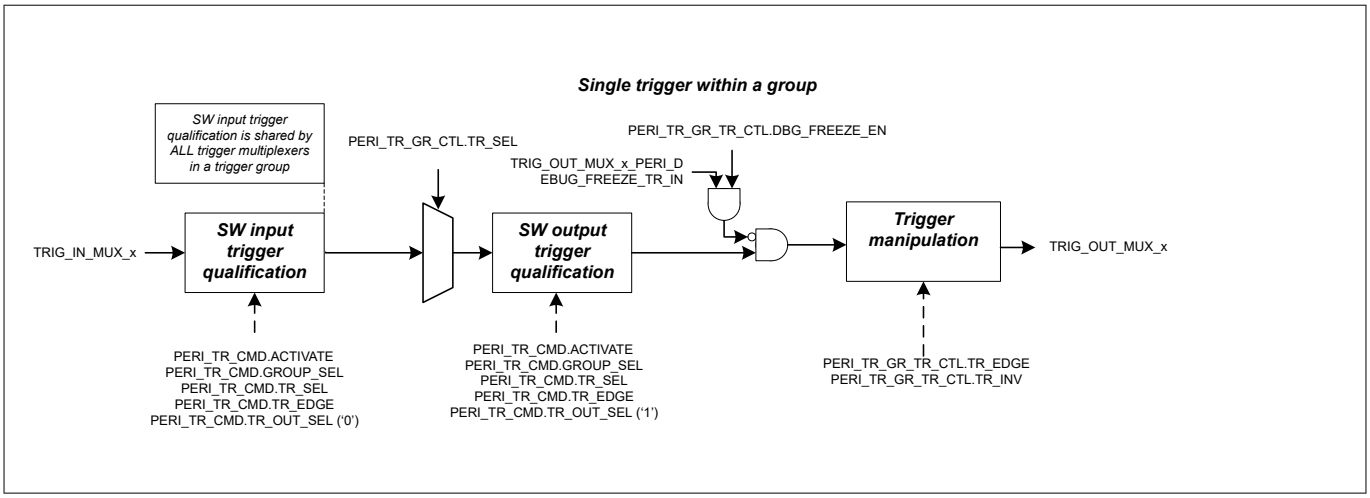


Figure 28 Multiplexer Trigger group

Group trigger configuration:

PERI[Peri Group Number]_TR_GR[Group Number].PERI_TR_GR_TR_CTL[Trigger Number].TR_SEL = Input trigger to be connected to

PERI[Peri Group Number]_TR_GR[Group Number].PERI_TR_GR_TR_CTL[Trigger Number].TR_INV = Invert the trigger or not

PERI[Peri Group Number]_TR_GR[Group Number].PERI_TR_GR_TR_CTL[Trigger Number].TR_EDGE = Edge or level-sensitive trigger

PERI[Peri Group Number]_TR_GR[Group Number].PERI_TR_GR_TR_CTL[Trigger Number].DBG_FREEZE_EN = Block the output trigger generation in debug mode or not

Activating the group trigger via software

PERI[Peri Group Number]_TR_CMD.TR_SEL = Input of Trigger Number

PERI[Peri Group Number]_TR_CMD.GROUP_SEL = Trigger Number Group_Nr

PERI[Peri Group Number]_TR_CMD.TR_EDGE = Edge or level-sensitive trigger

PERI[Peri Group Number]_TR_CMD.OUT_SEL = 1 for output trigger and 0 for input trigger

PERI[Peri Group Number]_TR_CMD.ACTIVATE = 1 signifies configured trigger activation

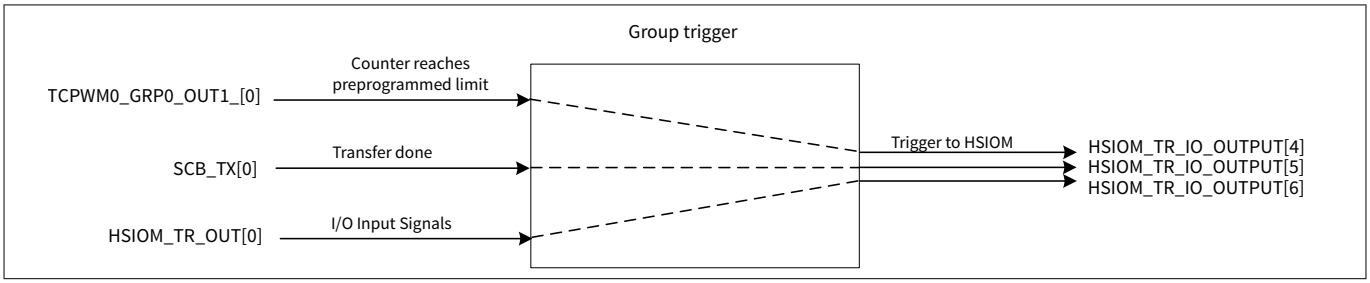


Figure 29 Example of Group Trigger

Figure 30 gives an overview of a 1-to-1 group.

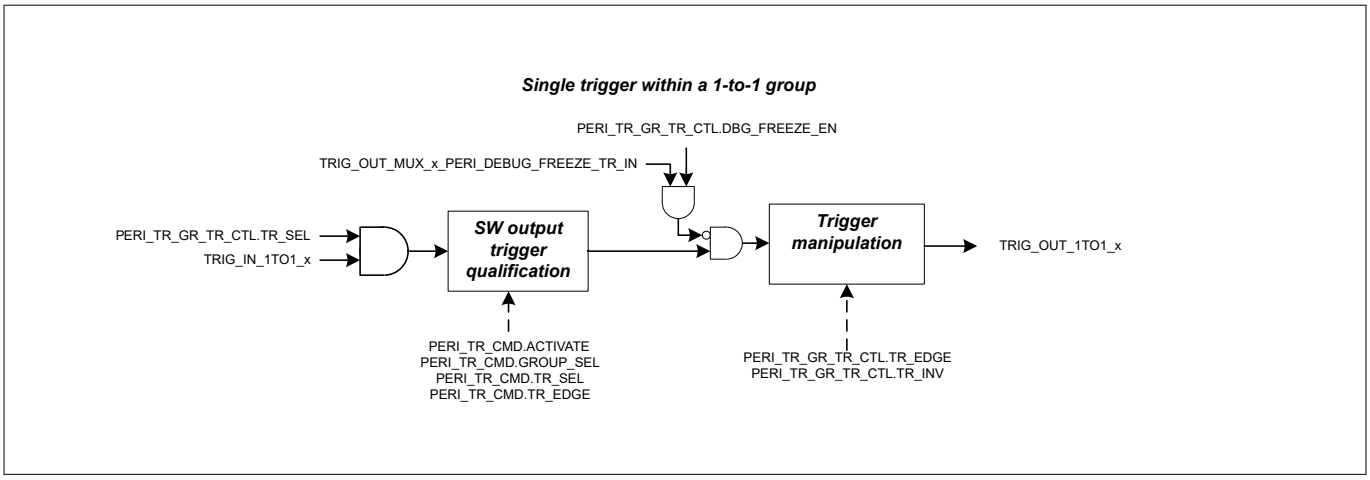


Figure 30 1-to-1 Trigger group

1TO1 trigger configuration:

PERI[Peri Group Number]_TR_1TO1_GR[Group Number]. PERITR_1TO1_GR_TR_CTL[Trigger Number].TR_SEL = True (input trigger) or False (constant signal level '0')

PERI[Peri Group Number]_TR_1TO1_GR[Group Number]. PERI_TR_1TO1_GR_TR_CTL[Trigger Number].TR_INV = Invert the output trigger or not

PERI[Peri Group Number]_TR_1TO1_GR[Group Number]. PERI_TR_1TO1_GR_TR_CTL[Trigger Number].TR_EDGE = Edge or level-sensitive trigger

PERI[Peri Group Number]_TR_1TO1_GR[Group Number]. PERI_TR_1TO1_GR_TR_CTL[Trigger Number].DBG_FREEZE_EN = Block the output trigger generation in debug mode or not

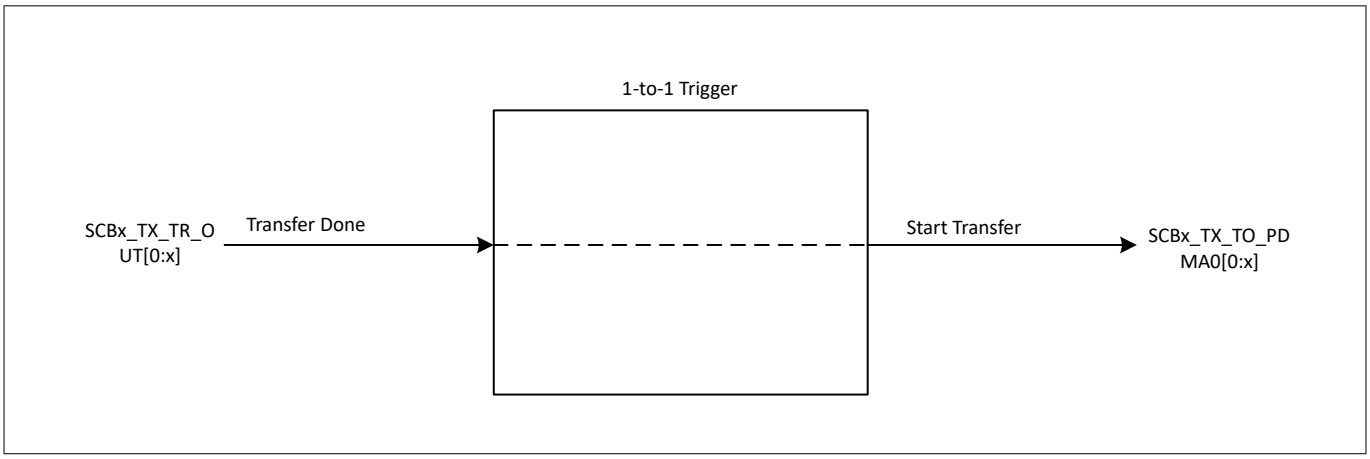


Figure 31 Example of 1-to-1 Trigger

As mentioned, the trigger manipulation logic provides asynchronous edge detection logic. The trigger manipulation is detailed in Figure 32.

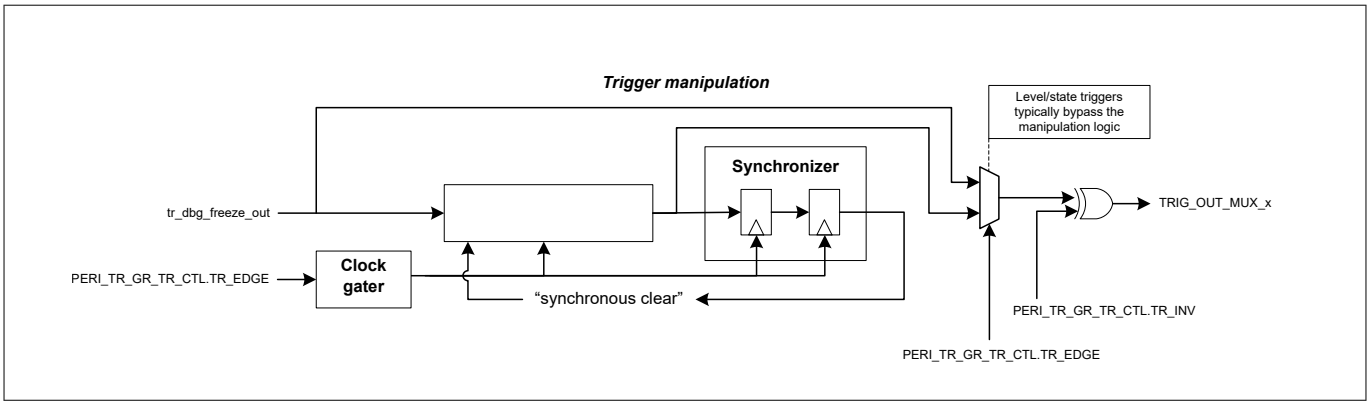


Figure 32 Trigger manipulation

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.

***Note**: There may be multiple combinations of triggers that may lead to a collision in the event that both triggers are enabled simultaneously.*

10.6 Trigger multiplexer connections

An overview of the Trigger multiplexer connections for PSC3M/P8, PSC3M/P7 is shown in Table 73

Table 73 Trigger MUX connections overview

In	Trigger multiplexer	Out
PDMA0[all] TCPWM[all out0, out1] IOSS[all] SCB[all] DEBUG600.cti_tr_out[0:1] CANFD.tmp_rtp_out[0:1] CANFD. tr_dbg_dma_req [0:1] CANFD.tr_fifo0/1 [0:1] LPCOMP[comp0, comp1] CPUSS.tr_fault[0] CRYPTOLITE.tr_trng_bitstream PPCA.dma_req[0:3] PPCA.tr_out	P-DMA0 trigger multiplexer	PDMA0_TR_IN[0:15]

(table continues...)

Table 73 (continued) Trigger MUX connections overview

In	Trigger multiplexer	Out
PDMA0[all] TCPWM[all out0, out1] TCPWM[all line, line_compl] IOSS[all] SCB[all] DEBUG600.cti_tr_out[0:1] CANFD.tmp_rtp_out[0:1] LPCOMP[comp0, comp1] CPUSS.tr_fault[0] CRYPTOLITE.tr_trng_bitstream	HSIOM0 trigger multiplexer	HSIOM_TR_IO_OUTPUT[0:59]
PPCA.tr_out[0:254]	HSIOM1 trigger multiplexer	HSIOM_TR_IO_OUTPUT[60:119]
PDMA0[all] TCPWM[all out0, out1] IOSS[all] SCB[all] DEBUG600.cti_tr_out[0:1] CANFD.tmp_rtp_out[0:1] LPCOMP[comp0, comp1] CPUSS.tr_fault[0] PPCA.tr_out	CPUSS debug multiplexer	CPUSS_CTI_TR_IN[0:1]
CANFD.tmp_rtp_out[0:1]	CAN TT synchronization triggers	CAN_TT_TR_IN[0:1]
CPUSS.tr_fault[0] DEBUG600.cti_tr_out[0:1]	PERI Freeze trigger multiplexer	PERI_DEBUG_FREEZE_TR_IN TCPWM_DEBUG_FREEZE_TR_IN PPCA_DEBUG_FREEZE_TR_IN
PDMA0[all] TCPWM[all out0, out1] IOSS[all] SCB[all] CANFD.tmp_rtp_out[0:1] CPUSS.tr_fault[0] PPCA.tr_out	TCPWM0 trigger multiplexer	TCPWM0_TR_ALL_IN[0:5] TCPWM0_TR_ONE_IN[0:3]

(table continues...)

Table 73 (continued) Trigger MUX connections overview

In	Trigger multiplexer	Out
PDMA0[all] TCPWM[all out0, out1] TCPWM[all line, line_compl] IOSS[all] SCB[all] DEBUG600.cti_tr_out[0:1] CANFD.tmp_rtp_out[0:1] LPCOMP[comp0, comp1] CPUSS.tr_fault[0] CRYPTOLITE.tr_trng_bitstream	PPCA trigger multiplexer	TR_PPCAIN[0:63]

***Note**: The DMA is referenced as the Data Wire (DW) in both register names and for triggers in this section.*

10.6.1 Trigger group inputs

Table 74 Trigger multiplexer group inputs

Input	Trigger	Description
Trigger multiplexer group 0: PDMA0_TR (P-DMA0 trigger multiplexer)		
1:4	PDMA0_TR_OUT[0:3]	Allow PDMA0 to chain to PDMA0. Channels 0 - 3 are for chaining
5:6	TCPWM0_GRP0_OUT0[0:1]	32-bit TCPWM counters out0
7:8	TCPWM0_GRP1_OUT0[0:1]	16-bit TCPWM counters out0
9:10	TCPWM0_GRP0_OUT1[0:1]	32-bit TCPWM counters out1
11:12	TCPWM0_GRP1_OUT1[0:1]	16-bit TCPWM counters out1
13:72	HSIOM_TR_OUT[0:59]	Triggers connected to pins in HSIOM (peri.tr_in)
73	SCB_I2C_SCL[0]	SCB0 I2C trigger
74	SCB_TX[0]	SCB0 TX trigger
75	SCB_RX[0]	SCB0 RX trigger
76	SCB_TGS_RELOAD[0]	SCB0 TGS reload event
77	SCB_TGS_STOP[0]	SCB0 TGS stop event
78	SCB_TGS_GATED[0]	SCB0 TGS gating event
79	SCB_I2C_SCL[1]	SCB1 I2C trigger
80	SCB_TX[1]	SCB1 TX trigger
81	SCB_RX[1]	SCB1 RX trigger
82	SCB_TGS_RELOAD[1]	SCB1 TGS reload event

(table continues...)

Table 74 (continued) Trigger multiplexer group inputs

Input	Trigger	Description
83	SCB_TGS_STOP[1]	SCB1 TGS stop event
84	SCB_TGS_GATED[1]	SCB1 TGS gating event
85	SCB_I2C_SCL[2]	SCB2 I2C trigger
86	SCB_TX[2]	SCB2 TX trigger
87	SCB_RX[2]	SCB2 RX trigger
88	SCB_TGS_RELOAD[2]	SCB2 TGS reload event
89	SCB_TGS_STOP[2]	SCB2 TGS stop event
90	SCB_TGS_GATED[2]	SCB2 TGS gating event
91	SCB_I2C_SCL[3]	SCB3 I2C trigger
92	SCB_TX[3]	SCB3 TX trigger
93	SCB_RX[3]	SCB3 RX trigger
94	SCB_TGS_RELOAD[3]	SCB3 TGS reload event
95	SCB_TGS_STOP[3]	SCB3 TGS stop event
96	SCB_TGS_GATED[3]	SCB3 TGS gating event
97	SCB_I2C_SCL[4]	SCB4 I2C trigger
98	SCB_TX[4]	SCB4 TX trigger
99	SCB_RX[4]	SCB4 RX trigger
100	SCB_TGS_RELOAD[4]	SCB4 TGS reload event
101	SCB_TGS_STOP[4]	SCB4 TGS stop event
102	SCB_TGS_GATED[4]	SCB4 TGS gating event
103	SCB_I2C_SCL[5]	SCB5 I2C trigger
104	SCB_TX[5]	SCB5 TX trigger
105	SCB_RX[5]	SCB5 RX trigger
106	SCB_TGS_RELOAD[5]	SCB5 TGS reload event
107	SCB_TGS_STOP[5]	SCB5 TGS stop event
108	SCB_TGS_GATED[5]	SCB5 TGS gating event
109	SCB_I2C_SCL[6]	SCB6 I2C trigger
110	SCB_TX[6]	SCB6 TX trigger
111	SCB_RX[6]	SCB6 RX trigger
112	SCB_TGS_RELOAD[6]	SCB6 TGS reload event
113	SCB_TGS_STOP[6]	SCB6 TGS stop event
114	SCB_TGS_GATED[6]	SCB6 TGS gating event
115:116	CTI_TR_OUT[0:1]	Debug events

(table continues...)

Table 74 (continued) Trigger multiplexer group inputs

Input	Trigger	Description
117:118	CANFD_TT_TR_OUT[0:1]	CAN TT sync outputs
119:120	CAN_DBG0/1	CAN debug event 0/1
121:122	CAN_FIFO0_CH0/1	CAN FIFO0 CH0/1 event
123:124	CAN_FIFO1_CH0/1	CAN FIFO1 CH0/1 event
125	LPCOMP_DSI_COMP0	LPCOMP Comparator#0
126	LPCOMP_DSI_COMP1	LPCOMP Comparator#1
127	CPUSS_TR_FAULT[0]	CPUSS fault trigger
128	TRNG_BITSTREAM	TRNG bit-stream sequence
129:132	PPCA_DMAREQ[0:3]	DMA Request from PPCA EPU (pulse request)
133:138	PPCA_TR[136:141]	PPCA CLB_OUT[5:0]
139:162	PPCA_TR[142:165]	PPCA ATOPSS ADC_EOC[23:0]
163:174	PPCA_TR[166:177]	PPCA ATOPSS DCMP_OUT[11:0]
175:183	PPCA_TR[178:186]	PPCA ATOPSS DCSG_OUT[8:0]
184:191	PPCA_TR[193:200]	PPCA EPU TR_EPU[7:0]
192:195	PPCA_TR[202:205]	PPCA HWFILT3P3Z0 TR_EOP[3:0]
196:197	PPCA_TR[206:207]	PPCA HWFILT3P3Z1 TR_EOP[1:0]
198:199	PPCA_TR[208:209]	PPCA CORDIC_0 TR_START/ TR_DONE
200:201	PPCA_TR[210:211]	PPCA CORDIC_1 TR_START/ TR_DONE
Trigger multiplexer group 1: HSIOM0 (HSIOM0 trigger multiplexer)		
1:16	PDMA0_TR_OUT[0:15]	P-DMA0 tr_out
17:18	TCPWM0_GRP0_OUT0[0:1]	TCPWM0 group 0 counters out0
19:20	TCPWM0_GRP1_OUT0[0:1]	TCPWM0 group 1 counters out0
21:22	TCPWM0_GRP0_OUT1[0:1]	TCPWM0 group 0 counters out1
23:24	TCPWM0_GRP1_OUT1[0:1]	TCPWM0 group 1 counters out1
25:26	TCPWM0_GRP0_LINE_[0:1]	TCPWM0 group 0 trigger line
27:28	TCPWM0_GRP0_LINE_COMPL_[0:1]	TCPWM0 group 0 trigger line_compl
29:30	TCPWM0_GRP1_LINE_[256:257]	TCPWM0 group 1 trigger line
31:32	TCPWM0_GRP1_LINE_COMPL_[256 :257]	TCPWM0 group 1 trigger line_compl
33:92	HSIOM_TR_OUT[0:59]	Triggers connected to pins in HSIOM (peri.tr_in)
93	SCB_I2C_SCL[0]	SCB0 I2C trigger

(table continues...)

Table 74 (continued) Trigger multiplexer group inputs

Input	Trigger	Description
94	SCB_TX[0]	SCB0 TX trigger
95	SCB_RX[0]	SCB0 RX trigger
96	SCB_TGS_RELOAD[0]	SCB0 TGS reload event
97	SCB_TGS_STOP[0]	SCB0 TGS stop event
98	SCB_TGS_GATED[0]	SCB0 TGS gating event
99	SCB_I2C_SCL[1]	SCB1 I2C trigger
100	SCB_TX[1]	SCB1 TX trigger
101	SCB_RX[1]	SCB1 RX trigger
102	SCB_TGS_RELOAD[1]	SCB1 TGS reload event
103	SCB_TGS_STOP[1]	SCB1 TGS stop event
104	SCB_TGS_GATED[1]	SCB1 TGS gating event
105	SCB_I2C_SCL[2]	SCB2 I2C trigger
106	SCB_TX[2]	SCB2 TX trigger
107	SCB_RX[2]	SCB2 RX trigger
108	SCB_TGS_RELOAD[2]	SCB2 TGS reload event
109	SCB_TGS_STOP[2]	SCB2 TGS stop event
110	SCB_TGS_GATED[2]	SCB2 TGS gating event
111	SCB_I2C_SCL[3]	SCB3 I2C trigger
112	SCB_TX[3]	SCB3 TX trigger
113	SCB_RX[3]	SCB3 RX trigger
114	SCB_TGS_RELOAD[3]	SCB3 TGS reload event
115	SCB_TGS_STOP[3]	SCB3 TGS stop event
116	SCB_TGS_GATED[3]	SCB3 TGS gating event
117	SCB_I2C_SCL[4]	SCB4 I2C trigger
118	SCB_TX[4]	SCB4 TX trigger
119	SCB_RX[4]	SCB4 RX trigger
120	SCB_TGS_RELOAD[4]	SCB4 TGS reload event
121	SCB_TGS_STOP[4]	SCB4 TGS stop event
122	SCB_TGS_GATED[4]	SCB4 TGS gating event
123	SCB_I2C_SCL[5]	SCB5 I2C trigger
124	SCB_TX[5]	SCB5 TX trigger
125	SCB_RX[5]	SCB5 RX trigger
126	SCB_TGS_RELOAD[5]	SCB5 TGS reload event

(table continues...)

Table 74 (continued) Trigger multiplexer group inputs

Input	Trigger	Description
127	SCB_TGS_STOP[5]	SCB5 TGS stop event
128	SCB_TGS_GATED[5]	SCB5 TGS gating event
129	SCB_I2C_SCL[6]	SCB6 I2C trigger
130	SCB_TX[6]	SCB6 TX trigger
131	SCB_RX[6]	SCB6 RX trigger
132	SCB_TGS_RELOAD[6]	SCB6 TGS reload event
133	SCB_TGS_STOP[6]	SCB6 TGS stop event
134	SCB_TGS_GATED[6]	SCB6 TGS gating event
135:136	CTI_TR_OUT[0:1]	Debug events
137:138	CANFD_TT_TR_OUT[0:1]	CAN TT sync outputs
139	LPCOMP_DSI_COMP0	LPCOMP Comparator#0
140	LPCOMP_DSI_COMP1	LPCOMP Comparator#1
141	CPUSS_TR_FAULT[0]	CPUSS fault trigger
142	TRNG_BITSTREAM	TRNG bit-stream sequence
Trigger multiplexer group 2: HSIOM1 (HSIOM1 trigger multiplexer)		
1:255	PPCA_TR[0:254]	PPCA Trigger
Trigger multiplexer group 3: CPUSS_CTI (CPUSS debug multiplexer)		
1:16	PDMA0_TR_OUT[0:15]	P-DMA0 tr_out
17:18	TCPWM0_GRP0_OUT0[0:1]	TCPWM0 group 0 counters out0
19:20	TCPWM0_GRP1_OUT0[0:1]	TCPWM0 group 1 counters out0
21:22	TCPWM0_GRP0_OUT1[0:1]	TCPWM0 group 0 counters out1
23:24	TCPWM0_GRP1_OUT1[0:1]	TCPWM0 group 1 counters out1
25:84	HSIOM_TR_OUT[0:59]	Triggers connected to pins in HSIOM (peri.tr_in)
85	SCB_I2C_SCL[0]	SCB0 I2C trigger
86	SCB_TX[0]	SCB0 TX trigger
87	SCB_RX[0]	SCB0 RX trigger
88	SCB_TGS_RELOAD[0]	SCB0 TGS reload event
89	SCB_TGS_STOP[0]	SCB0 TGS stop event
90	SCB_TGS_GATED[0]	SCB0 TGS gating event
91	SCB_I2C_SCL[1]	SCB1 I2C trigger
92	SCB_TX[1]	SCB1 TX trigger
93	SCB_RX[1]	SCB1 RX trigger
94	SCB_TGS_RELOAD[1]	SCB1 TGS reload event

(table continues...)

Table 74 (continued) Trigger multiplexer group inputs

Input	Trigger	Description
95	SCB_TGS_STOP[1]	SCB1 TGS stop event
96	SCB_TGS_GATED[1]	SCB1 TGS gating event
97	SCB_I2C_SCL[2]	SCB2 I2C trigger
98	SCB_TX[2]	SCB2 TX trigger
99	SCB_RX[2]	SCB2 RX trigger
100	SCB_TGS_RELOAD[2]	SCB2 TGS reload event
101	SCB_TGS_STOP[2]	SCB2 TGS stop event
102	SCB_TGS_GATED[2]	SCB2 TGS gating event
103	SCB_I2C_SCL[3]	SCB3 I2C trigger
104	SCB_TX[3]	SCB3 TX trigger
105	SCB_RX[3]	SCB3 RX trigger
106	SCB_TGS_RELOAD[3]	SCB3 TGS reload event
107	SCB_TGS_STOP[3]	SCB3 TGS stop event
108	SCB_TGS_GATED[3]	SCB3 TGS gating event
109	SCB_I2C_SCL[4]	SCB4 I2C trigger
110	SCB_TX[4]	SCB4 TX trigger
111	SCB_RX[4]	SCB4 RX trigger
112	SCB_TGS_RELOAD[4]	SCB4 TGS reload event
113	SCB_TGS_STOP[4]	SCB4 TGS stop event
114	SCB_TGS_GATED[4]	SCB4 TGS gating event
115	SCB_I2C_SCL[5]	SCB5 I2C trigger
116	SCB_TX[5]	SCB5 TX trigger
117	SCB_RX[5]	SCB5 RX trigger
118	SCB_TGS_RELOAD[5]	SCB5 TGS reload event
119	SCB_TGS_STOP[5]	SCB5 TGS stop event
120	SCB_TGS_GATED[5]	SCB5 TGS gating event
121	SCB_I2C_SCL[6]	SCB6 I2C trigger
122	SCB_TX[6]	SCB6 TX trigger
123	SCB_RX[6]	SCB6 RX trigger
124	SCB_TGS_RELOAD[6]	SCB6 TGS reload event
125	SCB_TGS_STOP[6]	SCB6 TGS stop event
126	SCB_TGS_GATED[6]	SCB6 TGS gating event
127:128	CTI_TR_OUT[0:1]	Debug events

(table continues...)

Table 74 (continued) Trigger multiplexer group inputs

Input	Trigger	Description
129:130	CANFD_TT_TR_OUT[0:1]	CAN TT sync outputs
131	LPCOMP_DSI_COMP0	LPCOMP Comparator#0
132	LPCOMP_DSI_COMP1	LPCOMP Comparator#1
133	CPUSS_TR_FAULT[0]	CPUSS fault trigger
133:145	PPCA_TR[52:63]	PPCA TCPWM_SS0 TR_OUT_0[11:0]
146:153	PPCA_TR[76:83]	PPCA TCPWM_SS0 MO_TR_OUT[7:0]
154:161	PPCA_TR[84:91]	PPCA TCPWM_SS1 TR_OUT_0[7:0]
162:169	PPCA_TR[100:107]	PPCA TCPWM_SS1 MO_TR_OUT[7:0]
170:173	PPCA_TR[108:111]	PPCA TCPWM_SS2 TR_OUT_0[3:0]
174:175	PPCA_TR[116:117]	PPCA TCPWM_SS3 TR_OUT_0[1:0]
176:181	PPCA_TR[136:141]	PPCA CLB_OUT[5:0]
182:205	PPCA_TR[142:165]	PPCA ATOPSS ADC_EOC[23:0]
206:217	PPCA_TR[166:177]	PPCA ATOPSS DCMP_OUT[11:0]
218:226	PPCA_TR[178:186]	PPCA ATOPSS DCSG_OUT[8:0]
227:234	PPCA_TR[193:200]	PPCA EPU TR_EPU[7:0]
235	PPCA_TR[212]	PPCA CPUSS CM33_0_TXEV
236	PPCA_TR[213]	PPCA CPUSS CM33_1_TXEV
237	PPCA_TR[214]	PPCA CPUSS CM33_0_LOCKUP
238	PPCA_TR[215]	PPCA CPUSS CM33_1_LOCKUP
239	PPCA_TR[216]	PPCA CPUSS CM33_0_HALTED
240	PPCA_TR[217]	PPCA CPUSS CM33_1_HALTED
Trigger multiplexer group 4: CANTT (CAN TT synchronization triggers)		
1	CAN_TT_TR_OUT[0]	CAN TT sync outputs
2	CAN_TT_TR_OUT[1]	CAN TT sync outputs
Trigger multiplexer group 5: PERI_FREEZE (PERI freeze trigger multiplexer)		
1	CPUSS_TR_FAULT[0]	CPUSS fault trigger
2:3	CTI_TR_OUT[0:1]	Debug events
Trigger multiplexer group 6: TCPWM0 (TCPWM0 trigger multiplexer)		
1:16	PDMA0_TR_OUT[0:15]	P-DMA0 tr_out
17:18	TCPWM0_GRP0_OUT0[0:1]	TCPWM0 group 0 counters out0
19:20	TCPWM0_GRP1_OUT0[0:1]	TCPWM0 group 1 counters out0
21:22	TCPWM0_GRP0_OUT1[0:1]	TCPWM0 group 0 counters out1

(table continues...)

Table 74 (continued) Trigger multiplexer group inputs

Input	Trigger	Description
23:24	TCPWM0_GRP1_OUT1[0:1]	TCPWM0 group 1 counters out1
25:84	HSIOM_TR_OUT[0:59]	Triggers connected to pins in HSIOM (peri.tr_in)
85	SCB_I2C_SCL[0]	SCB0 I2C trigger
86	SCB_TX[0]	SCB0 TX trigger
87	SCB_RX[0]	SCB0 RX trigger
88	SCB_TGS_RELOAD[0]	SCB0 TGS reload event
89	SCB_TGS_STOP[0]	SCB0 TGS stop event
90	SCB_TGS_GATED[0]	SCB0 TGS gating event
91	SCB_I2C_SCL[1]	SCB1 I2C trigger
92	SCB_TX[1]	SCB1 TX trigger
93	SCB_RX[1]	SCB1 RX trigger
94	SCB_TGS_RELOAD[1]	SCB1 TGS reload event
95	SCB_TGS_STOP[1]	SCB1 TGS stop event
96	SCB_TGS_GATED[1]	SCB1 TGS gating event
97	SCB_I2C_SCL[2]	SCB2 I2C trigger
98	SCB_TX[2]	SCB2 TX trigger
99	SCB_RX[2]	SCB2 RX trigger
100	SCB_TGS_RELOAD[2]	SCB2 TGS reload event
101	SCB_TGS_STOP[2]	SCB2 TGS stop event
102	SCB_TGS_GATED[2]	SCB2 TGS gating event
103	SCB_I2C_SCL[3]	SCB3 I2C trigger
104	SCB_TX[3]	SCB3 TX trigger
105	SCB_RX[3]	SCB3 RX trigger
106	SCB_TGS_RELOAD[3]	SCB3 TGS reload event
107	SCB_TGS_STOP[3]	SCB3 TGS stop event
108	SCB_TGS_GATED[3]	SCB3 TGS gating event
109	SCB_I2C_SCL[4]	SCB4 I2C trigger
110	SCB_TX[4]	SCB4 TX trigger
111	SCB_RX[4]	SCB4 RX trigger
112	SCB_TGS_RELOAD[4]	SCB4 TGS reload event
113	SCB_TGS_STOP[4]	SCB4 TGS stop event
114	SCB_TGS_GATED[4]	SCB4 TGS gating event
115	SCB_I2C_SCL[5]	SCB5 I2C trigger

(table continues...)

Table 74 (continued) Trigger multiplexer group inputs

Input	Trigger	Description
116	SCB_TX[5]	SCB5 TX trigger
117	SCB_RX[5]	SCB5 RX trigger
118	SCB_TGS_RELOAD[5]	SCB5 TGS reload event
119	SCB_TGS_STOP[5]	SCB5 TGS stop event
120	SCB_TGS_GATED[5]	SCB5 TGS gating event
121	SCB_I2C_SCL[6]	SCB6 I2C trigger
122	SCB_TX[6]	SCB6 TX trigger
123	SCB_RX[6]	SCB6 RX trigger
124	SCB_TGS_RELOAD[6]	SCB6 TGS reload event
125	SCB_TGS_STOP[6]	SCB6 TGS stop event
126	SCB_TGS_GATED[6]	SCB6 TGS gating event
127:128	CANFD_TT_TR_OUT[0:1]	CAN TT sync outputs
129	CPUSS_TR_FAULT[0]	CPUSS fault trigger
130:141	PPCA_TR[52:63]	PPCA TCPWM_SS0 TR_OUT_0[11:0]
142:149	PPCA_TR[76:83]	PPCA TCPWM_SS0 MO_TR_OUT[7:0]
150:157	PPCA_TR[84:91]	PPCA TCPWM_SS1 TR_OUT_0[7:0]
158:165	PPCA_TR[100:107]	PPCA TCPWM_SS1 MO_TR_OUT[7:0]
166:169	PPCA_TR[108:111]	PPCA TCPWM_SS2 TR_OUT_0[3:0]
170:171	PPCA_TR[116:117]	PPCA TCPWM_SS3 TR_OUT_0[1:0]
172:177	PPCA_TR[136:141]	PPCA CLB_OUT[5:0]
178:201	PPCA_TR[142:165]	PPCA ATOPSS ADC_EOC[23:0]
202:213	PPCA_TR[166:177]	PPCA ATOPSS DCMP_OUT[11:0]
214:222	PPCA_TR[178:186]	PPCA ATOPSS DCSG_OUT[8:0]
223:230	PPCA_TR[193:200]	PPCA EPU TR_EPU[7:0]
Trigger multiplexer group 7: PPCA (PPCA trigger multiplexer)		
1:16	PDMA0_TR_OUT[0:15]	P-DMA0 tr_out
17:18	TCPWM0_GRP0_OUT0[0:1]	TCPWM0 group 0 counters out0
19:20	TCPWM0_GRP1_OUT0[0:1]	TCPWM0 group 1 counters out0
21:22	TCPWM0_GRP0_OUT1[0:1]	TCPWM0 group 0 counters out1
23:24	TCPWM0_GRP1_OUT1[0:1]	TCPWM0 group 1 counters out1
25:26	TCPWM0_GRP0_LINE_[0:1]	TCPWM0 group 0 trigger line
27:28	TCPWM0_GRP0_LINE_COMPL_[0:1]	TCPWM0 group 0 trigger line_compl

(table continues...)

Table 74 (continued) Trigger multiplexer group inputs

Input	Trigger	Description
29:30	TCPWM0_GRP1_LINE_[256:257]	TCPWM0 group 1 trigger line
31:32	TCPWM0_GRP1_LINE_COMPL_[256:257]	TCPWM0 group 1 trigger line_compl
33:92	HSIOM_TR_OUT[0:59]	Triggers connected to pins in HSIOM (peri.tr_in)
93	SCB_I2C_SCL[0]	SCB0 I2C trigger
94	SCB_TX[0]	SCB0 TX trigger
95	SCB_RX[0]	SCB0 RX trigger
96	SCB_TGS_RELOAD[0]	SCB0 TGS reload event
97	SCB_TGS_STOP[0]	SCB0 TGS stop event
98	SCB_TGS_GATED[0]	SCB0 TGS gating event
99	SCB_I2C_SCL[1]	SCB1 I2C trigger
100	SCB_TX[1]	SCB1 TX trigger
101	SCB_RX[1]	SCB1 RX trigger
102	SCB_TGS_RELOAD[1]	SCB1 TGS reload event
103	SCB_TGS_STOP[1]	SCB1 TGS stop event
104	SCB_TGS_GATED[1]	SCB1 TGS gating event
105	SCB_I2C_SCL[2]	SCB2 I2C trigger
106	SCB_TX[2]	SCB2 TX trigger
107	SCB_RX[2]	SCB2 RX trigger
108	SCB_TGS_RELOAD[2]	SCB2 TGS reload event
109	SCB_TGS_STOP[2]	SCB2 TGS stop event
110	SCB_TGS_GATED[2]	SCB2 TGS gating event
111	SCB_I2C_SCL[3]	SCB3 I2C trigger
112	SCB_TX[3]	SCB3 TX trigger
113	SCB_RX[3]	SCB3 RX trigger
114	SCB_TGS_RELOAD[3]	SCB3 TGS reload event
115	SCB_TGS_STOP[3]	SCB3 TGS stop event
116	SCB_TGS_GATED[3]	SCB3 TGS gating event
117	SCB_I2C_SCL[4]	SCB4 I2C trigger
118	SCB_TX[4]	SCB4 TX trigger
119	SCB_RX[4]	SCB4 RX trigger
120	SCB_TGS_RELOAD[4]	SCB4 TGS reload event
121	SCB_TGS_STOP[4]	SCB4 TGS stop event

(table continues...)

Table 74 (continued) Trigger multiplexer group inputs

Input	Trigger	Description
122	SCB_TGS_GATED[4]	SCB4 TGS gating event
123	SCB_I2C_SCL[5]	SCB5 I2C trigger
124	SCB_TX[5]	SCB5 TX trigger
125	SCB_RX[5]	SCB5 RX trigger
126	SCB_TGS_RELOAD[5]	SCB5 TGS reload event
127	SCB_TGS_STOP[5]	SCB5 TGS stop event
128	SCB_TGS_GATED[5]	SCB5 TGS gating event
129	SCB_I2C_SCL[6]	SCB6 I2C trigger
130	SCB_TX[6]	SCB6 TX trigger
131	SCB_RX[6]	SCB6 RX trigger
132	SCB_TGS_RELOAD[6]	SCB6 TGS reload event
133	SCB_TGS_STOP[6]	SCB6 TGS stop event
134	SCB_TGS_GATED[6]	SCB6 TGS gating event
135:136	CTI_TR_OUT[0:1]	Debug events
137:138	CANFD_TT_TR_OUT[0:1]	CAN TT sync outputs
139	LPCOMP_DSI_COMP0	LPCOMP Comparator#0
140	LPCOMP_DSI_COMP1	LPCOMP Comparator#1
141	CPUSS_TR_FAULT[0]	CPUSS fault trigger
142	TRNG_BITSTREAM	TRNG bit-stream sequence

10.6.2 Trigger group outputs

Table 75 Trigger multiplexer group outputs

Output	Trigger	Description
Trigger multiplexer group 0: PDMA0_TR (P-DMA0 trigger multiplexer)		
0:15	PDMA0_TR_IN[0:15]	Triggers to P-DMA0
Trigger multiplexer group 1: HSIOM0 (HSIOM0 trigger multiplexer)		
0:59	HSIOM_TR_IO_OUTPUT[0:59]	Triggers to HSIOM
Trigger multiplexer group 2: HSIOM1 (HSIOM1 trigger multiplexer)		
0:59	HSIOM_TR_IO_OUTPUT[60:119]	Triggers to HSIOM
Trigger multiplexer group 3: CPUSS_CTI (CPUSS debug multiplexer)		
0:1	CPUSS_CTI_TR_IN[0:1]	Triggers to debug600.cti
Trigger multiplexer group 4: CANTT (CAN TT synchronization triggers)		
0	CAN_TT_TR_IN[0]	CAN TT sync input

(table continues...)

Table 75 (continued) Trigger multiplexer group outputs

Output	Trigger	Description
1	CAN_TT_TR_IN[1]	CAN TT sync input
Trigger multiplexer group 5: PERI_FREEZE (PERI freeze trigger multiplexer)		
0	PERI_DEBUG_FREEZE_TR_IN	Trigger to freeze PERI operation
1	TCPWM_DEBUG_FREEZE_TR_IN	Trigger to freeze TCPWM operation
2	PPCA_DEBUG_FREEZE_TR_IN	Trigger to freeze PPCA operation
Trigger multiplexer group 6: TCPWM0 (TCPWM0 trigger multiplexer)		
0:5	TCPWM0_TR_ALL_IN[0:5]	Triggers to TCPWM0
6	TCPWM0_TR_ONE_IN[0]	TCPWM0 Group #0 Counter #0 input trigger
7	TCPWM0_TR_ONE_IN[1]	TCPWM0 Group #0 Counter #1 input trigger
8	TCPWM0_TR_ONE_IN[2]	TCPWM0 Group #1 Counter #0 input trigger
9	TCPWM0_TR_ONE_IN[3]	TCPWM0 Group #1 Counter #1 input trigger
Trigger multiplexer group 7: PPCA (PPCA trigger multiplexer)		
0:63	TR_PPCAIN[0:63]	PPCA trigger inputs

10.7 Trigger one-to-one connections

The Trigger 1-to-1 connections for PSC3M/P8, PSC3M/P7 MCU can be found in the tables below:

Input	Label	Description
0	PDMA0_TR_OUT[0]_ACK_TO_CAN_0	CAN DW acknowledge
1	PDMA0_TR_OUT[0]_ACK_TO_CAN_1	CAN DW acknowledge

10.8 Registers summary

Table 76 Trigger multiplexer registers

Symbol	Name	Description
PERI[i]_TR_CMD	Trigger command Register	Control enables the software activation of a specific input trigger or output trigger of the trigger multiplexer structure

(table continues...)

Table 76 (continued) Trigger multiplexer registers

PERI[i]_TR_GR[x]_TR_CTL[y]	Trigger Group Control Register	Controls group trigger actions and connects a peripheral input trigger to multiple peripheral output triggers It can also invert the signal and specify if the output signal should be treated as a level-sensitive or edge-sensitive trigger. Specifies if the output trigger is blocked in debug mode. When set high, tr_dbg_freeze (bit 12) will block the output trigger generation. In the register format, 'i' is the peri group, 'x' is the trigger group, and 'y' is the output trigger line number from the multiplexer.
PERI[i]_TR_1TO1_GR[x]_TR_CTL[y]	Trigger 1-to-1 Group Control Register	Controls the 1-to-1 trigger actions and connects a peripheral input trigger to a specific peripheral output trigger. It also includes trigger manipulation that can invert the signal and specify if the output signal should be treated as a level-sensitive or edge-sensitive trigger. Specifies if the output trigger is blocked in debug mode. When set high, tr_dbg_freeze (bit 12) will block the output trigger generation. In the register format, 'i' is the peri group, 'x' is the trigger group, and 'y' is the output trigger line number from the multiplexer.

11 Debug and trace system

PSC3M/P8, PSC3M/P7 program and debug interface provides a communication gateway for an external device to perform programming or debugging. The external device can be an Infineon-supplied programmer and debugger, or a third-party device that supports programming and debugging. The serial wire debug (SWD) or the Joint Test Action Group (JTAG) interface can be used as the communication protocol between the external device and the PSC3M/P8, PSC3M/P7.

11.1 Features

The program and debug interface has the following features:

- Supports programming and debug through the JTAG or SWD interface
- CM33 supports 8-bit Instrumentation Trace Macrocell (ITM) and 4-bit Embedded Trace Macrocell (ETM) tracing
- Supports Cross Triggering Interface (CTI) and Cross Triggering Matrix (CTM)
- CM33 supports eight hardware breakpoints and four watch-points
- Provides read and write access to all memory and registers in the system while debugging, including the CM33 register banks when the core is running or halted

***Note**: PPCA CPU0/CPU1 cores have no ITM or ETM trace option.*

11.2 Architecture

Figure 33 shows the block diagram of the program and debug interface in the PSC3M/P8, PSC3M/P7 MCU. 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 PSC3M/P8, PSC3M/P7 MCU “target” using either the Serial Wire Debug (SWD) or Joint Test Action Group (JTAG) interfaces. The debug physical port pins communicate with the DAP through the high-speed I/O matrix (HSIOM). See the I/O subsystem chapter for details on the HSIOM.

The debug infrastructure is organized in the following groups:

- DAP (provides pin interfaces through which the debug host can connect to the target)
- Cortex®-M33 core debug components
- Other debug infrastructure (includes CTM, the TPIU, and the system ROM table)

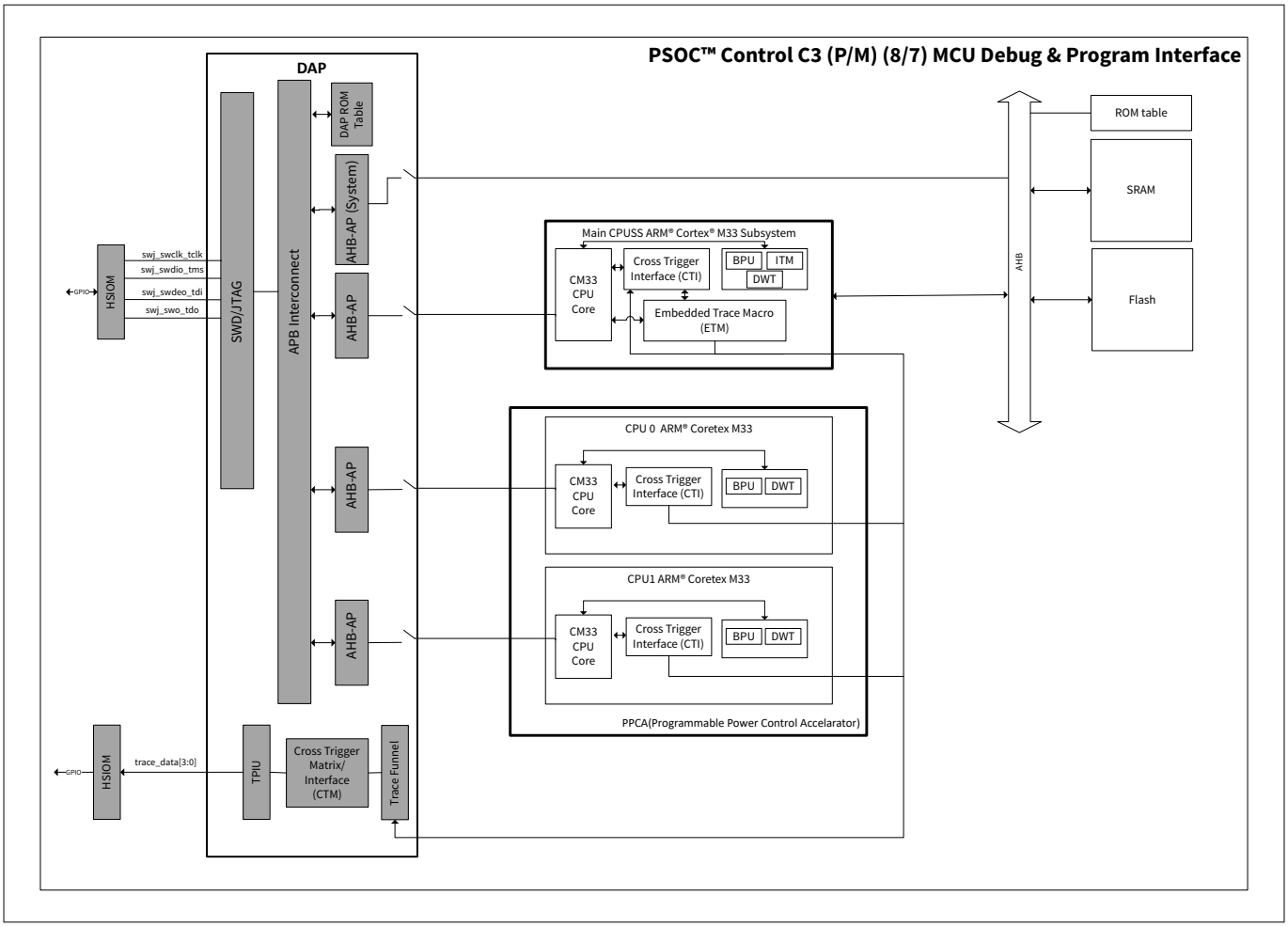


Figure 33 Program and debug architecture

The DAP communicates with the core using the Arm®-specified advanced high-performance bus (AHB) interface.

The external host can effectively take control of the entire device through the DAP to perform both programming and debugging operations, the DAP can be disabled in production devices for security, for security feature implementation, see Getting started with PSOC™ Control C3 security for detail.

The following are the various debug and trace components:

- Debug components
  - JTAG and SWD for debug control and access
- Trace source components
- Trace sink components
- Trace port interface unit (TPIU) to drive the trace information out of the chip to an external trace port analyzer
- Cross-triggering components
  - Cross-trigger interface (CTI)
  - Cross-trigger matrix (CTM)

***Note**: Trace components and cross-triggering components are only usable to main CM33, which are not applicable for PPCA CPU cores.*

11.2.1 Debug access port (DAP)

The DAP consists of a combined SWD/JTAG interface (SWJ). The SWJ includes an SWD listener that decides whether the JTAG interface (default after reset) or SWD interface is active. The JTAG and SWD interfaces are mutually exclusive because they share pins.

The debug port (DP) communicates with debug components using the Advanced Peripheral Bus Interconnect (APBIC) infrastructure, which then connects to one of the two AHB Access ports:

- The Cortex®-M33 AP connects directly to the AHB debug port and gives access to the Cortex®-M33 internal debug components. This allows access to the rest of the system through the Cortex®-M33 AHB master interface. The AP provides the debug host the same view as an application running on Cortex®-M33. This includes access to the MMIO of other debug components in the Cortex® -M33 subsystem. These debug components can be accessed by the Cortex®-M33 CPU, but cannot be reached through the other APs or by the Cortex®-M33 core.
- The System-AP, gives access to the rest of the system, such as user SRAM and Flash, through the AHB mux. This allows access to the System ROM table, which is not intended to be reached any other way. The system ROM table provides the MCU ID.

Both the debug and access port registers are Arm®IPs, which can be found in the Arm® Debug Interface Architecture Specification ADIv6.0.

***Note**: The DAP runs asynchronously with the Cortex® M33.*

11.2.1.1 DAP power domain

The entire debugger uses the same power domain as the Cortex®-M33 subsystem. Almost all debug components are part of the Active power domain. The only exception is the SWD as it uses two pins, SWCLKTCK and SWDIOTMS, which are operational in Deep Sleep. This allows for the SWD debug host to connect during Deep Sleep, while the application is running, or while powered down.

After the debugger is connected to the chip, it must bring the chip to the Active state before any operation. For this, the SWD/JTAG-DP has a register (DP_CTL_STAT) with two power request bits: CDBGPWRUPREQ and CSYSPWRUPREQ. These bits request for debug power and system power, respectively. These bits must remain set for the duration of the debug session.

The JTAG pins are only operational in Active mode, which means that the JTAG debug and JTAG boundary scan are not available when the system is in Deep Sleep. JTAG functionality is available only after a chip power-on reset.

Move the chip to the Active state before any operation. When the debugger attaches, it switches the CM33 into the active power domain. All other resources must be turned to an Active state by authenticated or debugging software.

11.2.1.2 DAP security

The DAP pins are controlled by the policy file, for detail, please see Getting started with PSOC™ Control C3 security. A device from the factory with a standard policy file sets up the DAP pins so that a SWD or JTAG interface can be connected without configuration. When using a secure policy file, configure the DAP pins and enable them in the secure policy file. For more information on the secure policy file, see the secure boot section.

Cortex®-M33 has its own CPUSS_AP_CTL register where the debug features, such as invasive/non-invasive debug and secure privileged invasive/non-invasive debug, can be configured. These debug features are configured in the policy file, which is used by the secure enclave to set the AP_CTL registers in the protected region of Flash and SFLASH. The SYS_AP is controlled by the Cortex® M33 AP_CTL register and is enabled by setting AP_CTL.SYS_DISABLE to ‘00’ in the policy file.

All AP_CTL policy field names that end with _DISABLE are two bits, with '00' being enable, '01' being disable, and '1x' being permanently disabled. For any secure/non-secure debug features to be enabled, AP_CTL.CM33_0_DISABLE must be set to '00'.

The following table shows the register configuration in the policy file for Cortex® -M33 for non-secure debugging.

Table 77 Non-secure debug

Debug mode	AP_CTL.CM33_0_DBG_DISABLE	AP_CTL.CM33_0_NID_DISABLE
Invasive	00	01
Non-Invasive	01	00

Table 78 Cortex® -M33 secure debug

Debug mode	AP_CTL.CM33_0_SPID_DISABLE[1)]	AP_CTL.CM33_0_SPNID_DISABLE	AP_CTL.CM33_0_SECURE_DISABLE
Secure Invasive	00	01	00
Secure Non-Invasive	01	00	00

1) For secure debug features, set the following in the policy file for Cortex®-M33

***Note**: To enable secure invasive debug, set CM33_0_DBG_DISABLE to '00'. To enable secure non-invasive debug, set CM33_0_NID_DISABLE to '00'.*

11.2.2 Trace

The Cortex® -M33 include Instrumentation Trace Macrocell (ITM) and Embedded Trace Macrocell (ETM). These components capture the program execution flow from the CPUs and generate trace outputs on the Advanced Trace Bus (ATB). These trace outputs are passed to the core-sight trace funnel (CSTF), which combines multiple trace streams on to a single bus.

Built into the debugger is a 4-bit Trace Port Interface Unit (TPIU). This is used to drive external pins through the IOSS interface, so that the trace can be captured by an external trace port analyzer (TPA). For more information, refer to the Arm® Debug Interface Architecture Specification ADIv6.0.

***Note**: Trace is not available for the PPCA CPU0 and CPU1 cores.*

11.2.3 ROM tables

The CoreSight architecture allows for discoverable debug topology. This means that a debug host can find all debug components available on a chip without knowing the identity of the chip. To discover the chip identity and the debug topology, use the CoreSight ROM tables.

The ROM tables are organized in a tree hierarchy. Each AP has a register that contains a 32-bit address pointing to the base of the root ROM table for that AP. The root ROM table in the debugger is the 'Dap ROM Table', which points to the different APs. The Coresight DP component has a register called BASEPTR that points to the DAP ROM table. For PSC3M/P8, PSC3M/P7 MCU there are three such root ROM tables (DAP, M33, and System).

Each ROM table contains 32-bit entries that have an address pointer that either points to the base of the next level ROM table or the base of 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 number. For all ROM tables in PSC3M/P8, PSC3M/P7 MCU, these IDs are the same. Each ROM table and CoreSight compliant component also contains component identification registers.

11.2.4 Cross trigger functionality

The debug infrastructure implements cross trigger functionality that provides ability to trigger (send or receive) events between different debug components and CPUs.

The debug infrastructure provides a Cross Trigger Interface (CTI) that primarily connects events to the TPIU, ETF and system triggers.

Each CPU has an internal CTI that connects events between CPU core and debug components within the CPU (like ETM, DWT etc.). All these three CTIs are connected via Cross Trigger Matrix (CTM) which provides ability to send/receive events between these three CTIs.

Figure 34 shows overall cross trigger functionality.

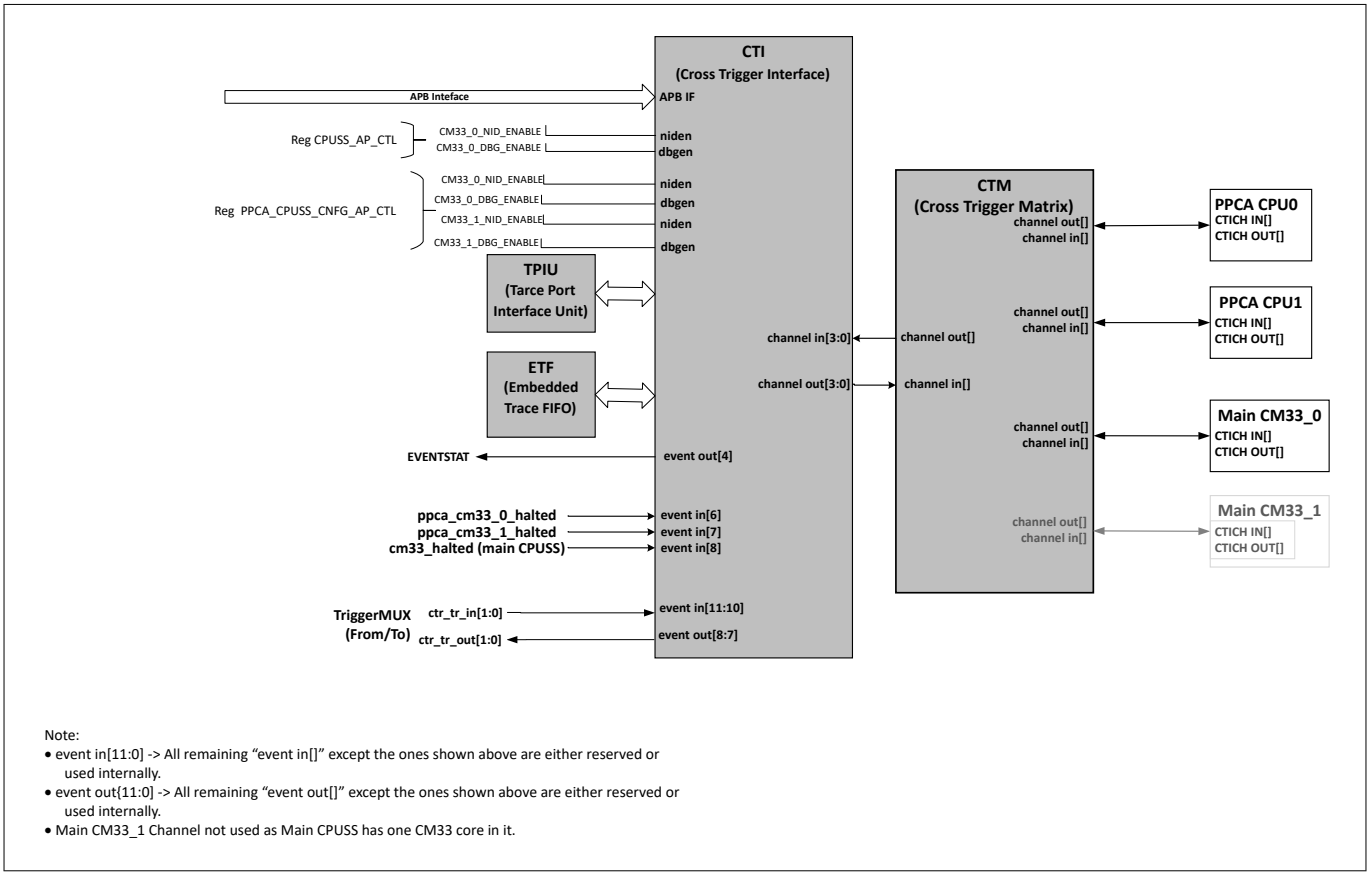


Figure 34 Cross trigger functionality

The EVENT_IN_LEVEL, SW_HANDSHAKE, “tinidensel[]” and “todbgensel[]” constants corresponding to “dp_eventstatus”, TPIU and ETF triggers are set according to CoreSight SoC-600 configuration and integration manual.

The trigger inputs “cm33_main_halted” and PPCA_cm33_0/1_halted” are treated as level trigger inputs. Hence EVENT_IN_LEVEL for these triggers is set as ‘1’. The level halted signals of this CTI are primarily used to implement debug freeze functionality in peripherals (like TCPWMs and Watchdog timers etc).

- The “cti_tr_in[]” and “cti_tr_out[]” triggers treatment requires system level verification. SW_HANDSHAKE[] constant corresponding to “cti_tr_out[]” are set to ‘0’ and this also need to be verified at system level and may need change based on verification feedback.

Refer Arm® Cortex®-M33 Processor Technical Reference Manual for the details about CM33 CTI connections to the respective debug components within CM33.

11.2.4.1 Freeze peripherals in debug mode

The level halted signals of the Cross-Trigger Interface (CTI) are primarily used to implement debug freeze functionality in peripherals (like TCPWMs etc).

For example, to configure the system for freezing TCPWM in debug mode, follow the steps given below:

- Initialize TCPWM IP or PPCA TCPWM IP for debug freeze operation, setup functions for which debug-freeze can be experimented
- Make sure that the debug session is active by checking the DEBUG_SESSION bit of the PWR_CTL register before enabling the freeze and suspend function

  ***Note**: The freeze and suspend function can only be enabled when a debug session is active.*

- Setup the debug-freeze on specific core, only one core can be enabled at a time, to implement the stop function using the CM33 HALT action, perform the following steps:

  1. Set up EVENT for the specific core HALT action
  - **a.** PPCA core0: Set up EVENT(6) for the CM33_0 HALT action
  - **b.** PPCA core1: Set up EVENT(7) for the CM33_1 HALT action
  - **c.** main core: Set up EVENT(8) for the CM33_main HALT action

  2. Enable the cross triggering mapping logic functionality using the CTI control register

  3. Enable a cross trigger event to the corresponding channel using the CTI trigger to channel enable register

  4. Enable PPCA by configuring PPCA_CNFG_CTRL register

  5. Enable the debug functionality, configure PPCA_CNFG_CNFG_CNFGEXTDBGCONN register
  - **a.** main core: assign to main CPU
  - **b.** PPCA core0/core1: assign to PPCA CPU

  ***Note**: If freeze PPCA core1, it should set up Device configurator tool to enable General → CPU1, PPCA configuration option enable requested assets to CPU1 instead of CPU0 (by default), save the settings.*

- Now the counter value will be frozen when the tr_dbg_freeze signal is activated, usually the tr_dbg_freeze signal is intended to be activated from the cpuss.cti_tr_out[x] signal, both CTI_TR_OUT0 or CTI_TR_OUT1 can be selected, use the trigger mux function to connect the CTITRIGOUT output to the tr_debug_freeze input of the TCPWM, as described in the TriggersMux chapter. This configuration enables the routing of cti_tr_out[0:1] to TCPWM for trigger-to-freeze operation

The following code snippet can be referred:

```
#define TRC_CTICONTROL (*((volatile uint32_t*)(0xE0080000)))
#define TRC_CTIINEN(n) (*((volatile uint32_t*)(0xE0080020 + 4*n)))
#define TRC_CTIOUTEN(n) (*((volatile uint32_t*)(0xE00800A0 + 4*n)))


/*CAUTION: Only one to be enabled at a time */
#define TEST_DBG_FREEZE_MAIN_CORE  1
#define TEST_DBG_FREEZE_PPCA_CORE0 0
#define TEST_DBG_FREEZE_PPCA_CORE1 0

    setup_debug_freeze_pwm();
    setup_debug_freeze_ppca_pwm();

    if(1U == _FLD2VAL(SRSS_PWR_CTL_DEBUG_SESSION, SRSS_PWR_CTL))
    {

    /*Setup debug-freeze for testing of the cores*/
#if TEST_DBG_FREEZE_MAIN_CORE == 1
      setup_debug_freeze_main_core();
#elif TEST_DBG_FREEZE_PPCA_CORE0 == 1
      setup_debug_freeze_ppca_core0();
#elif TEST_DBG_FREEZE_PPCA_CORE1 == 1

    /* To experiment this feature we should setup Device Configurator tool
     * Enable General -> CPU1, PPCA Configuration option enable requested assets to CPU1
instead of CPU0 (by default), save the settings
     * And then experiment the step */

      setup_debug_freeze_ppca_core1();
#endif
    }

void setup_debug_freeze_main_core(void)
{
    TRC_CTICONTROL = 0x1;
    TRC_CTIINEN(8) = 0x1;   /* main_core*/
    TRC_CTIOUTEN(7) = 0x1;
    PPCA_CNFG->CTRL |= (1 << 31);
    PPCA_CNFG_CNFG->CNFGEXTDBGCONN = 0;

}

void setup_debug_freeze_ppca_core0(void)
{
    TRC_CTICONTROL = 0x1;
    TRC_CTIINEN(6) = 0x1;   /*ppca_core0*/
    TRC_CTIOUTEN(7) = 0x1;
    PPCA_CNFG->CTRL |= (1 << 31);
    PPCA_CNFG_CNFG->CNFGEXTDBGCONN = 0xFFFF;
}
void setup_debug_freeze_ppca_core1(void)
```

```
{
    TRC_CTICONTROL = 0x1;
    TRC_CTIINEN(7) = 0x1;   /*ppca_core1*/
    TRC_CTIOUTEN(7) = 0x1;
    PPCA_CNFG->CTRL |= (1 << 31);
    PPCA_CNFG_CNFG->CNFGEXTDBGCONN = 0xFFFF;
}

void setup_debug_freeze_pwm(void)
{
    /*below code tests TCPWM debug freeze feature on the Main Core , TCPWM block outside
the PPCA */
    if (CY_TCPWM_SUCCESS != Cy_TCPWM_PWM_Init((TCPWM_Type*)MY_PWM_HW, MY_PWM_NUM,
&MY_PWM_config))
    {
        CY_ASSERT(0);
    }

    /* Enable the initialized PWM_W */
    Cy_TCPWM_PWM_Enable((TCPWM_Type*)MY_PWM_HW, MY_PWM_NUM);
    TCPWM0_GRP0_CNT0->CTRL |= (1 << 30) | (1 << 29); /*enable the debug functionality for
PROT_TCPWM0_GRP0_CNT0_CNT*/

    /*Connect the debug-freeze signal*/
    Cy_TrigMux_Connect(TRIG_IN_MUX_5_CTI_TR_OUT0, TRIG_OUT_MUX_5_TCPWM_DEBUG_FREEZE_TR_IN,
0, TRIGGER_TYPE_LEVEL);

    /*Start the trigger timer*/
    Cy_TCPWM_TriggerStart_Single((TCPWM_Type*)MY_PWM_HW, MY_PWM_NUM);
}

void setup_debug_freeze_ppca_pwm(void)
{
    Cy_TCPWM_PWM_Init(PPCA_PWM_HW, PPCA_PWM_NUM, &PPCA_PWM_config);
    Cy_TCPWM_PWM_Enable(PPCA_PWM_HW, PPCA_PWM_NUM);
    PPCA_TCPWM0_GRP0_CNT0->CTRL |= (1 << 30) | (1 << 29);
    Cy_TrigMux_Connect(TRIG_IN_MUX_5_CTI_TR_OUT0, TRIG_OUT_MUX_5_PPCA_DEBUG_FREEZE_TR_IN,
0, TRIGGER_TYPE_LEVEL);
    Cy_TCPWM_TriggerStart_Single(PPCA_PWM_HW, PPCA_PWM_NUM);
}
```

11.3 Serial wire debug (SWD) interface

PSC3M/P8, PSC3M/P7 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 an 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) consists of three phases:

- Host packet request phase – The host issues a request to the PSC3M/P8, PSC3M/P7 MCU target
- Target acknowledge response phase – The PSC3M/P8, PSC3M/P7 MCU target sends an acknowledgement to the host
- Data transfer phase – The host or target writes data to the bus, depending on the direction of the transfer

For more information on the SWD implementation, refer to Arm® Debug Interface Architecture Specification ADIv6.0.

11.4 JTAG interface

In response to higher pin densities on ICs, the Joint Test Action Group (JTAG) proposed a method to test circuit boards by controlling the pins on the ICs (and reading their values) via a separate test interface. The solution,later formalized as IEEE Standard 1149.1-2001, is based on the concept of a serial shift register routed across all of the pins of the IC – hence the name “boundary scan.” The circuitry at each pin is supplemented with a multipurpose element called a boundary scan cell. In the PSC3M/P8, PSC3M/P7 MCU, most GPIO port pins have a boundary scan cell associated with them (see the GPIO block diagrams in the I/O subsystem section). 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). A clock signal (TCK) that clocks the other signals is also included and a debug TAP reset (TRST) is also present which is an optional input signal. TDI, TMS, and TCK are all inputs to the device and TDO is the output from the device. For more information on the JTAG interface, refer to Arm® Debug Interface Architecture Specification ADIv6.0.

11.5 Programming the MCU

PSC3M/P8, PSC3M/P7 MCU is programmed using the following sequence:

1. Acquire the SWD port in the PSC3M/P8, PSC3M/P7 MCU
2. Enter the programming mode
3. Execute the device programming routines such as Silicon ID Check, Flash Programming, Flash Verification, and Check-sum Verification

***Note**: For detail, see PSOC™ Control C3 MCU Programming specification.*

11.5.1 SWD port acquisition

When the device is reset by asserting the external reset (XRES) pin, the DP defaults to the JTAG operation. To switch to the SWD protocol, reset the device using XRES to put it into its reset state. The host must send the following SWD connect sequence:

1. Set SWDIO HIGH for at least five SWCLK cycles. This places the JTAG TAP state machine into the Test Logic Reset state and selects the IDCODE instruction
2. Change the device from the JTAG to dormant state by writing the 31-bit sequence (0x2EEEEEE6, MSB first) on SWDIO. The device is now in a dormant state
3. Set SWDIO HIGH for at least eight SWCLK cycles. This ensures that the device is not in the middle of a Selection Alert sequence
4. On SWDIO, send the 128-bit Selection Alert sequence (0x49CF9046 A9B4A161 97F5BBC7 45703D98, MSB first)
5. Set SWDIO LOW for four SWCLK cycles
6. On SWDIO, send the required activation code sequence (0b0101_1000, MSB first)
7. Place the target into a known state by setting SWDIO HIGH for at least 50 SWCLK cycles. This makes sure the SWD interface is in the line reset state. The device is now in SWD configuration. The PSC3M/P8, PSC3M/P7 MCU uses Dormant operation to move between JTAG to Dormant, and then Dormant to SWD

For more information about this operation, refer to the Arm® Debug Interface Architecture Specification ADIv6.0.

11.5.2 SWD programming mode entry

After the SWD port is acquired, the host must enter the device programming mode within a specific time window. This is done by setting the TEST_MODE bit (bit 31) in the test mode control register (MODE register).

Make sure to configure the debug port before entering the device programming mode.

11.5.3 SWD programming routine executions

When the device is in programming mode, the external programmer can start sending the SWD packet sequence for performing programming operations such as flash erase, flash program, checksum verification, and so on.

11.6 PSC3M/P8, PSC3M/P7 MCU SWD debug interface

The Cortex® -M33 DAP debugging features are classified into two types: invasive debugging and noninvasive debugging. Invasive debugging includes program halting and stepping, breakpoints, and data watch-points. Noninvasive debugging includes instruction address profiling and device memory access, which includes the flash memory, SRAM, and other peripheral registers.

The DAP has three major debug subsystems:

- Debug Control and Configuration registers
- Breakpoint Unit (BPU) – Provides breakpoint support
- Debug Watchpoint (DWT) – Provides watchpoint support

See the Armv8-M architecture reference manual for complete details on the debug architecture.

***Note**: The BPU is part of the Flashpoint and Breakpoint (FPB) unit in Arm® Debug Interface Architecture Specification ADIv6.0.*

11.6.1 Debug control and configuration registers

The debug control and configuration registers are used to execute firmware debugging. The registers and their key functions are as follows (see the Arm®v8-M architecture reference manual for the complete bit-level definitions of these registers):

- Debug Halting Control and Status Register (DHCSR): This register contains the control bits to enable debug, halt the CPU, and perform a single-step operation. It also includes status bits for the debug state of the processor.
- Debug Fault Status Register (DFSR): This register describes the reason a debug event occurs and includes debug events, which are caused by a CPU halt, breakpoint event, or watchpoint event.
- Debug Core Register Data Register (DCRDR): This register is used to store the data to write to or read.
- Debug Exception and Monitor Control Register (DEMCR): This register contains the enable bits for global debug watchpoint (DWT) block enable, reset vector catch, and hard fault exception catch.

11.6.2 Break point units (BPU)

The BPU provides breakpoint functionality on instruction fetches. The Cortex® -M33 in the PSC3M/P8, PSC3M/P7 MCU supports four hardware breakpoints. Along with the hardware breakpoints, any number of software breakpoints can be created by using the BKPT instruction. The BPU has the following two types of registers:

- The breakpoint control register (FPB_CTRL) is used to enable the BPU and store the number of hardware breakpoints supported by the debug system.
- Each hardware breakpoint has a Breakpoint Compare Register (FPB_COMPx). It contains the enable bit for the breakpoint and the compare address value. The typical use case is that when an instruction fetch address matches the compare address of a breakpoint, a breakpoint event is generated and the processor is halted.

11.6.3 Data watchpoints (DWT)

The DWT provides watchpoint support on a data address access or a program counter (PC) instruction address.

The DWT supports two watchpoints for the Cortex® -M33. It also provides an external program counter sampling using a PC sample register, which can be used for noninvasive coarse profiling of the program counter.

- The watchpoint compare (DWT_COMPx) registers store the compare values that are used by the watchpoint comparator for the generation of watchpoint events. Each watchpoint has an associated DWT_COMPx register.
- The watchpoint function (DWT_FUNCTIONx) registers store the conditions that trigger the watchpoint events. They may be program counter watchpoint event or data address read/write access watchpoint events. A status bit is also set when the associated watchpoint event occurs.
- The watchpoint comparator PC sample register (DWT_PCSR) stores the current value of the program counter. This register is used for coarse, non-invasive profiling of the program counter register.

11.6.4 Debugging the PSC3M/P8, PSC3M/P7 device

The host debugs the target by accessing the debug control and configuration registers, BPU registers, and DWT registers. All registers are accessed through the SWD interface; in the Cortex® -M33, DAP converts the SWD packets to appropriate register access through the DAP-AHB interface.

The first step in debugging the target is to acquire the SWD port. The acquire sequence can be found in the SWD port acquisition section. The SWD port is acquired when the correct Cortex® -M33 DAP SWDID is read from the target device. For the debug transactions to occur on the SWD interface, do not use the corresponding pins for any other purpose. See the I/O subsystem section to understand how to configure the SWD port pins, allowing them to be used only for the SWD interface or for other functions such as LCD and GPIO. If debugging is required, do not use the SWD port pins for other purposes.

If only programming support is needed, the SWD pins can be used for other purposes. When the SWD port is acquired, the external debugger sets the C_DEBUGEN bit in the DHCSR register to enable debugging. Then, the different debugging operations such as stepping, halting, breakpoint configuration, and watchpoint configuration are carried out by writing to the appropriate registers in the debug system.

11.7 Debug handler

Debug handler logic generates the PERIPH_DEBUG_FREEZE signals, to be sent to the embedded PPCA peripherals, TCPWM, ATOPSS, and CLB.

The PERIPH_DEBUG_FREEZE signals are generated from the debug signals coming from the PPCA boundary (PPCA_TR_DEBUG_FREEZE) and from the CPUs (CPU0_HALTED and CPU1_HALTED signals)

The freeze happens only for the peripherals that are specifically allocated to the halted CPUs.

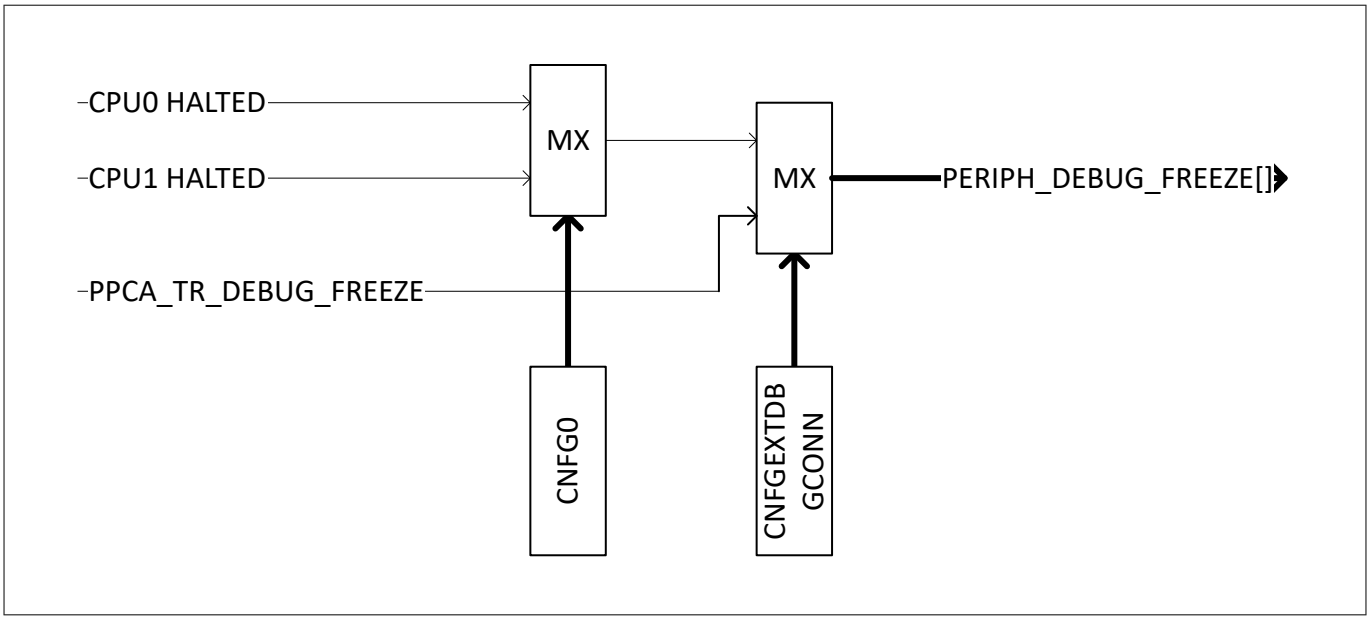


Figure 35 Debug handler

11.8 Registers summary

Table 79 Debug features registers

Name	Description
CPUSS_AP_CTL	Configure debug features

12 Cryptographic accelerator

The cryptographic accelerator block 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 (TRNG) functionality. The PSC3M/P8, PSC3M/P7 MCU cryptographic accelerator block name is CryptoLite.

12.1 Features

The following are the features of the cryptographic accelerator:

- Advanced Encryption Standard (AES)-128[1)] to accelerate block cipher functionality and only supports the encryption functionality of a single 128-bit block with a fixed 128-bit key
- Secure Hash Algorithm (SHA)-256/384/512 to accelerate hash functionality
- True random number generator (TRNG)
- Vector unit (VU) to support asymmetric key cryptography, such as RSA and Elliptic-Curve Cryptography (ECC)

12.2 Architecture

The following figure gives an overview of the CryptoLite block.

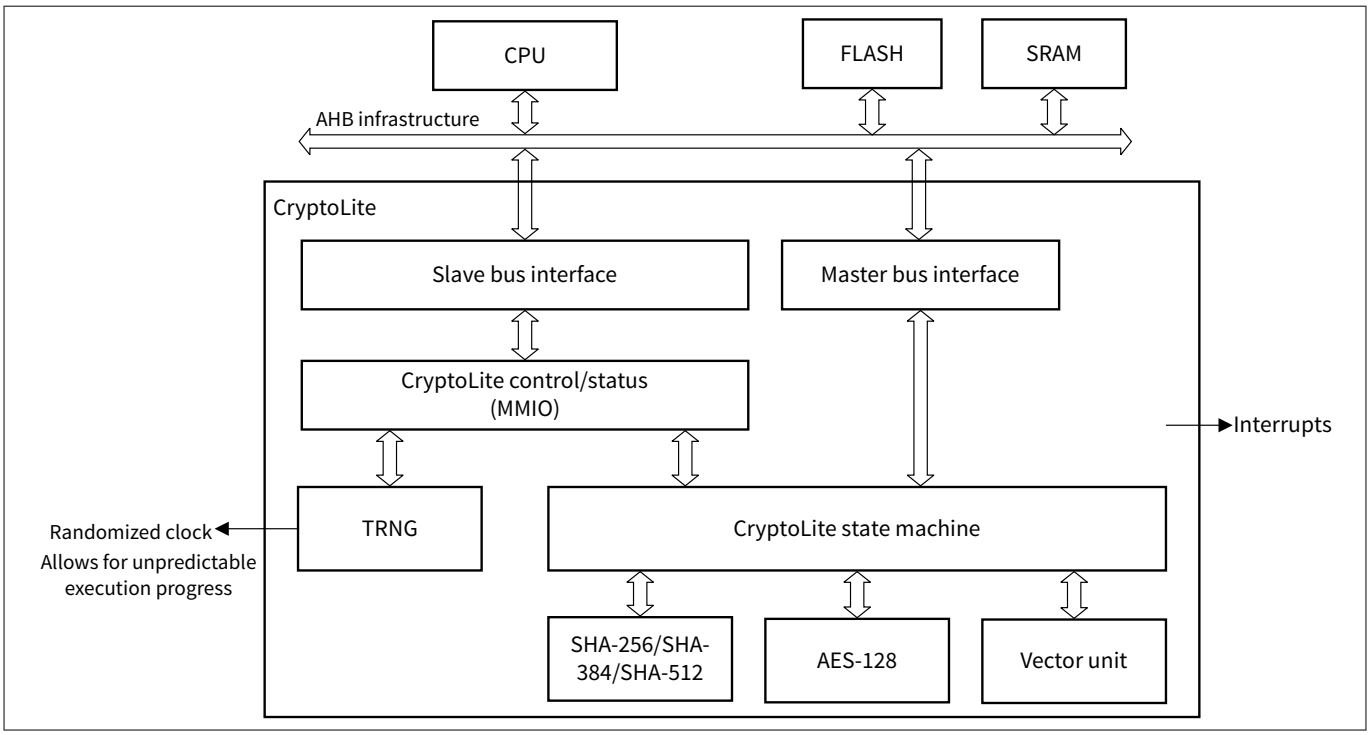


Figure 36 CryptoLite block diagram

The CryptoLite block has the following interfaces:

- AHB slave interface: External bus masters use this interface to access the block MMIO registers
- AHB master interface: The block uses this interface to access hardware function data structures and operands in memories (FLASH and SRAM)

1 The device also includes a dedicated AES block supporting both 128-bit and 256-bit AES, with additional security features. See Advanced Encryption Standard (AES) chapter for the details.

- Two interrupts: One interrupt for regular behavior, for example, TRNG is initialized. One interrupt for irregular/erroneous behavior, for example, when the AHB master interface encounters a bus error
- Randomized clock: This is the TRNG digitized analog signal (DAS) bit-stream sequence
- A clock and reset signal interface connects to the System Resources subsystem (SRSS)

The CryptoLite block provides hardware acceleration for a minimal set of cryptographic functions. The functionality is implemented by four components:

- AES-128 component: To accelerate block cipher functionality. The component supports:
  - Forward encryption of a single 128-bit block with a 128-bit key
- SHA component: To accelerate hash functionality. The component supports:
  - For SHA-256: Message schedule calculation and processing for a 512-bit message chunk
  - For SHA-384/SHA-512: Message schedule calculation and processing for a 1024-bit message chunk
- Vector unit (VU) component: To accelerate asymmetric key cryptography (for example, RSA and ECC). The component supports:
  - Large integer multiplication, addition, and so on
- TRNG component: Based on a set of ring oscillators, the TRNG includes a hardware health monitor. This component generates true random numbers using up to six ring oscillators

The component provides speeds up, for example, secure boot Digital Signature Authentication (DSA or ECDSA) through its SHA and VU hardware acceleration. The digital signature provides message authentication, integrity, and non-repudiation.

12.2.1 Randomized clock

The CryptoLite exposes the TRNG digitized analog signal (DAS) bit-stream sequence at its interface signal "*tr_trng_bitstream*". The sequence may be used to randomize the timing of functionality. This signal can be routed to GPIOs and DMA using the trigger multiplexer. See the Trigger multiplexer (Trigger MUX) chapter for more details.

The default value of the "*tr_trng_bitstream*" signal is set to constant '0' when oscillators are not enabled.

The TRNG functionality may be stopped by the monitor on repetition count (RC) or adaptive proportion (AP) error detection. If the TRNG functionality is stopped, both the DAS bit-stream and the randomized clock signal are set to constant '0'. If the "stop functionality" is not enabled, the DAS bit-stream and the randomized clock will continue to produce random bit sequences. See the True random number generator (TRNG) section for more details.

12.2.2 Hardware/Software interface

The SHA, AES-128, and VU components have a similar hardware/software interface. The following components have a dedicated MMIO register that provides a pointer to a data structure in memory:

- The SHA component uses the CRYPTOLITE_SHA_DESCR pointer register
- The AES-128 component uses the CRYPTOLITE_AES_DESCR pointer register
- The VU component uses the CRYPTOLITE_VU_DESCR pointer register

A write to one of these MMIO registers starts a hardware function. The block will read the data structure from memory, and the data structure specifies the desired hardware function and its operands. The data structures are hardware-function-specific. The block performs the specified hardware function. From start to finish, the hardware function is considered busy.

- While a hardware function (SHA, AES-128, VU) is busy, the CryptoLite block cannot accept a new MMIO register write transfer; a new MMIO register write transfer will be made pending, which blocks any other (write or read) MMIO register transfers
- While a hardware function is busy and the CryptoLite block is not blocked, the block can accept a new MMIO register read transfer. For example, a read transfer may read the CRYPTOLITE_STATUS register, which indicates if the CryptoLite block is busy

The above behavior has the following implications:

- Only one hardware function (SHA, AES-128, VU) can be busy/active at the same time; that is, they are mutually exclusive. This is desired as the implementation shares hardware resources
- The data structure pointer MMIO registers cannot be overwritten by software while a hardware function is busy
- Software may prepare a next data structure in memory and write its pointer to a data structure pointer MMIO register while a hardware function is not busy. The write transfer of the next data structure's pointer will block the cryptographic block. This prevents the data structure pointer MMIO register from being overwritten

***Note**: Software should ensure that the current (busy) and next data structures in memory do not overlap*

***Note**: The TRNG and a hardware function (SHA, AES-128, VU) can be active at the same time. However, a TRNG MMIO register write transfer will be made pending when a hardware function is busy. Therefore, it is advised that the TRNG functionality be setup/started before the hardware functions are used. A TRNG MMIO register read transfer (CRYPTOLITE_INTR_TRNG and CRYPTOLITE_TRNG_RESULT) is possible when a hardware function is busy.*

As mentioned, a data structure in memory specifies the hardware function and its operands. The operands are typically pointers to 8-bit bytes or 32-bit word sequences in memory. These sequences in memory may represent a secret key (AES-128), a message chunk (SHA), or a long unsigned integer number (VU). The following figure shows a high-level block diagram of the descriptor, data structure, and operands.

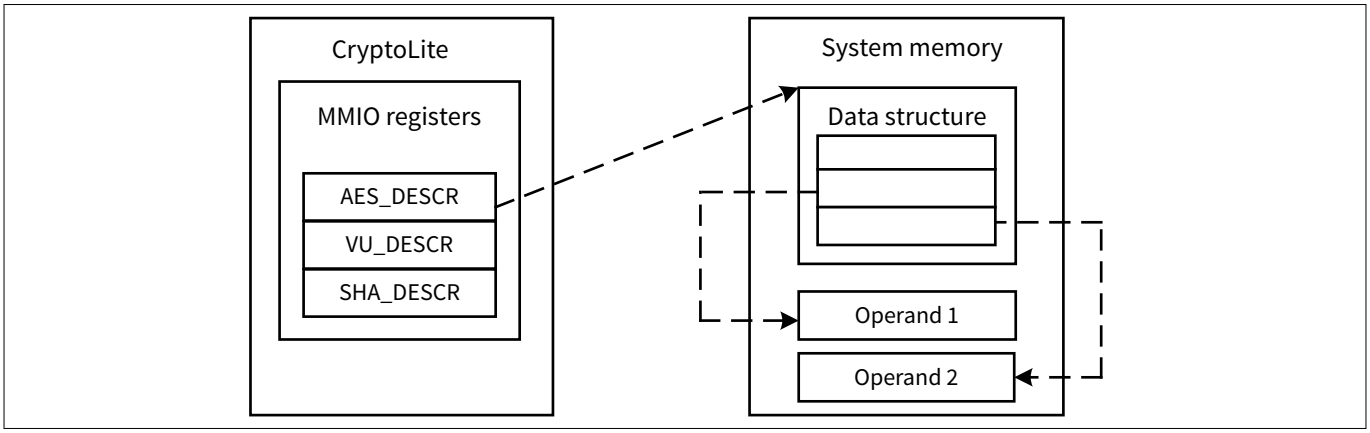


Figure 37 CryptoLite hardware/software interface

Most cryptographic functions are implemented using a combination of CPU software and hardware acceleration.

A hardware function (SHA, AES-128, VU) uses an AHB master interface to access memory. A bus error is reported (CRYPTOLITE_INTR_ERROR.BUS_ERROR), and the hardware function is terminated under an AHB bus error.

12.3 Block functions

12.3.1 AES-128

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

A block cipher mode of operation uses the AES-128 block cipher function in a specific configuration to provide message confidentiality and authenticity. The CryptoLite component provides AES-128 block cipher function hardware acceleration. A specific block cipher mode of operation is implemented using software and hardware acceleration. Typically, a message is broken up into multiple 128-bit blocks. The common modes of operation are:

- Electronic Code Book (ECB)
- Cipher Block Chaining (CBC)
- Cipher Feedback (CFB)
- Output Feedback (OFB)
- Counter (CTR)

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

The CryptoLite AES-128 block cipher function hardware acceleration consists of a single hardware function:

- **Encrypt**: Encrypt a 128-bit block P (plain-text) with a 128-bit SK (secret key) into a 128-bit block C (cipher-text)

A write to the CryptoLite block's CRYPTOLITE_AES_DESCR MMIO register starts a 128-bit AES encryption in the ECB mode. The CRYPTOLITE_AES_DESCR register expects a pointer to a three 32-bit word structure in memory. The pointer must be 32-bit word-aligned. The CryptoLite component retrieves the three control words that specify the hardware function.

- **Word 0**: Pointer to a 16 * 8-bit secret key SK. The secret key is a hardware function input
- **Word 1**: Pointer to a 16 * 8-bit block P. The block is a hardware function input
- **Word 2**: Pointer to a 16 * 8-bit block C. The block is a hardware function output

The following figure shows AES-128 data flow.

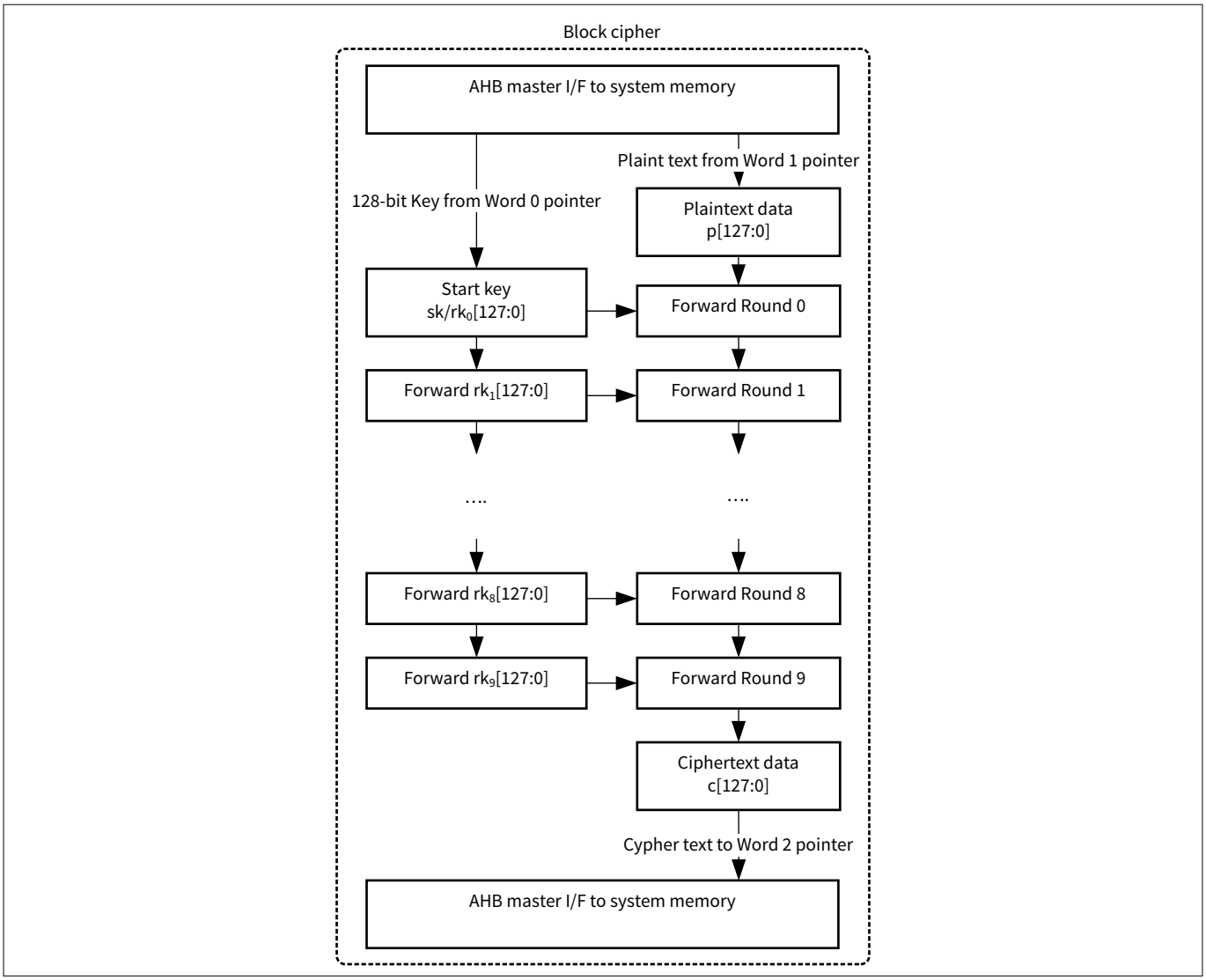


Figure 38 AES-128 data flow

***Note**: The AES-128 block cipher function operates on 8-bit bytes. As a result, no special endianness considerations are applicable from a hardware point of view.*

12.3.2 SHA

The CryptoLite supports the SHA-256, SHA-384, and SHA-512 algorithms acceleration. Before performing hash function, set the CRYPTOLITE_CTL.SHA_SEL register to select which the SHA operation enabled.

12.3.2.1 SHA-256/384/512

The SHA-256, SHA-384 and SHA-512 hash functions are a member of the SHA-2 hash family. To calculate a message's hash value (also known as message digest), a message is broken up into multiple 512-bit (SHA-256) or 1024-bit (SHA-384/SHA-512) message chunks. The SHA-256, SHA-384 and SHA-512 hash function process a single message chunk.

The SHA-256 hash function has an internal 256-bit (eight 32-bit words) hash state H[j] (with j = 0, …, 7), as well as the SHA-512 and SHA-384 hash function have an internal 512-bit (eight 64-bit words) hash state H[j] (with j = 0, ..., 7). For the first message chunk, the start hash state is initialized with SHA-256/SHA-384/SHA-512

(whichever is implemented based on CRYPTOLITE_CTL.SHA_SEL register configuration) pre-defined values as specified in NIST.FIPS.180-4. For each successive message chunk, the start hash state is the previous message chunk's end hash state. The final message chunk's end hash state is the message's hash value.

The SHA-256 hash function performs 64 rounds of a compression function, as well as the SHA-384 and SHA-512 hash function perform 80 rounds of a compression function. Each compression function round updates the internal hash state. For SHA-256, each round has a dedicated 32-bit round constant K[i] and a dedicated 32-bit message schedule value W[i]; for SHA-384 and SHA-512, each round has a dedicated 64-bit round constant K[i] and a dedicated 64-bit message schedule value W[i].

The round constants have SHA pre-defined values as specified in NIST.FIPS.180-4; they are fixed.

The message schedule values are created from the 512-bit (SHA-256) or 1024-bit (SHA-384/SHA-512) message chunk; they are message-specific.

The CryptoLite SHA-256 hash function hardware acceleration consists of two hardware functions:

- Message schedule function
  - For SHA-256, creation of 64 * 32-bit message schedule values W[i] (with = 0, …, 63) from a 512-bit message chunk M
  - For SHA-384 and SHA-512, creation of 80 * 64-bit message schedule values W[i] (with = 0, ..., 79) from a 1024-bit message chunk M
- Process function
  - For SHA-256, processing of a 256-bit hash state by performing 64 rounds of a compression function. The process inputs are a 256-bit hash state and a message schedule W (as created by the message schedule function). The process output is an updated 256-bit hash state. Note that the round constants K are an implicit rather than explicit process input (they are hardwired in the process function)
  - For SHA-384 and SHA-512, processing of 384-bit and 512-bit hash state respectively is done by performing 80 rounds of a compression function. The process inputs are a 512-bit hash state and a message schedule W (as created by the message schedule function). The process output is an updated 512-bit and 384-bit hash state. Note that the round constants K are an implicit, rather than explicit process input (they are hardwired in the process function)

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

A write to the CRYPTOLITE_SHA_DESCR MMIO register starts a hardware function. The CRYPTOLITE_SHA_DESCR register expects a pointer to a three 32-bit word structure in memory.

***Note**: The pointer must be 32-bit word-aligned.*

The CryptoLite component retrieves the three control words that specify the hardware function. The first word (word 0) is generic; the other two words (words 1 and 2) are hardware function-specific.

- **Word 0** bit position 28 is used to specify the hardware function
  - If WORD0[28] is '0', a message schedule hardware function is performed
  - If WORD0[28] is '1', a process hardware function is performed

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

- For SHA-256
  - **Word 1**: Pointer to a 16 * 32-bit (512 bit) message chunk M. The message chunk is a hardware function input. The pointer must be 32-bit word-aligned
  - **Word 2**: Pointer to a 64 * 32-bit (2048 bit) message schedule array W. The message schedule is a hardware function output. The pointer must be 32-bit word-aligned
- For SHA-384 and SHA-512
  - **Word 1**: Pointer to a 16 * 64-bit (1024 bit) message chunk M. The message chunk is a hardware function input. The pointer must be 64-bit word-aligned
  - **Word 2**: Pointer to an 80 * 64-bit (5120 bit) message schedule array W. The message schedule is a hardware function output. The pointer must be 64-bit word-aligned

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

- For SHA-256
  - **Word 1**: Pointer to an 8 * 32-bit (256 bit) hash state H. The hash state is both a hardware function input and a hardware function output (the input is overwritten by the output). The pointer must be 32-bit word-aligned
  - **Word 2**: Pointer to a 64 * 32-bit (2048 bit) message schedule array W. The message schedule is a hardware function input. The pointer must be 32-bit word-aligned
- For SHA-384 and SHA-512
  - **Word 1**: Pointer to an 8 * 64-bit (512 bit) hash state H. The hash state is both a hardware function input and a hardware function output (the input is overwritten by the output). The pointer must be 64-bit word-aligned
  - **Word 2**: Pointer to a 80 * 64-bit (5120 bit) message schedule array W. The message schedule is a hardware function input. The pointer must be 64-bit word-aligned

***Note**:* *The SHA-256 hash function operates on 32-bit words and assumes a big-endian memory layout of these words in memory. The PSC3M/P8, PSC3M/P7 MCU uses a little-endian memory layout. This has the following implications:*

- *When a message chunk's 8-bit bytes are used to construct the message schedule's 32-bit words W[i], a little-to-big-endian conversion ("LE2BE") is processed by hardware*
- *When a final hash value's 32-bit words H[j] are used as a message digest (assuming the message digest is considered a byte stream), a big-to-little-endian conversion ("BE2LE") is required by software*

***Note**:* *The SHA-384 and SHA-512 hash function operate on 64-bit words and assumes a big-endian memory layout of these words in memory. The PSC3M/P8, PSC3M/P7 MCU uses a little-endian memory layout. This has the following implications:*

- *When a message chunk's 8-bit bytes are used to construct the message schedule's 64-bit words W[i], a little-to-big-endian conversion ("LE2BE") is processed by hardware*
- *When a final hash value's 64-bit words H[j] are used as a message digest (assuming the message digest is considered a byte stream), a big-to-little-endian conversion ("BE2LE") is required by software*
- *SHA-384 writes eight 64-bit final hash digest to the memory, firmware is required to obtain by truncating the final hash value to its left-most 384 bits*

12.3.3 Vector unit (VU)

The vector unit (VU) supports long integer operations that are useful for asymmetric key cryptography such as RSA or Elliptic-curve cryptography (ECC).

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

The VU supports the following long integer operations:

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

  ***Note**: The operation’s “greater than equal” operator uses the source bit size (not the destination bit size), and the operation’s “subtraction” operator uses the destination bit size*

A write to the CRYPTOLITE_VU_DESCR register starts a VU operation. The CRYPTOLITE_VU_DESCR register expects a pointer to a four 32-bit word structure in memory. The pointer must be 32-bit word-aligned. The CryptoLite component retrieves the four control words that specify the hardware function. The first word (word 0) is generic; the other three words (words 1, 2, and 3) are hardware function-specific.

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

Table 80 Vector unit opcodes

Opcode	Operation
0_H	Multiplication (MUL)
1_H	Addition (ADD)
2_H	Subtraction (SUB)
3_H	Exclusive or (XOR)
4_H	Binary multiplication (XMUL)
5_H	Logical shift right by 1 (LSR1)

(table continues...)

Table 80 (continued) Vector unit opcodes

Opcode	Operation
6_H	Logical shift left by 1 (LSL1)
7_H	Logical shift right (LSR)
8_H	Conditional subtraction (COND_SUB)
9_H	Move (MOV)
A_H to F_H	Reserved

- Bit positions 27:25 are reserved
- Bit positions 24:16 specify the word length (minus 1) of the destination operand. A 9-bit word length allows for up to 512 32-bit word (16384-bit) destination operands
- Bit positions 15:8 specify the word length (minus 1) of source operand 1 (if present)
- Bit positions 7: 0 specify the word length (minus 1) of source operand 0 (if present). An 8-bit word length allows for up to 256 32-bit word (8192-bit) source operands

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

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

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

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

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

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

- **Word 1**: Pointer to word 0 (the least significant word) of the source operand. The pointer must be 32-bit word-aligned
- **Word 2**: Shift amount. Bit positions 12:0 specify the shift amount
- **Word 3**: Pointer to word 0 (the least significant word) of the destination operand. The pointer must be 32-bit word-aligned

Note on operand size:

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

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

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

  The following operations do not allow for the use of the same memory locations for the source and the destination operands:
  - MUL, XMUL, LSR, and COND_SUB operations

***Note**: The VU operates on 32-bit word multiples and assumes a little-endian memory layout of these words in memory. The PSC3M/P8, PSC3M/P7 MCU uses a little-endian memory layout. As a result, no endianness conversion is required.*

12.3.4 True random number generator (TRNG)

The true random number generator component (TRNG) generates true random numbers. The bit size of these generated numbers is fixed at 32 bits.

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

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

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

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

{r0, r1} = {0, 0}: no bit is produced

{r0, r1} = {0, 1}: a ‘0’ bit is produced (bit r0)

{r0, r1} = {1, 0}: a ‘1’ bit is produced (bit r0)

{r0, r1} = {1, 1}: no bit is produced

That is, the correction step only produces a bit on a ‘0’ to ‘1’ or ‘1’ to ‘0’ transition. Note that for a random input bit sequence, the correction step produces an output bit sequence of close to one-quarter the frequency of the input bit sequence (the input reduction bits are processed in non-overlapping pairs, and only half of the pair encodings result in an output bit).

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

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

- During the initial delay, the ring oscillator is not a reliable physical noise source
- After an initial delay, the same ring oscillator will show different oscillation behavior and provide a reliable physical noise source

Therefore, the DAS bits can be dropped during an initialization period (CRYPTOLITE_TRNG_CTL0.INIT_DELAY).

The following figure gives an overview of the TRNG component.

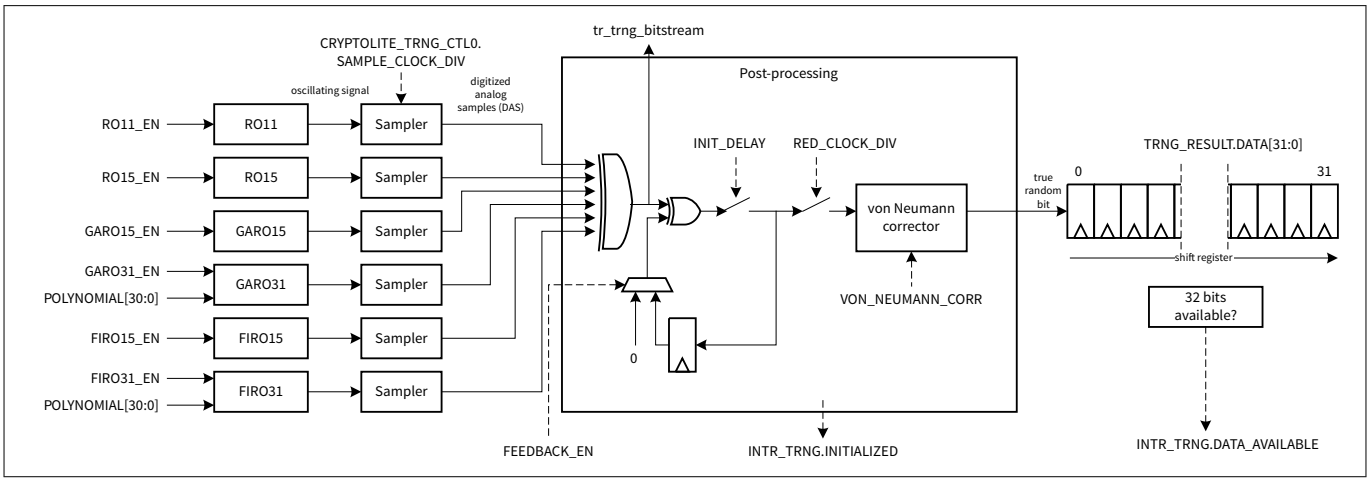


Figure 39 TRNG overview

The FEEDBACK_EN bitfield of the CRYPTOLITE_TRNG_CTL0 register controls the feedback. When '0', there is no feedback. This functionality is present, as it was found that the "von Neumann" corrector may be less effective with feedback enabled.

The "Sampler" logic digitizes an oscillating signal. The following figure illustrates the functionality.

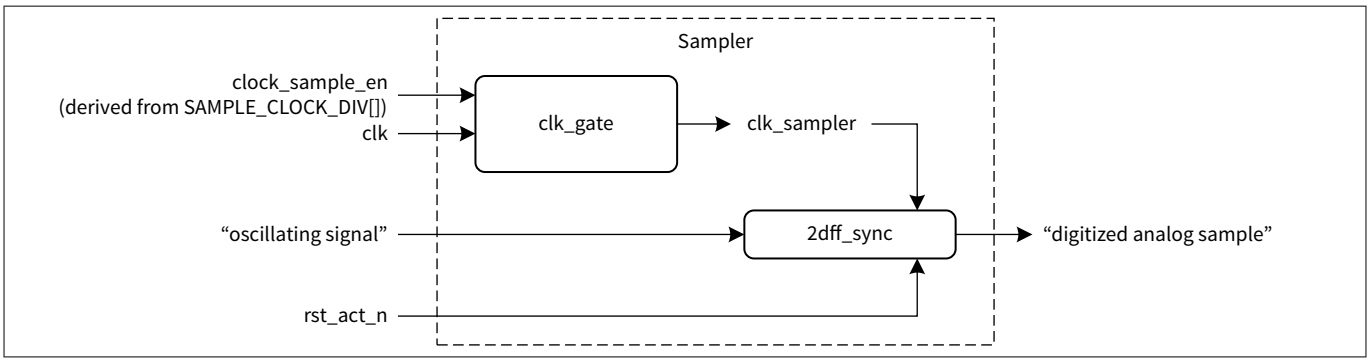


Figure 40 Sampler logic

***Note**: When a ring oscillator is disabled, the synchronization logic is reset. Therefore, the ring oscillator contributes a constant '0' to the reduction step of the post-processing.*

As mentioned, the TRNG relies on up to six ring oscillators:

- RO11: A fixed-ring oscillator consisting of 11 inverters
- RO15: A fixed-ring oscillator consisting of 15 inverters
- GARO15: A fixed Galois-based ring oscillator of 15 inverters
- GARO31: A flexible Galois-based ring oscillator of up to 31 inverters. A programmable polynomial of up to order 31 provides flexibility in the oscillator feedback
- FIRO15: A fixed Fibonacci-based ring oscillator of 15 inverters
- FIRO31: A flexible Fibonacci-based ring oscillator of up to 31 inverters. A programmable polynomial of up to order 31 provides flexibility in the oscillator feedback

Each ring oscillator can be enabled or disabled. When disabled, the ring is "broken" to prevent switching. The XXX_EN enable signals originate from the CRYPTOLITE_TRNG_CTL1 register.

***Note**: When using TRNG, all oscillators shall be active. The recommend polynomial of GARO31 is 0xB2D1AB70, and the recommend polynomial of FIRO31 is 0xE6B8C3B3.*

12.3.4.1 Fixed ring oscillators: RO11, RO15, GARO15, FIRO15

The following figure illustrates the schematics of the fixed-ring oscillators.

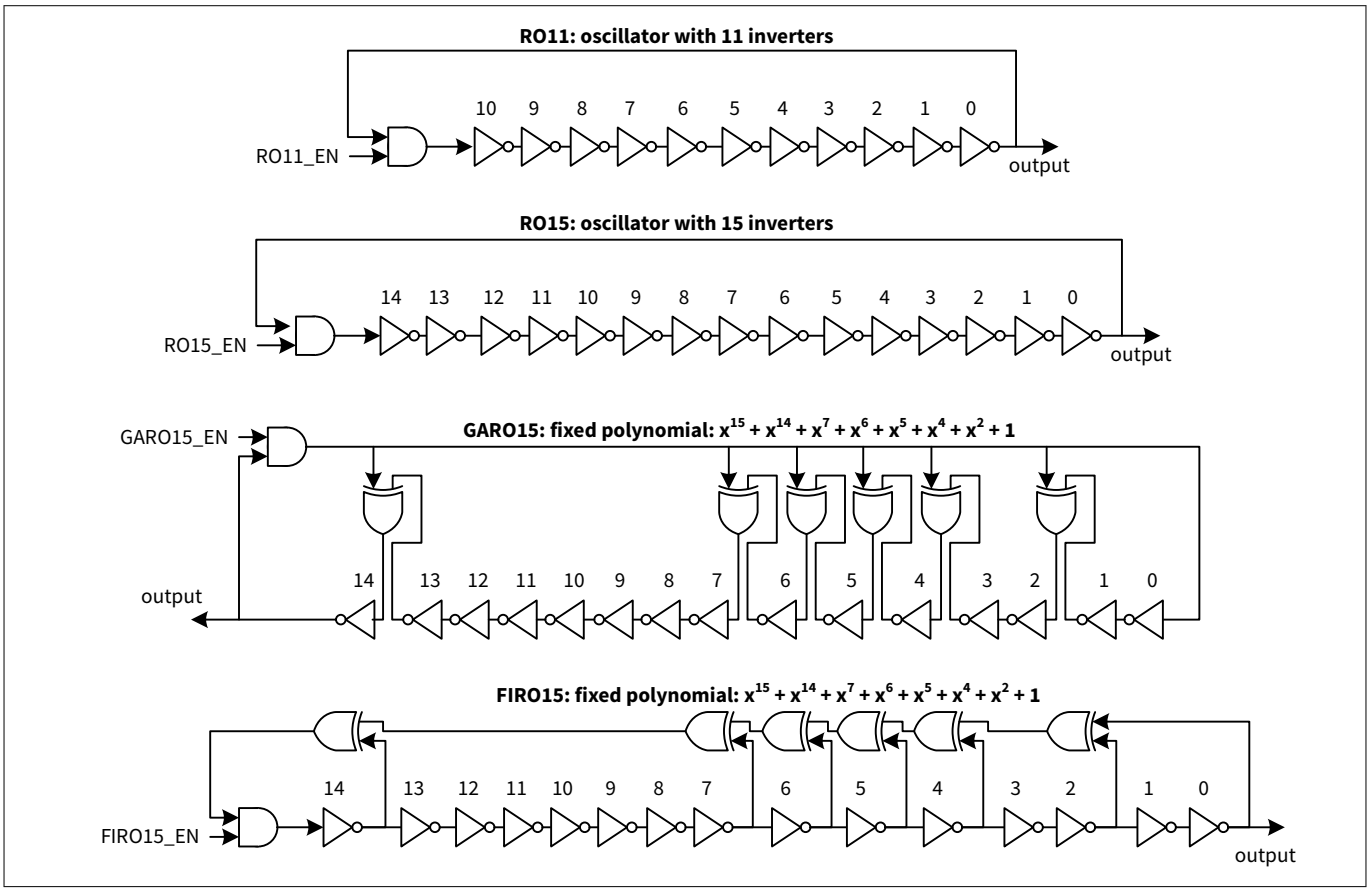


Figure 41 Four fixed-ring oscillators: RO11, RO15, GARO15, and FIRO15

12.3.4.2 Ring oscillator: GARO31

The flexible Galois based-ring oscillators rely on programmable polynomials to specify the oscillator feedback. This allows for rings of 1, 3, 5,…, 31 inverters (an odd number is required to generate an oscillating signal). The following figure gives an overview of the Galois-based ring oscillator.

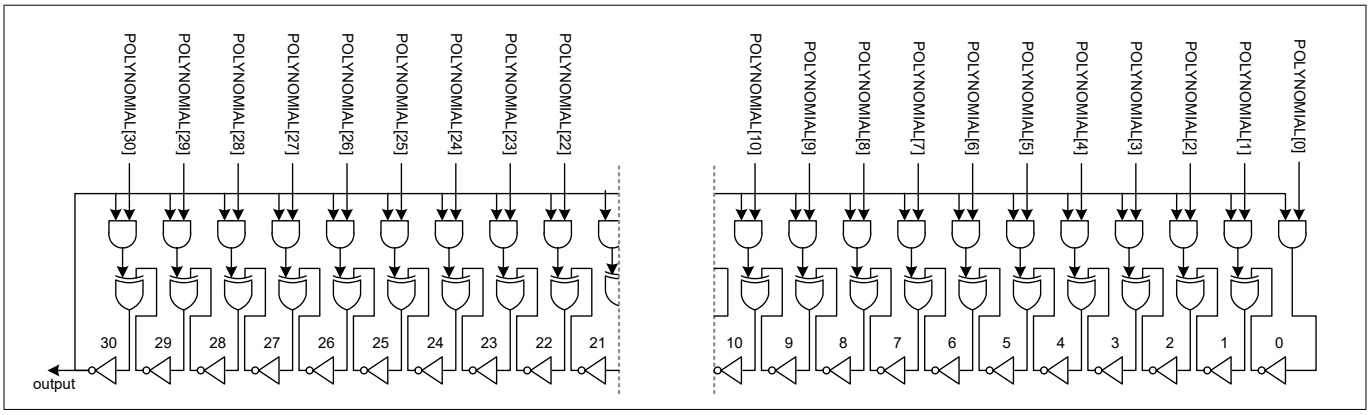


Figure 42 Flexible Galois-based ring oscillator: GARO31

When the ring oscillator is disabled (GARO31_EN bit of the CRYPTOLITE_TRNG_CTL1 register is '0'), the polynomial is forced to "0" and the ring is broken, as illustrated in the following figure.

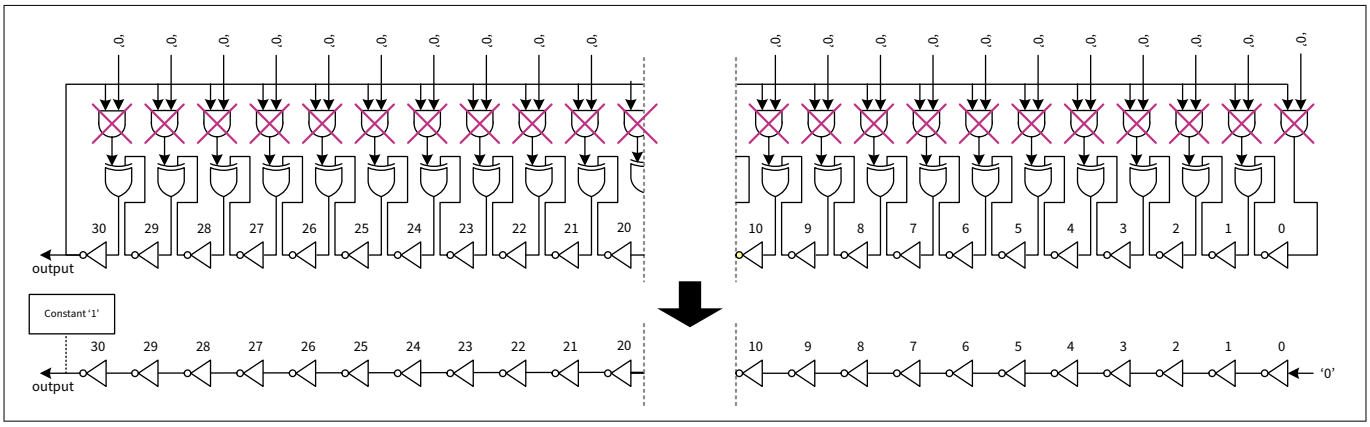


Figure 43 GARO31 stopped

The programmable polynomial specifies the oscillator feedback. The following figure illustrates two examples.

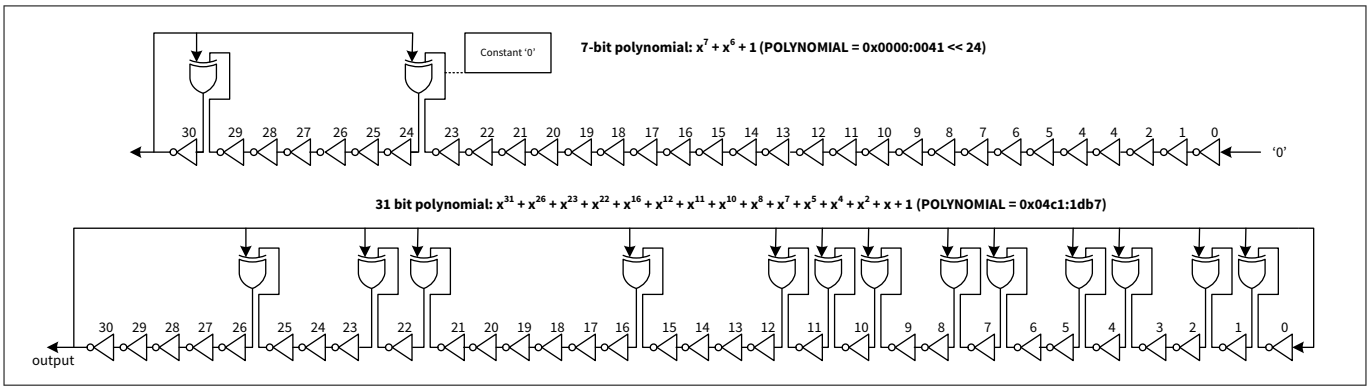


Figure 44 GARO31 examples

12.3.4.3 Ring oscillator: FIRO31

The flexible Fibonacci-based ring oscillators rely on programmable polynomials to specify the oscillator feedback. This allows for rings of 1, 3, 5,…, 31 inverters (an odd number is required to generate an oscillating signal). The following figure gives an overview of the Fibonacci-based ring oscillator.

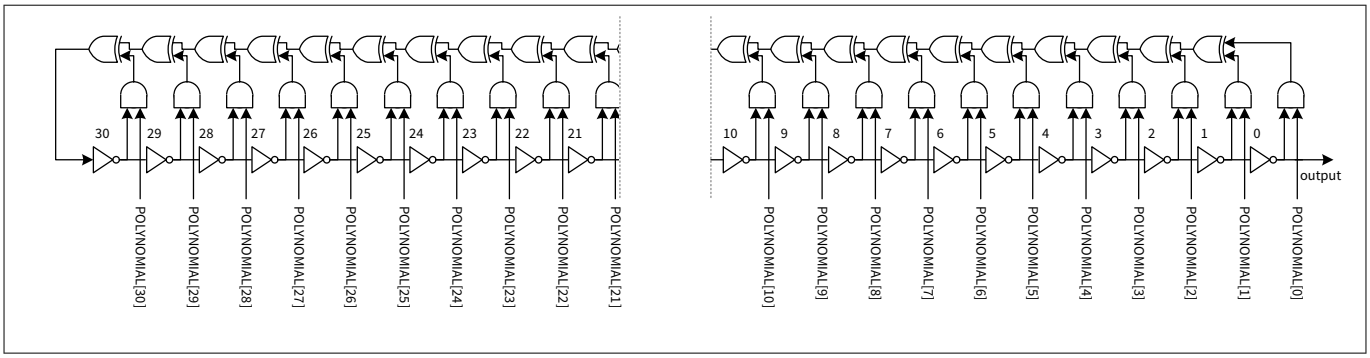


Figure 45 Flexible Fibonacci-based ring oscillator: FIRO31

When the ring oscillator is disabled (FIRO31_EN bit of the CRYPTOLITE_TRNG_CTL1 register is '0'), the polynomial is forced to "0" and the ring is broken, as illustrated in the following figure.

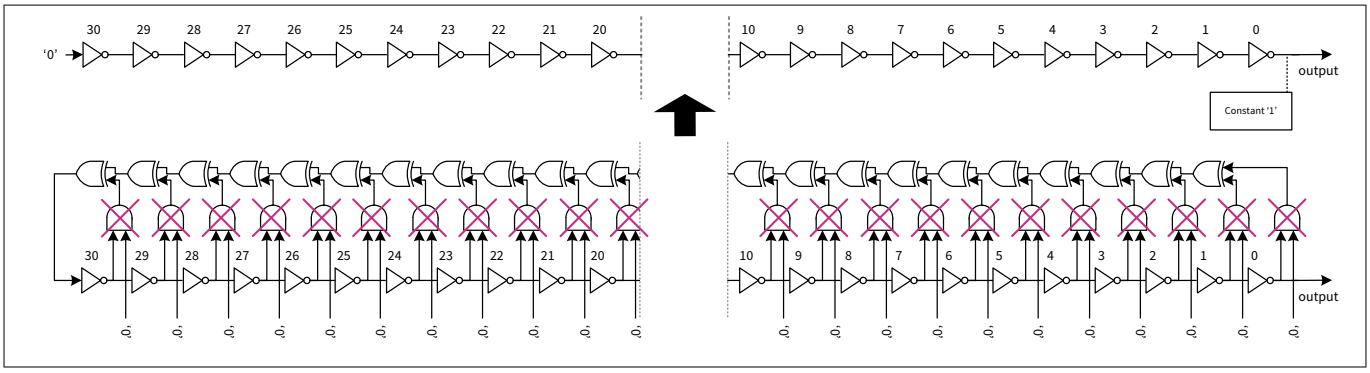


Figure 46 FIRO31 stopped

The programmable polynomial specifies the oscillator feedback. The following figure illustrates two examples.

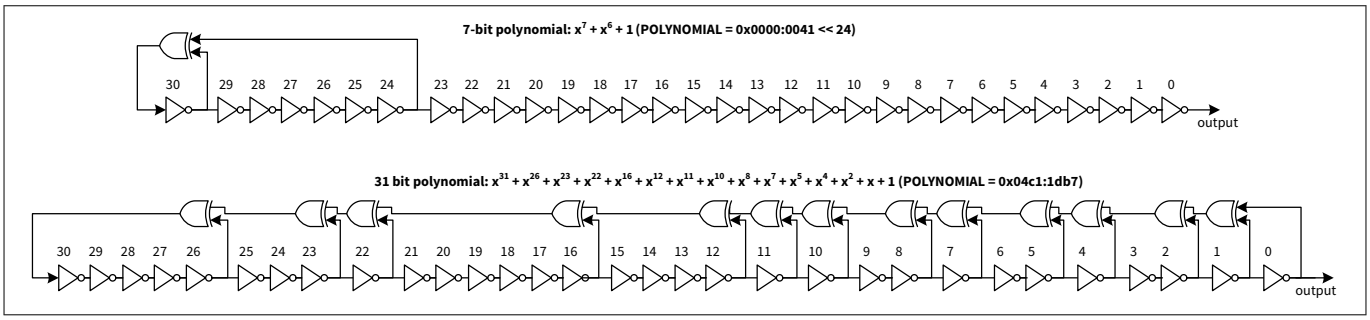


Figure 47 FIRO31 examples

12.3.4.4 TRNG result

A 32-bit TRNG value is provided through a read from the CRYPTOLITE_TRNG_RESULT register. This read is a blocking read; when 32-bit is not available, the read is blocked. A read from the CRYPTOLITE_TRNG_RESULT register returns "0" when the TRNG is not enabled (all XXX_EN fields of the CRYPTOLITE_TRNG_CTL1 register are '0').

12.3.4.5 TRNG health monitor

The TRNG has a built-in health monitor that performs tests on the digitized noise source to detect deviations from the intended behavior. For example, the health monitor detects "stuck at" faults in the digitized analog samples. The following figure illustrates the functionality of the health monitor.

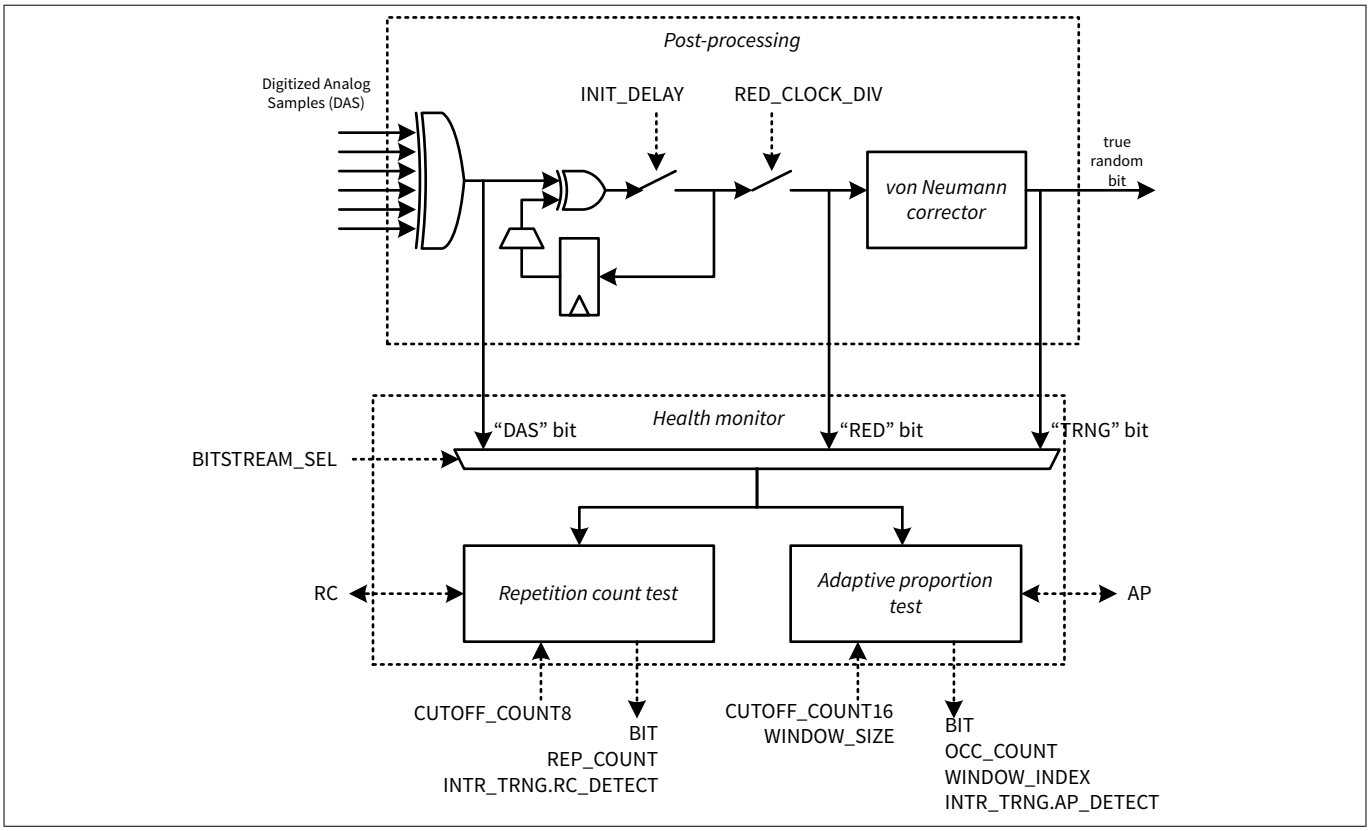


Figure 48 TRNG health monitor

The health monitor tests one out of three selected digitized bit streams:

- DAS bit-stream: This is the XOR of the digitized analog samples
- RED bit-stream: This is the bit-stream of reduction bits. Note that each reduction bit may be calculated over multiple DAS bits
- TR bit-stream: This is the bit-stream of true random bits (after the "von Neumann reduction" step)

The health monitor performs two different tests:

- **The repetition count test**

  This test checks for the repetition of the same bit value ('0' or '1') in a bit-stream. A detection indicates that a specific active bit value (specified by a status field BIT) has repeated for a pre-programmed number of bits (specified by a control field CUTOFF_COUNT8[7:0] of the CRYPTOLITE_RNG_MON_RC_CTL register). The test uses a counter to maintain the number of repetitions of the active bit value (specified by the status field REP_COUNT[7:0] of the CRYPTOLITE_TRNG_MON_RC_STATUS1 register).

  If the test is started (specified by the RC bitfield of the CRYPTOLITE_TRNG_MON_CTL register) and a change in the bit-stream value is observed, the active bit value BIT is set to the new bit value, and the repetition counter REP_COUNT[] is set to "1". If the bit-stream value is unchanged, the repetition counter REP_COUNT[] is incremented by "1".

  A detection stops the repetition count test (hardware sets the RC bit of the CRYPTOLITE_TRNG_MON_CTL register to '0'), sets the associated interrupt status field to '1', and ensures that hardware does not modify the status fields. When the test is stopped, REP_COUNT[] equals CUTOFF_COUNT8[].

  A detection stops the TRNG functionality (all XXX_EN fields of the CRYPTOLITE_TRNG_CTL1 register are set to '0' by hardware) if the STOP_ON_RC_DETECT bit of the CRYPTOLITE_TRNG_CTL0 register is set to '1'

- **The adaptive proportion test**

  This test checks for a disproportionate occurrence of a specific bit value ('0' or '1') in a bit stream. A detection indicates that a specific active bit value (specified by a status field BIT) has occurred a pre-

programmed number of times (specified by a control field CUTOFF_COUNT16[15:0] of the CRYPTOLITE_TRNG_MON_AP_CTL register) in a bit sequence of a specific bit window size (specified by a control field WINDOW_SIZE[31:16] of CRYPTOLITE_TRNG_MON_AP_CTL register). The test uses a counter to maintain an index in the current window (specified by WINDOW_INDEX[15:0] of the CRYPTOLITE_TRNG_MON_AP_STATUS1 register) and a counter to maintain the number of occurrences of the active bit value (specified by the status field OCC_COUNT[31:16] of the CRYPTOLITE_TRNG_MON_AP_STATUS1 register).

If the test is started (specified by the AP bit of the CRYPTOLITE_TRNG_MON_CTL register), the bit-stream is partitioned into bit sequences of a specific window size. At the first bit of a bit sequence, the active bit value BIT is set to the first bit value, the counter WINDOW_INDEX is set to “0”, and the counter OCC_COUNT is set to “1”. For all other bits of a bit sequence, the counter WINDOW_INDEX is incremented by “1”. If the new bit value equals the active bit value BIT, the counter OCC_COUNT[] is incremented by “1”. Note that the active bit value BIT is only set at the first bit of a bit sequence.

A detection stops the adaptive proportion test (hardware sets the AP bit of the CRYPTOLITE_TRNG_MON_CTL register to ‘0’), sets the associated interrupt status field to ‘1’, and ensures that hardware does not modify the status fields. When the test is stopped, OCC_COUNT[] equals CUTOFF_COUNT16[] and the WINDOW_INDEX identifies the bit sequence index on which the detection occurred.

A detection stops the TRNG functionality (all XXX_EN fields of the CRYPTOLITE_TRNG_CTL1 register are set to ‘0’ by hardware) if the STOP_ON_AP_DETECT bit of the CRYPTOLITE_TRNG_CTL0 register is set to ‘1’

12.4 Interrupts

The CryptoLite provides two interrupts:

- TRNG interrupt
- Error interrupt

12.4.1 TRNG interrupt

This interrupt is used to report TRNG behavior, and this interrupt is generated for the below scenarios:

- TRNG is initialized
- TRNG data is available in the CRYPTOLITE_TRNG_RESULT register
- TRNG monitor detects a "repetition count" error
- TRNG monitor detects an "adaptive proportion" error

The TRNG interrupt has an interrupt mask in the CRYPTOLITE_INTR_TRNG_MASK register. By making the interrupt mask low, the corresponding interrupt source is ignored. The TRNG interrupt is generated if the interrupt mask bit is high and the corresponding interrupt source is pending.

When servicing a TRNG interrupt, the interrupt service routine (ISR) can clear the interrupt source by writing a ‘1’ to the corresponding interrupt bit in the CRYPTOLITE_INTR_TRNG register.

The CRYPTOLITE_INTR_TRNG_MASKED register is the logical AND between the interrupt sources and the interrupt mask. This register provides a convenient way for the firmware to determine the source of the interrupt.

For verification and debug purposes, a set bit (such as DATA_AVAILABLE in the CRYPTOLITE_INTR_TRNG_SET register) is used to trigger each interrupt. This action allows the firmware to generate an interrupt without the actual event occurring.

12.4.2 Error interrupt

This interrupt is generated for an AHB bus error on the AHB master interface.

The error interrupt is enabled by setting the BUS_ERROR bit of the CRYPTOLITE_INTER_ERROR_MASK register and reads the CRYPTOLITE_INTR_ERROR_MASKED register to know the interrupt masked status forwarded to the CPU interrupt controller.

When servicing the error interrupt, the interrupt service routine (ISR) can clear the interrupt by writing a "1" to the BUS_ERROR bit of the CRYPTOLITE_INTR_ERROR register.

The BUS_ERROR bit of the CRYPTOLITE_INTR_ERROR_SET register can be used to assert an interrupt for firmware debugging.

***Note**: The cryptographic block terminates its AES, SHA, or VU functionality when it detects an error. The error is sticky; this allows software to check for an error after a series of operations rather than checking after every individual operation.*

12.5 Registers summary

Table 81 CryptoLite register summary

Register	Function
CRYPTOLITE_CTL	CryptoLite control register
CRYPTOLITE_STATUS	CryptoLite status register
CRYPTOLITE_AES_DESCR	AES descriptor pointer register
CRYPTOLITE_VU_DESCR	VU descriptor pointer register
CRYPTOLITE_SHA_DESCR	SHA descriptor pointer register
CRYPTOLITE_INTR_ERROR	CryptoLite error interrupt register
CRYPTOLITE_INTR_ERROR_SET	CryptoLite error interrupt set register
CRYPTOLITE_INTR_ERROR_MASK	CryptoLite error interrupt mask register
CRYPTOLITE_INTR_ERROR_MASKED	CryptoLite masked error interrupt output register
CRYPTOLITE_TRNG_CTL0	TRNG control 0 register
CRYPTOLITE_TRNG_CTL1	TRNG control 1 register
CRYPTOLITE_TRNG_STATUS	TRNG status register
CRYPTOLITE_TRNG_RESULT	TRNG result register
CRYPTOLITE_TRNG_GARO_CTL	TRNG Galois-based ring oscillator (GARO) control register
CRYPTOLITE_TRNG_FIRO_CTL	TRNG Fibonacci-based ring oscillator (FIRO) control register
CRYPTOLITE_TRNG_MON_CTL	TRNG health monitor control register
CRYPTOLITE_TRNG_MON_RC_CTL	TRNG health monitor repetition count test control register
CRYPTOLITE_TRNG_MON_RC_STATUS0	TRNG health monitor repetition count test status 0 register
CRYPTOLITE_TRNG_MON_RC_STATUS1	TRNG health monitor repetition count test status 1 register
CRYPTOLITE_TRNG_MON_AP_CTL	TRNG health monitor adaptive proportion test control register
CRYPTOLITE_TRNG_MON_AP_STATUS0	TRNG health monitor adaptive proportion test status 0 register
CRYPTOLITE_TRNG_MON_AP_STATUS1	TRNG health monitor adaptive proportion test status 0 register
CRYPTOLITE_INTR_TRNG	TRNG interrupt register

(table continues...)

Table 81 (continued) CryptoLite register summary

Register	Function
CRYPTOLITE_INTR_TRNG_SET	TRNG interrupt set register
CRYPTOLITE_INTR_TRNG_MASK	TRNG interrupt request mask register
CRYPTOLITE_INTR_TRNG_MASKED	TRNG masked interrupt output register

13 TrustZone

13.1 Features

- Arm® TrustZone enabled Cortex®-M33 CPU
- Secure and non-secure CPU states
- Attribution units - Secure Attribution Unit (SAU) and Implementation Defined Attribution Unit (IDAU)
- Secure and non-secure aliasing of the memory map
- Secure and non-secure interrupts
- Secure and non-secure peripherals
- Secure and non-secure debug

13.2 Architecture

Arm® TrustZone technology is an optional security extension designed for Armv8-M. The security extension is enabled on PSC3M/P8, PSC3M/P7 devices for improved security.

TrustZone implements system-wide security by partitioning the resources (memory and peripherals) into secure or non-secure worlds. TrustZone introduces two more states to the CPU - secure and Non-secure. These security states are orthogonal to the existing Thread and Handler modes, resulting in four modes: Secure Handler, Secure Thread, Non-secure Handler, and Non-secure Thread modes. In the PSC3M/P8, PSC3M/P7 device, with TrustZone extension enabled, the M33 CPU starts in a secure state. In the secure state the software has access to both secure as well as non-secure resources, while in the non-secure state, software has access only to non-secure resources. This allows execution of secure and non-secure software on the same CPU with an isolation of resources between the two worlds and which is based on the memory map. The partitioning of memory map is achieved with a programmable Secure Attribution Unit (SAU) and a fixed Implementation Defined Attribution Unit (IDAU).

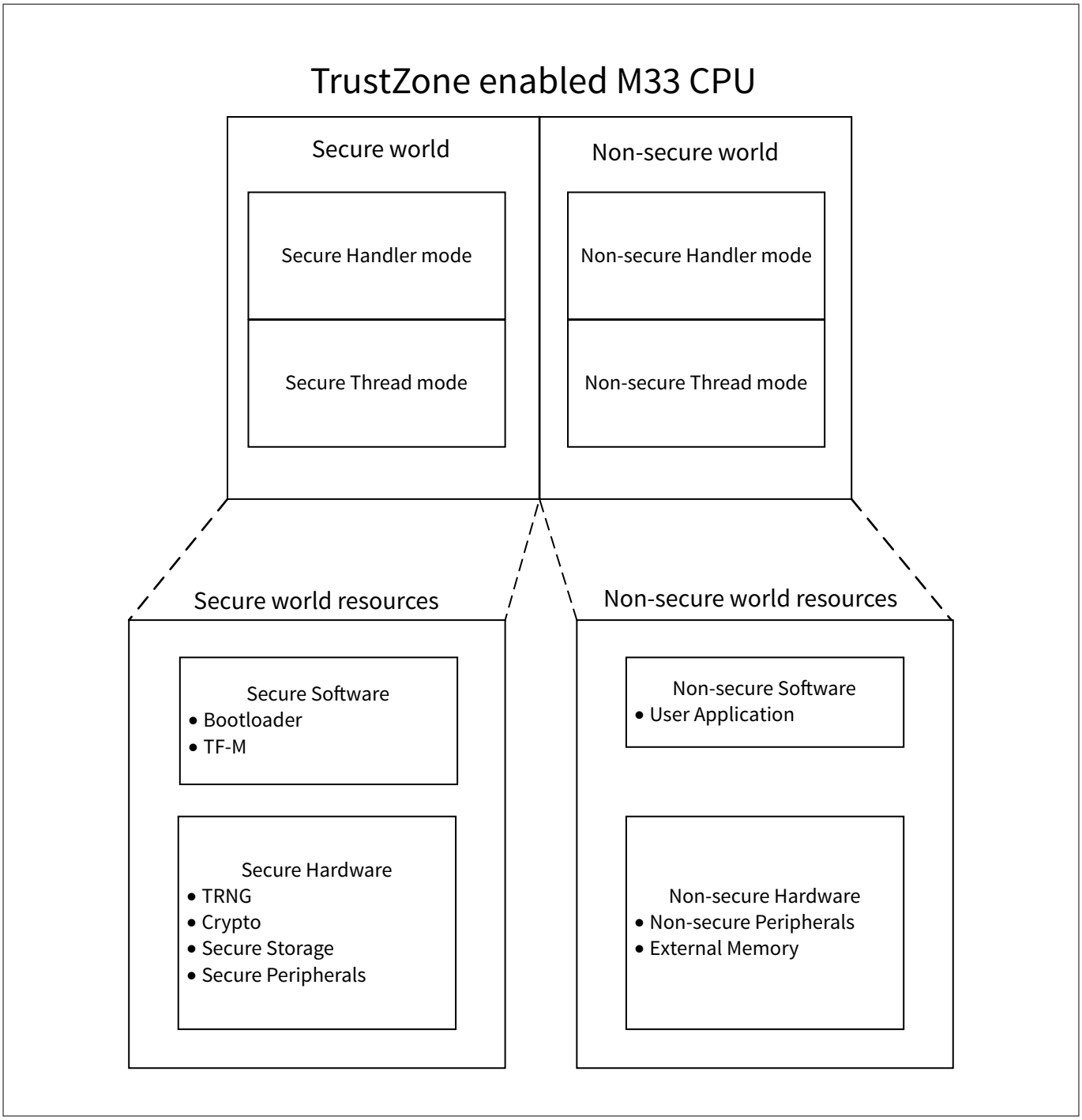


Figure 49 Arm® TrustZone enabled system overview

The following key internal resources are banked between secure and non-secure states:

- Stack pointers
- Interrupt Masking registers
- MPU
- SysTick timer
- VTOR
- Some other registers of System Control Block

Some registers are banked on a bit-by-bit basis. See Armv8-M Architecture Reference Manual for full list of banked registers. The M33 CPU has separate vector tables for secure and non-secure software. With the security extension enabled, a new SecureFault exception is added in the system.

13.3 Memory regions

The complete 4 GB address space is partitioned into secure and non-secure regions. These regions are aliased in the address space, meaning that one memory location is mapped to two different locations in the address map, one in the secure region and the other in the non-secure region. All memories and peripherals are aliased and placed in both regions.

The address space is partitioned into three regions:

- Secure (S):
  - Secure addresses are used for accessing memory and peripherals which are accessible only by Secure software
  - A CPU in secure state can only execute instructions from the secure region
  - Data in secure addresses is accessible by Secure software
- Non-secure callable (NSC):
  - Special type of secure region which holds the secure gateway (SG) instructions, allows the state transition from non-secure to secure state
- Non-secure (NS) :
  - Non-Secure addresses are used for accessing memory and peripherals which are accessible by both Secure and Non-secure software
  - A CPU in the Non-Secure state execute instructions only from the non-secure region
  - Data in Non-Secure addresses is accessible by secure and non-secure software

The attribution units, SAU, and IDAU assign security attributes to memory locations and partition the memory space. Any access violation originating from the security attribute checks generates a SecureFault exception.

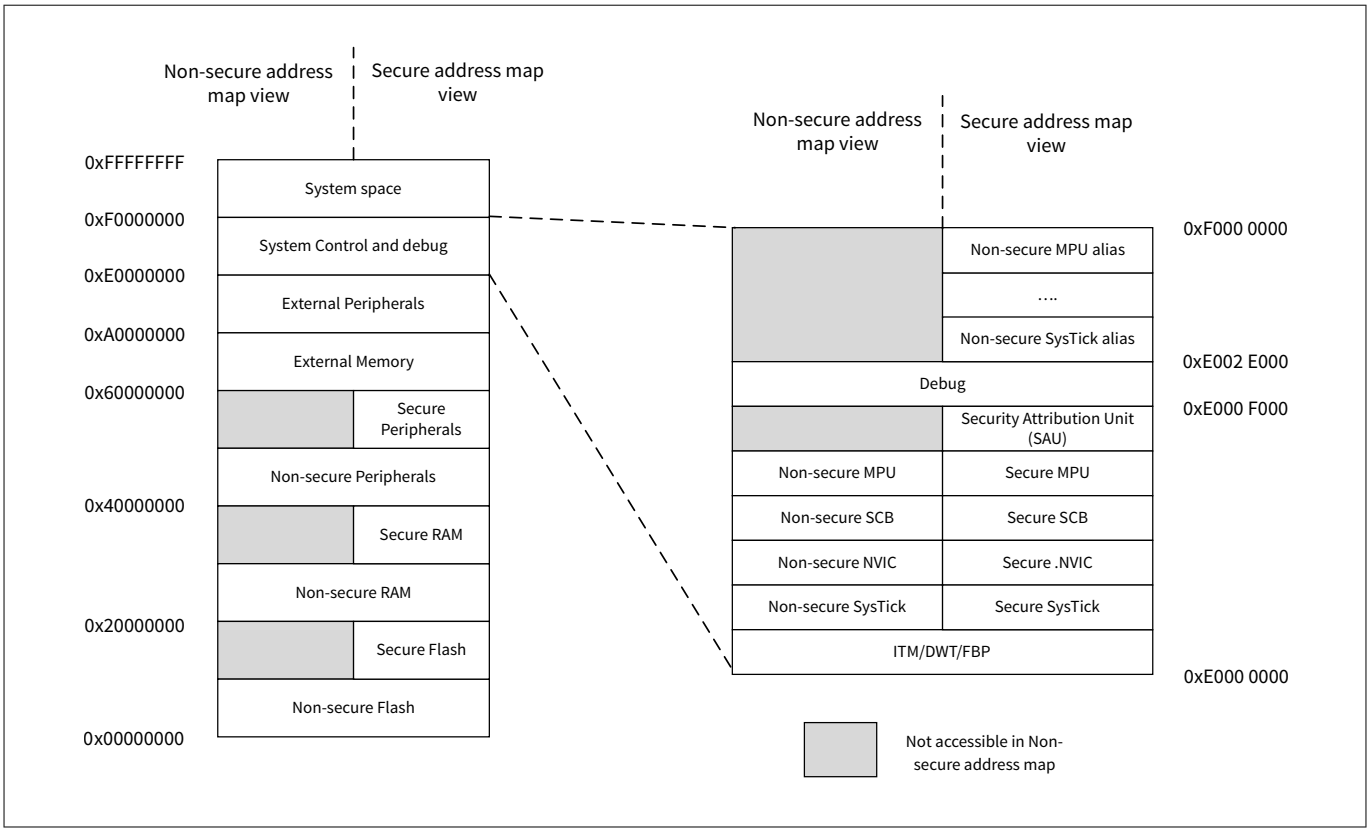


Figure 50 Memory space of TrustZone enabled system

13.4 Attribution units

The memory space is partitioned with Secure Attribution Unit (SAU) and Implementation defined Attribution Unit (IDAU). The SAU and IDAU work in conjunction to determine the security attribute of an address. The higher security level amongst the two will be applied. The HIGH to LOW security levels are as follows: Secure (S) > non-secure callable (NSC) > non-secure (NS)

Table 82 summarizes the resultant security attribute.

Table 82 Resultant security level

SAU	IDAU	Resultant security attribute
S	X	S
X	S	S
NSC	NSC or NS	NSC
NSC or NS	NSC	NSC
NS	NS	NS

***Note**: X = Don't Care*

13.4.1 Implementation Defined Attribution Unit (IDAU)

- IDAU provides an address aliasing scheme that allows a memory space to be partitioned into non-secure, non-secure callable and secure regions

- IDAU has a fixed implementation, meaning that the address aliasing scheme is not configurable
- IDAU uses the address bit 28 to segregate secure and non-secure memory regions
- Private peripheral bus (PPB) and external private peripheral bus (EPPB) regions are exempt from security attribute check
- IDAU address map partitioning can be overridden only with a SAU configuration of a higher security level
- IDAU implementation is attached externally to M33 CPU and all non-CPU masters on their AHB buses
- IDAU returns region number for all addresses, required for Test Target (TT) instruction

Table 83 shows the memory space partitioning defined by the fixed IDAU implementation:

Table 83 IDAU memory map partitioning

Region	Start address	End address	Size	Security attribute	Region number
Code NS	0x0000 0000	0x0FFF FFFF	256 MB	NS	0x0
Code S	0x1000 0000	0x1FFF FFFF	256 MB	NSC	0x1
RAM NS	0x2000 0000	0x2FFF FFFF	256 MB	NS	0x2
RAM S	0x3000 0000	0x3FFF FFFF	256 MB	NSC	0x3
Peripherals NS	0x4000 0000	0x4FFF FFFFF	256 MB	NS	0x4
Peripherals S	0x5000 0000	0x5FFF FFFF	256 MB	S	0x5
External memory NS	0x6000 0000	0x6FFF FFFF	256 MB	NS	0x6
External memory S	0x7000 0000	0x7FFF FFFF	256 MB	S	0x7
External memory reserved	0x8000 0000	0x8FFF FFFF	256 MB	NS	0x8
	0x9000 0000	0x9FFF FFFF	256 MB	S	0x9
External device reserved	0xA000 0000	0xAFFF FFFF	256 MB	NS	0xA
	0xB000 0000	0xBFFF FFFF	256 MB	S	0xB
	0xC000 0000	0xCFFF FFFF	256 MB	NS	0xC
	0xD000 0000	0xDFFF FFFF	256 MB	S	0xD
System address space	0xE000 0000	0xE00F FFFF	1 MB	Exempt	0xE
	0xE010 0000	0xEFFF FFFF	255 MB	NS	0xE
	0xF000 0000	0xF00F 0000	1 MB	Exempt	0xF
	0xF010 0000	0xFFFF FFFF	256 MB	S	0xF

***Note**: IDAU uses address bit 28 for S and NS indication. A single-bit can only indicate two states - 0: NS, 1: S. To indicate NSC regions, IDAU re-uses the S indication. IDAU marks the address ranges 0x1000_0000 - 0x1FFF_FFFF (CODE S) and 0x3000_0000 - 0x3FFF_FFFF (RAM S) as NSC by reusing the S indication.*

13.4.2 Secure Attribution Unit (SAU)

- SAU is a programmable unit and is a part of M33 CPU
- SAU configuration is only applicable to M33 CPU
- SAU supports eight programmable regions

- SAU is disabled at reset
- When the SAU is disabled, the address space attribution is controlled by SAU_CTRL.ALLNS bit field
- When the SAU is enabled, the eight programmable regions are used for marking memory as NS or NSC. Memory regions that are not covered by SAU are marked as Secure
- It is the job of secure software to configure SAU and to create NS and NSC regions that enables non-secure software execution

Table 84 shows an example SAU configuration.

Table 84 Example SAU configuration

Region	Purpose	Start address	End address	SAU security attribute	IDAU security attribute	Resultant security attribute
Code NS	Flash	0x02000000	0x0201FFFF	-	NS	S
		0x02020000	0x0207FFFF	NS		NS
	SRAM	0x04000000	0x04007FFF	-	NS	S
		0x04008000	0x0401FFFF	NS		NS
Code S	Flash	0x12000000	0x1201FEFF	-	NSC	S
		0x1201FF00	0x1201FFFF	NSC		NSC
		0x12020000	0x1207FFFF	-		S
	SRAM	0x14000000	0x1401FFFF	-	NSC	S
RAM NS	Flash	0x22000000	0x2201FFFF	-	NS	S
		0x22020000	0x2207FFFF	NS		NS
	SRAM	0x24000000	0x24007FFF	-	NS	S
		0x24008000	0x2401FFFF	NS		NS
RAM S	Flash	0x32000000	0x3207FFFF	-	NSC	S
	SRAM	0x34000000	0x3401FFFF	-	NSC	S
Peripheral NS	Peripherals	0x42000000	0x43FFFFFF	NS	NS	NS
Peripheral S	Peripherals	0x52000000	0x53FFFFFF		S	S
External memory NS	-	0x60000000	0x6FFFFFFF		NS	NS
External memory S	-	0x60000000	0x7FFFFFFF		S	S
External memory reserved	-	0x80000000	0x8FFFFFFF		NS	NS
		0x90000000	0x9FFFFFFF		S	S
External device reserved	-	0xA0000000	0xAFFFFFFF		NS	NS
		0xB0000000	0xBFFFFFFF		S	S
		0xC0000000	0xCFFFFFFF		NS	NS
		0xD0000000	0xDFFFFFFF		S	S

(table continues...)

Table 84 (continued) Example SAU configuration

Region	Purpose	Start address	End address	SAU security attribute	IDAU security attribute	Resultant security attribute
System address space	System control and debug	0xE0000000	0xE00FFFFF		Exempt	Exempt
		0xE0100000	0xEFFFFFFF		NS	NS
		0xF0000000	0xF00F0000		Exempt	Exempt
		0xF0100000	0xFFFFFFFF		S	S

13.5 State transition

Three instructions are defined for M33, to support security state transition:

- BXNS: Branch with exchange to non-secure
- BLXNS: Branch with link and exchange to non-secure
- SG: Secure gateway

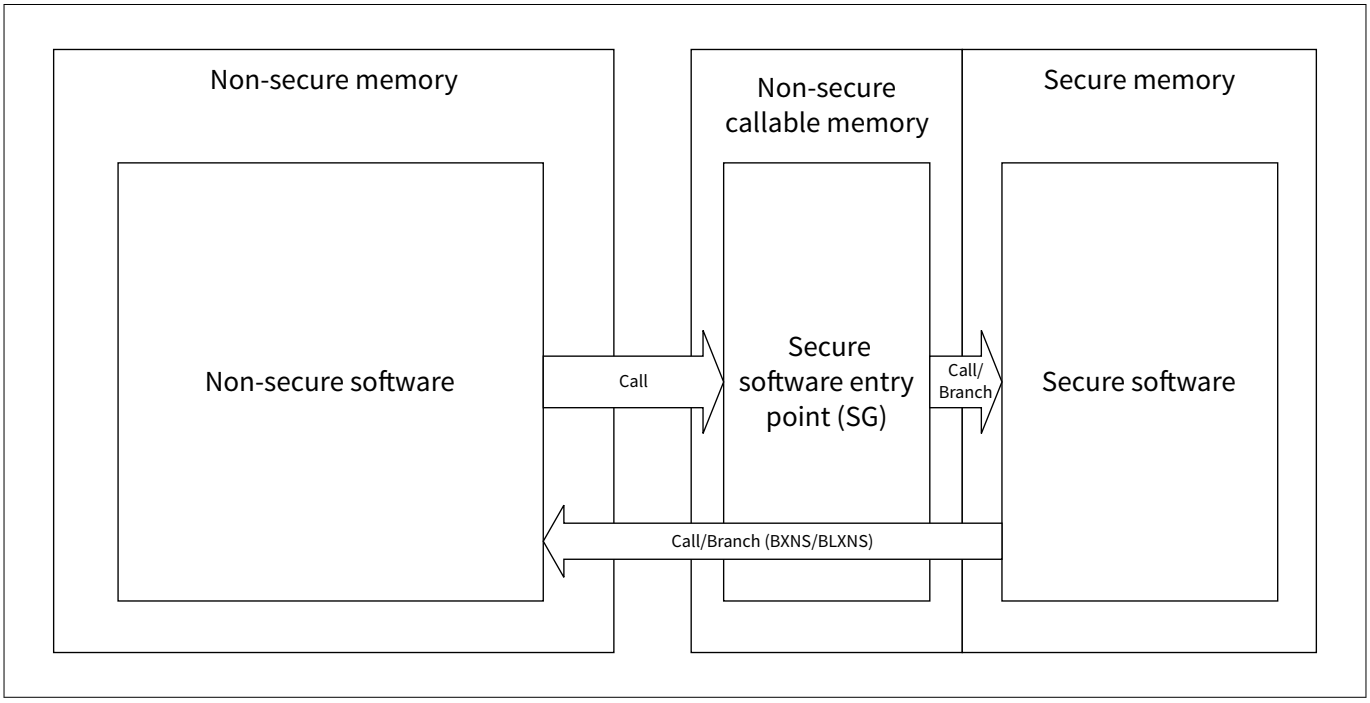


Figure 51 Security state transition

13.5.1 Secure to non-secure state transition

After a reset, the M33 CPU starts in the secure state. In the secure state, the CPU can only execute instructions from the secure memory. For the secure software to call non-secure software, the CPU state transition can be initiated with either BXNS or BLXNS instruction. Calling the BXNS instruction performs state transition and branches to non-secure address. BXLNS performs the state transition and calls a subroutine at a non-secure address.

These instructions are executed only when the CPU is in the secure state. BXNS/BLXNS are undefined if called from non-secure state.

13.5.2 Non-secure to Secure state transition

Non-secure software cannot directly access the Secure memory. For state transition SG instruction should be the first instruction executed from Non-secure Callable region. Successful execution of SG instruction allows CPU to transition to Secure state and now a call/branch to Secure software can be performed. Typically, Non-secure to Secure state transition is implemented to call a subroutine from Secure Software.

SG instruction should only be executed from Non-secure Callable region. If executed from Non-secure region the instruction behaves as a NOP.

13.5.3 State transition from exception or interrupt

The state transition can also take place due to an exception or interrupt. The exceptions and interrupts can be configured to the target either secure or non-secure state.

- If the CPU is in the secure state and an exception or interrupt targeting the non-secure state is triggered, this results in transition from the secure to the non-secure state
- If the CPU is in the non-secure state and an exception or interrupt targeting the secure state is triggered, this results in transition from the non-secure state to the secure state
- If the arriving interrupt has same state as the current CPU state, no state transition occurs

14 Protection units

The PSC3M/P8, PSC3M/P7 MCU implements protection units to enforce security by allowing or restricting bus transfers based on specific properties of a transfer. The rules that determine protection are implemented in MMIO registers of protection units, which define the protected address space and the protection attributes. The hardware that evaluates these protection attributes to restrict or permit access is the protection unit. The PSC3M/P8, PSC3M/P7 devices have different types of protection units such as the Security Attribution Unit (SAU), Implementation Defined Attribution Unit (IDAU), Memory Protection Unit (MPU), Memory Protection Controller (MPC), Peripheral Protection Controller (PPC), and Master Security Controller (MSC). Each type of protection unit has a distinct set of MMIO registers that help define different protection regions and their attributes.

The PSC3M/P8, PSC3M/P7 MCU platform is based on the Armv8-M architecture, and TrustZone security functions are part of this architecture. In a TrustZone-enabled system, software is divided into secure and non-secure domains. The processor handles the partitioning of the address space using the programmable SAU in combination with a device-specific logic block IDAU. For more details, see TrustZone.

While SAU and IDAU are essential, they may not be sufficient for partitioning the memory space into a large number of regions in security-driven applications. In such cases, it may be necessary to partition the address space into many more regions, such as at the page granularity. The MPC and PPC enable the partitioning of the memory space and the peripheral space into several regions. These units control the accessibility of each memory and peripheral region based on non-secure attributes and protection context. The MPU, a programmable unit part of the CPU, allows privileged software to define memory access permissions. Additionally, for bus masters without security extension, the MSC adds TrustZone capability.

The protection units allow or restrict bus transfers on the bus infrastructure based on access attributes such as:

- Read or Write attribute
- Execute attribute to distinguish a code access from a data access
- User or Privileged attribute to distinguish, for example, OS or kernel access from a task or thread access
- Secure or Non-secure attribute to distinguish a ‘Secure’ access from a 'Non-secure' access
- Protection context attribute to distinguish access from different protection contexts

The functioning of a secure system is based on the following components:

- **Bus masters**: This term refers to the bus masters in the architecture. In the PSC3M/P8, PSC3M/P7 device, examples of bus masters include a Cortex®-M core, DW, or the test controller
- **Protection units**: These are hardware blocks that implement the protection scheme to protect memory, peripherals, and shared resources from bus masters. They enforce the protection defined in the protection unit's MMIO registers
- **Protection MMIO registers**: MMIO registers form a register structure in memory that establishes rules based on which each protection unit evaluates a transfer. Each protection unit is associated with a register structure. An MMIO register structure describes the following:
  - A memory region on which the rule is applied. A memory region can be defined with either Start/End address or Start/End block index
  - A set of protection attributes, which are properties based on which a transfer is evaluated. These include:
    - R/W/X
    - User or Privileged
    - Secure or Non-secure
    - Protection context

The set of protection attributes that can be configured in MMIO register depends on the protection unit it is associated with.

Each bus master is associated with its own access attributes, which define the bus master's access privileges. A bus master is associated with attributes such as secure/non-secure, protection context, and user/privileged attribute. These are dynamic attributes that change based on the bus master's context and state. However, for some bus masters, attributes such as secure/non-secure and user/privileged attribute are fixed and cannot be changed.

Protection attributes appear in two places:

- Protection MMIO registers
- Bus master's protection attribute

The protection units evaluate these access attributes of masters against those of memory/peripheral access attributes. If the access attributes match (bus master and memory/peripheral), the protection units allow the transfer; otherwise, the transfer is blocked.

Protection units address the following:

- **Security requirements**: This includes preventing malicious attacks from accessing secure memory or peripherals
- **Safety requirements**: This includes detecting accidental (non-malicious) software errors and random hardware errors. It is important to enable failure analysis to investigate the root cause of a safety violation

14.1 Architecture

When a bus master accesses a memory or peripheral, the access is evaluated by a protection unit against the protection attributes of the memory or peripheral location being accessed. If the bus master’s protection attributes satisfy the protection attributes of the memory or peripheral, the protection unit then allows the access. In the case of an access violation, a fault condition is triggered, resulting in a bus error. MPC and PPC allow configuring the error response to either a Bus Error or Read-Zero Write Ignore (RAZ/WI).

The MPU, SAU, and IDAU are part of the M33 CPU subsystem and protect the entire 4-GB address space. For more information on SAU and IDAU, see Attribution units. MPCs are also part of the M33 CPU subsystem and protect memories from illegal access. A separate protection unit type, PPC, is provided for peripheral protection within the peripheral address space. Additionally, the Master Security Controller (MSC) is implemented for non-CPU bus masters without security extension to add TrustZone capability.

- **MPU**: The M33 bus master has dedicated MPUs and are implemented as part of the CPU, under the control of privileged software. Typically, MPU access attributes are dynamically updated by privileged software such as RTOS/TF-M SPM
- **MPC**: MPCs are designed for implementing protection in scenarios with multiple bus masters. They implement protection based on a concept called Protection Context, allowing them to distinguish between different protection contexts and secure from non-secure accesses, thereby providing effective protection in a multi-master scenario
- **PPC**: PPCs are protection units provided in the peripheral register space for peripheral protection. Their attributes are similar to MPCs, except that they are intended to protect peripheral spaces. PPCs distinguish between different protection contexts and secure from non-secure accesses, as well as user mode accesses from privileged mode accesses. They implement protection for fixed address regions corresponding to specific peripherals
- **MSC**: MSCs are instantiated to provide security extension attributes for adding TrustZone capability. They set up the master’s security attributes and are paired with masters without security extension to protect the transactions initiated on the bus. Examples of masters include DW and Crypto

The PSC3M/P8, PSC3M/P7 protection architecture consists of the following components:

Figure 52 gives an overview of the PSC3M/P8, PSC3M/P7 protection architecture.

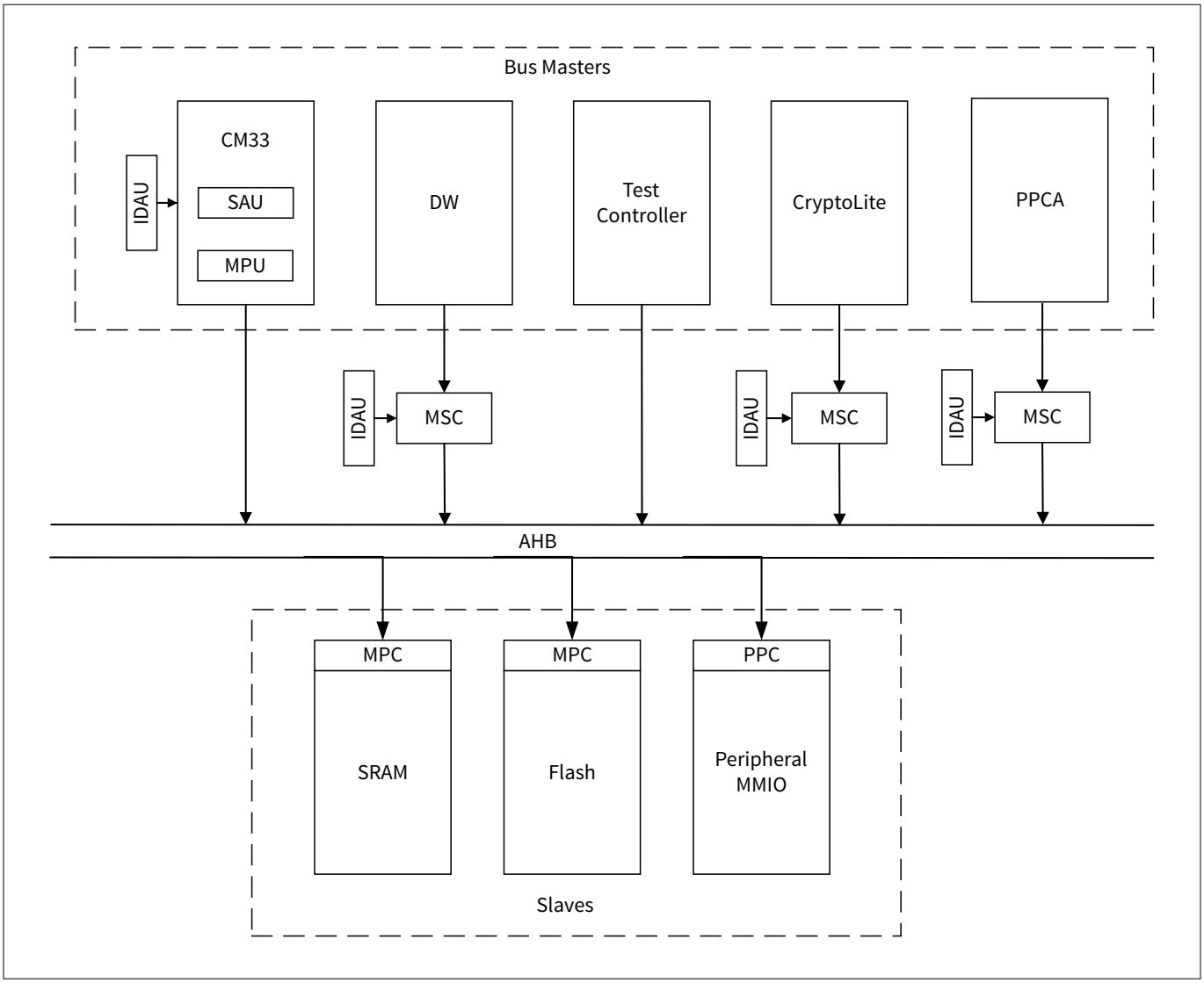


Figure 52 Protection architecture

A protection violation is caused by a mismatch between a bus master's access attributes and the protection attributes for the memory or peripheral region. These violations can be captured in the fault report structure to allow for failure analysis. The fault report structures are capable of generating an interrupt to indicate the occurrence of a fault. This is particularly 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. A protection violation results in a bus error, causing the bus transfer to be halted before reaching its intended target.

Figure 53 shows the attribute check precedence in the PSC3M/P8, PSC3M/P7 device.

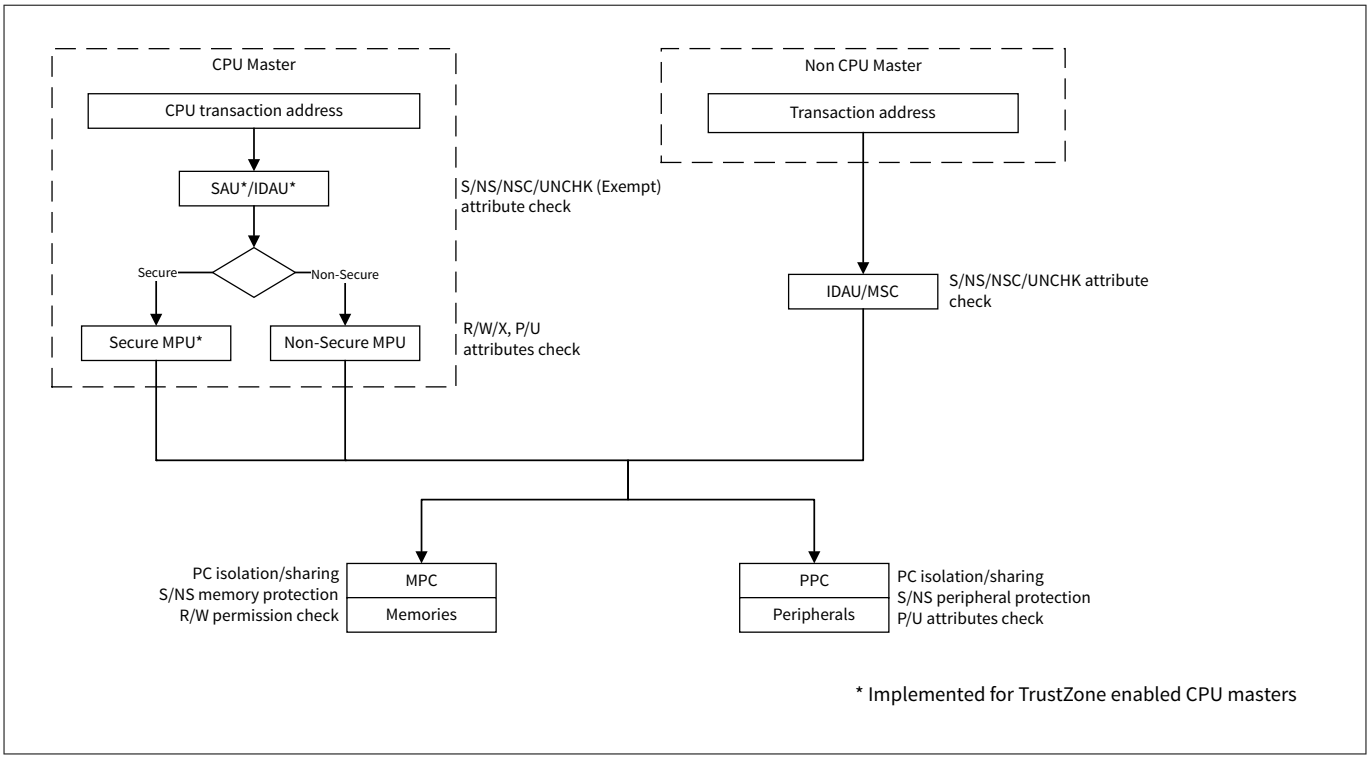


Figure 53 Attribute check precedence

The different types of protection units cater to different use cases, as described in the following table.

Table 85 PSC3M/P8, PSC3M/P7 protection units

Protection unit	Use	Number of instances
MPU	Used to protect memory between tasks within a single Arm® core. A task in one of the Arm® cores can protect its memory from access by another task in the same core.	M33 CPU: 1 Secure and 1 Non-secure MPU with 8 programmable regions for each MPU
MPC	Used to protect memory that is shared between multiple bus masters (with different protection contexts).	7
PPC	These units protect device-defined peripheral memory space. PPC has fixed regions and is used exclusively to protect the predefined peripheral address region (PPC region).	1
SAU	Used for partitioning the address space to isolate Secure and Non-Secure resources.	1
IDAU	Implements fixed partitioning of the address space to isolate Secure and Non-Secure resources.	3
MSC adapter	Used to add TrustZone capability to masters without security extension	2

14.2 Protection context

Protection context is an Infineon proprietary protection mechanism that enables the enforcement of access restrictions to memory and peripherals shared among multiple bus masters, as well as from the same bus master. Each bus master has a protection context field MS_PCx_PC.PC[3:0], which is utilized as the protection context attribute for all bus transfers initiated by the master. The MPCs and PPCs utilize this protection context attribute to allow or restrict bus transfers between a bus master and memory or peripheral.

Programming the master's MS_PCx_PC.PC[3:0] (PC[ ]) field allows for the setting of a bus master's protection context. Changes to the protection context are necessary for bus masters that may transition between multiple tasks, each catering to different protection contexts. As the protection context attribute enables or restricts bus transfers, changes to the protection context field should be carefully controlled to avoid compromising security. Each bus master has an MSx_CTL.PC_MASK[7:0] (PC_MASK[ ]) protection context mask field that identifies the allowed protection contexts for the bus master. This bit field is a 'one-hot' field specifying the allowed protection contexts for a bus master. Multiple bus masters can share a protection context

The protection context field PC[ ] indicates the active protection context (PC), and its modifications are restricted by the associated PC_MASK[ ] value. PC[ ] can only be set to 'i' if the corresponding mask bit PC_MASK[i] is '1'.

For example, when attempting to program MS_PC_STRUCT[x].PC[3:0] to '4':

- If PC_MASK[4] is '1', PC[ ] is set to '4'
- If PC_MASK[4] is '0', PC[ ] is not changed

The PSC3M/P8, PSC3M/P7MCU supports eight protection contexts, with each context predefined for a specific task. Table 86 shows the PC assignment for each task.

Table 86 Protection context assignment to firmware layers

PC	Firmware/software layer
0	ROM_BOOT/FLASH_BOOT
1	-
2	Protected firmware/OEM_BL/TF-M/OEM App
3-6	OEM App
7	SYS-AP (MXDEBUG600)

Table 87 specifies the bus master's protection attributes configured during device boot.

Table 87 Master security attributes

Master name	Master ID	MSx_CTL value			MS_PCx_PC value
		Privileged attribute	NS attribute	PC_MASK	Active PC
M33 CPU	0	-	-	PC2, PC3, PC4, PC5, PC6	PC2
DW	4, 5	Inherited			
CRYPTOLITE	7	Inherited			
PPCA	9	1	-	PC4, PC5, PC6	PC2
Test controller	31	-	-	PC7	PC7

For masters that provide their own access attributes, the corresponding bit fields in the MSx_CTL register structure are not implemented. For instance, the M33 CPU's MS0_CTL does not have P and NS bit fields, as it provides its own P and NS attributes.

***Note**: The DW controller and Crypto accelerator inherit the access control attributes of the bus transfer(s) that programmed the channels or components. For PPCA, the NS attribute bit is always set to 1 and cannot be modified; in other words, it always operates as a non-secure master.*

14.2.1 Special protection contexts

The active PC of the M33 CPU bus master can be changed by writing to the PC[ ] bit field. Access to this field is typically restricted to PC2 secure software, and the active PC cannot be directly modified by any other software. In cases where a switch to PC2/3 from PC4/5/6 is required, it can only be performed with a handler mechanism.

Out of the eight protection contexts PC2 and PC3 are treated as special: the entry to these special PCs is hardware-controlled. For special PCx (where x refers to the special PC number), a programmable exception handler address is provided at MXCM33_CM33_PCx_HANDLER.ADDR[31:0]. The bit field MXCM33_CM33_PC_CTL.VALID[x] indicates whether the handler for PCx, provided in MXCM33_CM33_PCx_HANDLER.ADDR[31:0], is valid. If a CPU exception handler fetch returns a handler address that matches the programmed MXCM33_CM33_PCx_HANDLER.ADDR[31:0] address value, it results in the CM33 PC being changed to PCx by the hardware. If a PC's MXCM33_CM33_PC_CTL.VALID[x] is not set, the PC is treated as an ordinary PC.

Writing to MXCM33_CM33_PCx_HANDLER.ADDR[31:0] and MXCM33_CM33_PC_CTL.VALID[x] bit fields from an OEM application is prohibited. These registers can only be programmed via the OEM provisioning process. The OEM policy provides a JSON object for programming these registers.

For the M33 bus master, the MS_PC0_PC register includes an additional MS_PC0_PC.PC_SAVED[19:16] bit field that holds the saved PC. The current PC and a saved PC implement a two-entry stack. Upon entry of a special exception handler, the hardware pushes the current PC to the stack. It is the responsibility of the exception handler to restore the active PC and saved PC before exiting from the handler.

When PCx is controlled by hardware (indicated by the bit MXCM33_CM33_PC_CTL.VALID[x] being ‘1’), the protection context cannot be selected by the software. In this case, the corresponding bit of the data written to PC_MASK[x] is forced to ‘0’. Note that PC switching with a handler is implemented only for the M33 CPU and not for other bus masters.

14.3 Master protection attributes

The protection attributes in protection unit's MMIO registers establish rules for different memory regions and their access attributes. The protection units use the bus master's own protection attributes to regulate access based on the rules set in the protection unit's MMIO registers. Not all bus masters provide all protection attributes associated with a bus transfer. For example, none of the bus masters have a native protection context attribute, which should be dynamically set based on the task executed by the bus master.

To ensure system-wide restricted access, the missing attributes are provided in the bus master's MMIO register fields. Some of these fields are set during the ROM boot process and cannot be altered by the user code.

- The PC_MASK[] and PC[] register fields provide protection context functionality
- The MSx_CTL.P register field provides the unprivileged or privileged attribute for masters that do not provide their own attribute
- The MSx_CTL.NS register field provides the secure or non-secure attribute for masters that do not provide their own attribute
- The execute attribute of masters that do not provide the execute attribute is set to '0'

See Table 87 for the bus master protection attribute set by the ROM boot code.

14.3.1 Master Security Controller (MSC)

The AHB5 TrustZone Master Security Controller (MSC) serves as an adapter for connecting a legacy AHB-lite master or an AHB5 master without the security extension to an AHB5 system and adding TrustZone for Armv8-M capability. A master without a security extension can be configured as secure or non-secure by either programming the NS bit field, which is a part of the MSx_CTL register, or by inheriting the attribute from the master that programmed it. The MSC evaluates the NS attribute of a security-unaware master against IDAU. If the master has the appropriate NS attribute, access is allowed; otherwise, it is blocked. The MSC can be programmed to behave RZWI or generate an MSC interrupt in case of access violation.

For additional details about the MSC, see the CoreLink SIE-200 technical reference manual from Arm®.

14.3.2 Memory Protection Unit (MPU)

The Armv8-M memory protection units (MPUs) are programmable units and are part of the CPU. They allow the privileged software to define memory access permissions (R/W/X) and memory attributes for normal memory or device memory, and also monitors instruction fetch and data access operations from the CPU. In the PSC3M/P8, PSC3M/P7 platform, the M33 CPU is equipped with one secure MPU and one non-secure MPU with eight programmable regions for each MPU. Any access violation observed by the MPU triggers a MemManage fault.

For more information on the architecture and registers, see the Armv8-M memory model and memory protection unit (MPU) user guide from Arm®.

14.4 Slave protection attributes

The MPC and PPC evaluate the access attributes of memory and peripheral slaves before every transaction with a master.

14.4.1 Memory Protection Controller (MPC)

The memory protection controller (MPC) is located on the AHB bus. The SRAM and Flash are equipped with the MPC.

The MPC provides the following functionality:

- Allocates protection context (PC) based on run-time memory separation or isolation and sharing
- Acts as an Armv8-M TrustZone-M compliant memory protection controller between the secure and non-secure world
- Makes the memory accessible and visible either in its secure address alias or its non-secure address alias
- Divides the memory space into fixed-sized blocks/pages, with SRAM blocks being 2 KB and Flash blocks being 2 KB
- Configures the NS/R/W attributes of each block for all PCs
- Blocks any non-secure access targeting the secure region
- Blocks any secure access targeting the non-secure region
- Offers configurable error responses:
  - MPCx_CFG.RESPONSE=1'b0: hardware performs Read Zero Write Ignore (RZWI)
  - MPCx_CFG.RESPONSE=1'b1: hardware provides bus error

The MPC comprises MMIO registers that program the LUT_SRAM. The LUT_SRAM contains a look-up-table of the security attributes and is connected to an attribution check logic block, which allows or blocks the transaction.

Figure 54 shows the architecture of the MPC.

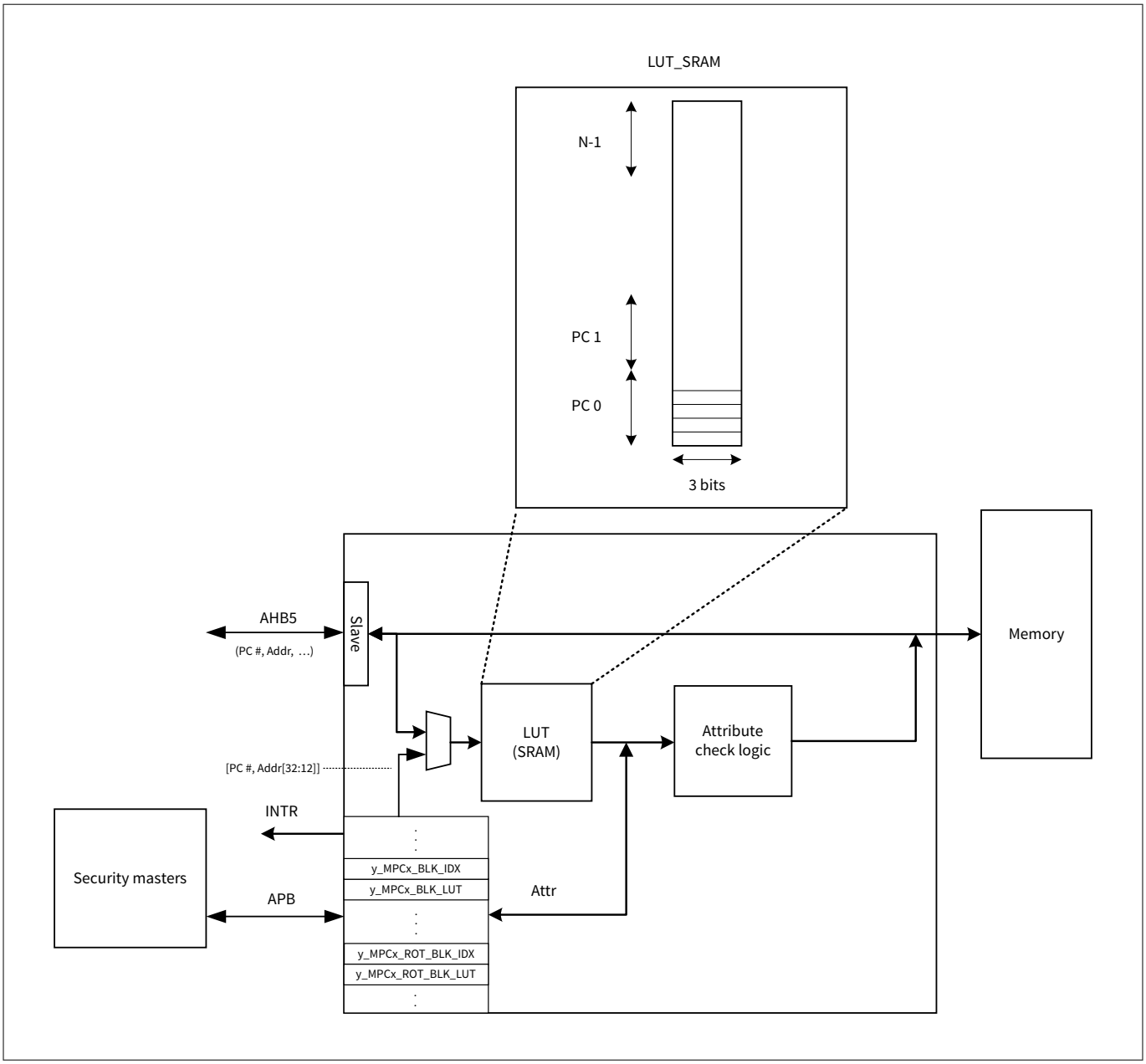


Figure 54 MPC architecture

***Note**: The Secure/Non-secure (NS) attribute distinguishing a 'secure' access from a 'non-secure' access, is a reflection of the resultant security attribute of the location being accessed. For a data access, if a CPU in the S state makes an access to an NS location, the access is deemed as NS. Conversely, if the transaction is made to an S location, the access is deemed as S. Similarly, a CPU in the NS state making a data access to an NS location is deemed as NS. Likewise, for an instruction access, the Secure/Non-secure (NS) access attribute represents the security attribute of the memory location being accessed.*

14.4.1.1 Software view

The MPC provides two access control register sets. The RoT register set configures Read, Write, and Non-secure (NS) attributes for all regions and protection contexts. The other register set is limited to configuring the NS

attribute of the current protection context (R/W attributes cannot be modified). Figure 56 shows the MPC register structure.

The MPC utilizes a lookup table implemented in SRAM (SRAM_LUT) that contains 3-bit NS/R/W attributes per memory block for all PCs.

- For each PC, NSx1, Rx1, and Wx1 attribute bits are reserved per memory block and stored in the LUT (LUT_SRAM). The total number of LUT_SRAM bits is 3 * PC_NR * the number of blocks for a memory module. PC_NR is the number of protection contexts, which is 8 for PSC3M/P8, PSC3M/P7
- Both register sets update the same SRAM_LUT. Figure 55 shows the software view of the MPC
- On reset, the hardware state machine initializes the LUT_SRAM into secure memory with R/W access for all PCs
- For Flash MPC, the boot firmware configures the R/W/NS attributes for each PC's isolation (MPC RoT Configuration). For Flash MPC, the OEM application can only modify the NS permission for the current protection context
- The R/W/NS permissions of all protection context (MPC RoT Configuration) for SRAM can be updated by a secure OEM application
- Refer to PSOC™ Control C3M/P8, C3M/P7 Register Reference Manual for MPC register definition and programming model

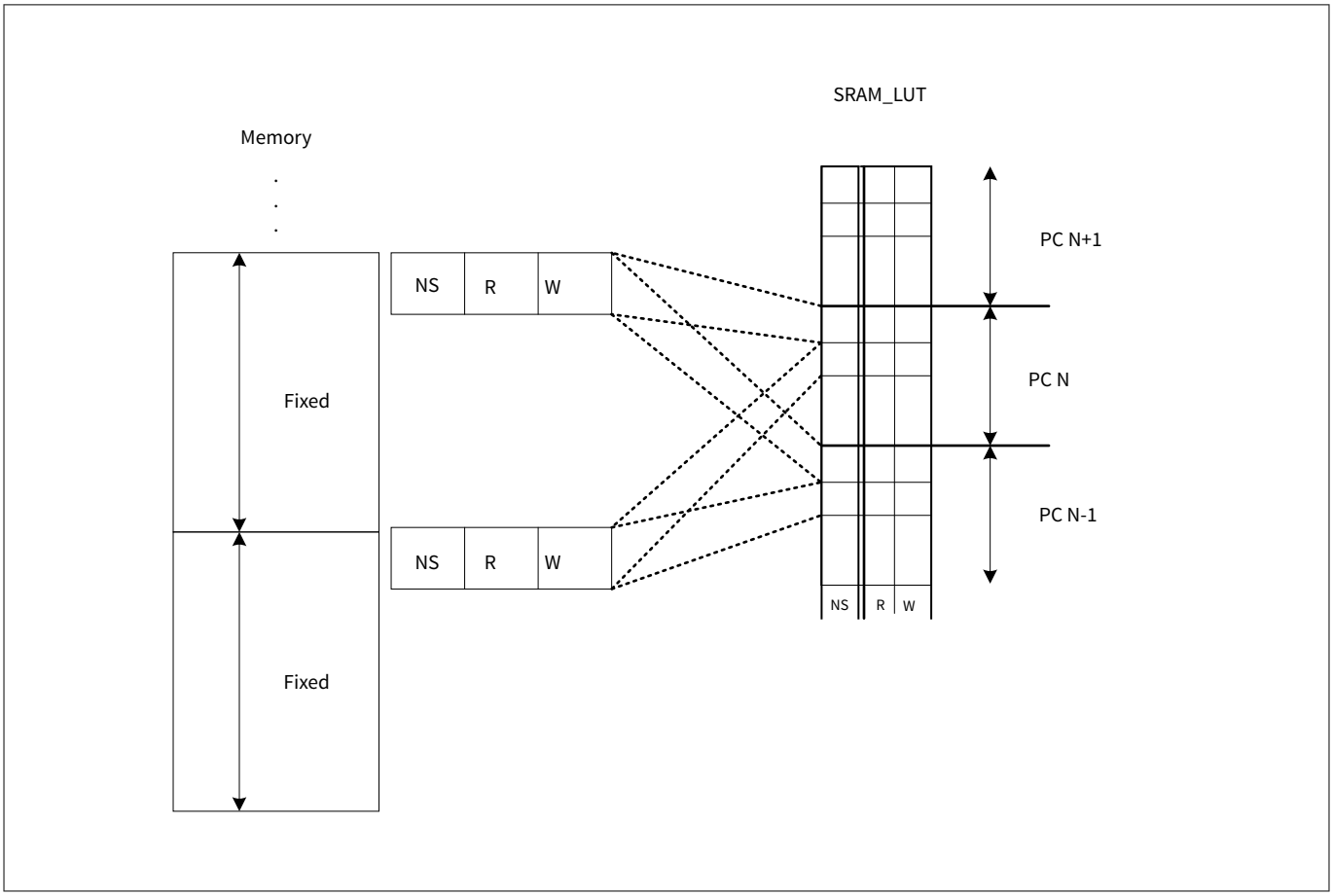


Figure 55 MPC software view

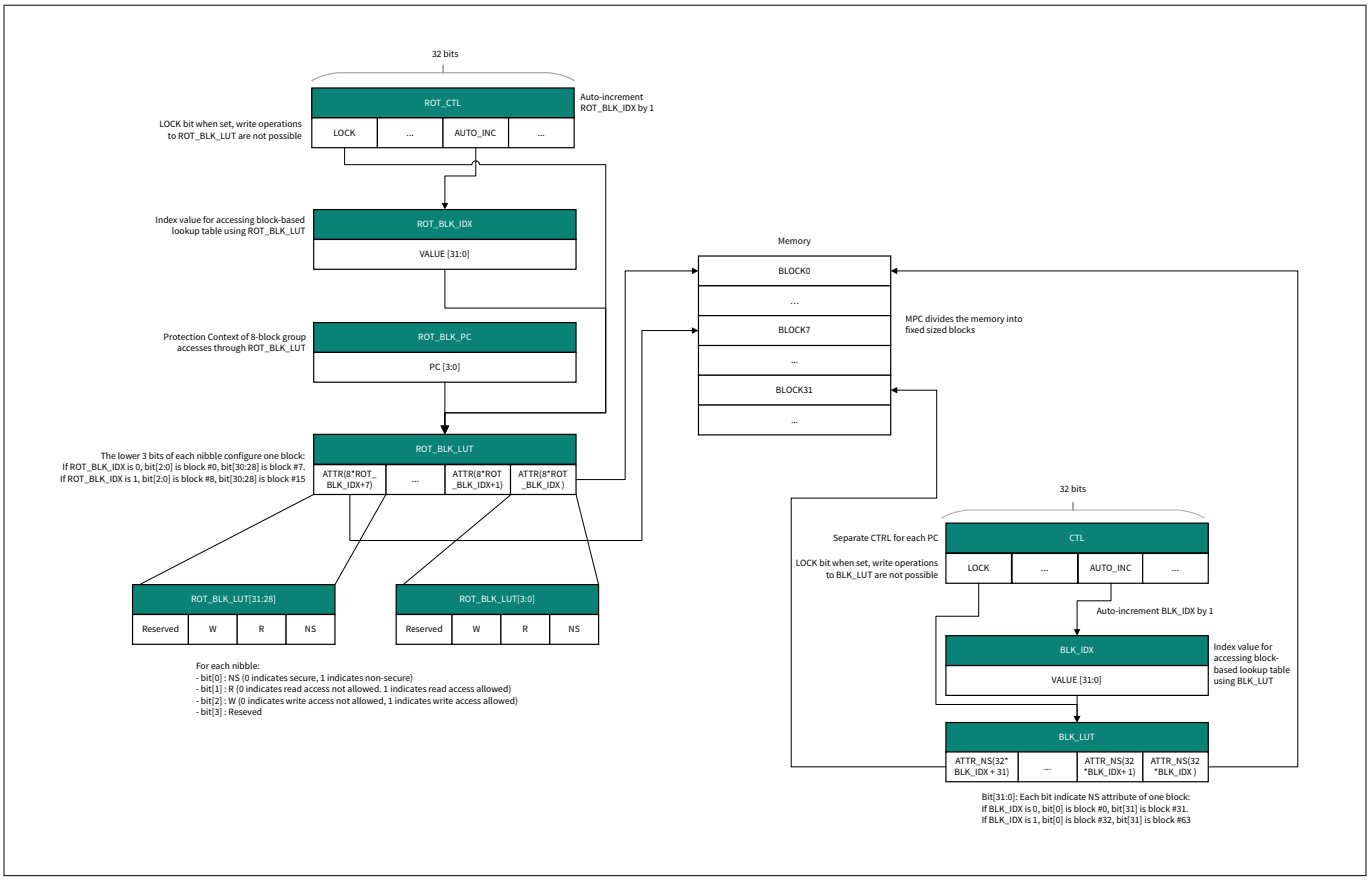


Figure 56 MPC MMIO registers

14.4.1.2 MPC MMIO register protection

The MPC MMIO registers are protected by the PPC, allowing only trusted software to make modifications to the MPC configuration. The x_PPC_MPC_MAIN, x_PPC_MPC_PC, and x_PPC_MPC_ROT PPC regions control access to the MPC MMIO registers. In the PSC3M/P8, PSC3M/P7 platform, the PPC region for the MPC is configured by ROM such that modification of the flash MPC RoT configuration is restricted to ROM code executing in PC0, while the other configurations can be modified by the secure OEM application..

14.4.2 Peripheral Protection Controller (PPC)

The PPC provides peripheral access control based on Arm® TrustZone-M secure or non-secure (NS), privileged or unprivileged (P), and protection context (PC) attributes. Situated between bus masters and peripherals, the PPC groups the MMIO registers of a peripheral that typically require the same security permissions into ‘regions.’ For example, the registers of SCB0 are grouped together in one region. Each PPC region can be configured with PC/NS/P attributes, and a peripheral can have multiple PPC regions depending on its access control needs. The PPC checks the security attribute of a peripheral region against the transaction attribute from the masters for a security violation. If the transaction has a security violation, it is then blocked.

The PPC provides the following functionality:

- TrustZone-compatible peripheral protection with AHB5 bus interface
- Blocking non-secure transfers to secure regions and secure transfers to non-secure regions are considered as violations

- Blocking non-privileged transfers if the peripheral region is configured as privileged
- Configurable error responses:
  - Read Zero Write Ignore (RZWI)
  - Bus fault

14.4.3 PPC access control rules

The PPC configuration is primarily controlled by the following registers:

Table 88 PPC registers

Register name	Function
LOCK_MASK	This register provides a lock mechanism for each PC.
PC_MASK	This register indicates the PCs that have access to the peripheral region, with one PC_MASK for each peripheral region.
NS_ATT	This register indicates whether a peripheral region is non-secure or secure. Each bit corresponds to a peripheral region.
S_P_ATT	This register indicates the secure privileged or unprivileged status for a peripheral region, with each bit corresponding to a peripheral region.
NS_P_ATT	This register indicates the non-secure privileged or unprivileged status for a peripheral region, with each bit corresponding to a peripheral region.

The following convention is used in the subsequent section to describe the PPC restrictions, where "i" stands for a PC value and "j" stands for a peripheral region.

- After a reset or power cycle, PC_MASK bits are by default set to high, S_P_ATT and NS_ATT bits set to low, meaning secure privileged access is provided to all PCs
- If PC_MASK[j].PC_MASK[i]=0x0 and if LOCK_MASK[i]=0, PC "i" can have Read and Write access to PC_MASK[j], NS_ATT[j], NS_P_ATT[j], S_P_ATT[j]
- LOCK_MASK[i] can only be written from the corresponding PC "i". The LOCK_MASK bits that are accessed by PC "i" and are not associated with PC "i" return RZWI
- PC "i" has Read or Write access to the peripheral region "j" if PC_MASK[j].PC_MASK[i] = 1’b1
- If (LOCK_MASK[i] & PC_MASK[j].PC_MASK[i]) == 1’b1:
  - The peripheral region "j" security attributes cannot be changed (with exception on the NS_P_ATT registers). This means that PC "i" is “locked” as its corresponding LOCK_MASK bit is set to 1’b1
  - PC "i" that is 'locked' cannot write to any PC_MASK
  - PC "i" that is 'locked' cannot be given additional access to any peripheral region nor be removed from the peripheral region
  - Additionally, for any PC "k", if PC_MASK[j].PC_MASK[k] is set to 1’b1, PC "k" can only clear the PC_MASK that is not 'locked', that is, remove access from other PCs to the peripheral region "j". PC "k" cannot provide additional access to other PCs as PC "i" has a 'locked' peripheral region "j"
- If LOCK_MASK[i]==1’b1 and if PC_MASK[j].PC_MASK[i]=0x0:
  - PC "i" is subjected to the 'locked' condition specified in the requirements above and does not have Read or Write access to the peripheral region "j". PC "i" cannot change the security attributes for the peripheral region "j" but PC "i" has read access to the security attributes for the peripheral region "j"

Figure 57 shows the PPC MMIO register structure.

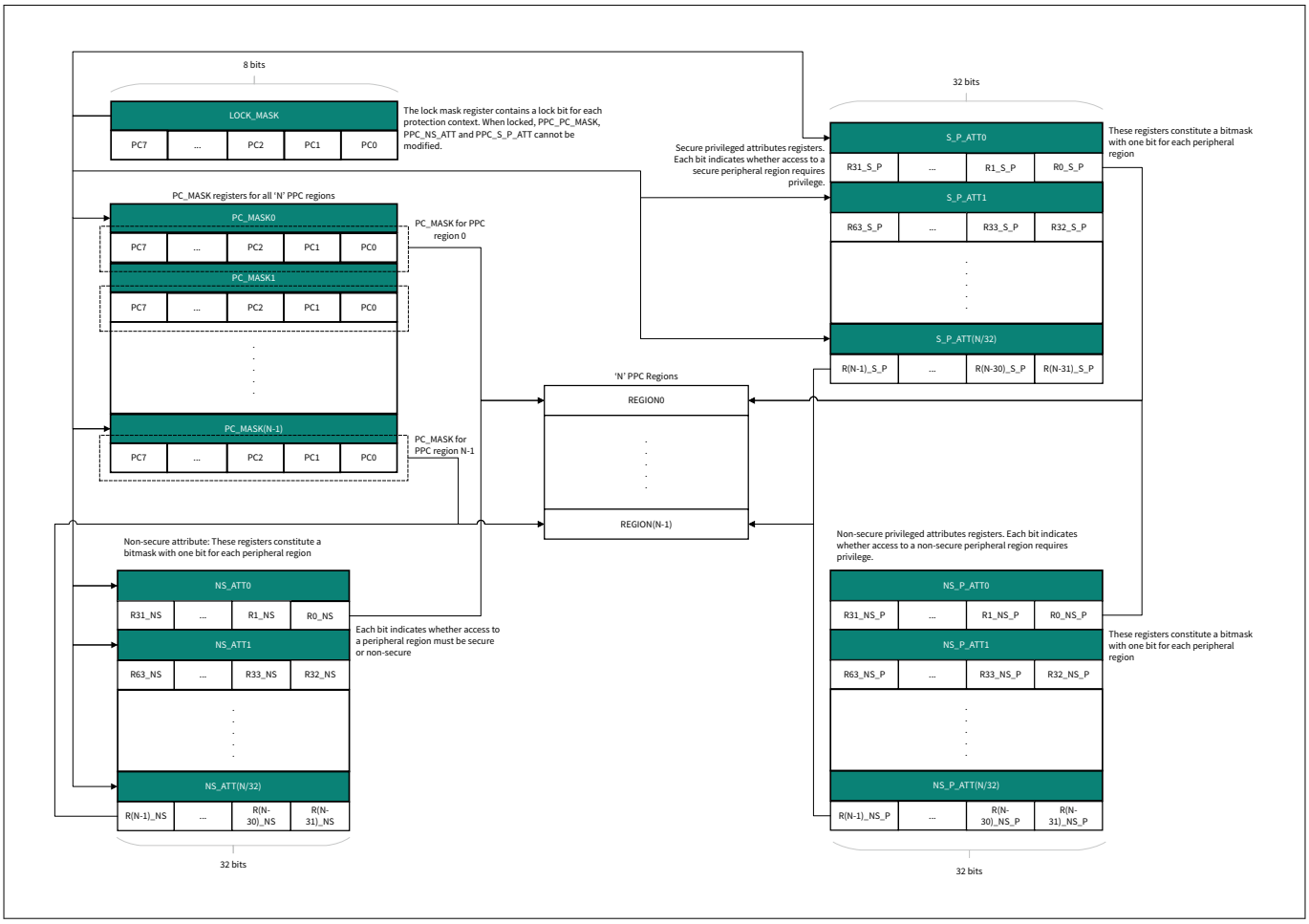


Figure 57 PPC MMIO structure

The ROM boot code configures and locks some of the PPC regions, which are shared with PC0. See Table 89 for a list of regions shared with PC0.

For the PPC register definition and programming model, see the PSOC™ Control C3M/P8, C3M/P7 register reference manual.

14.4.3.1 Security aware peripherals

In PSC3M/P8, PSC3M/P7, peripherals are categorized as security aware or security unaware. The security unaware PPC peripheral region (SECURITY_AWARE = 0) adheres to the following PPC rules:

- Secure transactions can access secure regions only, and non-secure transactions can access non-secure regions only

The security aware PPC peripheral region (SECURITY_AWARE = 1) adheres to the following PPC rules:

- NS = 0: This region allows only secure transactions to access secure resources
- NS = 1: This region allows secure transactions to access both secure and non-secure resources, and non-secure transactions can access only non-secure resources. The expectation is that the peripheral itself can perform the security check, and hence the PPC allows both secure and non-secure transactions to pass through if other attributes match

For a list of security aware peripherals, see Table 89.

14.4.3.2 PPC regions

The PPC not only protects the peripherals but also the MMIO registers of other blocks, such as MPC, peripheral interconnect, SRSS, and so on. There are a total of 297 regions in the PPC. The ROM boot code configures the NS_ATT, S_P_ATT, NS_P_ATT, and PC_MASK registers.

The ROM boot code configures the attributes of all regions of PPC in the following manner:

- NS_ATT = 0 - PPC region is secure
- S_P_ATT= 1 - Privilege and non-privilege access set for all but four regions. S_P_ATT is set to 0 for following four regions: PPC0_PPC_PPC_SECURE, PPC0_PPC_PPC_NONSECURE, CPUSS_AP, MS_PC0_PRIV
- NS_P_ATT = 1 for all regions
- Sets lock bit for PC0. PC2 software is responsible for configurations of the PPC access permissions and attributes for the rest of the PCs

Table 89 shows the list of all PPC0 regions and their PC_MASK configured by the boot code.

Table 89 PPC regions

PPC region number	PPC region name	Start address	Size	Accessible by PCx	Security aware
0	PERI0_MAIN	0x42000000	4000	2, 3, 4, 5, 6, 7	0
1	PERI0_GR0_GROUP	0x42004010	8	2, 3, 4, 5, 6, 7	0
2	PERI0_GR1_GROUP	0x42004040	20	2, 3, 4, 5, 6, 7	0
3	PERI0_GR2_GROUP	0x42004080	20	2, 3, 4, 5, 6, 7	0
4	PERI0_GR3_GROUP	0x420040C0	20	2, 3, 4, 5, 6, 7	0
5	PERI0_GR4_GROUP	0x42004100	20	2, 3, 4, 5, 6, 7	0
6	PERI0_GR5_GROUP	0x42004150	8	2, 3, 4, 5, 6, 7	0
7	PERI0_GR0_BOOT	0x42004020	4	0	0
8	PERI0_GR1_BOOT	0x42004060	4	0	0
9	PERI0_GR2_BOOT	0x420040A0	4	0	0
10	PERI0_GR3_BOOT	0x420040E0	4	0	0
11	PERI0_GR4_BOOT	0x42004120	4	0	0
12	PERI0_GR5_BOOT	0x42004160	4	0	0
13	PERI0_TR	0x42008000	8000	2, 3, 4, 5, 6, 7	0
14	PPC0_PPC_PPC_SECURE	0x42020000	4000	0, 2, 3	0
15	PPC0_PPC_PPC_NONSECURE	0x42024000	4000	0, 2, 3	0
16	PERI_PCLK0_MAIN	0x42040000	10000	2, 3, 4, 5, 6, 7	0
17	CPUSS	0x42100000	10000	2, 3, 4, 5, 6, 7	0
18	RAMC0_CM33	0x42110000	40	2, 3, 4, 5, 6, 7	0
19	RAMC1_CM33	0x42120000	40	2, 3, 4, 5, 6, 7	0
20	RAMC0_BOOT	0x42110100	8	0	0
21	RAMC1_BOOT	0x42120100	8	0	0

(table continues...)

Table 89 (continued) PPC regions

PPC region number	PPC region name	Start address	Size	Accessible by PCx	Security aware
22	RAMC0_RAM_PWR	0x42110200	100	2, 3, 4, 5, 6, 7	0
23	RAMC1_RAM_PWR	0x42120200	100	2, 3, 4, 5, 6, 7	0
24	RAMC0_MPC0_PPC_MPC_MAIN	0x42114000	4	2, 3, 4, 5, 6, 7	0
25	RAMC1_MPC0_PPC_MPC_MAIN	0x42124000	4	2, 3, 4, 5, 6, 7	0
26	RAMC0_MPC0_PPC_MPC_PC	0x42114100	20	2, 3, 4, 5, 6, 7	0
27	RAMC1_MPC0_PPC_MPC_PC	0x42124100	20	2, 3, 4, 5, 6, 7	0
28	RAMC0_MPC0_PPC_MPC_ROT	0x42114200	20	2, 3, 4, 5, 6, 7	0
29	RAMC1_MPC0_PPC_MPC_ROT	0x42124200	20	2, 3, 4, 5, 6, 7	0
30	PROMC_CM33	0x42140000	4	2, 3, 4, 5, 6, 7	0
31	PROMC_MPC0_PPC_MPC_MAIN	0x42141000	4	0	0
32	PROMC_MPC0_PPC_MPC_PC	0x42141100	20	0	0
33	PROMC_MPC0_PPC_MPC_ROT	0x42141200	20	0	0
34	FLASHC_BOOT	0x42150000	8	2, 3, 4, 5, 6	0
35	FLASHC_BOOT1	0x42150100	20	2, 3, 4, 5, 6, 7	0
36	FLASHC_MAIN	0x42150200	10	2, 3, 4, 5, 6	0
37	FLASHC_DFT	0x42150400	80	0	0
38	FLASHC_ECC	0x42150800	10	2, 3, 4, 5, 6	0
39	FLASHC_MPC0_PPC_MPC_MAIN	0x42151000	4	2, 3, 4, 5, 6, 7	0
40	FLASHC_MPC0_PPC_MPC_PC	0x42151100	20	2, 3, 4, 5, 6, 7	0
41	FLASHC_MPC0_PPC_MPC_ROT	0x42151200	20	0	0
42	FLASHC_FM_CTL_FM_DFT	0x42152000	4	0	0
43	FLASHC_FM_CTL_FM_BOOT	0x42152040	8	0	0
44	FLASHC_FM_CTL_FM_MAIN	0x42152800	800	2, 3, 4, 5, 6	0

(table continues...)

Table 89 (continued) PPC regions

PPC region number	PPC region name	Start address	Size	Accessible by PCx	Security aware
45	MXCM33_CM33	0x42160000	100	2, 3, 4, 5, 6, 7	0
46	MXCM33_CM33_S	0x42161000	4	2, 3, 4, 5, 6, 7	0
47	MXCM33_CM33_NS	0x42161004	4	2, 3, 4, 5, 6, 7	0
48	MXCM33_BOOT_PC0	0x42162000	80	0	0
49	MXCM33_BOOT_PC1	0x42162100	4	0	0
50	MXCM33_BOOT_PC2	0x42162140	4	0	0
51	MXCM33_BOOT_PC3	0x42162180	4	0	0
52	MXCM33_BOOT	0x421621C0	4	0	0
53	MXCM33_CM33_INT	0x42168000	400	2, 3, 4, 5, 6, 7	0
54	DW0_DW	0x42180000	80	2, 3, 4, 5, 6, 7	0
55	DW0_DW_CRC	0x42180100	80	2, 3, 4, 5, 6, 7	0
56	DW0_CH_STRUCT0_CH	0x42188000	40	2, 3, 4, 5, 6, 7	0
57	DW0_CH_STRUCT1_CH	0x42188040	40	2, 3, 4, 5, 6, 7	0
58	DW0_CH_STRUCT2_CH	0x42188080	40	2, 3, 4, 5, 6, 7	0
59	DW0_CH_STRUCT3_CH	0x421880C0	40	2, 3, 4, 5, 6, 7	0
60	DW0_CH_STRUCT4_CH	0x42188100	40	2, 3, 4, 5, 6, 7	0
61	DW0_CH_STRUCT5_CH	0x42188140	40	2, 3, 4, 5, 6, 7	0
62	DW0_CH_STRUCT6_CH	0x42188180	40	2, 3, 4, 5, 6, 7	0
63	DW0_CH_STRUCT7_CH	0x421881C0	40	2, 3, 4, 5, 6, 7	0
64	DW0_CH_STRUCT8_CH	0x42188200	40	2, 3, 4, 5, 6, 7	0
65	DW0_CH_STRUCT9_CH	0x42188240	40	2, 3, 4, 5, 6, 7	0
66	DW0_CH_STRUCT10_CH	0x42188280	40	2, 3, 4, 5, 6, 7	0
67	DW0_CH_STRUCT11_CH	0x421882C0	40	2, 3, 4, 5, 6, 7	0
68	DW0_CH_STRUCT12_CH	0x42188300	40	2, 3, 4, 5, 6, 7	0
69	DW0_CH_STRUCT13_CH	0x42188340	40	2, 3, 4, 5, 6, 7	0
70	DW0_CH_STRUCT14_CH	0x42188380	40	2, 3, 4, 5, 6, 7	0
71	DW0_CH_STRUCT15_CH	0x421883C0	40	2, 3, 4, 5, 6, 7	0
72	CPUSS_ALL_PC	0x421C0000	80	2, 3, 4, 5, 6, 7	0
73	CPUSS_DDFT	0x421C0080	4	0	0
74	CPUSS_CM33_S	0x421C0100	4	2, 3, 4, 5, 6, 7	0
75	CPUSS_CM33_NS	0x421C0120	4	2, 3, 4, 5, 6, 7	0
76	CPUSS_MSC_INT	0x421C0200	10	2, 3, 4, 5, 6, 7	0
77	CPUSS_AP	0x421C1000	4	0, 2, 3	0

(table continues...)

Table 89 (continued) PPC regions

PPC region number	PPC region name	Start address	Size	Accessible by PCx	Security aware
78	CPUSS_BOOT	0x421C2000	8	0	0
79	MS0_MAIN	0x421C4000	4	0	0
80	MS4_MAIN	0x421C4040	4	0	0
81	MS7_MAIN	0x421C4070	4	0	0
82	MS9_MAIN	0x421C4090	4	0	0
83	MS31_MAIN	0x421C41F0	4	0	0
84	MS_PC0_PRIV	0x421C5000	4	0, 2, 3	0
85	MS_PC9_PRIV	0x421C5090	4	2, 3	0
86	MS_PC31_PRIV	0x421C51F0	4	2, 3	0
87	MS_PC0_PRIV_MIR	0x421C5004	4	2, 3, 4, 5, 6, 7	0
88	MS_PC9_PRIV_MIR	0x421C5094	4	2, 3, 4, 5, 6, 7	0
89	MS_PC31_PRIV_MIR	0x421C51F4	4	2, 3, 4, 5, 6, 7	0
90	MSC_ACG	0x421C6000	40	2, 3, 4, 5, 6, 7	0
91	CPUSS_SL_CTL_GROUP	0x421C8000	8	2, 3, 4, 5, 6, 7	0
92	IPC_STRUCT0_IPC	0x421D0000	20	2, 3, 4, 5, 6, 7	1
93	IPC_STRUCT1_IPC	0x421D0020	20	2, 3, 4, 5, 6, 7	1
94	IPC_STRUCT2_IPC	0x421D0040	20	2, 3, 4, 5, 6, 7	1
95	IPC_STRUCT3_IPC	0x421D0060	20	2, 3, 4, 5, 6, 7	1
96	IPC_STRUCT4_IPC	0x421D0080	20	2, 3, 4, 5, 6, 7	1
97	IPC_STRUCT5_IPC	0x421D00A0	20	2, 3, 4, 5, 6, 7	1
98	IPC_STRUCT6_IPC	0x421D00C0	20	2, 3, 4, 5, 6, 7	1
99	IPC_STRUCT7_IPC	0x421D00E0	20	2, 3, 4, 5, 6, 7	1
100	IPC_INTR_STRUCT0_INTR	0x421D1000	10	2, 3, 4, 5, 6, 7	1
101	IPC_INTR_STRUCT1_INTR	0x421D1020	10	2, 3, 4, 5, 6, 7	1
102	IPC_INTR_STRUCT2_INTR	0x421D1040	10	2, 3, 4, 5, 6, 7	1
103	IPC_INTR_STRUCT3_INTR	0x421D1060	10	2, 3, 4, 5, 6, 7	1
104	FAULT_STRUCT0_MAIN	0x421E0000	100	2, 3, 4, 5, 6, 7	0
105	SRSS_GENERAL	0x42200000	400	2, 3, 4, 5, 6, 7	0
106	SRSS_GENERAL2	0x42200400	40	2, 3, 4, 5, 6, 7	0
107	SRSS_HIB_DATA	0x422008A0	10	2, 3, 4, 5, 6, 7	0
108	SRSS_MAIN	0x42201000	1000	2, 3, 4, 5, 6, 7	0
109	SRSS_SECURE	0x42202000	2000	2, 3, 4, 5, 6, 7	0
110	RAM_TRIM_SRSS_SRAM	0x42204000	8	2, 3, 4, 5, 6, 7	0

(table continues...)

Table 89 (continued) PPC regions

PPC region number	PPC region name	Start address	Size	Accessible by PCx	Security aware
111	SRSS_DPLL	0x42204200	40	2, 3, 4, 5, 6, 7	0
112	SRSS_WDT	0x4220C000	10	2, 3, 4, 5, 6, 7	0
113	MAIN	0x4220D000	40	2, 3, 4, 5, 6, 7	0
114	PWRMODE_PWRMODE	0x42210000	4000	2, 3, 4, 5, 6, 7	0
115	BACKUP_BACKUP	0x42220000	100	2, 3, 4, 5, 6, 7	0
116	BACKUP_B_BREG0	0x42221000	10	2, 3, 4, 5, 6, 7	0
117	BACKUP_B_BREG1	0x42221010	10	2, 3, 4, 5, 6, 7	0
118	BACKUP_B_BREG2	0x42221020	20	2, 3, 4, 5, 6, 7	0
119	BACKUP_B_BREG3	0x42221080	40	2, 3, 4, 5, 6, 7	0
120	BACKUP_BACKUP_SECURE	0x4222FF00	4	0	0
121	CRYPTOLITE_MAIN	0x42230000	100	2, 3, 4, 5, 6, 7	0
122	CRYPTOLITE_TRNG	0x42230100	100	2, 3, 4, 5, 6, 7	0
123	MXWOUND_MAIN	0x42240000	4	0	0
124	MXWOUND_MXWOUND_PROP0_PROP	0x42240100	4	0	0
125	DEBUG600_DEBUG600	0x42250000	4	2, 3, 4, 5, 6, 7	0
126	SVGS_SVGS_MAIN	0x42260000	40	2, 3, 4, 5, 6, 7	0
127	SVGS_SVGS_AUX	0x42260040	40	0	0
128	MXAES	0x42270000	2000	2, 3, 4, 5, 6, 7	0
129	HSIOM_PRT0_PRT	0x42400000	8	2, 3, 4, 5, 6, 7	1
130	HSIOM_PRT1_PRT	0x42400010	8	2, 3, 4, 5, 6, 7	1
131	HSIOM_PRT2_PRT	0x42400020	8	2, 3, 4, 5, 6, 7	1
132	HSIOM_PRT3_PRT	0x42400030	8	2, 3, 4, 5, 6, 7	1
133	HSIOM_PRT4_PRT	0x42400040	8	2, 3, 4, 5, 6, 7	1
134	HSIOM_PRT5_PRT	0x42400050	8	2, 3, 4, 5, 6, 7	1
135	HSIOM_PRT6_PRT	0x42400060	8	2, 3, 4, 5, 6, 7	1
136	HSIOM_PRT7_PRT	0x42400070	8	2, 3, 4, 5, 6, 7	1
137	HSIOM_PRT8_PRT	0x42400080	8	2, 3, 4, 5, 6, 7	1
138	HSIOM_PRT9_PRT	0x42400090	8	2, 3, 4, 5, 6, 7	1
139	HSIOM_SECURE_PRT0_SECURE_PRT	0x42401000	4	2, 3, 4, 5, 6, 7	1
140	HSIOM_SECURE_PRT1_SECURE_PRT	0x42401010	4	2, 3, 4, 5, 6, 7	1

(table continues...)

Table 89 (continued) PPC regions

PPC region number	PPC region name	Start address	Size	Accessible by PCx	Security aware
141	HSIOM_SECURE_PRT2_SECURE_PRT	0x42401020	4	2, 3, 4, 5, 6, 7	1
142	HSIOM_SECURE_PRT3_SECURE_PRT	0x42401030	4	2, 3, 4, 5, 6, 7	1
143	HSIOM_SECURE_PRT4_SECURE_PRT	0x42401040	4	2, 3, 4, 5, 6, 7	1
144	HSIOM_SECURE_PRT5_SECURE_PRT	0x42401050	4	2, 3, 4, 5, 6, 7	1
145	HSIOM_SECURE_PRT6_SECURE_PRT	0x42401060	4	2, 3, 4, 5, 6, 7	1
146	HSIOM_SECURE_PRT7_SECURE_PRT	0x42401070	4	2, 3, 4, 5, 6, 7	1
147	HSIOM_SECURE_PRT8_SECURE_PRT	0x42401080	4	2, 3, 4, 5, 6, 7	1
148	HSIOM_SECURE_PRT9_SECURE_PRT	0x42401090	4	2, 3, 4, 5, 6, 7	1
149	HSIOM_AMUX	0x42402000	20	2, 3, 4, 5, 6, 7	1
150	HSIOM_MON	0x42402200	10	2, 3, 4, 5, 6, 7	1
151	GPIO_PRT0_PRT	0x42410000	40	2, 3, 4, 5, 6, 7	1
152	GPIO_PRT1_PRT	0x42410080	40	2, 3, 4, 5, 6, 7	1
153	GPIO_PRT2_PRT	0x42410100	40	2, 3, 4, 5, 6, 7	1
154	GPIO_PRT3_PRT	0x42410180	40	2, 3, 4, 5, 6, 7	1
155	GPIO_PRT4_PRT	0x42410200	40	2, 3, 4, 5, 6, 7	1
156	GPIO_PRT5_PRT	0x42410280	40	2, 3, 4, 5, 6, 7	1
157	GPIO_PRT6_PRT	0x42410300	40	2, 3, 4, 5, 6, 7	1
158	GPIO_PRT7_PRT	0x42410380	40	2, 3, 4, 5, 6, 7	1
159	GPIO_PRT8_PRT	0x42410400	40	2, 3, 4, 5, 6, 7	1
160	GPIO_PRT9_PRT	0x42410480	40	2, 3, 4, 5, 6, 7	1
161	GPIO_PRT0_CFG	0x42410040	40	2, 3, 4, 5, 6, 7	1
162	GPIO_PRT1_CFG	0x424100C0	40	2, 3, 4, 5, 6, 7	1
163	GPIO_PRT2_CFG	0x42410140	40	2, 3, 4, 5, 6, 7	1
164	GPIO_PRT3_CFG	0x424101C0	40	2, 3, 4, 5, 6, 7	1
165	GPIO_PRT4_CFG	0x42410240	40	2, 3, 4, 5, 6, 7	1
166	GPIO_PRT5_CFG	0x424102C0	40	2, 3, 4, 5, 6, 7	1
167	GPIO_PRT6_CFG	0x42410340	40	2, 3, 4, 5, 6, 7	1

(table continues...)

Table 89 (continued) PPC regions

PPC region number	PPC region name	Start address	Size	Accessible by PCx	Security aware
168	GPIO_PRT7_CFG	0x424103C0	40	2, 3, 4, 5, 6, 7	1
169	GPIO_PRT8_CFG	0x42410440	40	2, 3, 4, 5, 6, 7	1
170	GPIO_PRT9_CFG	0x424104C0	40	2, 3, 4, 5, 6, 7	1
171	GPIO_SEC_GPIO	0x42417000	4	2, 3, 4, 5, 6, 7	1
172	GPIO_GPIO	0x42418000	40	2, 3, 4, 5, 6, 7	1
173	GPIO_TEST	0x42419000	8	2, 3, 4, 5, 6, 7	1
174	LPCOMP	0x42430000	10000	2, 3, 4, 5, 6, 7	0
175	TCPWM0_GRP0_CNT0_CNT	0x42500000	100	2, 3, 4, 5, 6, 7	0
176	TCPWM0_GRP0_CNT1_CNT	0x42500100	100	2, 3, 4, 5, 6, 7	0
177	TCPWM0_GRP1_CNT0_CNT	0x42510000	100	2, 3, 4, 5, 6, 7	0
178	TCPWM0_GRP1_CNT1_CNT	0x42510100	100	2, 3, 4, 5, 6, 7	0
179	TCPWM0_TR_ALL_SYNC_BYPASS_TR_ALL_SYN_BYPASS	0x42590000	4	2, 3, 4, 5, 6, 7	0
180	DFT	0x42600000	1000	0	0
181	EFUSE_CTL1	0x42610000	4	2, 3, 4, 5, 6, 7	0
182	EFUSE_CTL2	0x42610100	80	0	0
183	EFUSE_CTL3	0x42610180	4	2, 3, 4, 5, 6, 7	0
184	EFUSE_DATA_BOOT1	0x42610800	80	2, 3, 4, 5, 6, 7	0
185	CANFD0_CH0_CH	0x42800000	200	2, 3, 4, 5, 6, 7	0
186	CANFD0_CH1_CH	0x42800200	200	2, 3, 4, 5, 6, 7	0
187	CANFD0_MAIN	0x42801000	40	2, 3, 4, 5, 6, 7	0
188	CANFD0_BUF	0x42810000	10000	2, 3, 4, 5, 6, 7	0
189	SCB0	0x42820000	10000	2, 3, 4, 5, 6, 7	0
190	SCB1	0x42840000	10000	2, 3, 4, 5, 6, 7	0
191	SCB2	0x42850000	10000	2, 3, 4, 5, 6, 7	0
192	SCB3	0x42860000	10000	2, 3, 4, 5, 6, 7	0
193	SCB4	0x42870000	10000	2, 3, 4, 5, 6, 7	0
194	SCB5	0x43800000	10000	2, 3, 4, 5, 6, 7	0
195	SCB6	0x43810000	10000	2, 3, 4, 5, 6, 7	0

(table continues...)

Table 89 (continued) PPC regions

PPC region number	PPC region name	Start address	Size	Accessible by PCx	Security aware
196	PPSS_PPSS_CNFG_PPSSCNFG	0x43000000	4	2, 3, 4, 5, 6, 7	0
197	PPSS_PPSS_CNFG_CNFG_CNFG	0x43000100	100	2, 3, 4, 5, 6, 7	0
198	PPSS_PPSS_CNFG_IMC	0x43000300	80	2, 3, 4, 5, 6, 7	0
199	PPSS_PPSS_CNFG_ADCM	0x43000400	100	2, 3, 4, 5, 6, 7	0
200	PPSS_PPSS_CNFG_DISPERI_BOOT	0x43000800	4	2, 3, 4, 5, 6, 7	0
201	PPSS_EPU	0x43004000	1000	2, 3, 4, 5, 6, 7	0
202	PPSS_RAM0	0x43010000	10000	2, 3, 4, 5, 6, 7	0
203	PPSS_RAM1	0x43020000	10000	2, 3, 4, 5, 6, 7	0
204	PPSS_RAM2	0x43030000	10000	2, 3, 4, 5, 6, 7	0
205	PPSS_RAM3	0x43040000	10000	2, 3, 4, 5, 6, 7	0
206	PPSS_SHMEM	0x43050000	10000	2, 3, 4, 5, 6, 7	0
207	PPSS_PPSS_IPC_STRUCT0_IPC	0x43060000	20	2, 3, 4, 5, 6, 7	1
208	PPSS_PPSS_IPC_STRUCT1_IPC	0x43060020	20	2, 3, 4, 5, 6, 7	1
209	PPSS_PPSS_IPC_STRUCT2_IPC	0x43060040	20	2, 3, 4, 5, 6, 7	1
210	PPSS_PPSS_IPC_STRUCT3_IPC	0x43060060	20	2, 3, 4, 5, 6, 7	1
211	PPSS_PPSS_IPC_INTR_STRUCT0_INTR	0x43061000	10	2, 3, 4, 5, 6, 7	1
212	PPSS_PPSS_IPC_INTR_STRUCT1_INTR	0x43061020	10	2, 3, 4, 5, 6, 7	1
213	PPSS_PPSS_IPC_INTR_STRUCT2_INTR	0x43061040	10	2, 3, 4, 5, 6, 7	1
214	PPSS_PPSS_IPC_INTR_STRUCT3_INTR	0x43061060	10	2, 3, 4, 5, 6, 7	1
215	PPSS_CPUSS_CNFG_PPSS_MXCM330_CM33	0x43080000	100	2, 3, 4, 5, 6, 7	0
216	PPSS_CPUSS_CNFG_PPSS_MXCM331_CM33	0x43090000	100	2, 3, 4, 5, 6, 7	0
217	PPSS_CPUSS_CNFG_PPSS_MXCM330_CM33_NS	0x43081004	4	2, 3, 4, 5, 6, 7	0

(table continues...)

Table 89 (continued) PPC regions

PPC region number	PPC region name	Start address	Size	Accessible by PCx	Security aware
218	PPSS_CPUSS_CNFG_PPSS_MXCM331_CM33_NS	0x43091004	4	2, 3, 4, 5, 6, 7	0
219	PPSS_CPUSS_CNFG_PPSS_MXCM330_BOOT	0x430821C0	4	2, 3, 4, 5, 6, 7	0
220	PPSS_CPUSS_CNFG_PPSS_MXCM331_BOOT	0x430921C0	4	2, 3, 4, 5, 6, 7	0
221	PPSS_CPUSS_CNFG_PPSS_MXCM330_CM33_INT	0x43088000	100	2, 3, 4, 5, 6, 7	0
222	PPSS_CPUSS_CNFG_PPSS_MXCM331_CM33_INT	0x43098000	100	2, 3, 4, 5, 6, 7	0
223	PPSS_CPUSS_CNFG_PPSS_RAMC0_CM33	0x430A0000	40	2, 3, 4, 5, 6, 7	0
224	PPSS_CPUSS_CNFG_PPSS_RAMC1_CM33	0x430B0000	40	2, 3, 4, 5, 6, 7	0
225	PPSS_CPUSS_CNFG_PPSS_RAMC2_CM33	0x430C0000	40	2, 3, 4, 5, 6, 7	0
226	PPSS_CPUSS_CNFG_PPSS_RAMC3_CM33	0x430D0000	40	2, 3, 4, 5, 6, 7	0
227	PPSS_CPUSS_CNFG_PPSS_RAMC4_CM33	0x430E0000	40	2, 3, 4, 5, 6, 7	0
228	PPSS_CPUSS_CNFG_PPSS_RAMC0_BOOT	0x430A0100	8	2, 3, 4, 5, 6, 7	0
229	PPSS_CPUSS_CNFG_PPSS_RAMC1_BOOT	0x430B0100	8	2, 3, 4, 5, 6, 7	0
230	PPSS_CPUSS_CNFG_PPSS_RAMC2_BOOT	0x430C0100	8	2, 3, 4, 5, 6, 7	0
231	PPSS_CPUSS_CNFG_PPSS_RAMC3_BOOT	0x430D0100	8	2, 3, 4, 5, 6, 7	0
232	PPSS_CPUSS_CNFG_PPSS_RAMC4_BOOT	0x430E0100	8	2, 3, 4, 5, 6, 7	0
233	PPSS_CPUSS_CNFG_PPSS_RAMC0_RAM_PWR	0x430A0200	100	2, 3, 4, 5, 6, 7	0
234	PPSS_CPUSS_CNFG_PPSS_RAMC1_RAM_PWR	0x430B0200	100	2, 3, 4, 5, 6, 7	0
235	PPSS_CPUSS_CNFG_PPSS_RAMC2_RAM_PWR	0x430C0200	100	2, 3, 4, 5, 6, 7	0
236	PPSS_CPUSS_CNFG_PPSS_RAMC3_RAM_PWR	0x430D0200	100	2, 3, 4, 5, 6, 7	0

(table continues...)

Table 89 (continued) PPC regions

PPC region number	PPC region name	Start address	Size	Accessible by PCx	Security aware
237	PPSS_CPUSS_CNFG_PPSS_RAMC4_RAM_PWR	0x430E0200	100	2, 3, 4, 5, 6, 7	0
238	PPSS_CPUSS_CNFG_PPSS_RAMC0_PPSS_MPC0_PPC_MPC_MAIN	0x430A4000	4	2, 3, 4, 5, 6, 7	0
239	PPSS_CPUSS_CNFG_PPSS_RAMC1_PPSS_MPC0_PPC_MPC_MAIN	0x430B4000	4	2, 3, 4, 5, 6, 7	0
240	PPSS_CPUSS_CNFG_PPSS_RAMC2_PPSS_MPC0_PPC_MPC_MAIN	0x430C4000	4	2, 3, 4, 5, 6, 7	0
241	PPSS_CPUSS_CNFG_PPSS_RAMC3_PPSS_MPC0_PPC_MPC_MAIN	0x430D4000	4	2, 3, 4, 5, 6, 7	0
242	PPSS_CPUSS_CNFG_PPSS_RAMC4_PPSS_MPC0_PPC_MPC_MAIN	0x430E4000	4	2, 3, 4, 5, 6, 7	0
243	PPSS_CPUSS_CNFG_PPSS_RAMC0_PPSS_MPC0_PPC_MPC_PC	0x430A4100	20	2, 3, 4, 5, 6, 7	0
244	PPSS_CPUSS_CNFG_PPSS_RAMC1_PPSS_MPC0_PPC_MPC_PC	0x430B4100	20	2, 3, 4, 5, 6, 7	0
245	PPSS_CPUSS_CNFG_PPSS_RAMC2_PPSS_MPC0_PPC_MPC_PC	0x430C4100	20	2, 3, 4, 5, 6, 7	0
246	PPSS_CPUSS_CNFG_PPSS_RAMC3_PPSS_MPC0_PPC_MPC_PC	0x430D4100	20	2, 3, 4, 5, 6, 7	0
247	PPSS_CPUSS_CNFG_PPSS_RAMC4_PPSS_MPC0_PPC_MPC_PC	0x430E4100	20	2, 3, 4, 5, 6, 7	0
248	PPSS_CPUSS_CNFG_PPSS_RAMC0_PPSS_MPC0_PPC_MPC_ROT	0x430A4200	20	2, 3, 4, 5, 6, 7	0
249	PPSS_CPUSS_CNFG_PPSS_RAMC1_PPSS_MPC0_PPC_MPC_ROT	0x430B4200	20	2, 3, 4, 5, 6, 7	0
250	PPSS_CPUSS_CNFG_PPSS_RAMC2_PPSS_MPC0_PPC_MPC_ROT	0x430C4200	20	2, 3, 4, 5, 6, 7	0

(table continues...)

Table 89 (continued) PPC regions

PPC region number	PPC region name	Start address	Size	Accessible by PCx	Security aware
251	PPSS_CPUSS_CNFG_PPSS_RAMC3_PPSS_MPC0_PPC_MPC_ROT	0x430D4200	20	2, 3, 4, 5, 6, 7	0
252	PPSS_CPUSS_CNFG_PPSS_RAMC4_PPSS_MPC0_PPC_MPC_ROT	0x430E4200	20	2, 3, 4, 5, 6, 7	0
253	PPSS_CPUSS_CNFG_AP	0x430F0000	8	2, 3, 4, 5, 6, 7	0
254	PPSS_CPUSS_CNFG_RAMC_PPU0_PPU	0x430F8000	1000	2, 3, 4, 5, 6, 7	0
255	PPSS_CPUSS_CNFG_RAMC_PPU1_PPU	0x430F9000	1000	2, 3, 4, 5, 6, 7	0
256	PPSS_CPUSS_CNFG_RAMC_PPU2_PPU	0x430FA000	1000	2, 3, 4, 5, 6, 7	0
257	PPSS_CPUSS_CNFG_RAMC_PPU3_PPU	0x430FB000	1000	2, 3, 4, 5, 6, 7	0
258	PPSS_CPUSS_CNFG_RAMC_PPU4_PPU	0x430FC000	1000	2, 3, 4, 5, 6, 7	0
259	PPSS_TCPWM0_TCPWM_TCPWM0_GRP0_TCPWM0_CNT0_CNT	0x43100000	100	2, 3, 4, 5, 6, 7	0
260	PPSS_TCPWM0_TCPWM_TCPWM0_GRP0_TCPWM0_CNT1_CNT	0x43100100	100	2, 3, 4, 5, 6, 7	0
261	PPSS_TCPWM0_TCPWM_TCPWM0_GRP0_TCPWM0_CNT2_CNT	0x43100200	100	2, 3, 4, 5, 6, 7	0
262	PPSS_TCPWM0_TCPWM_TCPWM0_GRP0_TCPWM0_CNT3_CNT	0x43100300	100	2, 3, 4, 5, 6, 7	0
263	PPSS_TCPWM0_TCPWM_TCPWM0_GRP1_TCPWM0_CNT0_CNT	0x43110000	100	2, 3, 4, 5, 6, 7	0
264	PPSS_TCPWM0_TCPWM_TCPWM0_GRP1_TCPWM0_CNT1_CNT	0x43110100	100	2, 3, 4, 5, 6, 7	0
265	PPSS_TCPWM0_TCPWM_TCPWM0_GRP1_TCPWM0_CNT2_CNT	0x43110200	100	2, 3, 4, 5, 6, 7	0

(table continues...)

Table 89 (continued) PPC regions

PPC region number	PPC region name	Start address	Size	Accessible by PCx	Security aware
266	PPSS_TCPWM0_TCPWM_TCPWM0_GRP1_TCPWM0_CNT3_CNT	0x43110300	100	2, 3, 4, 5, 6, 7	0
267	PPSS_TCPWM0_TCPWM_TCPWM0_GRP2_TCPWM0_CNT0_CNT	0x43120000	100	2, 3, 4, 5, 6, 7	0
268	PPSS_TCPWM0_TCPWM_TCPWM0_GRP2_TCPWM0_CNT1_CNT	0x43120100	100	2, 3, 4, 5, 6, 7	0
269	PPSS_TCPWM0_TCPWM_TCPWM0_GRP2_TCPWM0_CNT2_CNT	0x43120200	100	2, 3, 4, 5, 6, 7	0
270	PPSS_TCPWM0_TCPWM_TCPWM0_GRP2_TCPWM0_CNT3_CNT	0x43120300	100	2, 3, 4, 5, 6, 7	0
271	PPSS_TCPWM0_TCPWM_TCPWM0_TR_ALL_SYNC_BYPASS_TR_ALL_SYN_BYPASS	0x43190000	4	2, 3, 4, 5, 6, 7	0
272	PPSS_TCPWM0_TCPWM_TCPWM0_MOTIF_GRP2_TCPWM0_MOTIF0_MOTIF	0x431A8000	200	2, 3, 4, 5, 6, 7	0
273	PPSS_TCPWM1_TCPWM_TCPWM1_GRP0_TCPWM1_CNT0_CNT	0x43200000	100	2, 3, 4, 5, 6, 7	0
274	PPSS_TCPWM1_TCPWM_TCPWM1_GRP0_TCPWM1_CNT1_CNT	0x43200100	100	2, 3, 4, 5, 6, 7	0
275	PPSS_TCPWM1_TCPWM_TCPWM1_GRP1_TCPWM1_CNT0_CNT	0x43210000	100	2, 3, 4, 5, 6, 7	0
276	PPSS_TCPWM1_TCPWM_TCPWM1_GRP1_TCPWM1_CNT1_CNT	0x43210100	100	2, 3, 4, 5, 6, 7	0
277	PPSS_TCPWM1_TCPWM_TCPWM1_GRP2_TCPWM1_CNT0_CNT	0x43220000	100	2, 3, 4, 5, 6, 7	0
278	PPSS_TCPWM1_TCPWM_TCPWM1_GRP2_TCPWM1_CNT1_CNT	0x43220100	100	2, 3, 4, 5, 6, 7	0

(table continues...)

Table 89 (continued) PPC regions

PPC region number	PPC region name	Start address	Size	Accessible by PCx	Security aware
279	PPSS_TCPWM1_TCPWM_TCPWM1_GRP2_TCPWM1_CNT2_CNT	0x43220200	100	2, 3, 4, 5, 6, 7	0
280	PPSS_TCPWM1_TCPWM_TCPWM1_GRP2_TCPWM1_CNT3_CNT	0x43220300	100	2, 3, 4, 5, 6, 7	0
281	PPSS_TCPWM1_TCPWM_TCPWM1_TR_ALL_SYNC_BYPASS_TR_ALL_SYN_BYPASS	0x43290000	4	2, 3, 4, 5, 6, 7	0
282	PPSS_TCPWM1_TCPWM_TCPWM1_MOTIF_GRP2_TCPWM1_MOTIF0_MOTIF	0x432A8000	200	2, 3, 4, 5, 6, 7	0
283	PPSS_TCPWM2_TCPWM_TCPWM2_GRP0_TCPWM2_CNT0_CNT	0x43300000	100	2, 3, 4, 5, 6, 7	0
284	PPSS_TCPWM2_TCPWM_TCPWM2_GRP0_TCPWM2_CNT1_CNT	0x43300100	100	2, 3, 4, 5, 6, 7	0
285	PPSS_TCPWM2_TCPWM_TCPWM2_GRP0_TCPWM2_CNT2_CNT	0x43300200	100	2, 3, 4, 5, 6, 7	0
286	PPSS_TCPWM2_TCPWM_TCPWM2_GRP0_TCPWM2_CNT3_CNT	0x43300300	100	2, 3, 4, 5, 6, 7	0
287	PPSS_TCPWM2_TCPWM_TCPWM2_TR_ALL_SYNC_BYPASS_TR_ALL_SYN_BYPASS	0x43390000	4	2, 3, 4, 5, 6, 7	0
288	PPSS_TCPWM3_TCPWM_TCPWM3_GRP0_TCPWM3_CNT0_CNT	0x43400000	100	2, 3, 4, 5, 6, 7	0
289	PPSS_TCPWM3_TCPWM_TCPWM3_GRP0_TCPWM3_CNT1_CNT	0x43400100	100	2, 3, 4, 5, 6, 7	0
290	PPSS_TCPWM3_TCPWM_TCPWM3_TR_ALL_SYNC_BYPASS_TR_ALL_SYN_BYPASS	0x43490000	4	2, 3, 4, 5, 6, 7	0
291	PPSS_HWFILTSS0	0x43500000	1000	2, 3, 4, 5, 6, 7	0
292	PPSS_HWFILTSS1	0x43510000	1000	2, 3, 4, 5, 6, 7	0

(table continues...)

Table 89 (continued) PPC regions

PPC region number	PPC region name	Start address	Size	Accessible by PCx	Security aware
293	PPSS_ADCM_ADCM	0x43520000	100	2, 3, 4, 5, 6, 7	0
294	PPSS_CLB_CLB	0x43530000	8000	2, 3, 4, 5, 6, 7	0
295	PPSS_CORDIC	0x43540000	20000	2, 3, 4, 5, 6, 7	0
296	PPSS_ATOPSS_ATOPSS	0x43580000	10000	2, 3, 4, 5, 6, 7	0

***Note**: Access to the TCPWM0_MOTIF_GRP1_MOTIF0_MOTIF region is allowed for all PCs if the part supports MOTIF. Refer to the part's datasheet to determine if the part supports MOTIF. If the part does not support MOTIF, access to this region is restricted to PC0.*

15 Secure boot

Secure boot is defined as the process of obtaining, validating, and launching the product firmware. PSC3M/P8, PSC3M/P7 MCU BootROM consists of ROM_BOOT and FLASH_BOOT. After reset, the immutable ROM_BOOT always begins and verifies the integrity of SFLASH memory, which consists of FLASH_BOOT, Infineon and OEM policy assets. After ROM_BOOT transfer control to FLASH_BOOT, the pre-programmed FLASH_BOOT extends the functionality of ROM_BOOT and applies full trims as needed before authenticating the next stage image (OEM application). FLASH_BOOT authenticates and launches the next stage application or edge protect bootloader. The following chapter describes each stage of the secure boot process in detail.

15.1 Basic definitions

The following terms and definitions will help you to understand the terms used in this document. Many of these terms will be discussed in more detail within this section.

- **BootROM**: The PSC3M/P8, PSC3M/P7 consists of two firmware components: ROM_BOOT and FLASH_BOOT. Collectively, these firmware components are referred to as the immutable BootROM
- **Chain of Trust (CoT)**: The root of trust begins with the Infineon code residing in ROM, which cannot be altered. Chain of Trust is established by validating the blocks of software starting from the root of trust (RoT) ROM_BOOT
- **Debug Access Port (DAP)**: Interface between an external debugger/programmer and PSC3M/P8, PSC3M/P7 MCU for programming and debugging.
- **Digest**: A digest is computed by applying a hash function on a data set, such as the binary of an application
- **Digital signature**: Encrypting of the digest (hash of a data set). For example, the encrypted hash of the user application
- **Device integrity exam**: A test used prior to provisioning to verify that this is a genuine device and is in a known state
- **Hash:** A crypto algorithm that generates a repeatable but unique signature for a given block of data. This function is not reversible
- **Hardware security module (HSM)**: This is a physical computing or programming device that protects and manages digital keys, it can perform digital software signing and encryption as well as other cryptographic functions
- **Hardware unique key (HUK):** A 256-bit device root key for deriving all device-specific keys
- **Inter-process communication (IPC):** IPC hardware used to facilitate communication between secure and non-secure memory
- **Life cycle stage (LCS):** LCS is the security mode in which the device is operating
- **Memory protection controller (MPC):** Enables to isolate the parts of memory by means of PCs
- **Master security controller (MSC):** Helps to protect the transactions initiated on the bus
- **Non-volatile counter (NV counter):** Security counter to offer anti-rollback protection
- **OTP**: One-time programmable memory
- **Protection context (PC):** This parameter allows each bus master a security state level from '0' to '7'. A bus master can be assigned a PC value that stays static or that is changed during application execution. PC provides a more precise way of applying memory restrictions. The PC state works together with protection units
- **Public-key cryptography (PKC):** Otherwise known as asymmetrical cryptography. Public-key cryptography is an encryption technique that uses a paired public and private key (or asymmetric key)

algorithm for secure data. It is used to secure a message or block of data. The private key is used to encrypt data and must be kept secure, and the public key is used to decrypt but can be disseminated widely
  - **Public key:** The public key can be shared, but it should be authenticated or secure so it cannot be modified
  - **Private key:** The private key must be kept in a secure location, so it cannot be viewed or stolen. It is used to encrypt a block of data that will be decrypted using an associated public key
- **Policy:** Policies are a collection of predefined (name, value) pairs that describe what is and is not allowed on the device to which the policy applies
- **Peripheral protection controller (PPC):** Allows configuring the peripherals such that they are accessible only in pre-configured PCs assigned by the RoT
- **Provisioning:** The process by which keys, policies and secrets are injected into the device. Once provisioned, the device can be accessed or modified only with the keys injected adhering to the relevant policies
- **Platform security architecture (PSA):** Firmware security specifications and open-source reference implementations, to help building best-practice security into products
- **RMA:** Return merchandise authorization
- **SESIP:** Security Evaluation Standard for IoT (Internet of Things) Platforms
- **SHA-256:** A cryptographic hash algorithm is used to create a digest for a block of data or code. This hash algorithm produces a 256-bit unique digest of the data, no matter the size of the data block

15.2 Features

The PSC3M/P8, PSC3M/P7 MCU provides the following secure boot features:

- Each boot stage verifies the next stage image before launching them; therefore, offers a chain of trust until launches the first user application. Further user can also expand the chain of trust by incorporating their own OEM bootloader or EPB (Edge Protect Bootloader).
- Implements a hardware root of trust with immutable trust anchor with the help of pre-programmed multistage BootROM (consists of ROM_BOOT and FLASH_BOOT firmware)
- Provides a conventional development LCS to get started with and transition to production LCS to productize the device
- Built-in immutable key storage area that can be loaded with required OEM keys and other user keys during provisioning
- Supports dual bank. The device user flash is divided into two equal-sized banks, known as bank 0 and bank 1. This feature is enabled/disabled through provisioning
- Supports protected firmware. This feature enables customer to protect their firmware IP
- Supports secured debug and secured provisioning from Infineon to the users
- Device provisioning and firmware upgrade can be performed through either SWD/JTAG interface or BootROM DFU Serial Interface

15.3 Architecture

15.3.1 Policies and keys

15.3.1.1 Policies

- **IFX_POLICY:** Infineon factory provisions the IFX_POLICY. It consists of pre-programmed Infineon assets and a securely bootable and upgradable piece of pre-programmed software. These are factory-programmed and locked by Infineon
- **PROT_FW_POLICY:** PROT_FW_POLICY consists of PROT_FW_ROT_KEYS and protected firmware attributes. PROT_FW_POLICY is provisioned in development LCS
- **OEM_POLICY:** OEM provisions the OEM_POLICY in development LCS. Policy consists of device life cycle state (LCS), debug policy, boot policy, dfu policy, OEM_ROT_KEYS and other assets. Policy can be changed several times until "prov_oem_complete" flag is set to "true"

15.3.1.2 Keys

The keys that are described in this section are asymmetric ECC 521-bit ECDSA key pairs. Each pair includes both a PRIVATE and a PUBLIC Key.

- **IFX_ROT_KEY:** IFX_ROT_KEY is an asymmetric ECC 521-bit ECDSA key pair used to validate the assets signed and installed by Infineon during the initial manufacturing. Infineon stores the IFX_ROT_PRIVATE_KEY securely in internally controlled HSM and stores the IFX_ROT_PUBLIC_KEY on the device OTP. RAM applications are signed using the stored IFX_ROT_PRIVATE_KEY and validated using the corresponding IFX_ROT_PUBLIC_KEY stored on the device.
- **PROT_FW_ROT_KEY:** PROT_FW_ROT_KEY is used to authenticate the protected firmware. To revoke this key, use PROT_FW_ROT_KEY_REVOCATION.
- **OEM_ROT_KEY:** OEM_ROT_KEY is used to authenticate the OEM application images. By default, OEM_ROT_KEY0 is used to authenticate the images. If needed, OEM can revoke the key to use OEM_ROT_KEY1. The OEM can revoke the key only once through provisioning, and it cannot be reverted.

  During the provisioning process, the OEM has the ability to configure the OEM_ROT_KEY pair to utilize either ECDSA-521, ECDSA-384, or ECDSA-256 bit ECC. After the OEM has made their selection, the appropriate public key should be used to sign the OEM image. By default, the policy will have ECDSA-521 set as the default configuration.

15.3.2 Life cycle stages (LCS)

PSC3M/P8, PSC3M/P7 MCUs have configurable, non-volatile life cycle stages that play a critical role in device security. These stages follow a strict, one-way progression, controlled by blowing an eFuse, which ensures that once a stage is completed, it cannot be reversed. For instance, once a device transitions from the development stage to production, it cannot be reverted back to development. Instead, it can remain in production or move forward to the return merchandise authorization (RMA) stage, as determined by the OEM.

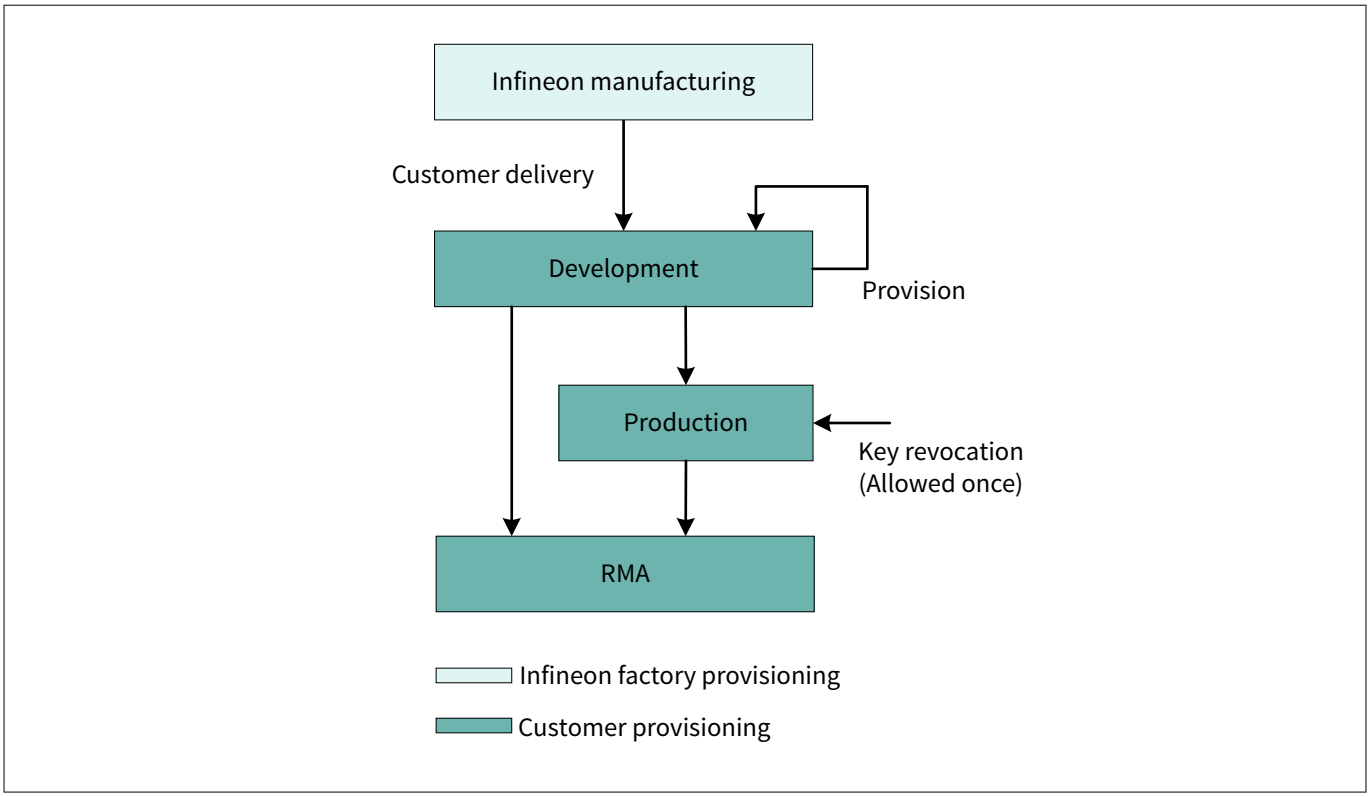


Figure 58 Life cycle stages

15.3.2.1 Infineon manufacturing

In Infineon Fab, parts are produced, tailored, and programmed in the IFX_MFG LCS. All the necessary testing and trimming takes place in this LCS and Infineon assets are loaded to the device. Device is transitioned to development LCS before shipping from the Infineon factory.

15.3.2.2 Development LCS

PSC3M/P8, PSC3M/P7 MCU devices are shipped in development LCS to the customer. Before the devices enter this state, all the trimming and factory testing is completed, and the devices are shipped out of Infineon in this LCS. Customers must provision OEM assets during development LCS and develop their end application for secure boot. At this stage, the device can be re-provisioned any number of times, and the OEM_ROT_KEY and assets can be replaced as needed through OEM provisioning.

***Note**: If a customer intends to reprovision the device in the development LCS, then do not set the prov_oem_complete parameter value as "true", system "ap" value as "Permanently Disable", and "dfu_enable" to "false".*

15.3.2.3 Production LCS

This is the in-field operational LCS of a secured device. Before moving onto this phase, signed valid application code must have been programmed and the HASH of the OEM keys must be provisioned (OEM_ROT_KEYS). In this LCS, device will boot only if application is authenticated successfully. If authentication fails at any stage, next application will not boot, and device boot process terminates here.

During transition of production LCS, the provisioning RAM Application performs the following:

- Provisions OEM_ROT_KEYS

- Verifies all policies are provisioned with the authorized production keys
- Writes SECURE_HASH to the OTP. A 32-byte SHA-512 SECURE_HASH consists of hash of OEM_ROT_KEYs and OEM_POLICY
- Writes the OTP bit to move irreversibly to the production state

  Debug access restrictions must be defined before entering the production LCS. For example, the debug policy can be set to permanently disable the debug in this LCS. Alternately, the debug policy can be set to disable if debug capabilities are required in this LCS and the device can be debugged with a secure debug certificate. Infineon recommends keeping the debug ports closed in production LCS.

15.3.2.4 RMA

The device can transition to the RMA lifecycle when a customer require Infineon to perform the failure analysis (FA). The device can be moved to RMA mode by invoking `TransitionToRMA RAMApp` with a certificate signed by OEM_ROT_KEY. Customer should erase all OEM-owned assets on the device before invoking the `RAMApp` and transition to the RMA LCS state. Once the device is transitioned to the RMA life cycle stage, the device can no longer be provisioned or programmed again. After the device is moved into RMA mode and when it boots, it does not attempt to execute any code in the user flash; instead, it waits for the `OpenRMA` token through the debug port, signed by Infineon, to enable further internal debug and analysis of the device.

15.3.3 Boot sequence

Secure boot authenticates the application images through the use of a public key cryptography security protocol defined by an industry-specific standard. When a device is power cycle/reset, ROM_BOOT trims and initializes the hardware. It validates the FLASH_BOOT before it jumps to execute FLASH_BOOT. FLASH_BOOT extends the functionality of ROM_BOOT, applies IFX and OEM policies, validates, and launches the edge protect bootloader (optional) or OEM application. Edge protect bootloader validates the first OEM/customer application and launches it as shown in Figure 59. These BootROM and edge protect bootloader use hashing extensively to authenticate data and code stored in the device. OEM can have multiple images based on BOOT_CFG_ID defined in the OEM policy as mentioned in Table 90.

Table 90 Boot configuration

BOOT_CFG_ID	Image options
Simple application configuration	
`DUAL_BANK_SIMPLE_APP`	• In this configuration the user flash memory is split in to two equal size(256KB each) banks as bank0 and bank1 • BootROM follows rest all features as same as `BOOT_CONFIG_APP` i.e. BootROM launches the active bank CM33 secure application without authentication
Secure boot configuration	
The following configurations are supported for secure boot : • `BOOT_SECURE_APP` • `DUAL_BANK_SECURE_APP` • `PROT_FW`	

(table continues...)

Table 90 (continued) Boot configuration

BOOT_CFG_ID	Image options
BOOT_SECURE_APP	• The BootROM shall authenticate multiple OEM images as specified in BOOT_APP_LAYOUT[] and launches the first CM33 secure application • The BOOT_APP_LAYOUT[] address list determines the number of images that the BootROM will authenticate • The BootROM shares Boot status information, providing the pass/fail status for image authentication • This configuration is designed to support production and RMA LCS
DUAL_BANK_SECURE_APP	• This configuration is same as DUAL_BANK_SIMPLE_APP and additionally BootROM expects the OEM image should be signed with OEM_ROT_KEY and aligned on MCUBoot format • BootROM authenticate active bank OEM images and launches it after successful authentication
PROT_FW	• This configuration is meant for protected firmware, which has read, write restriction and allows only execution permission • The protected firmware should signed with PROT_ROT_KEY and aligned with MCUboot format • The NS or PPCA application shall communicate to protected firmware via only IPC communication • The BootROM launches the protected firmware in the PC2 context, allows debug access only with a debug token, and responds to secure service requests from the non-secure world

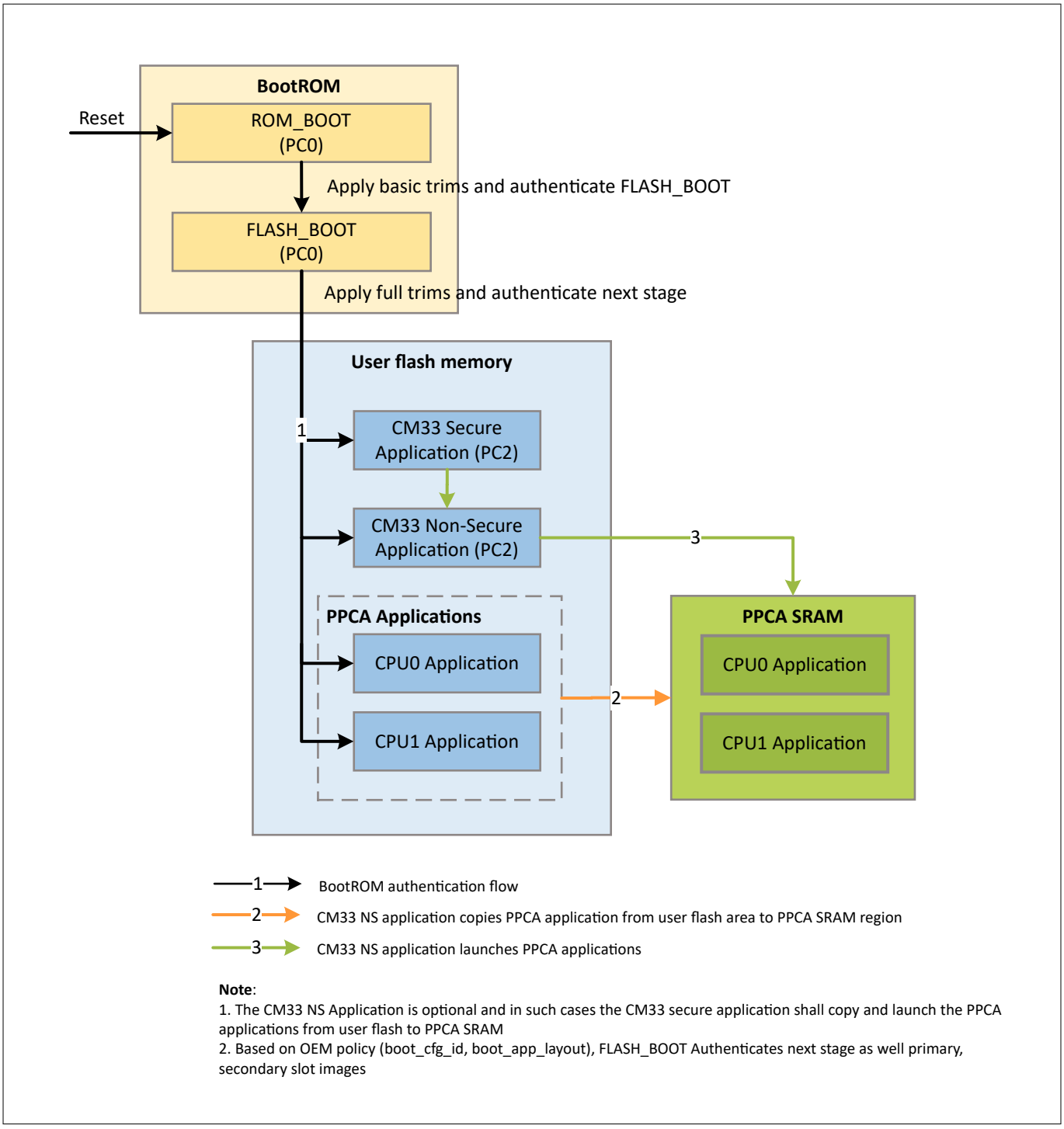


Figure 59 Secure bootflow

15.3.3.1 Secure boot

The BootROM provides the fundamental security features required to provide a secure system. When device is shipped from Infineon in development LCS, "secure boot" is disabled in the policy and BootROM boots to the OEM Application, without validating it. "secure boot" can be enabled in development LCS by provisioning OEM assets and programming OEM Application image in MCUboot format. Alternately, when device transition to production LCS, "secure boot" gets enabled automatically.

When "secure boot" is enabled in the policy, BootROM validates the first application using OEM_ROT_KEY_PUBLIC and creates a chain of trust between BootROM and the first OEM application to provide the highest order of security.

15.3.3.2 Protection states

A life cycle stage is a strict irreversible progression dictated by writing one time program (OTP) eFuse bits. Every time a boot occurs, protecting settings (MPC and PPC) of the device are applied by boot process. If a critical error, such as a hardware initialization error or any Infineon provisioned asset corruption or mismatch detection is found by Boot, the device enters the corrupted state.

In the event that secure boot is enabled and the authentication of the first user application image fails, the device will enter a 'dead branch' state. To recover from this state, the user will need to re-provision the device with the correct OEM assets in the development life cycle stage.

When a device encounters a debug token verification failure, it will not grant access to its debug port. To configure the debug policy, you need to modify the debug section of the OEM provisioning policy.

15.3.4 BootROM serial interface

The immutable BootROM includes a built-in DFU protocol that enables secure and reliable firmware upgrades through a serial interface. By default, DFU functionality is disabled and must be explicitly enabled in the OEM policy. The device must also be provisioned before DFU can be used.

DFU mode is selected during device boot-up based on the state of the *P8.2* pins, as described in Table 91. When the device enters DFU mode, pins *P8.1* and *P8.2* function as the serial communication interface. Use pull-up or pull-down resistors of 1 kΩ or less connected to VDDIO_1 or VSS to set the DFU select pin (*P8.2*) to logic high or low. For a High-Z state, the DFU select pin (*P8.2*) can be left floating.

Table 91 DFU mode and serial communication pins

Pin	UART
P8.0	UART - Rx
P8.1	UART - Tx
P8.2	DFU enable (VSS)/DFU disable (VDDIO_1/High-Z)

The following parameters are used for DFU-UART communication:

UART configuration:

1. **Baud rate**: 115200
2. **Data bits**: 8
3. **Stop bits**: 1
4. **RTS/CTS**: No
5. **Parity**: None

15.3.5 Secured provisioning and re-provisioning

Provisioning is the act of configuring a device with an authorized set of keys, certificates, credentials, and firmware images. Provisioning is performed using the provisioning applications (RAM application) that are built to run from the device SRAM. These applications program the Infineon and OEM assets into OTP, SFLASH and advance the device to development or production LCS. The device is ready to launch the user application in these LCSs.

Development LCS allows you to re-provision the OEM assets during the development phase any number of times. Production LCS will not allow to re-provision the OEM assets rather allows only once revoking the OEM_ROT_KEY key.

15.3.6 Protection context used in secure boot

Protection context (PC) is not one specific block of the hardware. However, it is used for all the protection units, such as MPU, MPCs, and PPCs. Multiple protection contexts are used to isolate the different security levels within the PSC3M/P8, PSC3M/P7 device. Secure boot process uses PC0, PC1, PC2 as shown in Figure 59 and Table 86.

16 Advanced Encryption Standard (AES)

This chapter describes the advanced encryption standard (AES) accelerator module, this module provides high-performance AES encryption or decryption with key lengths of 128/256 bits.

16.1 Introduction

The AES cryptographic peripheral is a dedicated hardware that implements a full algorithm according to `AES_FIPS_PUB_197` and `ISO_IEC_18033_3` standards.

The hardware can be easily programmed to perform an AES encryption or decryption with key lengths 128 and 256 bits. It has built-in security features, such as masking and redundancy mechanisms to protect the key as well as the input and output data against power analysis, fault, and probing attacks. The special function registers (SFRs) are specified below in the registers section. Functional software behavior of the module is described below.

16.2 Overview

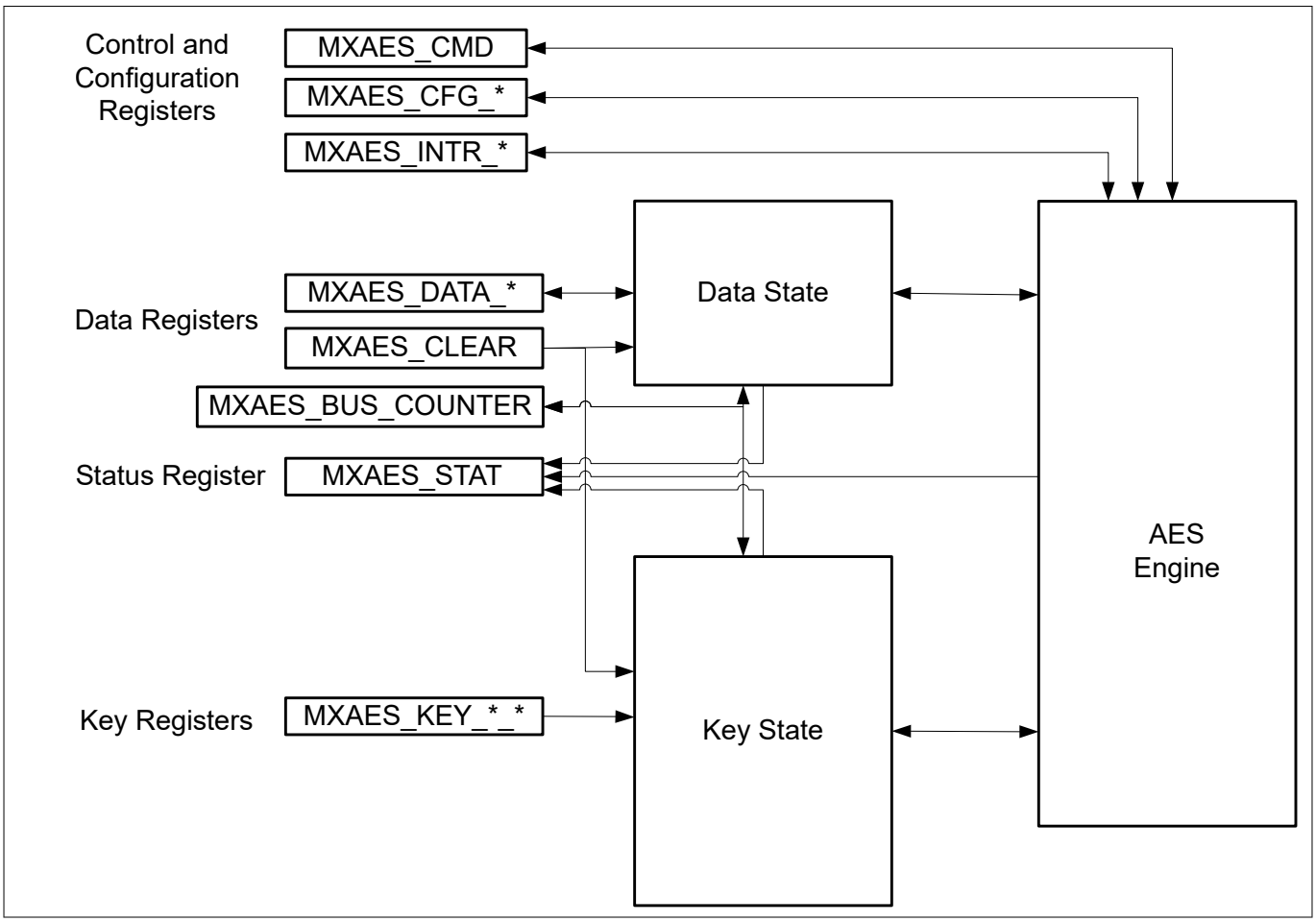


Figure 60 AES user interface

The AES SFR interface has the following types of registers:

- Configuration and command registers: These registers are used to let the software select the AES calculation mode, key length and, for example, set the security level required. The command register is used to start a calculation

- Status register: It provides general status information, for example, whether the AES engine is busy with a calculation and whether the result is ready or not
- Key registers: These registers are used to load directly to key shares into the AES key state
- Data registers: These are the input and output registers for the plain and cipher texts

16.3 Programming model

The programming sequence given in the below flow chart guides the user with the steps mandatory to execute to perform an AES encryption with this hardware accelerator module.

***Note**: The order of writing of MXAES_CFG_2 and MXAES_KEY_* and MXAES_DATA_* does not matter.*

***Note**: Clearing data is needed for starting a new calculation otherwise a DATA_X_Y word write is XORed with current DATA_X_Y content.*

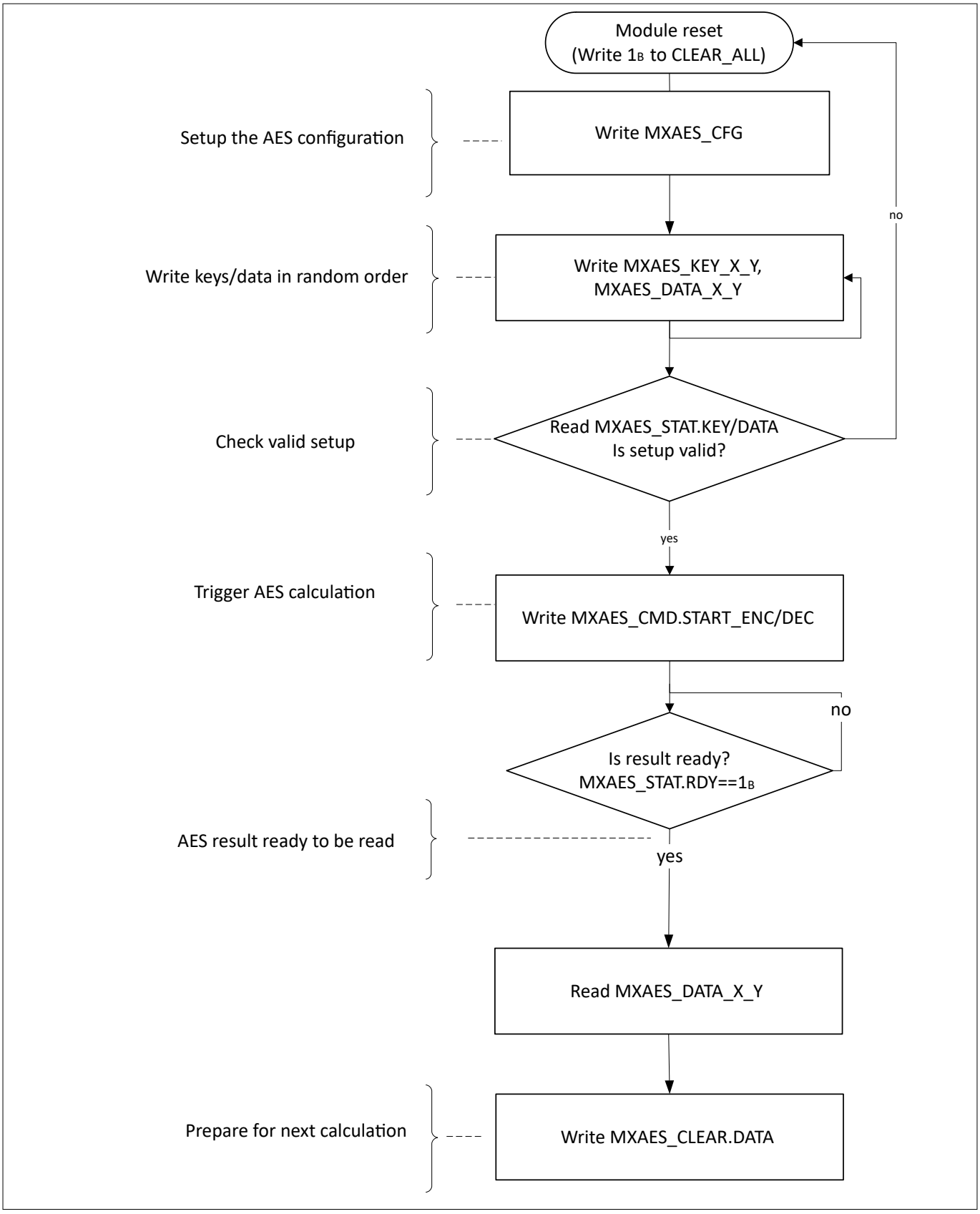


Figure 61 AES programming sequence

16.4 Functional description

16.4.1 Module reset

Through the module, reset the AES can be reset separately without the need to reset the whole system. The module reset can be triggered from software by setting the CLEAR_ALL bit to 1. This clears all SFRs, data, keys, and masks internal states except the internal seed.

16.4.2 Endianess

When writing a 128-bit plaintext block 0x1234ABCD9876EEFF1122334455667788, 0xCDAB3412 has to be written to AES_DATA_x_0, 0xFFEE7698 to AES_DATA_x_2, 0x44332211 to AES_DATA_x_3, and 0x88776655 to AES_DATA_x_3. However, the chronological order of reading and writing the data shares does not matter.

16.4.3 BusFault scenarios

To reduce the verification effort and the module complexity, the AES engine reacts to some scenarios with BusFaults such as not allowed read or write accesses or not allowed read or write accesses when the engine is not idle or specific not allowed subsequent accesses, non-allowed combinations of some modes and useless subsequent operations. The not allowed read or write accesses are given in the access modes tables below each register's definition but the rest of cases to which the engine reacts with a BusFault are listed below:

- Writing to or reading from unmapped registers
- Writing KEY_X_Y while the key state contains a KEY_X_Y decryption key

16.4.4 Handling of key state clearing

The clear register gives the user the possibility to clear keys. Clearing a key also clears the corresponding valid flag. The table below lists the keys which have been cleared according to which CLEAR.AES_KEY_* field has been set. Reading the STAT status bits after having written the CLEAR bit allows the user to check if the key-clearing settings through CLEAR write access – have been executed correctly.

Table 92 Key location cleared by CLEAR register

Field	List of keys implicitly cleared	List of key registers to reload	List of CRYPT_MODE keys not affected	List of keys not affected
AES_KEY_128LO	AES_KEY_256	KEY_0 KEY_1 KEY_2 KEY_3	AES128HI	AES_KEY_128HI
AES_KEY_128HI	AES_KEY_256	KEY_4 KEY_5 KEY_6 KEY_7	AES128LO	AES_KEY_128LO
AES_KEY_256	AES_KEY_128LO AES_KEY_128HI	AES_KEY_*	-	-

16.4.5 Fault structure

This module is a fault generating component. It acknowledges a fault in case the AES hardware has encountered a malicious behavior. It implements a fault interface: A fault request output signal, a fault acknowledge input signal and a fault data output information to support the handshake protocol as specified by the CPUSS fault report structure.

16.4.6 Timing

The total execution time for AES consists of the I/O overhead and the actual computation time. The I/O overhead strongly depends on the software that controls the AES module. The table below shows the actual hardware computation time. It gives first the time during which the engine is busy with a computation and the second the time elapsed for the AES result to be ready depending on the SEC_LEVEL configuration. For SEC_LEVEL=0, the result of an AES 128 operation is ready in 469 clock cycles.

Table 93 AES calculations timing

Engine Operation	AES-128	AES-256
Engine busy [clock cycles]	$352+192*2^{SEC_LEVEL}$	$472+192*2^{SEC_LEVEL}$
Result ready [clock cycles]	$277+192*2^{SEC_LEVEL}$	$373+192*2^{SEC_LEVEL}$

16.4.7 Debug support

This section describes the behavior of the AES module for debug accesses and during Debug state. The debugger has access to the module even if disabled via the System/Product PAU. However, there are no extra access rights for debug purposes and the module does not support debug registers. For debug accesses the AES operates as in normal mode of operation. It does not stop or suspend its current operation when a Debug state is entered.

***Note**: The differences in CPU and debug accesses are specified in the access rights table in cases where register access rights are different from CPU to debug accesses. Additionally, no side effects occur with debug accesses.*

16.5 AES registers interface

This section defines the programmer's view of the AES accelerator. The AES memory-mapped address region is 8 KB large. AES address space is partitioned into two address spaces: non protected (AES_R0 and AES_R1 regions) and protected (AES_R2 and AES_R3 regions). Protected AES registers can be identified through the value 1B of the thirteenth bit of the offset address. Second, each of these two times 4 K bytes address spaces is in turn partitioned into two address spaces: One for software higher access rights protected, so-called protection control (PC0/PC1) and a second for the rest of the software. The AES registers mapped to the latter two times 2 KB regions (AES_R1 and AES_R3) can be identified through the value 1B of the twelfth bit of the offset address.

Both regions sizes are power of two integers:

- R1=OFFSET_END_R1-OFFSET_START_R1+1=2^{11} Bytes
- R3=OFFSET_END_R3-OFFSET_START_R3+1=2^{11} Bytes

The following figure gives an overview of these four AES address-mapped regions.

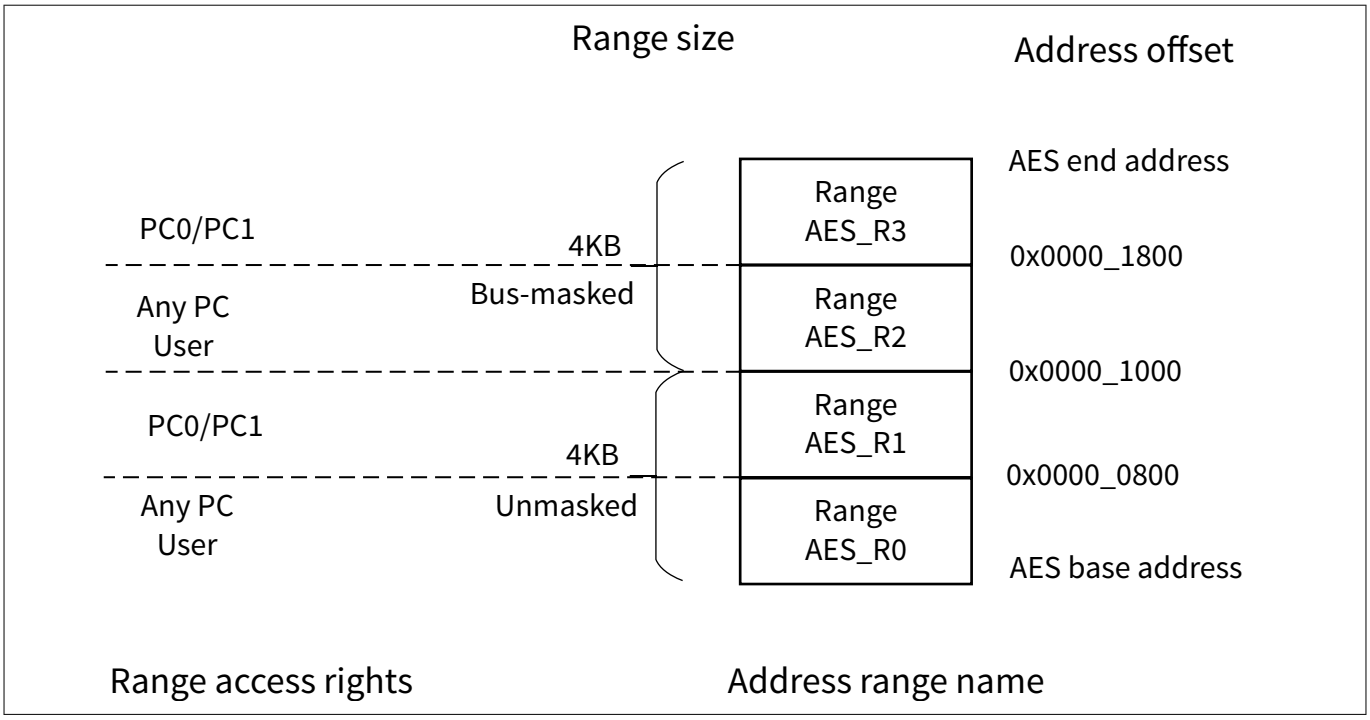


Figure 62 AES address space

16.5.1 Registers overview

Table 94 Registers overview

Register	Description	Address
MXAES_CFG	AES configuration register	0x4227_0000
MXAES_CMD	AES command register	0x4227_000C
MXAES_STAT	AES status register	0x4227_0014
MXAES_CFG_2	AES configuration register 2	0x4227_0060
MXAES_INTR	AES interrupt cause register	0x4227_0080
MXAES_INTR_SET	AES interrupt set register	0x4227_0084
MXAES_INTR_MASK	AES interrupt mask register	0x4227_008C
MXAES_INTR_MASKED	AES masked interrupt register	0x4227_0090
MXAES_CLEAR	AES clear register	0x4227_00C0
MXAES_BUS_COUNTER	AES bus counter register	0x4227_00DC
MXAES_QCHAN_CTL	AES Q-channel response register	0x4227_00E0
MXAES_TEST_1	AES test register 1	0x4227_0800
MXAES_SEED_U	AES user seeding register	0x4227_1000
MXAES_KEY_x_y (x=0-1;y=0-7)	AES key register (x for index of shares; y for index of word)	0x4227_1100 + x * 0x60 + y * 12

(table continues...)

Table 94 (continued) Registers overview

Register	Description	Address
MXAES_DATA_x_y (x=0-1;y=0-3)	AES data register (x for index of shares; y for index of word)	0x4227_1400 + x * 0x30 + y * 12
MXAES_DATA_RR_x_ y (x=0-1;y=0-3)	AES data reread register (x for index of shares; y for index of word)	0x4227_1490 + x * 0x30 + y * 12
MXAES_TEST_2	AES second test register	0x4227_1800

In tables where register bits or bit fields are defined, the following conventions are used to indicate the access types.

Table 95 Bit function terminology

Abbreviation	Description
r	The bit or bit field can only be read (read-only).
w	The bit or bit field can only be written (write-only). A read to this register will always give a default value back that is described in the register documentation.
rw	The bit or bit field can be read and written.
rw1ch	The bit or bit field can be read and written, but the bit or bit field can be cleared only by hardware.
wh	The bit or bit field can only be written (write-only), but the bit or bit field can be also set or reset by hardware.
rwh	As rw, but the bit or bit field can also be set or reset by hardware.
rh	This bit or bit field can be modified by hardware (read-hardware, typical example: status flags). A read of this bit or bit field gives the actual status of this bit or bit field back. Writing to this bit or bit field has no effect on the setting of this bit or bit field.
rw1s	As rw, only the software can set it by writing 1.

16.5.2 Access conditions

Table 96 Access conditions of AES

Condition	Description
AES_BUSY	The AES engine is busy. This condition defines a register's access behavior when the AES is busy doing a cryptographic operation; no key can be written. This mode is indicated by: AES_STAT.BSY = BSY
AES_IDLE	The AES engine is not busy and not ready simultaneously. It is the so-called idle state in the AES module finite state machine. This condition defines a register's access behavior indicated by: AES_STAT.RDY = NRDY and AES_STAT.BSY = NBSY
AES_NBSY	The AES engine is not busy with a calculation. AES_STAT.BSY = NBSY

(table continues...)

Table 96 (continued) Access conditions of AES

Condition	Description
AES_NRDY	The data is being loaded or data result is not ready. This condition defines a register's access behavior indicated by: AES_STAT.RDY = NRDY
AES_RDY	The AES result is ready. This condition defines a register's access behavior when the AES result is ready; data can be read but not written. Note that the ready access mode is not the complementary of the busy access mode. This mode is indicated by: AES_STAT.RDY = RDY
CPU	Access mode for CPU.
DEBUG	Access mode for debugger.
User	In the User mode, all SFRs are accessible.

16.5.3 Resets

Table 97 Resets of AES

Reset	Description
Module reset	A software synchronous local clearing of AES module is triggered when CLEAR_ALL is set.
Power-on reset	Reset through cold reset
System reset	System reset

16.5.4 AES configuration register

The configuration register is used for the two main settings of AES, which are the selection of the algorithm and the selection of the mode of calculation. Writing operations with undefined values for the fields CRYPT_MODE or CALC_MODE are ignored (i.e. the register keeps its previous value). NOMODE setting and any combination with NOMODE are illegal writes. Any legal write to MXAES_CFG clears all the AES data registers and the AES_STAT.RDY flag. A switch CFG.CRYPT_MODE from AES128LO/HI to AES256 or vice versa clears KEY_X_Y and the key state content.

MXAES_CFG Offset address: 0x0000

AES configuration register Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res			CRYPT_MODE					Res			CALC_MODE				
r			rw					r			rw				

Field	Bits	Type	Description
CALC_MODE	4:0	rw	**Mode of AES calculation** This field selects the mode of operation. • 0x00 **NOMODE**: No calculation mode selected • 0x03 **CIPHER**: Set the cipher mode This mode corresponds to a encrypt or decrypt calculation with masked key and data. To select the type of operation and start a calculation set MXAES_CMD.START_ENC or START_DEC
CRYPT_MODE	12:8	rw	**Cryptographic mode** This field selects the key's length and key state location of the AES cryptographic engine. • 0x00 **NOMODE**: No key size selection • 0x10 **AES128LO**: Set AES mode using lower half 128-bit key • 0x13 **AES128HI**: Set AES mode using higher half 128-bit key • 0x16 **AES256**: Set AES mode using 256-bit key

Table 98 Access modes of MXAES_CFG

Condition	Access mode	Description
AES_BUSY and Debug	read/write	This register is read only. Writes are ignored.
AES_BUSY	read only	This register is read only in this mode. Writes to this register trigger a BusFault.
(default)	read/write	This register has no access restriction.

16.5.5 AES command register

This command register of AES can be used to launch an operation. Note that setting START_ENC, START_DEC at the same time is ignored and has no impact. The setting START_ENC=START_DEC=1 is ignored. Only 0b10 and 0b01 cause an operation (either START_DEC or START_ENC bit is set). Setting more than one bit will be ignored. Writing 0b00 does not cause any action in the module. Writing 0b11 is ignored.

MXAES_CMD Offset address: 0x000C

AES command register Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res														START_DEC	START_ENC
r														rw1ch	rw1ch

Field	Bits	Type	Description
CALC_ENC	0	rw1ch	**Launch an AES encryption operation** Setting MXAES_CMD.START_ENC launches an encryption. Any write to this bit is ignored if data is not valid and key for the selected key length in MXAES_STAT is not valid. The START_ENC is cleared only by hardware as soon as the flag BSY is cleared. • 0x0 **NENC**: No encryption operation is launched • 0x1 **ENC**: An encryption operation is launched
CRYPT_DEC	1	rw1ch	**Launch an AES decryption operation** Setting MXAES_CMD.START_DEC launches a decryption. Any write to this bit is ignored if data is not valid and key for the selected key length in MXAES_STAT is not valid. The START_DEC is cleared only by hardware as soon as the flag BSY is cleared. • 0x0 **NDEC**: No decryption operation is launched • 0x1 **DEC**: A decryption operation is launched

Table 99 Access modes of MXAES_CMD

Condition	Access mode	Description
AES_BUSY and Debug	read/write	This register is read only. Writes are ignored.
AES_BUSY	read only	This register is read only in this mode. Writes to this register trigger a BusFault.
(default)	read/write	This register has no access restriction.

16.5.6 AES status register

The status register provides information about the general status of the AES engine and the key state. Key flags are a feedback information for the software to check the keys status given that key registers are not readable. In particular, the below flags indicate which keys are loaded and whether AES is busy/ready or not. The AES_STAT.xxx_KEY_xxx flags are set only after all 2 key shares are loaded. Only the AES_STAT.xxx_KEY_xxx flags which correspond to the set CRYPT_MODE are reliable. Changing the CRYPT_MODE sets the AES_STAT.xxx_KEY_xxx flags corresponding to that mode if enough key portions have been written for this selected mode. This avoids reloading the key when switching to a different CRYPT_MODE. For example, only when AES_CFG={AES128HI, CIPHER} and all key words AES_KEY_x_y (for x=0 to 1 and y=4 to 7) have been loaded will the AES_STAT.AES_KEY_128HI flag be set to 1B. No AES operation can be started until the relevant key status flag is set.

MXAES_STAT Offset address: 0x0014

AES status register Reset value: Table 101

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	START_DEC_RR	START_ENC_RR	CRYPT_MODE_RR					Res			CALC_MODE_RR				
r	rh	rh	rh					r			rh				

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BSY	RDY	SEED	RES	SEC_LEVEL_RR				Res			AES_KEY_256	Res	AES_KEY_128HI	AES_KEY_128LO	DATA
rh	rh	rh	r	rh				r			rh	r	rh	rh	rh

Field	Bits	Type	Description
DATA	0	rh	**AES data valid** This data validation flag becomes valid when a complete 128-bit data block is written to the registers DATA_0_0, DATA_0_1, DATA_0_2, DATA_0_3, DATA_1_0, DATA_1_1, DATA_1_2 and DATA_1_3 registers. • 0x0 **DNV**: Data has not been completely loaded • 0x1 **DV**: Data is valid
AES_KEY_128LO	1	rh	**AES 128 low key valid** • 0x0 **ALIV**: 128-bit AES key low has not been completely loaded • 0x1 **ALV**: 128-bit AES key low is valid
AES_KEY_128HI	2	rh	**AES 128 high key valid** • 0x0 **AHIV**: 128-bit AES key high has not been completely loaded • 0x1 **AHV**: 128-bit AES key high is valid
AES_KEY_256	4	rh	**AES 256 Key valid** • 0x0 **A2IV**: 256-bit AES key has not been completely loaded • 0x1 **A2V**: 256-bit AES key is valid
SEC_LEVEL_RR	11:8	rh	**SEC_LEVEL reread bits**
SEED	13	rh	**Seeding status bit** Reading this bit gives the programmer the information if a seeding has been performed. This bit is permanently set to '1'. • 0x0 **NO_SEED**: Seeding has not been performed yet • 0x1 **SEED**: Seeding already done
RDY	14	rh	**Ready flag** The ready flag indicates whether the result of an operation is available. It is asserted as soon as a valid result is available. This flag is deasserted in one of the following cases: MXAES_CLEAR.DATA or CLEAR_ALL bit is set, any DATA_X_Y register is written or any legal write to CFG register. • 0x0 **AES_NRDY**: Result is not ready to be read • 0x1 **AES_RDY**: Result is valid and ready to be read

(table continues...)

(continued)

Field	Bits	Type	Description
BSY	15	rh	**Busy flag** The busy flag indicates whether the AES engine is busy or not. Attention this flag indicates whether an encryption or decryption is undergoing or not. • 0x0 **NBSY**: Module is not busy with an AES operation. In this status keys can be written • 0x1 **BSY**: : Module is busy doing a cryptographic operation: neither data nor keys can be written
CALC_MODE_RR	20:16	rh	**AES CALC_MODE reread bits**
CRYPT_MODE_RR	28:24	rh	**AES CRYPT_MODE reread bits**
START_ENC_RR	29	rh	**AES START_ENC reread bit** This bit is set by hardware as soon as CMD.START_ENC is set by software. This bit is cleared by hardware as soon as STAT.BSY is cleared.
START_DEC_RR	30	rh	**AES START_DEC reread bit** This bit is set by hardware as soon as CMD.START_DEC is set by software. This bit is cleared by hardware as soon as STAT.BSY is cleared.

Table 100 Access modes of MXAES_STAT

Condition	Access mode	Description
Debug	read/write	This register is read only. Writes are ignored.
(default)	ready only	This register is read only. Writes trigger a BusFault.

Table 101 Reset values of MXAES_STAT

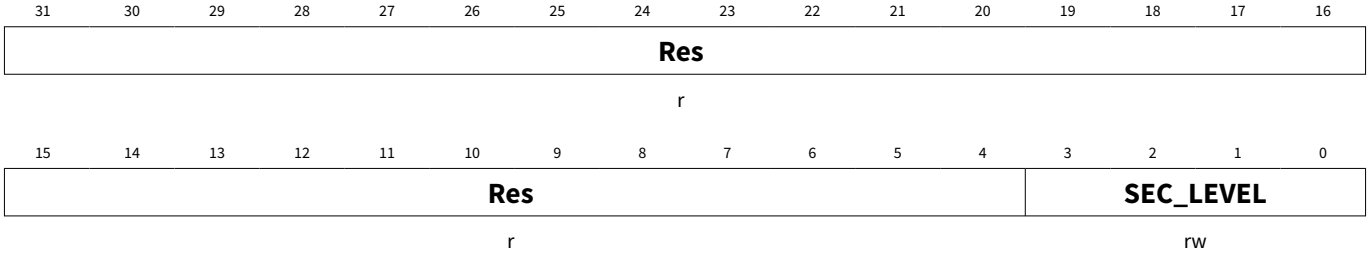
Condition	Access mode	Description
Power-on reset	0000 0000 0000 0000 00X0 0000 0000 0000b	Value set by power-on reset. This value depends on the NRNGA_INTEGRATION parameter.
System reset	0000 0000 0000 0000 00X0 0000 0000 0000b	Value set by system reset. SEED is not changed.
Module reset	0000 0000 0000 0000 00â€“0 0000 0000 0000b	Value set by CLEAR_ALL. SEED is not changed.

16.5.7 AES configuration register 2

The second configuration register allows to configure the overall security level of AES. Modifications of MXAES_CFG_2 are allowed as long as the module is not busy with an operation. Otherwise it triggers a BusFault. Note that it is not necessary to write this register if the default reset value is convenient for the application.

MXAES_CFG_2 Offset address: 0x0060

AES configuration register 2 Reset value: 0x0000 0000



Field	Bits	Type	Description
SEC_LEVEL	3:0	rw	**Security level** This field defines the overall security level. This feature consists of randomizing the AES execution and increases the resistance against side channel attacks targeting a key extraction as well as the strength of the backward calculation against differential fault analysis. This feature has a significant impact on the AES performance. The default value 0x0 gives the least randomization level with least performance drop impact. The security level can be increased by setting any valid values from 0x1 to 0xA to this field. Any value greater than 0xA is evaluated as 0xA.

Table 102 Access modes of MXAES_CFG_2

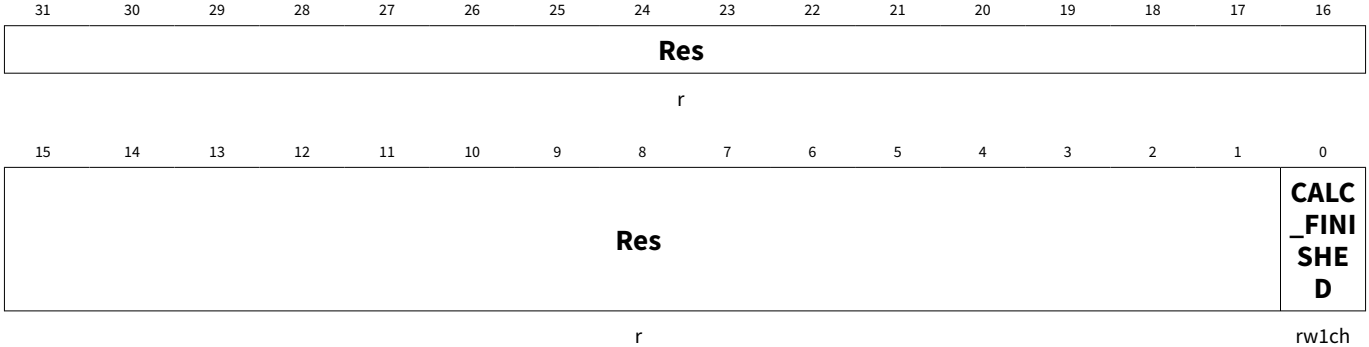
Condition	Access mode	Description
AES_BUSY and Debug	read/write	This register is read only. Writes are ignored.
AES_BUSY	read only	This register is read only in this mode. Writes to this register trigger a BusFault.
(default)	read/write	This register has no access restriction.

16.5.8 AES interrupt cause register

This register holds the interrupt causes of the module. It is set to 1 by hardware and set to 0 by software (RW1C attribute in the memory map (Readable and write one to clear)).

MXAES_INTR Offset address: 0x0080

AES interrupt cause register Reset value: 0x0000 0000



Field	Bits	Type	Description
CALC_FINISHED	0	rw1ch	**Interrupt cause calculation finished** This cause bit is set to 1 by hardware when the AES engine is not busy anymore with a calculation (also notified by the STAT.BSY flag). This is the unique interrupt cause of the module. This interrupt is cleared by software by either writing a 1 or by module reset or with the CLEAR_ALL bit. The software clear overrules the hardware set in case both happen simultaneously. • 0x0 **CLEARED**: No interrupt or interrupt cleared by software or module reset • 0x1 **CLEAR**: Clear interrupt

Table 103 Access modes of MXAES_INTR

Condition	Access mode	Description
AES_BUSY and Debug	read/write	This register is read only. Writes are ignored.
AES_BUSY	read only	This register is read only in this mode. Writes to this register trigger a BusFault.
(default)	read/write	This register has no access restriction.

16.5.9 AES interrupt set register

This register provides software with the capability to set a specific interrupt cause bit in MXAES_INTR register. The cause bit is set to 1 by software (RW1S attribute in the memory map). This functionality is useful for debug purposes. This register does not require dedicated storage elements, but reuses the MXAES_INTR register.

MXAES_INTR_SET Offset address: 0x0084

AES interrupt set register Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res															CALC_FINISHED
r															rw1s

Field	Bits	Type	Description
CALC_FINISHED	0	rw1s	**Interrupt set calculation finished** Interrupt set bit corresponding to interrupt cause CALC_FINISHED bit. This bit is readable. Only the software can set it by writing 1.

Table 104 Access modes of MXAES_INTR_SET

Condition	Access mode	Description
AES_BUSY and Debug	read/write	This register is read only. Writes are ignored.
AES_BUSY	read only	This register is read only in this mode. Writes to this register trigger a BusFault.
(default)	read/write	This register has no access restriction.

16.5.10 AES interrupt mask register

This register holds the interrupt mask bits of co-located interrupt cause bits in INTR register.

MXAES_INTR_MASK Offset address: 0x008C

AES interrupt mask register Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res															CALC_FINISHED
r															rw

Field	Bits	Type	Description
CALC_FINISHED	0	rw	**Interrupt mask calculation finished** The interrupt cause bit will only cause an interrupt when its mask bit is set.

Table 105 Access modes of MXAES_INTR_MASK

Condition	Access mode	Description
AES_BUSY and Debug	read/write	This register is read only. Writes are ignored.
AES_BUSY	ready only	This register is read only in this mode. Writes to this register trigger a BusFault.
(default)	read/write	This register has no access restriction.

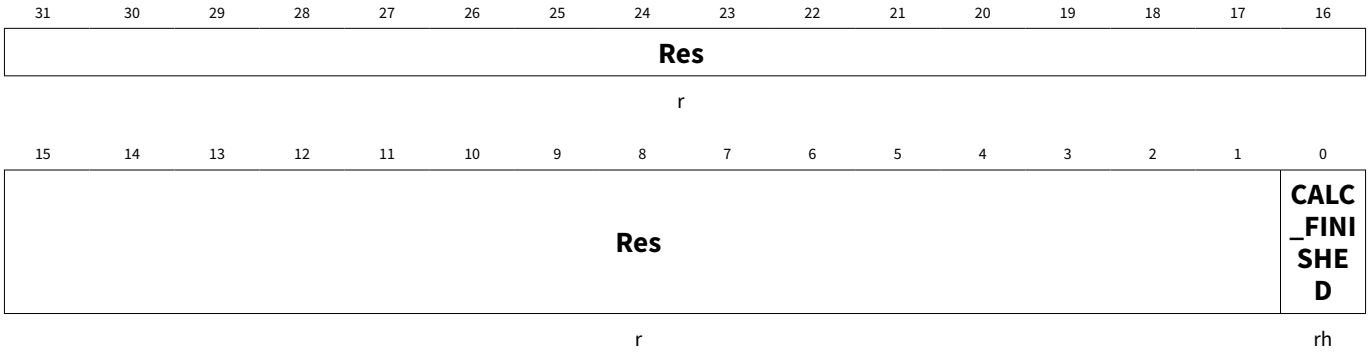
16.5.11 AES masked interrupt register

This register is the logical AND of the MXAES_INTR and MXAES_INTR_MASK registers. Typically, an Interrupt Service Routine (ISR) starts by identifying the cause bit that caused the interrupt. Without this register, two MMIO register reads are required: one read for MXAES_INTR and one read for MXAES_INTR_MASK. With the

register, only a single MMIO register read is required. This gain seems small, but ISRs are typically latency sensitive, and the MMIO register reads may travel a long way over the chip's bus infrastructure. Furthermore, this register comes cheap as it does not require dedicated storage elements, but uses the existing AES_INTR and AES_INTR_MASK registers.

MXAES_INTR_MASKED Offset address: 0x0090

AES masked interrupt register Reset value: 0x0000 0000



Field	Bits	Type	Description
CALC_FINISHED	0	rh	**Masked interrupt calculation finished** Logical AND of the MXAES_INTR.CALC_FINISHED and MXAES_INTR_MASK.CALC_FINISHED

Table 106 Access modes of MXAES_INTR_MASKED

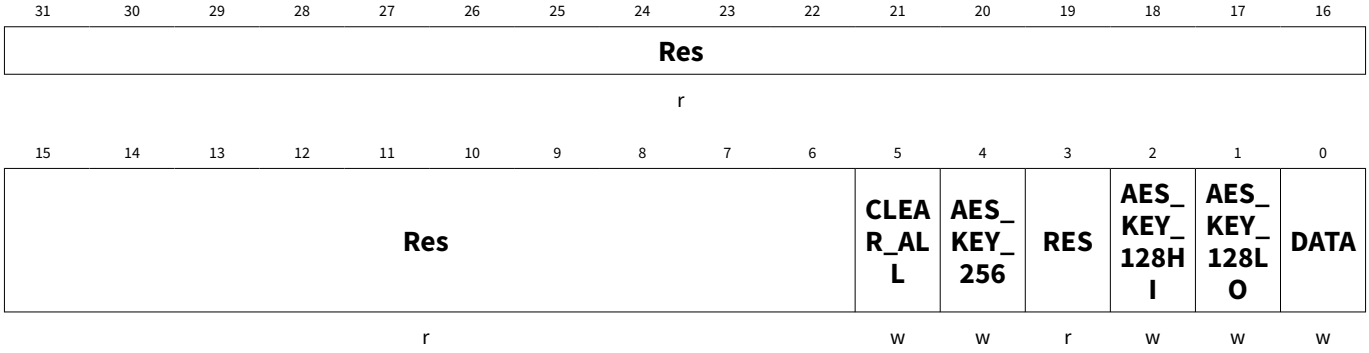
Condition	Access mode	Description
Debug	read/write	This register is read only. Writes are ignored.
(default)	read only	This register is read only. Writes trigger a BusFault.

16.5.12 AES clear register

The clear register gives the user the possibility to clear keys from the key state or clear data status flags (valid or not). Clearing a status flag also clears all words and shares of the corresponding data or key state keys. This is a write-only register. Reading the MXAES_STAT status bits after having written the MXAES_CLEAR allows the user to check if the flag clearing settings - through MXAES_CLEAR write access - have been executed correctly. A CLEAR_ALL set overrules when combined with any other bit set.

MXAES_CLEAR Offset address: 0x00C0

AES clear register Reset value: 0x0000 0000



Field	Bits	Type	Description
DATA	0	w	**Clear data** • 0x0 **NCLR**: Data keeps its value • 0x1 **CLR**: Data clearing triggered
AES_KEY_128LO	1	w	**Clear AES 128 low key** • 0x0 **NCLR**: 128-bit AES key low keeps its value • 0x1 **CLR**: 128-bit AES key is cleared
AES_KEY_128HI	2	w	**Clear AES 128 high Key** • 0x0 **NCLR**: 128-bit AES key high keeps its value • 0x1 **CLR**: 128-bit AES key high is cleared
AES_KEY_256	4	w	**Clear AES 256 key** • 0x0 **NCLR**: 256-bit AES key keeps its value • 0x1 **CLR**: 256-bit AES key is cleared if it is set in AES_STAT. This clears AES_KEY_128HI and LO keys
CLEAR_ALL	5	w	**Clear all SFRs and AES internal registers** Setting this bit behaves as a module reset. It clears all AES keys, data, all SFRs and all AES internal registers. • 0x0 **NCLR**: Clear is disabled • 0x1 **CLR**: A clear all is triggered

Table 107 Access modes of MXAES_CLEAR

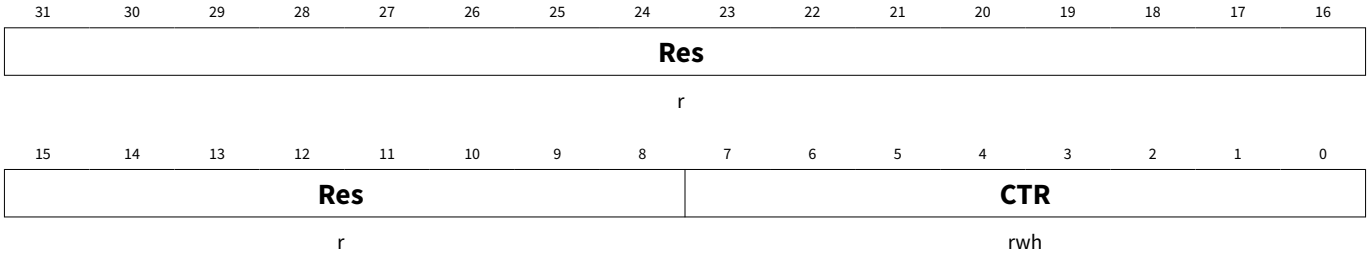
Condition	Access mode	Description
AES_BUSY and Debug	read/write	This register is write only. Writes are ignored and reads return 0xDEADDEAD.
AES_BUSY	none	Any access to this register triggers a BusFault.
Debug	read/write	No restrictions
(default)	write only	This register is write only. Reads trigger a BusFault.

16.5.13 AES bus counter register

This register supports the software in keeping track of the number of data and key shares written.

MXAES_BUS_COUNTER Offset address: 0x00DC

AES bus counter register Reset value: 0x0000 0000



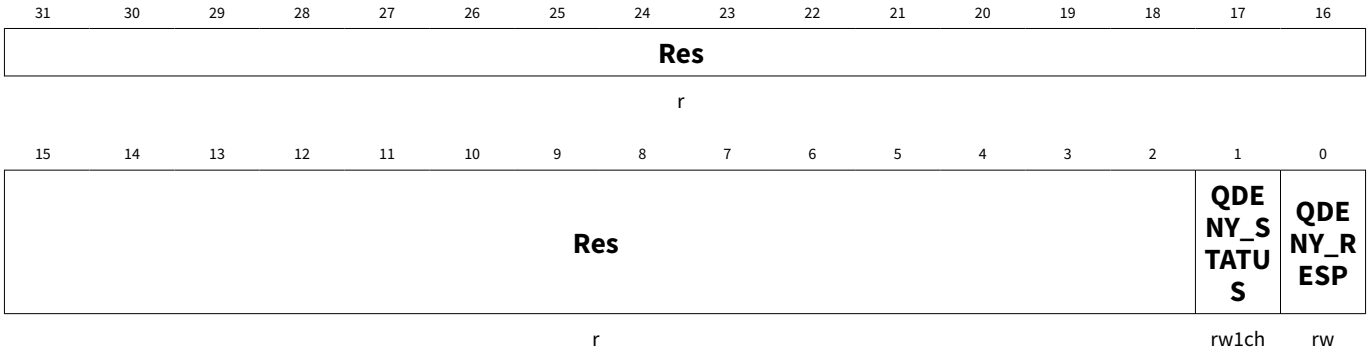
Field	Bits	Type	Description
CTR	7:0	rwh	**Key and data write counter** The counter is incremented on every CPU bus write access of data or key. On Debug write access of key or data registers the counter is not incremented. It is reset on module reset but never reset otherwise. When it reaches 0xFF and a key or data bus write happens, it is incremented to 0x00 while the overflow is ignored.

16.5.14 AES Q-channel response register

This register stores the Q-CHANNEL response bit. It allows the software to determine whether to assert qdeny or qaccept when qreq is asserted. If the product does not use Q-Channel, this register has no effect and can be ignored.

MXAES_QCHAN_CTL Offset address: 0x00E0

AES Q-channel response register Reset value: 0x0000 0000



Field	Bits	Type	Description
QDENY_RESP	0	rw	**Q-Channel QDENY Response Policy** This policy bit provides software flexibility to determine whether to assert qdeny or qaccept when qreq is asserted but the IP is not quiescent. Note: the AES always accepts the request. • 0x0 **ACCEPT**: A quiescence request is always accepted • 0x1 **DENY**: A quiescence request may be denied

(table continues...)

(continued)

Field	Bits	Type	Description
QDENY_STATUS	1	rw1ch	**Q-Channel QDENY Status** This status bit allows dynamic debug when a qdeny occurs. • 0x0 **NODENY**: No quiescence request was denied • 0x1 **DENIED**: Q quiescence request was denied

16.5.15 AES user seeding register

The SEED_U register can be used to trigger the internal seeding of AES that brings the internal mask units into a random state. This register provides a random number for the mask initialization. This register is write only.

MXAES_SEED_U Offset address: 0x1000

AES user seeding register Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PRNG_SEED															
w															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PRNG_SEED															
w															

Field	Bits	Type	Description
PRNG_SEED	31:0	w	**Seed for internal masking** Writing to this register triggers the internal seeding of the PRNG mask generator of AES. The content of this register is used to seed the AES mask generator in addition to the NRNGA internal noise source. It does not reset the PRNG, it actually adds up entropy to the entropy of the NRNGA internal noise source. This register is not readable however writable at any time also during STAT.BSY status.

Table 108 Access modes of MXAES_SEED_U

Condition	Access mode	Description
Debug	read/write	This register is write only. Reads return 0xDEADDEAD.
(default)	write only	This register is write only. Reads trigger a BusFault.

16.5.16 AES Key registers

This section describes the SFR interface of the key state. 'x' refers to the index of shares and y for the index of the word. For security reasons, key registers cannot be read. Note that any SFR write is XORed with the previous register content. The chronological order of writing of the key words and shares does not matter.

AES_KEY_0_0 corresponds to the bits 31 down to 0 bits in the key state and AES_KEY_0_7 to the bits 255 down to 224 of the first share.

AES_KEY_1_0 corresponds to the bits 31 down to 0 bits of the second share and AES_KEY_1_7 to the bits 255 down to 224 of the second share.

Note that writing two key shares into the words KEY_0-1_y is mandatory. A first complete key block must be written into the registers KEY_0_0 to KEY_0_* and a second into AES_KEY_1_0 to KEY_1_*. For example, for AES256 all key words KEY_x_y(x=0-1, y=0-7) has to be written at least once. This sets the key valid flag corresponding to its CRYPT_MODE. Writing at least once KEY_0_4, KEY_1_4, KEY_0_5, KEY_1_5, KEY_1_6, KEY_1_6, KEY_0_7, KEY_1_7 sets the key valid flag of AES128HI. Note that updating an AES key after a decryption operation has been performed leads to an unpredictable behavior or key state's content.

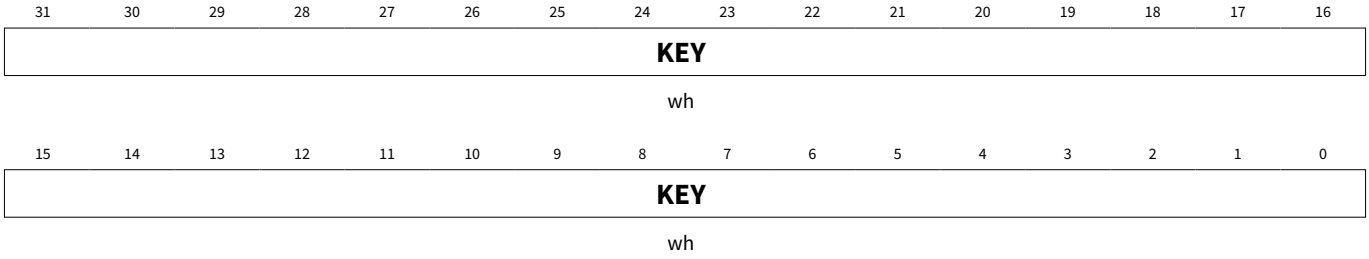
All MXAES_KEY_X_Y registers of the key state are cleared in these cases:

- Change CRYPTO_MODE from AES128LO/HI to AES256 and conversely
- Perform a clear with the clear register bit fields AES_256_KEY or CLEAR_ALL

16.5.16.1 AES key register (x for index of shares; y for index of word)

MXAES_KEY_x_y (x=0-1;y=0-7) Offset address: 0x1100+x*0x60+y*12

AES key register [x for index of shares; y for index of word) Reset value: 0x0000 0000



Field	Bits	Type	Description
KEY	31:0	wh	**Key value**

Table 109 Access modes of MXAES_KEY_x_y (x=0-1;y=0-7)

Condition	Access mode	Description
AES_IDLE and Debug	read/write	This register is write only. Reads return 0xDEADDEA.
AES_BUSY and Debug	read/write	Writes are ignored and reads return 0xDEADDEAD.
AES_IDLE and CPU	write only	Reads trigger a BusFault.
AES_BUSY and CPU (default)	none	Any access to this register triggers a BusFault.

16.5.17 AES Data registers

This section describes AES data shares and how and when to read or write the yth word of the xth data share.

After Reset, once the module has been configured by setting the crypto mode and loading its corresponding key words in shares as appropriate into the key state, the module is ready to receive data. The chronological order of writing/reading of the data words and shares does not matter. DATA_X_Y and KEY_X_Y words and shares writes can be interleaved as wished.

Note that any data SFR write is XORed with the data register content. A first complete data block must be written into the registers AES_DATA_0_0 to AES_DATA_0_3 and a second into AES_DATA_1_0 to AES_DATA_1_3. This sets the data valid flag. The encrypted/decrypted data result can be read from this register if AES_STAT.RDY=1B. DATA_0_0 corresponds to the bits 31 down to 0 bits of the 128-bit first share of data to load and DATA_0_3 to the bits 127 down to 96. DATA_1_0 corresponds to the bits 31 down to 0 bits of the 128-bit second share of data to load and DATA_1_3 to the bits 127 down to 96. Note that writing two data shares is mandatory and as soon as the corresponding eight words are written the AES_STAT.DATA flag is set.

This DATA_RR_x_y registers allow the user to check that data transfers to and from the AES_DATA_x_y registers have been performed correctly by means of a verification transfer after a AES_DATA_x_y data transfer. Write or read the yth word of the xth data share. DATA_RR_0_0 corresponds to the bits 31 down to 0 bits of the 128-bit first share of data to load and DATA_RR_0_3 to the bits 127 down to 96. DATA_RR_1_0 corresponds to the bits 31 down to 0 bits of the 128-bit second share of data to load and DATA_RR_1_3 to the bits 127 down to 96.

16.5.17.1 AES data register (x for index of shares; y for index of word)

AES data to encrypt/decrypt can be load from/to this register. When ready, the result can be read from this register.

MXAES_DATA_x_y (x=0-1;y=0-3) Offset address: 0x1400+x*0x30+y*12

AES data register [x for index of shares; y for index of word) Reset value: 0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATA															
rwh															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA															
rwh															

Field	Bits	Type	Description
DATA	31:0	rwh	**Data value**

Table 110 Access modes of MXAES_DATA_x_y (x=0-1;y=0-3)

Condition	Access mode	Description
AES_RDY and AES_NBSY	read/write	No restrictions
AES_NRDY and AES_NBSY and Debug	read/write	This register is writable in this mode. Reads return 0xDEADDEAD.
AES_RDY and AES_BUSY and Debug	read/write	This register is readable in this mode. Writes are ignored.
AES_NRDY and AES_BUSY and Debug	read/write	Reads return 0xDEADDEAD. Writes are ignored.
CPU and AES_NRDY and AES_BUSY	none	Any access triggers a BusFault.
CPU and AES_NRDY and AES_NBSY	write only	This register is writable in this mode. Reads return a BusFault.
CPU and AES_RDY and AES_BUSY (default)	ready only	This register is readable in this mode. Writes return a BusFault.

16.5.17.2 AES data reread register (x for index of shares; y for index of word)

This register content is used to check the integrity of the corresponding yth word of the xth data share in DATA_X_Y register.

MXAES_DATA_RR_x_y (x=0-1;y=0-3) Offset address: 0x1490+x*0x30+y*12

AES data reread register [x for index of shares; y for index of word) Reset value: 0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATA															
rh															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA															
rh															

Field	Bits	Type	Description
DATA	31:0	rh	**Data value**

Table 111 Access modes of MXAES_DATA_RR_x_y (x=0-1;y=0-3)

Condition	Access mode	Description
CPU and AES_RDY	read only	This register is read only
CPU and AES_NBSY	read only	DATA_RR_x_y register can be read only if DATA_x_y has been written, otherwise return 0xDEADDEAD.
AES_NRDY and AES_BUSY and CPU	none	Any access triggers a BusFault.
AES_NRDY and AES_BUSY and Debug (default)	read/write	Writes are ignored and reads return 0xDEADDEAD.

17 Power supply and monitoring

The PSC3M/P8, PSC3M/P7 MCU operates using a single regulated VDDD supply within the range of 1.71 V to 3.63 V. In addition, there is an optional VBACKUP supply that can be used, which has a range of 1.71 V to 3.63 V. It integrates multiple LDO regulators to power the blocks within the device in various power modes.

The device supports multiple power supply rails, such as VDDD, VDDA, VDDIOs, and VBACKUP, which enable the application to use a separate supply for different blocks within the device. For example, VDDA powers analog peripherals such as ADC, DAC, and DCSG. The PSC3M/P8, PSC3M/P7 MCU family supports power-on-reset (POR), brown-out-detect (BOD), and low voltage-detect (LVD) circuits for power supply monitoring and failure protection purposes.

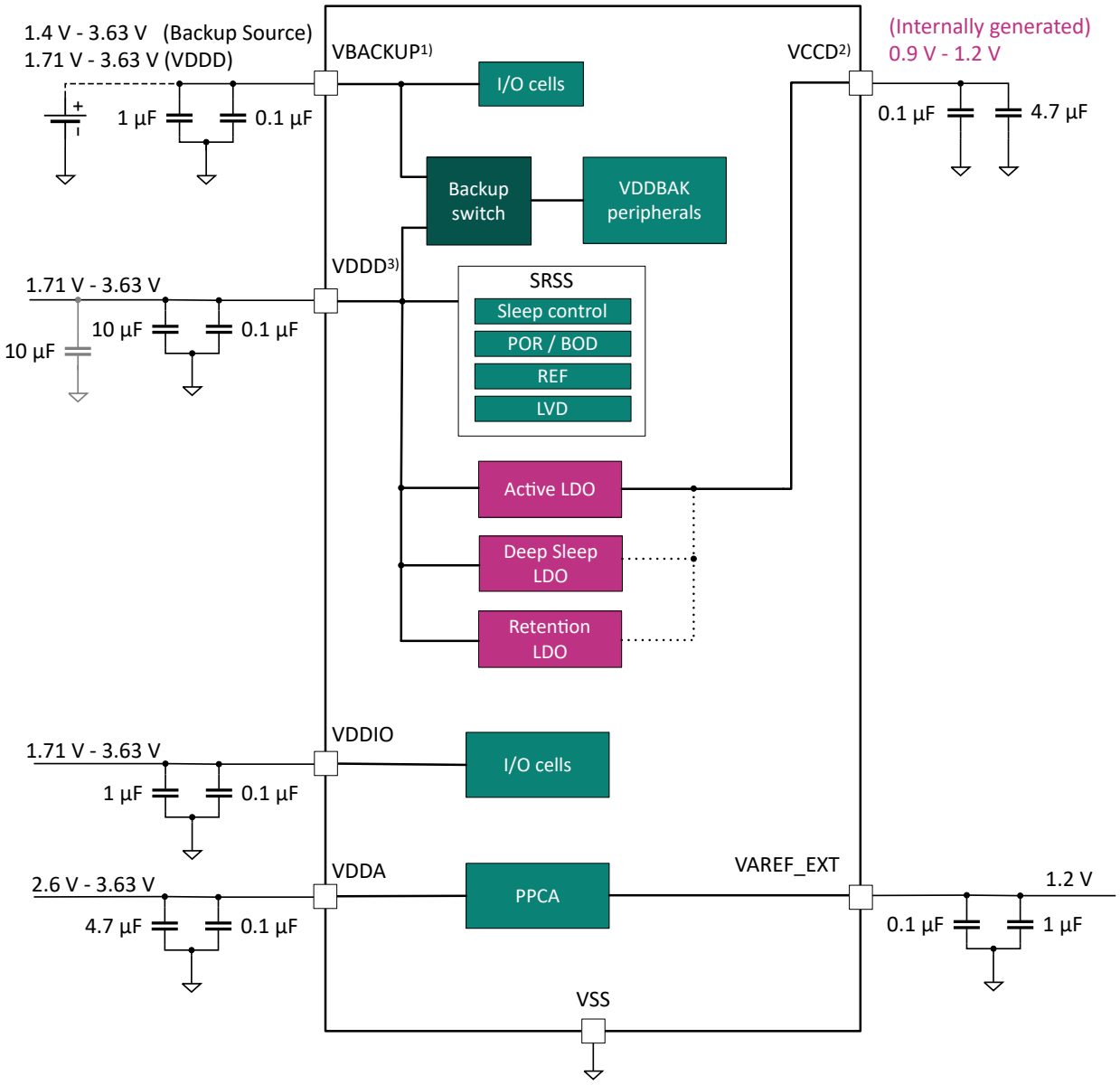
17.1 Features

The power supply and monitoring subsystem of the PSC3M/P8, PSC3M/P7 MCU supports the following features:

- Operating voltage range:
  - VDDIO: 1.71 V to 3.63 V
  - VDDA: 2.6 V to 3.63 V
  - VDDD: 1.71 V to 3.63 V
  - VBACKUP: 1.71 V to 3.63 V
- User-selectable core logic operation:
  - Ultra Low Power (ULP): Core logic runs at 0.9 V
  - Medium Frequency (MF): Core logic runs at 1.0 V
  - Low Power (LP): Core logic runs at 1.1 V
  - Overdrive (OD): Core logic runs at 1.2 V
- Multiple independent supply rails:
  - VDDD
  - VDDA
  - VDDIOs
  - VBACKUP
- Multiple on-chip linear regulators:
  - One Active LDO
  - Multiple low-power regulators to power the peripherals operating in different power modes
- Two POR/BOD circuits to monitor VDDD and VCCD
- One low voltage detect (LVD) circuit to monitor VDDD

***Note**: Refer to the device datasheet for information regarding the available supplies and the available user-selectable core logic operating voltages.*

17.2 Architecture



1) VBACKUP can be connected to a coin cell (1.4 V to 3.63 V) or a supercapacitor, depending on the application. If not used, it should be connected to VDDD.
2) VCCD is internally generated. It is only used to connect external bypass capacitors for the internal LDO. Do not use it to power external circuits.
3) A 10 µF capacitor on VDDD supports full operating voltage range for Active/Hibernate modes and Deep Sleep with VDDD ≥ 2.3 V. Use 2x10 µF capacitors or a single 22 µF capacitor for Deep Sleep operation down to 1.71 V.

Figure 63 Power supply and monitoring system block diagram

Figure 63 shows the typical powering configuration and VBACKUP connected to a coin cell. Isolated sources can be used; for example, to isolate the analog circuits, VDDA can be connected to an independent regulated supply between 2.6 V to 3.63 V.

The regulators and supply pins/rails shown in Figure 63 power various blocks inside the device. The availability of various supply rails/pins for an application depends on the device package selected. Refer to the device datasheet for details.

All core regulators draw their input power from the VDDD supply pin. The VCCD supply generated by the Active LDO provides power to all active domain peripherals in Active/Sleep mode. Deep Sleep LDO powers the Deep Sleep peripherals, while Retention LDO powers the retention logic for SRAM in Deep Sleep and Deep Sleep-RAM power modes. In Hibernate mode, all internal regulators are switched off, and Hibernate peripherals such as LPComp and Sleep Control are powered directly from the VDDD supply.

The I/O cells operate from various VDDIOx pins, depending on the port where they are located. The VCCD supply drives the logic inside the I/O cells from core logic peripherals. VDDA powers the PPCA (SAR ADC, DAC, and DCSG) and the analog logic, such as analog mux switches inside the I/O cells. To know which I/Os operate from which supply, refer to the device datasheet.

The device includes multiple VDDIO pins to power I/O cells, except those in the backup domain. VDDIO can either share the same supply as VDDD or use an independent supply voltage within the valid operating range. This flexibility allows different logic levels for port pins powered by separate VDDIO supplies. The VDDIO supply can be switched off if the port I/Os are not in use. However, switching off the VDDIO supply is permitted only in Active or Deep Sleep modes. It must not be switched off when the device is in Hibernate mode.

In addition to the power rails and regulators, the device provides options to monitor supply rails and provide protection against potential supply failures. These include POR/BOD circuits and an LVD circuit. The Sleep Control block provides the circuits required for entry and exit from low power modes.

17.3 Power supply

The device includes the following core regulators to power peripherals in various power modes:

17.3.1 Active regulator

The device includes a linear LDO regulator to power the Active and Sleep mode peripherals. This active linear regulator generates the core voltage (VCCD) required for Active mode operation of the peripherals from VDDD. The regulator is capable of providing 0.9 V, 1.0 V, 1.1 V, and 1.2 V[2)] for core operation. When changing the active regulator output voltage, it takes some time to stabilize, especially when changing to a higher voltage. This should be taken into account if the core voltage is changed by directly writing to the configuration registers. The PDL function includes the stabilization delay.

The regulator is available in Active and Sleep power modes. This regulator implements two sub-modes:

- High-current mode
- Low-current mode

The high-current mode is the normal operating mode; that is, the device operates to its full capacity in Active power mode. In the low-current mode, the current output from the regulator as well as its quiescent current is limited. This mode implements the LPActive, Sleep and LPSleep power modes. The low-current mode sets limitations on the capabilities and availability of resources in the Sleep mode. For example, in case of LPActive and LPSleep sub-modes, the maximum supported clock frequency is limited. For details, see Device power modes. The device always starts in Active mode; therefore, the Active LDO is always turned on at power on or when going to Active mode from any other operating mode. In power modes other than Active and Sleep, the active linear regulator is turned off.

2 Refer to the device datasheet for available core voltages.

17.3.2 Deep Sleep regulator

The Deep Sleep regulator powers the peripherals available in Deep Sleep mode. The regulator is optimized for low quiescent current operation to reduce the device's operating current in low power modes.

17.3.3 Retention regulator

The retention LDO reduces the Deep Sleep current by supplying a lower voltage to the retention flops and SRAM core in Deep Sleep mode. The lower supply voltage ensures the minimum retention voltage requirements are met while minimizing the leakage current.

17.3.4 Backup domain

A backup domain is an independent power domain. During normal operation, the VDDD supply provides power to the backup domain peripherals. In the absence of VDDD supply, the backup switch goes to the normally closed position, thereby utilizing the VBACKUP supply input to power the backup domain peripherals such as RTC. The user can either connect a backup source, such as a coin cell or supercapacitor, to the VBACKUP pin or connect it externally to the VDDD if backup functionality is not required.

17.4 SRSS reference block

The SRSS reference block contains two instances of 0.8 V band gap reference (BGR) voltage generators and a 250 nA current reference. These are used by the internal regulators, voltage monitors, PLLs, and so on. The Active BGR available during Active/Sleep modes consumes 20 uA of current, while the Deep Sleep BGR available during Deep Sleep mode consumes only 0.2 uA of current.

17.5 Voltage monitoring

The PSOC™ Control series MCU offers multiple voltage monitoring and supply failure protection options. This includes POR, BOD, and LVD.

17.5.1 Power-on-reset (POR)

POR circuits provide a reset pulse during the initial power ramp. POR circuits monitor VDDD and VCCD voltages. POR circuits are used during the initial chip power-up and then disabled. Refer to the device datasheet for details on the POR trip-point levels.

17.5.2 Brown-out detection (BOD)

The BOD circuit protects the operating or retention logic from possibly unsafe supply conditions by applying a reset to the device. The BOD monitors the VDDD voltage and generates a reset if the voltage excursion dips below the minimum VDDD voltage required for safe operation (see the device datasheet for details). The system will not come out of RESET until the supply is detected to be valid again. The BOD circuit guarantees a reset in Active, Sleep, Deep Sleep-RAM, and Deep Sleep-Off power modes before the system crashes.

There is no on-chip BOD support in Hibernate mode. Applications that require BOD support during Hibernate need to use external supervisor circuits. There is no configuration register for BOD. For details on the BOD trip points, supported supply ramp rate, and brown-out detection response time, refer to the device datasheet.

17.5.3 Low voltage detect (LVD)

An LVD circuit monitors the external supply voltage and generates an interrupt or fault while the device is still within its operating limits. Firmware can utilize the LVD interrupt to implement proactive measures for prevention. Use the PWR_LVD_CTL register to customize and adjust the trip point of the LVD. The PSC3M/P8, PSC3M/P7 MCU incorporates a single LVD on the VDDD rail. It is active and functional during Active/Sleep modes. LVD is not available during Deep Sleep and Hibernate modes. By default, the LVD is disabled to conserve power since its operation consumes energy. It is important to note that LVD does not serve as a reset source; instead, the system reset responsibility lies with the BOD circuits. However, if desired, the firmware can initiate a system reset through the LVD interrupt.

The HVLVD1_TRIPSEL [3:0] bit-field is used to select one of the sixteen supported voltage thresholds. See the PWR_LVD_CTL register description in the PSC3M/P8, PSC3M/P7 MCU Register TRM for threshold value details. The HVLVD_SRCSEL[6:4] bit-field only supports VDDD monitoring by using the default value of 0x0 or by using 0x1 and 0x4 to select the VDDD branch through AMUXBUSA or AMUXBUSB, respectively. Other values are reserved. The HVLVD1_EN [7] bit is used to enable the LVD. Ensure that the LVD is disabled before changing any other configuration bits in the PWR_LVD_CTL register. The HVLVD1_EDGE_SEL [17:16] bit-field is used to select which edge (s) will trigger an action when the threshold is crossed. This can be configured to disabled, rising edge, falling edge, or both rising and falling edges.

17.6 Sleep control

Sleep control contains the required logic and circuits to enable power mode transitions from Active/Sleep to Deep Sleep and Hibernate and wake up to Active/Sleep mode. There is a possibility that co-processors and peripherals are operating without CPU intervention. To ensure that these operations are not impacted when entering Deep Sleep, sleep control implements a quiescence request and response scheme. When the Deep Sleep entry is requested, the CPU sends a quiescence request to PERI, which expands this request to the relevant peripherals. These requests are sent on dedicated interfaces called Q-channels. If any peripheral denies the request, the Deep Sleep entry is aborted.

The Wake-up Interrupt Controller (WIC) is used for sensing the inputs that can wake up the system from Deep Sleep and Hibernate mode. The WIC forwards the wake-up request to the main Power Policy Unit (PPU), and the PPU in turn executes the wake-up sequence to bring the device into Active mode. For further details, refer to Device power modes.

17.7 Registers summary

Table 112 Power Supply and Monitoring register summary

Register	Function
PWR_LVD_CTL	High Voltage/Low Voltage Detector (HVLVD) configuration register
PWR_LVD_STATUS	High Voltage/Low Voltage Detector (HVLVD) status register
PWR_CTL2	POR/BOD, voltage/current reference, LDO control register

18 Device power modes

The PSC3M/P8, PSC3M/P7 device supports multiple power modes intended to minimize average power consumption in an application. PSC3M/P8, PSC3M/P7 supports six low-power modes to achieve the best compromise between low-power consumption, short start-up time, available peripherals, and available wake-up sources.

The Peripheral Driver Library (PDL) supports all device power mode transitions and is the recommended method for transitioning and configuring the PSC3M/P8, PSC3M/P7 power resources.

18.1 Features

The PSC3M/P8, PSC3M/P7 power modes provide the following features:

- Software can utilize power modes to optimize power consumption in an application
- Deep Sleep mode supports multiple wake-up sources and configurable amounts of SRAM and CPU state retention
- Hibernate mode allows wake-up from I/O, LPComp, RTC, and timer alarms

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

- Enabling and disabling clocks to peripherals
- Powering on/off clock sources
- Powering on/off peripherals and parts inside PSC3M/P8, PSC3M/P7

18.2 Architecture

The PSC3M/P8, PSC3M/P7 MCU supports the following power modes, in the order of decreasing power consumption:

- **Active/Sleep**: In Active mode, all peripherals and the CPU are available. In Sleep mode, all peripherals except the CPU are available
  - **Low-Power Active (LPACTIVE)/Low-Power Sleep (LPSLEEP)**: These modes are low-power profiles implemented as register configurations within Active/Sleep modes
- **Deep Sleep**: In this mode, active logic is frozen, but low-speed functions are available. Most configurations and memory states are retained. High-speed clock sources are turned off. Wakeup returns the processor to where it left off
- **Deep Sleep RAM**: Active mode logic is not retained, which necessitates a processor reset upon wakeup. The entire system RAM can be retained. Wakeup causes the processor to ‘warm’ reboot
- **Deep Sleep OFF**: Active mode logic and memories are not retained. Wakeup causes the processor to ‘cold’ reboot
- **Hibernate**: The lowest power mode where VDDD is still present. The device and I/O states are frozen, and the device resets on wake up

Two embedded linear voltage regulators supply power to most of the digital circuitry in the device: the linear active regulator and the Deep Sleep regulator. The linear active regulator is utilized in the Active and Sleep modes, while the Deep Sleep regulator, along with the retention and NWELL regulators, is used in the Deep Sleep, Deep Sleep RAM, and Deep Sleep OFF modes.

The device supports multiple voltage operating points, allowing for dynamic voltage scaling to optimize power consumption in the Active mode. The voltage from the linear active regulator, which powers the logic core (VCCD), can be adjusted based on the system’s maximum operating frequency.

The power management unit system operates in the following mode selections:

- **Ultra Low Power (ULP) mode**: 0.9 V setting with CPU running at 50 MHz
- **Mid Frequency (MF) mode**: 1.0 V setting with CPU running at 70 MHz

- **Low Power (LP) mode**: 1.1 V setting with CPU running at 150 MHz
- **Over Drive (OD) mode**: 1.2 V setting with CPU running at 180 MHz

The PSC3M/P8, PSC3M/P7 device operates with a single external supply (VDDD) in the range of 1.71 V to 3.6 V and features six power modes, with transitions managed by the power subsystem. The backup power supply (VBACKUP) operates within the range of 1.71 V to 3.6 V. When the external VBACKUP is not connected, the supply on VDDD is used as the backup power supply.

18.2.1 Power modes

The PSC3M/P8, PSC3M/P7 device supports the following power modes:

Table 113 PSC3M/P8, PSC3M/P7 MCU power modes

Power mode	Description	Entry condition	Wake-up source	Wake-up action
Active	• Primary mode of operation with all peripherals and CPU available • Current consumption as low as 7 mA (Typ.)	Wake up from Sleep, all Deep Sleep modes, Hibernate reset, or any other reset	Not applicable	Not applicable
Sleep	• CPU is in Sleep mode; all other peripherals are available • Current consumption as low as 4 mA (Typ.)	Register writes from Active mode or wake up from Deep Sleep through debugger	Any interrupt to CPU	Interrupt
Deep Sleep	• All high-frequency clocks and peripherals are turned off • Low-frequency clocks and low-power analog and digital peripherals are available for operation and as wake-up sources • Current consumption as low as 16 µA (Typ.)	Register writes from Active	GPIO interrupt, Low-power comparator, SCB-instance, watchdog timer, RTC alarms, and debugger	Interrupt or debug

(table continues...)

Table 113 (continued) PSC3M/P8, PSC3M/P7 MCU power modes

Power mode	Description	Entry condition	Wake-up source	Wake-up action
Deep Sleep RAM	• A low-power mode, similar to Deep Sleep, except most configuration states are not retained • Wakeup causes the processors to reboot • The software can use the retained SRAM contents (programmable) to more quickly recover the application state, enabling a 'warm' reboot[1)]	Register writes from Active	GPIO interrupt, Low-power comparator, SCB-instance, watchdog timer, and RTC alarms	Warm reboot
Deep Sleep OFF	• A low-power mode, similar to Deep Sleep, except most configuration states and memories are not retained • Wakeup causes the processors to perform a 'cold' reboot	Register writes from Active	GPIO interrupt, Low-power comparator, SCB-instance, watchdog timer, and RTC alarms	Cold reboot

(table continues...)

Table 113 (continued) PSC3M/P8, PSC3M/P7 MCU power modes

Power mode	Description	Entry condition	Wake-up source	Wake-up action
Hibernate	• GPIO states are frozen; all high-frequency clocks and peripherals are switched off • Low-frequency clocks and WCO can function • Device resets on wake-up event • Current consumption as low as 300 nA (Typ.)	Register writes from Active	WAKEUP pins, Low-power comparator, RTC alarms, and watchdog timer	Hibernate reset

1) Boot uses the last 8K of RAM for internal variables/stack

18.2.1.1 Active and Sleep

Active mode: The device enters Active mode upon any reset, which is the normal operating power mode where all device resources used by the system are active and clocked. In this mode, the CPU executes code along with all logic and is memory-powered. The firmware may decide to enable or disable specific peripherals and power domains based on the application and power requirements. All peripherals are available for use in Active mode.

Sleep mode: In Sleep mode, the CPU clock is turned off, and the CPU halts code execution to enter sleep. All peripherals available in Active mode are also available in Sleep mode. Peripheral activity can continue; if the CPU is required, it can be activated via clock gating instantaneously. Any unmasked interrupt can wake up the CPU to Active mode.

LPACTIVE/LPSLEEP: They are low-power profiles implemented as register configurations within Active/Sleep modes to reduce power consumption during Active or Sleep mode. These low-power profiles are similar to Active/Sleep, except the current is limited, clocks may run at a lower frequency, and some functions are not available or are limited.

- **LPACTIVE**:
  - Clock for clk_hf0 (main logic clock) is selected as IMO, which runs at 8 MHz nominal
  - Optional peripherals are turned off, including reference voltage buffer, low voltage detection (LVD), and other clock sources
  - Code runs from flash
  - POR/BOD and references are put in their low-power mode, limiting the maximum external ramp rate on VDDD
- **LPSLEEP**: LPSLEEP relates to LPACTIVE in the same way that Sleep relates to Active

18.2.1.2 Deep Sleep

Deep Sleep mode is a low-power mode in which the active logic is frozen while low-speed functions remain available. Most configuration and memory states are retained, and GPIO states are frozen. In this mode, high-speed clock sources such as the internal main oscillator (IMO), external crystal oscillator (ECO), and phase-

locked loop (PLL) are turned off, rendering the CPU, SRAM, high-speed logic, and high-speed peripherals unusable. However, low-speed clock sources like the watch crystal oscillator (WCO) and internal low-speed oscillator (ILO), as well as peripherals configured and enabled by the firmware, continue to operate. The ILO can be turned off to conserve more power. Wake-up returns the processor to where they left off.

This mode allows the operation to resume at the same program counter value, with the state allowing the resumption of operation fully retained. Stack and critical parameters can also be retained, and interrupts from low-speed asynchronous or low-power analog peripherals can cause a CPU to wake up from Deep Sleep mode. Upon a normal wake-up from Deep Sleep, the processor returns to either Active or Sleep, depending on the programmed behavior for the particular wake-up interrupt, while a debug wake-up from Deep Sleep returns to Sleep.

This mode is used in applications where the processor does not need to run the code, but peripherals are required to function at a speed lower than the system clock speed.

PPCA behavior in Deep Sleep mode

1. PPCA is an Active peripheral:

   The PPCA is designed to operate only in ACTIVE mode. It does not support low-power functionalities like Deep Sleep operations
2. Behavior During Deep Sleep (DS):

   Entering DS is equivalent to shutting down the PPCA. Exiting DS is treated as a reboot of the PPCA, where the subsystem starts afresh. Despite shutting down, the PPCA retains certain registers in Deep Sleep to preserve critical information needed after reboot
3. PPCA Power Down Logic:

   The PPCA_CNFG DISPERI register is critical for disabling unused or unconfigured CPU cores. Any core not explicitly disabled through this register will remain in "halt mode," preventing the PPCA from executing its internal shutdown procedures correctly. This can disrupt efficient power management during transition to Deep Sleep
4. Recommendations for Deep Sleep Transitions:

   Before the Main CPU enters Deep Sleep, the PPCA must already be in its shutdown state. Failure to follow this recommendation could lead to improper system behavior, inefficiencies, or potential errors during power transitions

   Main CPU Deep Sleep Entry Sequence

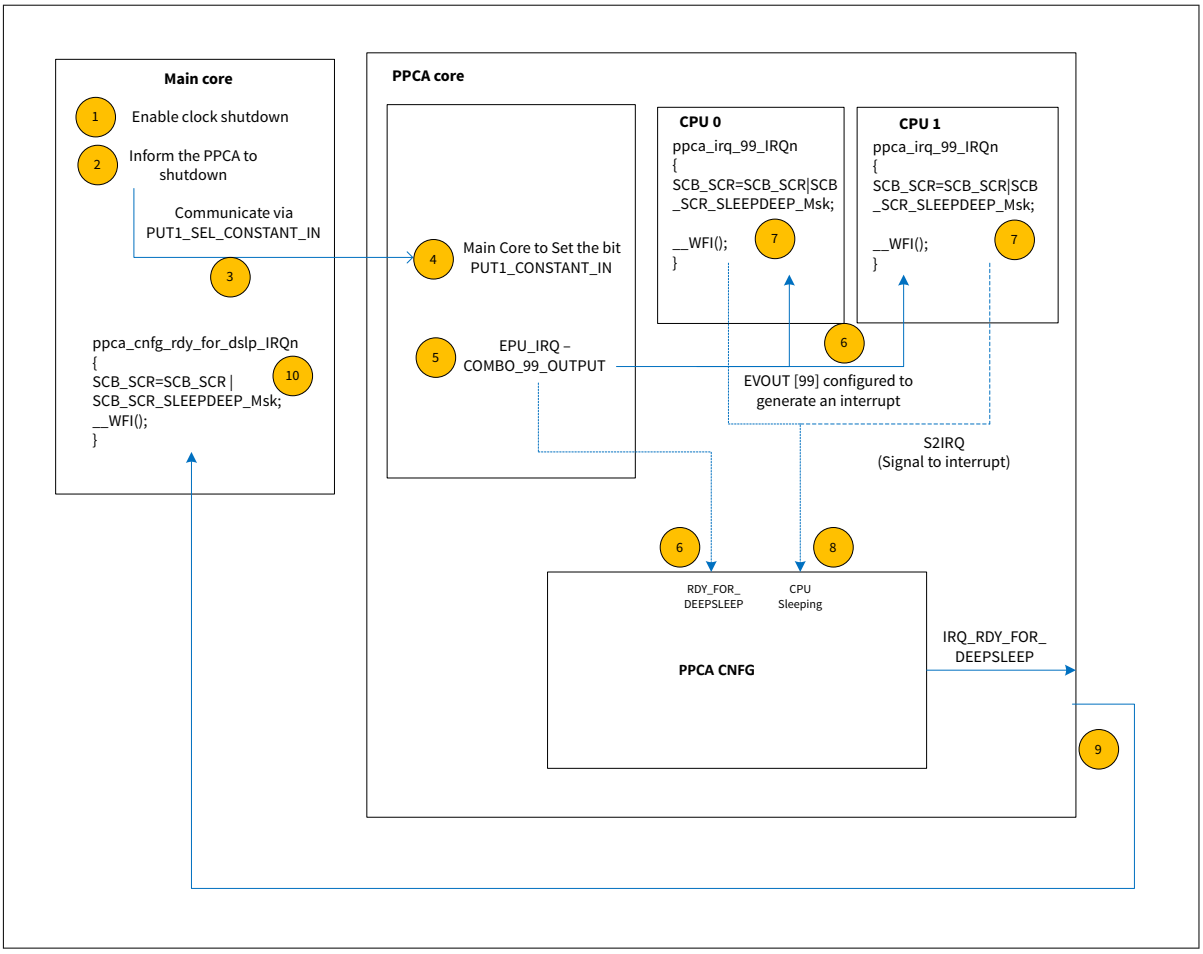


Figure 64 Deep Sleep entry sequence

Main CPU Core - Set PPCA_CLK_SH_DWN_EN Bit

The Main CPU initiates the process by enabling the clock shutdown for the PPCA subsystem. Setting the PPCA_CLK_SH_DWN_EN bit signals that the Main CPU is ready to transition into a low-power state and asks the PPCA to begin its own shutdown preparation

Main CPU Core - Inform PPCA to Shutdown

The Main CPU explicitly notifies the PPCA to enter shutdown mode. This step involve communication via EPU interrupt

PPCA Core - Assess Shutdown Readiness

Once the PPCA receives the shutdown request, The PPCA evaluates whether it is safe to transition to the shutdown state

If the PPCA determines that it is "OK" to shutdown (e.g., all dependent peripherals are in a safe state, no critical tasks are active), it begins preparing for a graceful shutdown

If the PPCA determines that it is "NOT OK" to shutdown (e.g., pending tasks or peripherals are active), it sends a rejection response back to the Main CPU. The process then halts unless the Main CPU retries

PPCA Core - Enter WFI with SCB.DS Bit Set

The PPCA sets the SCB.DS (System Control Block Deep Sleep) bit, indicating that it is entering a low-power state during shutdown. The PPCA then executes a WFI (Wait-For-Interrupt) instruction to stop its execution and allow hardware to take over the transition

PPCA Core - Start the Shutdown Sequence

After entering the WFI state, the PPCA hardware or firmware begins its shutdown sequence

PPCA Core - Signal Main CPU (IRQ_RDY_FOR_SLEEP)

Once the PPCA has successfully entered its shutdown state, it sends an interrupt or signal (IRQ_RDY_FOR_SLEEP) to the Main CPU. This signal informs the Main CPU that the PPCA is now ready for the system-wide low-power state

Main CPU Core - Enter WFI

After receiving the IRQ_RDY_FOR_SLEEP signal from the PPCA, the Main CPU also sets its own low-power state configurations and executes a WFI instruction. This effectively stops the Main CPU's execution and allows the system to fully transition to the Deep Sleep state

At this point, both the Main CPU and PPCA are in the Deep Sleep (DS) state. The system remains in this state until a wake-up event (such as an interrupt, timer, or external signal) brings the system back to an active state

Main CPU Deep Sleep Exit Sequence

Main CPU core : Deep Sleep wake-up interrupt occurs, Main CPU is ACTIVE
Main CPU core : Start Reboot sequence on PPCA, clear IRQ_RDY_FOR_SLEEP.
Main CPU core : Enable PPCA as done during the cold boot again.
Main CPU core : Enable PPCA CPU's by clearing CM33_CTL.CPU_WAIT bit on both Cores.
At this point, both PPCA cores starts booting again as fresh boot. Main CPU along with PPCA are in ACTIVE state.

18.2.1.3 Deep Sleep RAM

In Active mode, logic is not retained, requiring a processor reset upon wake-up. However, in Deep Sleep RAM mode, the entire SRAM is retained. The software can store crucial information in the system RAM, enabling a ‘warm’ reboot that returns the application to its previous state more quickly than a full ‘cold’ reboot. While in Deep Sleep mode, logic is operational, but only the SRAM is retained. When a wake-up occurs, it results in a reset, known as a warm reboot.

Note:
- *The SRAM region (SRAM size minus 8 KB) is utilized by Infineon firmware during the boot operation. Although this region is available to the user, data retention across resets, including return from Deep Sleep RAM, is not guaranteed in this area, as it can be overwritten by Infineon boot firmware*
- *On warm boot (wakeup from Deep Sleep RAM/Deep Sleep OFF), the CPU will be in secure mode with the Protection Context set to 2. The value stored in SRSS BOOT ENTRY must be accessible to Secure/PC2 programs*
- *Peripherals that are inactive during Deep Sleep mode will be reset upon waking from Deep Sleep RAM or Deep Sleep OFF, requiring re-initialization after a warm boot*

18.2.1.4 Deep Sleep OFF

In Active mode, logic and memories are not retained at all, necessitating a full 'cold' reboot to restore the application to its previous state.

During Deep Sleep, logic is functional, but the Active logic and SRAM are powered off. When a wakeup occurs, it results in a reset (cold reboot).

In Deep Sleep OFF mode, SRAM retention is not available.

Table 114 outlines the PSC3M/P8, PSC3M/P7 wake-up behavior after a specific Deep Sleep state.

Table 114 Deep Sleep states and wake-up behavior

Deep Sleep state	After wakeup
Deep Sleep	Wake up from WFI/WFE

(table continues...)

Table 114 (continued) Deep Sleep states and wake-up behavior

Deep Sleep state	After wakeup
Deep Sleep RAM	Warm reboot
Deep Sleep OFF	Cold reboot

18.2.1.5 Hibernate

Hibernate mode is the lowest power mode of the device, designed for applications in a dormant state while external supplies are still present and XRES is de-asserted. In this mode, both the Active and Deep Sleep regulators are turned off, and all clocks are stopped. Additionally, GPIO states are frozen, and only asynchronous events, such as transitions on designated I/O pins and possible reset events, are supported. It is possible to use the comparators in this mode, at the expense of low-speed comparator current. This mode requires the device to undergo a reset.

Upon emerging from Hibernate mode, the Boot ROM code detects the chip's arrival at that state and executes the minimum required code to resume operations, taking advantage of retained data as appropriate. The GPIO state is retained in this mode. Wake-up from Hibernate mode is facilitated through dedicated wake-up pins (P2.1 and P7.0) and a low-power comparator output. The low-power comparator operation in Hibernate mode requires external voltages for wake-up comparison. Additionally, an RTC alarm and the watchdog timer (WDT) interrupt can generate a Hibernate wake-up signal. The configuration of the wake-up source from Hibernate mode can be done using the PWR_HIB_WAKE_CTL register, for example, by setting the HIB_WAKE_RTC of the PWR_HIB_WAKE_CTL register to enable the RTC alarm wake-up from Hibernate mode. User code can differentiate between a hibernate reset and a cold reset by using the Cy_SysLib_GetResetReason PDL function.

To prevent accidental entry into Hibernate mode in applications that cannot meet its requirements, an option to disable the Hibernate mode is provided. This can be achieved by setting the HIBERNATE_DISABLE bit [30] of the PWR_HIBERNATE register to disable Hibernate mode in the device. Note that this bit is a write-once bit during execution and is cleared only on reset. When entering Hibernate mode, debug functionality is lost, and the debugger disconnects.

The primary difference between Hibernate and Deep Sleep OFF modes is that in the latter, the Deep Sleep logic and regulators are operational, while in Hibernate, only VDDD-supplied logic and backup domain are available.

18.2.2 Other operation states

In addition to the power modes discussed in the previous sections, the device can also be in two other states – Reset and Off states. These states are determined by the external power supply and XRES connections. Entering these modes does not require any firmware action, and exiting them does not necessitate an interrupt or wake-up event. The inclusion of the discussion on the Reset and Off states aims to cover all potential modes and states that the device can enter. These states can be utilized in a system to optimize further power consumption.

18.2.2.1 Reset state (XRES)

Reset is the device state when an external reset (XRES pin pulled low) is applied or when power-on-reset/brownout detection is asserted. Reset does not function as a power mode. During the reset state, all the components in the device are powered down, and I/Os are tristated, keeping the power consumption to a minimum.

18.2.2.2 Off state (OFF)

The off-state simply represents the device state with no power applied.

18.3 Power mode transitions

Figure 65 shows the various states that the device can be in, along with possible power mode transition paths.

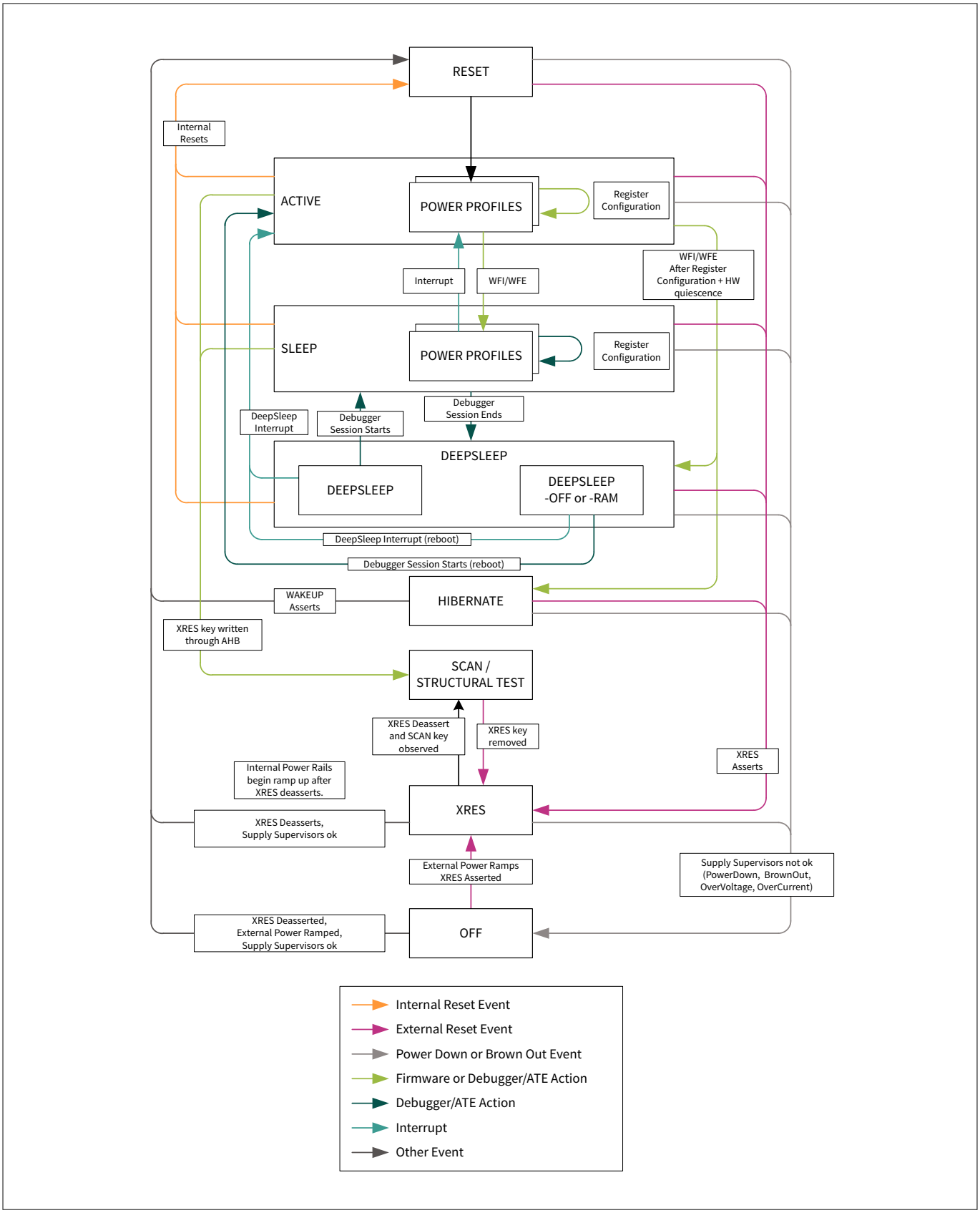


Figure 65 Power mode transitions

18.4 Summary

Table 115 shows the available resources that are active in certain power modes and their corresponding wake-up parameters.

Table 115 Power modes and wake-up parameters

	Active/Sleep	Deep Sleep	Deep Sleep-RAM	Deep Sleep-OFF	Hibernate	Off
Parameters						
Wake source[1)]	Any interrupt	DS peripherals	DS peripherals	DS peripherals	RTC/HIB peripherals	Power on
Wake action	Resume	Resume	Warm boot	Reset/cold boot	Reset	Reset
Wake time	One CPU cycle	<20 µs	Deep Sleep + warm boot	Deep Sleep + cold boot	POR + cold boot <= 4 ms	
Resources						
ECO	On/Off	On/Off	On/Off	On/Off	Off	Off
IHO	On	Off	Off	Off	Off	Off
IMO	On	On/Off	Off	Off	Off	Off
ILO	On/Off	On/Off	On/Off	On/Off	Off	Off
WCO	On/Off	On/Off	On/Off	On/Off	On/Off	Off
CPU	On/Sleep	Retention	Off	Off	Off	Off
PPCA (except SRAM)	On	Off	Off	Off	Off	Off
System SRAM	On	On/Off	On/Off	Off	Off	Off
PPCA SRAM	On	On	On [2)]	Off	Off	Off

1) See Table 116 for the list of peripherals available in DS (Deep Sleep) and HIB (Hibernate) power modes.
2) Individually configurable via the PPUs for each SRAM

The operational states available in low-power modes are generally limited in functionality and parametric performance compared to their capabilities in the Active modes. Additionally, blocks that do not support low-power modes, such as Deep Sleep and Hibernate, cannot wake up the CPUs from these modes.

Table 116 outlines the availability of various device components during the device power modes/states.

Table 116 Resources available in different power modes

Block	Power mode				
	Active	Sleep	Deep Sleep	Hibernate	Backup
CPUSS					
CPU	Y	N	N	N	N
NVIC	Y	Y	N	N	N
WIC	Y	Y	Y	N	N
FLASH	Y	Y	N	N	N

(table continues...)

Table 116 (continued) Resources available in different power modes

Block	Power mode				
	Active	Sleep	Deep Sleep	Hibernate	Backup
SRAM	Y	Y	Y	N	N
DMA	Y	Y	N	N	N
Peripherals					
TCPWM (also in PPCA)	Y	Y	N	N	N
SCB	Y	Y	Y[1)]	N	N
CAN FD	Y	Y	N	N	N
LPComp	Y	Y	Y [2)]	Y	N
PPCA					
CPUs	Y	Y	N	N	N
SRAM	Y	Y	Y	N	N
CORDIC	Y	Y	N	N	N
ADC	Y	Y	N	N	N
DCSG	Y	Y	N	N	N
Custom logic block	Y	Y	N	N	N
Hardware filters	Y	Y	N	N	N
I/O					
GPIO	Y	Y	Y	Y[3)]	N
Backup					
RTC	Y	Y	Y	Y	Y
Registers	Y	Y	Y	Y	Y

1) Only SCB 0 (I^2C, SPI)
2) Only in Deep Sleep mode. Not available in Deep Sleep RAM and Deep Sleep OFF modes.
3) Only hibernate_wakeup pins (P2.1 and P7.0) are operational and capable of waking up the device from Hibernate mode.

18.5 Registers summary

Table 117 Register list

Name	Description
PWR_CTL	Controls the device's power mode options and enables observation of the current state.
PWR_CTL2	
PWR_HIBERNATE	Controls various Hibernate mode entry/exit related options.

(table continues...)

Table 117 (continued) Register list

Name	Description
PWR_HIB_WAKE_CTL	The Hibernate Wake-up Mask register sets the hibernate assignment wake-up and triggers recording.
PWR_HIB_WAKE_CTL2	The Hibernate Wake-up Polarity register selects the polarity for the corresponding hibernate wake-up sources.
PWR_HIB_WAKE_CAUSE	The Hibernate Wake-up Cause register records the hibernate unmasked wake-up causes and retains them during the Hibernate mode.

19 Clocking system

The PSC3M/P8, PSC3M/P7 MCU family provides flexible clocking options with on-chip oscillators, phase-lock loop (PLL), and supports multiple external clock sources.

19.1 Features

The PSC3M/P8, PSC3M/P7 MCU clock system includes the following resources:

- Three internal clock sources:
  - 8-MHz internal main oscillator (IMO)
  - 48-MHz internal high-speed oscillator (IHO)
  - 32-kHz internal low-speed oscillator (ILO)
- Three external clock sources:
  - Up to 80-MHz external clock (EXT_CLK) input from an I/O pin
  - 4- to 35-MHz external crystal oscillator (ECO)
  - 32.768-kHz external watch crystal oscillator (WCO)
- One frequency lock loop (FLL) with 24- to100-MHz output range
- Two digital phase-locked loops (DPLL0 and DPLL1) with 25- to 250-MHz output range
- Each clock source has a clock supervisor (CSV)

19.2 Architecture

Figure 66 shows the generic view of the clocking system in the PSC3M/P8, PSC3M/P7 MCU family.

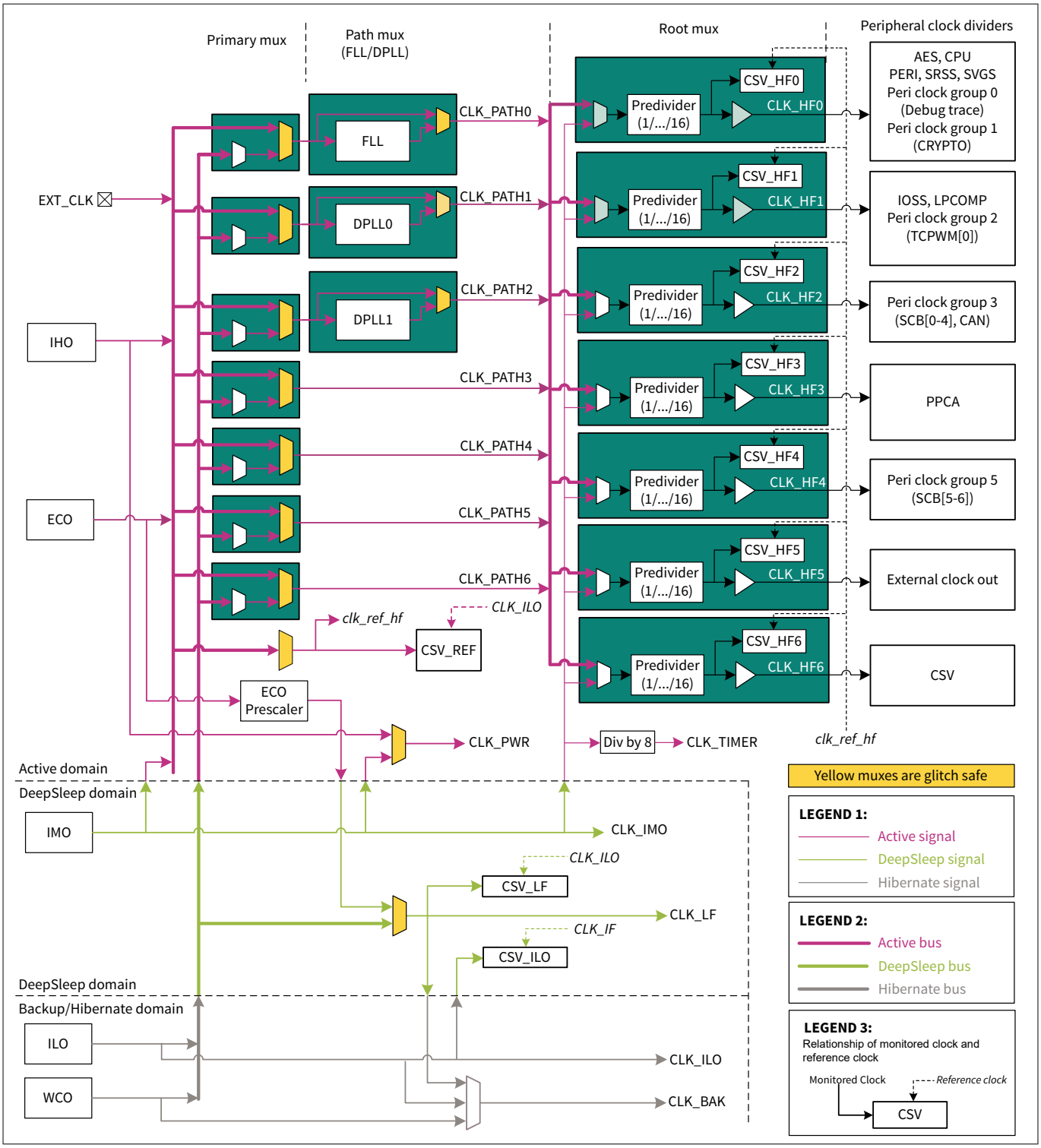


Figure 66 Clocking system block diagram

Note: *See Peripheral clock dividers section for more details about the peripheral clock dividers.*

19.3 Clock sources

19.3.1 Internal main oscillator (IMO)

The IMO is a high-speed internal (crystal-less) main oscillator with fast wake-up and low jitter that produces a fixed frequency of 8-MHz, and has a tolerance of ±2%. The IMO is the default clock source after power-on-reset (POR) or any other reset. The DPLL or FLL uses the IMO to generate a wide range of higher frequency clocks or the high-frequency root clocks use the IMO directly. The IMO is available in Active, Sleep, and DeepSleep modes.

19.3.2 Internal high-frequency oscillator (IHO)

The IHO is an accurate, high-speed internal (crystal-less) oscillator that produces a fixed frequency 48 MHz without any external components, and has a tolerance of ±1%. A high-speed clock can be derived using the IHO plus a DPLL. The IHO is only available in Active and Sleep modes.

19.3.3 Internal low-speed oscillator (ILO)

The ILO is a low-speed internal oscillator that operates with no external components, and outputs a stable clock at 32-kHz nominal frequency. The ILO 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 brown-out detection (BOD), the ILO_BACKUP bit must be set in the CLK_ILO_CONFIG register.

The ILO can be used as the clock source for CLK_LF as well as a source for the backup domain (CLK_BAK). CLK_BAK runs the real-time clock (RTC). Although the ILO is not suitable as an RTC due to its poor accuracy, it can be used as a HIBERNATE wake-up source using the wake-up alarm facility in the RTC. In this case, the VDDD rail must be supplied during hibernate for the ILO to run, and the ILO_BACKUP bit must be set in the CLK_ILO_CONFIG register.

The ILO can be used for the source of the watchdog timer (WDT). The ILO is enabled and disabled with the ENABLE bit of the CLK_ILO_CONFIG register. Leave the ILO enabled if it is the source of the WDT. If the WDT is enabled, the only way to disable the ILO is to first clear the WDT_LOCK bit in the WDT_CTL register, and then clear the ENABLE bit in the CLK_ILO_CONFIG register. If the WDT_LOCK bit is set, any register write to disable the ILO is ignored. Enabling the WDT automatically enables the ILO.

The calibration counters described in the Clock calibration counters chapter can be used to measure the ILO against a high-accuracy clock such as the ECO. This result can then be used to determine how the ILO must be adjusted. The ILO can be trimmed using the CLK_TRIM_ILO_CTL register.

19.3.4 External clock (EXT_CLK)

The external clock is a 0.001 to 80-MHz range clock that can be sourced from a signal on a designated I/O pin. This clock can be used as the source clock for either the DPLL or FLL, or can be used directly by the high-frequency clocks.

When manually configuring a pin as an input to EXT_CLK, set the drive mode of the pin to high-impedance digital to enable the digital input buffer. See the I/O subsystem chapter for more details. Consult the device datasheet to determine the specific pin used for EXT_CLK.

19.3.5 External crystal oscillator (ECO)

The PSC3M/P8, PSC3M/P7 MCU contains an oscillator to drive an external 4-MHz to 35-MHz crystal. 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 PSC3M/P8, PSC3M/P7 MCU. The hardware automatically disables the ECO during Hibernate, DeepSleep, and XRES. ECO operates only in Active mode.

The ECO can be enabled by using the CLK_ECO_CONFIG.ECO_EN register bit-fields. After the ECO is enabled, check the CLK_ECO_STATUS register to ensure it is ready. The ECO clock can be a choice for CLK_HF and CLK_LF (prescaled).

19.3.6 Watch crystal oscillator (WCO)

The WCO is a highly accurate 32.768-kHz clock source. It is the primary clock source for the RTC. The WCO can also be used as a source for CLK_LF and CLK_BAK. The WCO can be enabled and disabled by setting the WCO_EN bit in the BACKUP_CTL register 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 the WCO_BYPASS bit in the BACKUP_CTL register for the backup domain.

The WCO can also operate from external clock/sine wave inputs. In these modes, bypass the WCO by setting the WCO_BYPASS bit in the BACKUP_CTL register before enabling the WCO. Also, configure the GPIOs for the WCO_OUT and WCO_IN signals in analog mode.

19.3.7 ECO prescaler

The ECO prescaler divides the ECO, and creates a clock that can be used with a CLK_LF clock. This feature is available only during Active and Sleep modes. It cannot be used during DeepSleep or Hibernate modes.

The division function has a 10-bit integer divider and an 8-bit fractional divider. This function is configured using the CLK_ECO_PRESCALE register.

19.4 Clock generation

The PSC3M/P8, PSC3M/P7 MCU has fully integrated clocks and provides clocks to all blocks that require clocks. It switches between different clock sources without glitches, and ensures that there are no metastable conditions.

19.4.1 Digital phase-locked loop (DPLL)

DPLL is a digital PLL for the PSC3M/P8, PSC3M/P7 MCU that supports output clock frequencies of 25 to 250 MHz. The PLL uses a digital filter and an internal 250- to 500-MHz digital controlled oscillator (DCO) with an output duty cycle of 48–52%. It also contains an LDO to reject supply noise and improve jitter performance.

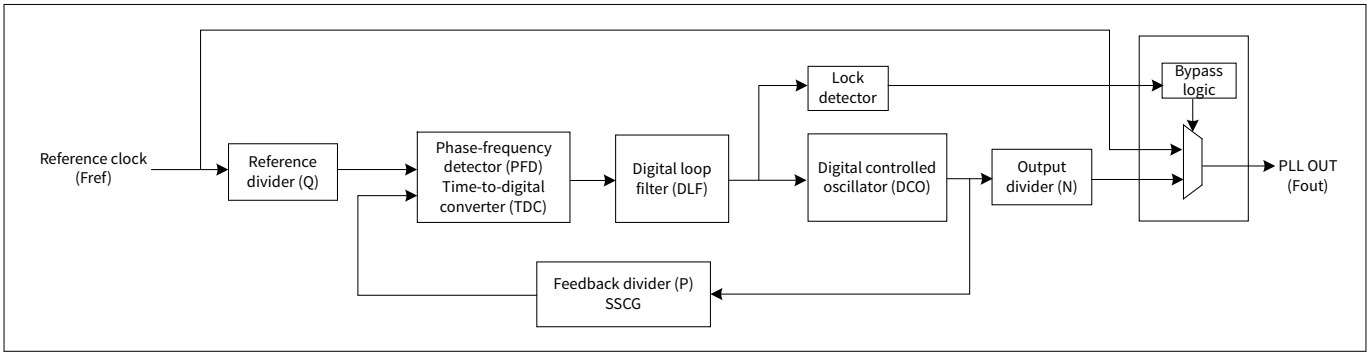


Figure 67 DPLL block diagram

Figure 67 shows the DPLL block diagram. The time to digital converter (TDC) block translates the difference between the reference clock and feedback clock into the code, which will be input to the digital loop filter (DLF). The DCO has an input of digital code and generates the output frequency accordingly.

The PLL provides multiple frequency-synthesis configurations, using programmable input and output integer dividers (not only binary) and a feedback divider (integer and fractional divider). The PLL has three programmable dividers:

- P (product): This divider is in the feedback path between the DCO output and the phase-frequency detector (PFD) input. It sets the multiplication factor of the DCO
- Q (quotient): This divider is between the input (reference) clock and the PFD. It divides the input clock before passing the same to the PFD
- N (output/post divider): This divider accepts the output clock of the DCO and divides it down to provide the desired frequency and output of the PLL

All dividers (P, Q, and N) are digitally programmable to divide their incoming clock by an integer number, while the feedback clock also supports division by a fractional number. The PLL generates a specific output frequency based on the setting of P, Q, and N dividers, and application of an appropriate reference clock, .

The PLL supports the following operation modes:

- **Integer operation mode**: The PLL generates integer modification of the input frequency by configuring P, Q, and N - it sets the PLL output frequency.

  PLL output frequency equation (Integer mode):

$$F_{out} = F_{ref} \times \left(P/Q/N\right) \tag{1}$$

  ***Note**: Fref is the input frequency of the PLL, that is, the frequency of the high-frequency clock source selected using the PATH_MUX (for example, 8 MHz from the IMO)*

- **Fractional operation mode**: The fractional frequency is defined by 24 input bits. The frequency resolution is $1/2^{24}$.

  PLL output frequency equation (Fractional mode):

$$F_{out} = \left(F_{ref}/Q\right) \times \left(P + Frac_div\right)/N \tag{2}$$

  ***Note**: The configuration of fractional operation uses the CLK_DPLL_LPx_CONFIG2 register. Frac_div is the value set by CLK_DPLL_LPx_CONFIG2.FRAC_DIV divided by 2^{24} . Frac_div = CLK_DPLL_LPx_CONFIG2.FRAC_DIV / 2^{24}.*

  For example: Non-standard frequency 40.456 MHz generated from Fref = 8 MHz

  P = 40, Q = 1, N = 8, Frac_div =7650410

  F_{out} = (8 * (40 + 7650410/16777216)) / 8 = 40.456 MHz

Follow these steps to configure the PLL:

1. Determine the desired reference clock frequency (Fref) and the desired output frequency (Fout). Calculate the reference (Q), feedback (P), and output (N) dividers subject to the following constraints:
   - **a.** PFD frequency (frequency constraints: 4 - 8 MHz). Fpfd = Fref/Q. There may be multiple reference divider values that meet this constraint
   - **b.** DCO frequency (frequency constraints: 250 - 500 MHz). DCO = Fpfd × P. There may be multiple feedback divider values that meet this constraint with different REFERENCE_DIV choices
   - **c.** Output frequency. PLL_OUT = DCO/N. 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 CLK_DPLL_LPx_CONFIG register. Do not enable the PLL during the same register write as configuring the dividers. Do not change the divider settings while the PLL is enabled
3. Configure the fractional divider in CLK_DPLL_LPx_CONFIG2 register and SSCG in CLK_DPLL_LPx_CONFIG3 register if needed

4. Enable the PLL (CLK_PLL_CONFIGx.ENABLE = 1). Wait at least 1 µs for PLL circuits to start
5. 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_DPLL_LPx_CONFIG.BYPASS_SEL. The status of the PLL can be checked by reading CLK_DPLL_LPx_STATUS. This register contains a bit indicating the PLL has locked. It also contains a bit indicating if the PLL lost the lock status

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, so 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_DPLL_LPx_CONFIG3 register.

SSCG has the following parameters:

- **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. However, the system constraints on the frequency variation must be evaluated while selecting the Modulation Depth
- **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 "F_{out} = Fc − ΔF"

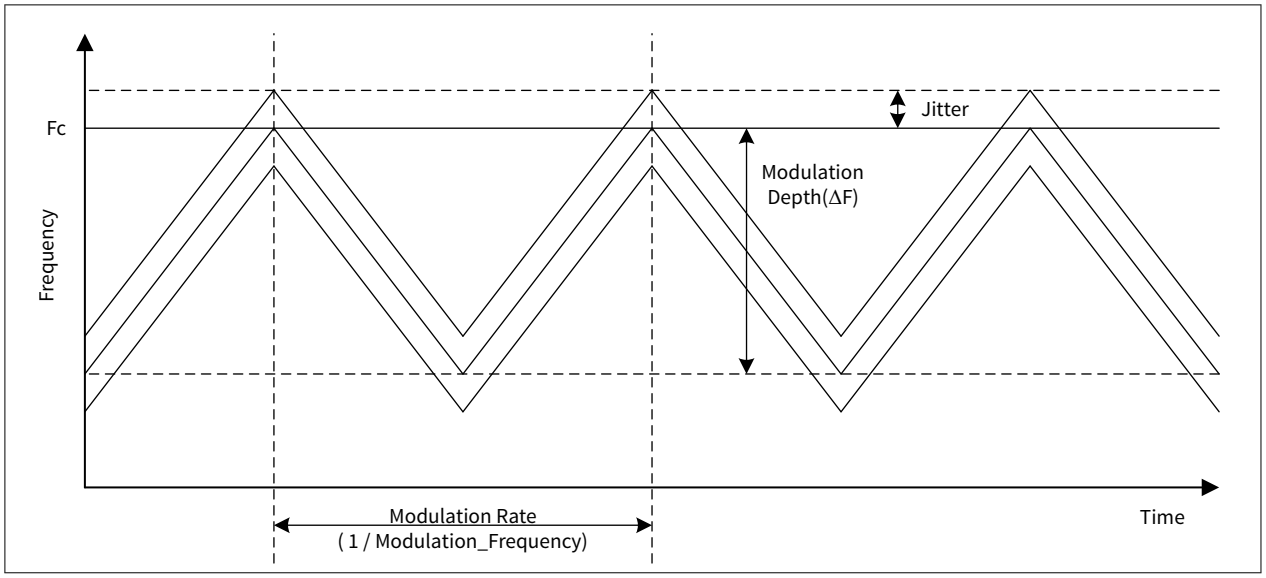


Figure 68 Down spreading example

$\overline{Fout}$ = the modulated output clock frequency

Fc = the source or carrier frequency

ΔF = total frequency deviation (min to max)

19.4.2 Frequency lock loop (FLL)

The PSC3M/P8, PSC3M/P7 MCU contains one frequency lock loop (FLL), which resides on CLK_PATH0. The FLL is capable of generating a clock output in the range 24 MHz to 100 MHz. The input frequency must be between 0.008 to 100 MHz, and must be at least 2.5 times less than the current-controlled oscillator (CCO) frequency. This makes it possible to use the IMO to generate much higher clock frequencies for the rest of the system.

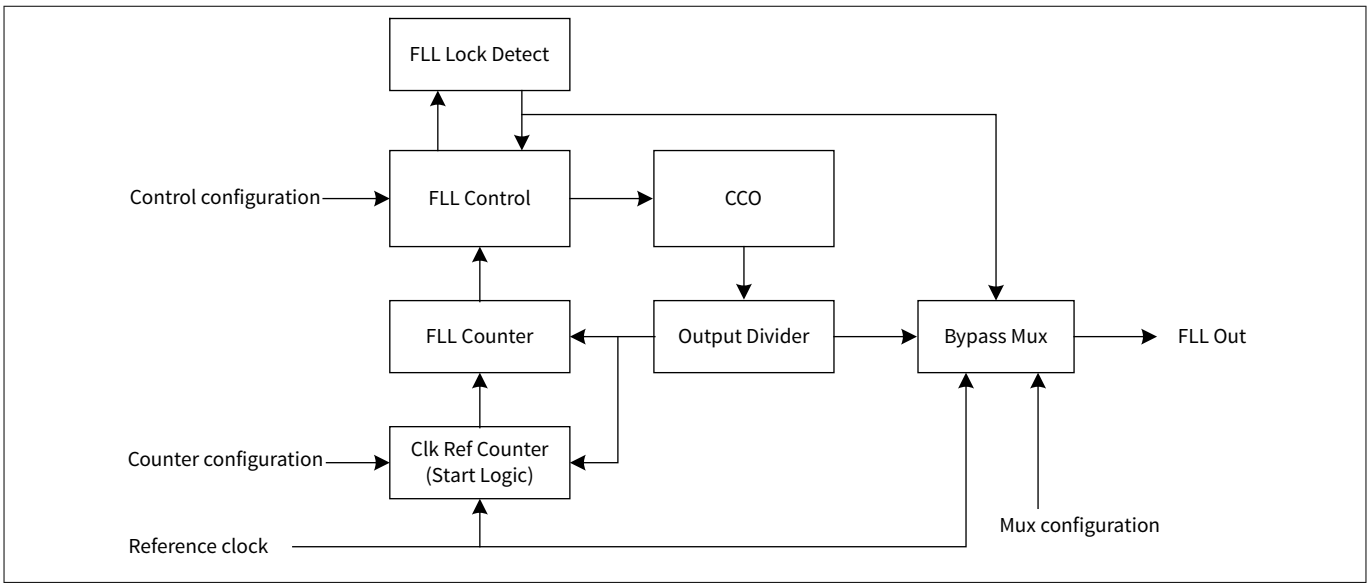


Figure 69 FLL block diagram

The FLL is similar in purpose to a PLL but is not equivalent; there are some differences:

- 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 and is explained in detail later in this section
- FLL can produce up to 100-MHz clock with good duty cycle through its divided clock output
- FLL reference clock can be the WCO (32.768 kHz), IMO (8 MHz), or any other periodic clock source

The CCO can output a stable frequency in the 48 MHz to 200 MHz range. This range is divided into five sub-ranges as shown by Table 118.

Table 118 CCO frequency ranges

CCO range	0	1	2	3	4
f_{min}	48 MHz	64 MHz	85 MHz	113 MHz	150 MHz
f_{Maximum}	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.*

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 below.

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 counts 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 the FLL_REF_DIV field in the CLK_FLL_CONFIG2 register.

After the CCO clocks are counted, they are compared against an ideal value and an error is calculated. The ideal value is programmed into the FLL_MULT field of the CLK_FLL_CONFIG register.

As an example, the reference clock is the IMO (8 MHz), the desired CCO frequency is 100 MHz, the value for FLL_REF_DIV is set to 146. This means that the FLL counts 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 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 FLL_LF_IGAIN and FLL_LF_PGAIN found in CLK_FLL_CONFIG3. Figure 70 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. The CCO_LIMIT field in the CLK_FLL_CONFIG4 can be used to put an upper limit on the trim adjustment; this is not needed for most situations.

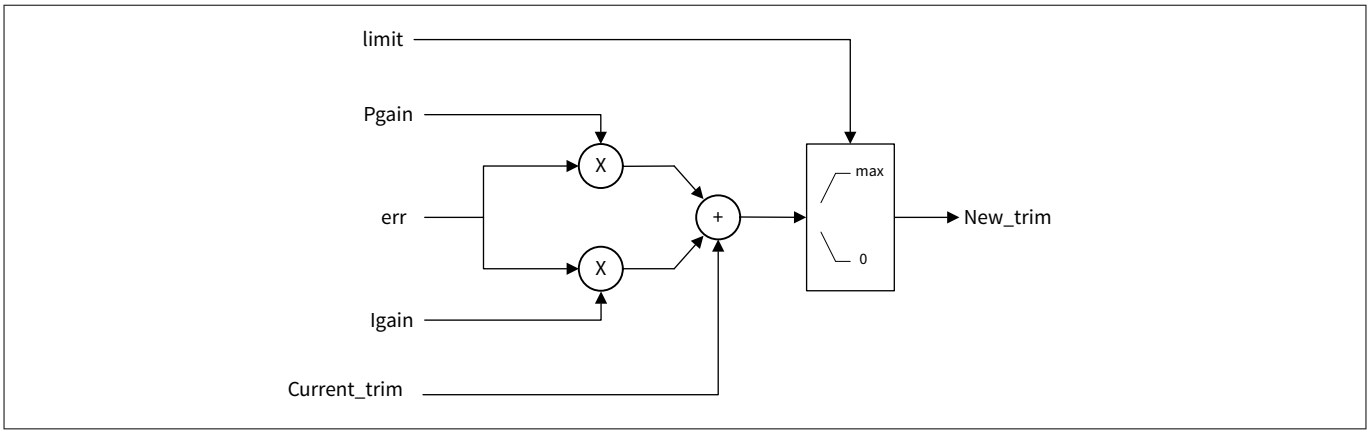


Figure 70 FLL error correction diagram

The FLL determines whether it is “locked” by comparing the error term with the LOCK_TOL field in the CLK_FLL_CONFIG2 register. 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 SETTLING_COUNT field of CLK_FLL_CONFIG3. 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 REF_DIV. Higher REF_DIV values lead to higher accuracy, whereas lower 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 REF_DIV is set to ‘1’, then FLL_MULT must be set to either ‘13’ or ‘12’. This results 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 REF_DIV. However, the larger REF_DIV is, the longer each measurement cycle takes, so increasing the lock time. In this example, REF_DIV was set to 146. This means that each measurement cycle takes 146 * (1/8 MHz) = 18.25 µs, whereas when REF_DIV is set to 1, each measurement cycle takes 1 * (1/8 MHz) 0.125 µs.

Another issue with lower REF_DIV values is that the minimum LOCK_TOL is 1, so the output of the CCO can have an error of ±1. In the example where REF_DIV = 1 and FLL_MULT = 13, the MULT value can really be 12, 13, or 14 and still be locked. This means that the output of the FLL may vary between 96 and 112 MHz, which may not be desirable.

Thus, a choice must be made between faster lock times and more accurate FLL outputs. The biggest change to make for this is the value of REF_DIV. The CLK_FLL_STATUS register checks the status of FLL.

Enabling and disabling the FLL

The FLL requires firmware sequencing when enabling, disabling, and entering/exiting DeepSleep.

Follow these steps to enable FLL:

1. Enable the CCO by writing CLK_FLL_CONFIG4.CCO_ENABLE = 1 and wait until CLK_FLL_STATUS.CCO_READY == 1
2. Ensure that the reference clock has stabilized and CLK_FLL_CONFIG3.BYPASS_SEL = FLL_REF
3. Write FLL_ENABLE = 1 and wait until CLK_FLL_STATUS.LOCKED == 1
4. Write CLK_FLL_CONFIG3.BYPASS_SEL = FLL_OUT to switch to the FLL output

Follow these steps to disable FLL:

1. Ensure that the processor is operating from a different clock than Clock Path 0. If the muxes are changed, wait four FLL output clock cycles for it to complete
2. Write CLK_FLL_CONFIG3.BYPASS_SEL = FLL_REF and read the same register to ensure that the write completes
3. Wait at least six FLL reference clock cycles and disable it with FLL_ENABLE = 0
4. Disable the CCO by writing CLK_FLL_CONFIG4.CCO_ENABLE = 0

Before entering DeepSleep, either disable the FLL using the above sequence or use the following procedure to deselect/select it before/after DeepSleep. Before entering DeepSleep, write CLK_FLL_CONFIG3.BYPASS_SEL =FLL_REF to change the FLL to use its reference clock. After DEEPSLEEP wake-up, wait until CLK_FLL_STATUS.LOCKED==1 and then write CLK_FLL_CONFIG3.BYPASS_SEL = FLL_OUT to switch to the FLL output.

***Note**: It is not recommended to use the FLL_AUTO option in the BYPASS_SEL field.*

19.5 Clock trees

The PSC3M/P8, PSC3M/P7 MCU clocks are distributed throughout the device, as shown in Figure 66. The clock trees are as follows:

- Path clocks
- High-frequency root clocks (CLK_HF)
- Low-frequency clock (CLK_LF)
- Backup clock (CLK_BAK)
- Timer clock (CLK_TIMER)
- Power clock (CLK_PWR)

19.5.1 Clock path

The PSC3M/P8, PSC3M/P7 MCU has seven clock paths: CLK_PATH0 contains the FLL, CLK_PATH1 contains DPLL0, CLK_PATH2 contains DPLL1, CLK_PATH3, CLK_PATH4, and CLK_PATH5 and CLK_PATH6 are a direct connection to the high-frequency root clocks. Note that the FLL and DPLL 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 which source clock will clock that path. This configuration is done in the CLK_PATH_SELECTx register.

Table 119 Clock path source selections

Name	Description
PATH_MUX[2:0]	Selects the source for CLK_PATHx 0: IMO - Internal main oscillator 1: EXT_CLK - External clock pin 2: ECO - External-crystal oscillator 3: Reserved. Do not use 4: DSI_MUX 5: Reserved. Do not use 6: IHO - Internal high-speed oscillator 7: Reserved. Do not use

***Note**: Switching away from the originally selected clock requires four cycles of it. Do not disable the original clock during this time.*

The DSI mux is configured through the CLK_DSI_SELECTx register. Table 120 shows the DSI mux source selections.

Table 120 DSI mux source selections

Name	Description
DSI_MUX[4:0]	Selects the source for the DSI_MUXx 0-15: Reserved. Do not use 16: ILO - Internal low-speed oscillator 17: WCO - Watch crystal oscillator 18-31: Reserved. Do not use

19.5.2 High-frequency root clocks (CLK_HF)

The PSC3M/P8, PSC3M/P7 MCU has seven high-frequency root clocks (CLK_HF[0 - 6]). Each CLK_HF has a particular destination on the device, as shown in Table 121. See the Peripheral clock divider groups chapter for more details on the Peri clock group.

Table 121 CLK_HF destinations

Name	Description
CLK_HF[0]	AES, CPU, PERI, AHB infrastructure, SRSS, SVGS, Peri clock group 0 (Debug trace), Peri clock group 1 (CRYPTO)
CLK_HF[1]	IOSS, LPCOMP, Peri clock group 2 (TCPWM[0])
CLK_HF[2]	Peri clock group 3 (CAN, SCB [0 - 4])
CLK_HF[3]	PPCA
CLK_HF[4]	Peri clock group 5 (SCB[5-6])
CLK_HF[5]	Clock output on clk_ext pin (when used as an output)
CLK_HF[6]	CSV

Each high-frequency root clock has a mux to determine its source and predivider. The root source can be IMO or ROOT_MUX by the DIRECT_MUX bit of CLK_DIRECT_SELECT[i] register selection show in.

Table 122 Clock root direct select register

Name	Description
DIRECT_MUX[8]	CLK_HF the direct select mux configuration: 0: IMO 1: ROOT_MUX, see CLK_ROOT_SELECT register

The ROOT_MUX and predivider configuration is done in the CLK_ROOT_SELECT[i] register shown in.

Table 123 Clock root select register

Name	Description
ROOT_MUX[3: 0]	CLK_HF input clock path selection: 0: Select CLK_PATH0 1: Select CLK_PATH1 2: Select CLK_PATH2 3: Select CLK_PATH3 4: Select CLK_PATH4 5: Select CLK_PATH5 6: Select CLK_PATH6 [7: 15]: Reserved
ROOT_DIV[11: 8]	Selects predivider value for the clock root: 0: No Divider 1: Divide selected clock source by 2 2: Divide selected clock source by 3 3: Divide selected clock source by 4 4: Divide selected clock source by 5 5: Divide selected clock source by 6 6: Divide selected clock source by 7 7: Divide selected clock source by 8 8: Divide selected clock source by 9 9: Divide selected clock source by 10 10: Divide selected clock source by 11 11: Divide selected clock source by 12 12: Divide selected clock source by 13 13: Divide selected clock source by 14 14: Divide selected clock source by 15 15: Divide selected clock source by 16
ENABLE	Enable for this clock root. All clock roots default to disabled (ENABLE==0) except HFCLK0, which cannot be disabled.

***Note**: CLK_HF[1-6] can be enabled and disabled. CLK_HF[0] is always enabled as it is the source of the CPU. To enable and disable CLK_HF[1-6] set the ENABLE bit in the CLK_ROOT_SELECT register.*

19.5.3 Low-frequency clock (CLK_LF)

The low-frequency clock (CLK_LF) in the PSC3M/P8, PSC3M/P7 MCU has three input options: ILO, WCO, and ECO prescaler. CLK_LF is the source for the multi-counter watchdog timers (MCWDTs) and the RTC.

The source of CLK_LF is set in the LFCLK_SEL bits of the CLK_SELECT register. Writes to this field are ignored unless the WDT is unlocked using the WDT_LOCK register; see the Watchdog timer chapter for the details. It takes four cycles of the originally selected clock to switch away from it. Do not disable the original clock during this time.

Table 124 LFCLK input selection bits LFCLK_SEL

Name	Description
LFCLK_SEL[2:0]	LFCLK input clock selection: 0: ILO - Internal low-speed oscillator 1: WCO - Watch-crystal oscillator. Requires Backup domain to be present and properly configured (including external watch crystal, if used) 2 - 4: Reserved 5: ECO_PRESCALER - External-crystal oscillator after prescaling. Does not work in DEEPSLEEP or HIBERNATE modes. Intended for applications that operate in ACTIVE/SLEEP modes only. This option is only valid when ECO is present in the product 6: Reserved

19.5.4 Backup clock (CLK_BAK)

The backup clock is used to clock the backup domain, specifically the RTC. For more information, see Backup chapter.

19.5.5 Timer clock (CLK_TIMER)

The timer clock (CLK_TIMER) can be used as a clock source for the CPU SYSTICK timer. The timer clock is standardized as a 1-MHz (fixed) clock derived from IMO, which is always running in Active/Sleep modes. It turns off during other modes, such as DeepSleep. In the PSC3M/P8, PSC3M/P7 MCU, there is no pre-divider and it only provides a fixed 1-MHz frequency, which is also a divided-by-8 clock from CLK_IMO. The SRSS constraints automatically create the relevant internal clocks.

19.5.6 Power clock (CLK_PWR)

The power clock (CLK_PWR) is used to clock the Arm® power architecture components, Power Police Unit (PPU) and Power Dependency Control Matrix (PDCM). The CLK_PWR is always running in Active/Sleep modes. It turns off during other modes, such as DeepSleep mode.

The source of power clock can be the IMO or IHO; it is set in the CLK_PWR_MUX bits of the PWRMODE_CLK_SELECT register. The divide value is set by the CLK_PWR_DIV bits of this register.

Table 125 Power mode clock select register

Name	Description
CLK_PWR_DIV[7: 0]	CLK_PWR is generated by dividing the CLK_PWR_MUX selection by (CLK_PWR_DIV+1)
CLK_PWR_MUX[17: 16]	Selects the source for the power clock: 0: IMO 1: IHO 2: Reserved

19.6 CLK_HF distribution

The PSC3M/P8, PSC3M/P7 MCU has several CLK_HFs, which connect the CPU subsystem, SRSS, and peripheral functions. Also, a part of CLK_HF connects to CSV. See the High-frequency root clocks (CLK_HF) chapter for more details. Some peripherals connect to CLK_HF by the peripheral clock divider (see the Peripheral clock dividers chapter for the details).

The following table lists the details of the CLK_HF distribution network, and the CPU and peripheral maximum frequency under the different active modes. The PSC3M/P8, PSC3M/P7 MCU supports four Active modes: Ultra Low Power (ULP) mode (0.9 V), Medium Frequency (MF) mode (1.0 V), Low Power (LP) mode (1.1 V), and Over Driver (OD) mode (1.2 V). See the Device power modes chapter for more the details of power modes.

Table 126 Clock distribution and Active mode performance parameters

CLK_HF	Peripherals	Recommend source	Max permitted clock frequency [1)]			
			OD (1.2 V)	LP (1.1 V)	MF (1.0 V)	ULP (0.9 V)
CLK_HF0	AES, CPU, PERI, SRSS, SVGS, Peri clock group 0 (Debug trace), Peri clock group 1 (CRYPTO)	DPPL0	180 MHz	150 MHz	70 MHz	50 MHz
CLK_HF1	IOSS, LPCOMP, Peri clock group 2 (TCPWM[0])	DPLL0	180 MHz	150 MHz	70 MHz	50 MHz
CLK_HF2	Peri clock group 3 (CAN, SCB[0:4])	FLL	100 MHz	100 MHz	25 MHz	25 MHz
CLK_HF3	PPCA	DPLL1	200 MHz	150 MHz	70 MHz	50 MHz
CLK_HF4	Peri clock group 5 (SCB[5-6])	DPLL1 (For fast SPI)	200 MHz	150 MHz	70 MHz	50 MHz
		FLL (For UART/I^2C)	100 MHz	100 MHz	25 MHz	25 MHz
CLK_HF5	EXT_CLK_OUT	All	80 MHz	80 MHz	80 MHz	50 MHz
CLK_HF6	CSV	All	8 MHz	8 MHz	8 MHz	8 MHz

1) Maximum clock frequency after the corresponding clock source (DPLL/FLL + dividers). All internal tolerances and affects are covered by these frequencies.

19.7 Peripheral clock dividers

The PSC3M/P8, PSC3M/P7 MCU peripherals, such as SCBs and TCPWMs, require a clock. These peripherals can be clocked only by a peripheral clock divider. The PSC3M/P8, PSC3M/P7 MCU has 24 peripheral clock (PCLK) dividers. The output of any of these dividers can be routed to any peripheral within the group.

Clocks can be divided down to generate synchronous clocks for the analog and digital peripherals. The following integer and fractional clock dividers are provided:

- Thirteen 8-bit integer clock dividers
- Five 16-bit integer clock dividers
- Four 16.5 fractional clock dividers (16 integer bits, 5 fractional bits)
- Two 24.5 fractional clock divider (24 integer bits, 5 fractional bits)

19.7.1 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 16.5-bit divider with an integer divide value of 4 generates a 12-MHz clock from a 48-MHz CLK_PERI. A 16.5-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; some will have a 16-MHz period and others will have a 12-MHz period, such that the average is 13.7 MHz.

Fractional dividers are useful when a high-precision clock is required (for example, for 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.

19.7.2 Peripheral clock divider configuration

The peripheral clock dividers are configured using registers from the peripheral block; specifically DIV_CMD, DIV_8_CTL, DIV_16_CTL, DIV_16_5_CTL, DIV_24_5_CTL, and CLOCK_CTL registers.

First, configure the clock divider via the DIV_8_CTL, DIV_16_CTL, DIV_16_5_CTL, and DIV_24_5_CTL registers. There is one register for each divider; for example, there are eight DIV_8_CTL registers as there are eight 8-bit dividers. In these registers, set the value of the integer divider; if it is a fractional divider, then set the fraction portion as well.

After the divider is configured, use the DIV_CMD register to enable the divider. This is done by setting the DIV_SEL to the divider number you want to enable, and setting the TYPE_SEL to the divider type. For example, if you want to enable the 0th 16.5-bit divider, write ‘0’ to DIV_SEL and ‘2’ to TYPE_SEL. If you want to enable the tenth 16-bit divider, write ‘10’ to DIV_SEL and ‘1’ to TYPE_SEL. See the registers reference manual for more details.

To connect a peripheral to a specific divider, the CLOCK_CTL register is used. There is a CLOCK_CTL register corresponding to each PCLK of the clock divider group mentioned in the Peripheral clock divider groups chapter. For example, to select the second 8-bit divider for PCLK with group 5 and output 3, configure DIV_SEL to ‘2’ and TYPE_SEL to ‘1’ in the PERI_PCLK_GR5_CLOCK_CTL3 register. Also, the ‘Output’ order matches the order of y of the PERI_PCLK_GRx_CLOCK_CTLy register.

Phase-aligning dividers

For specific use cases, 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 appear as follows.

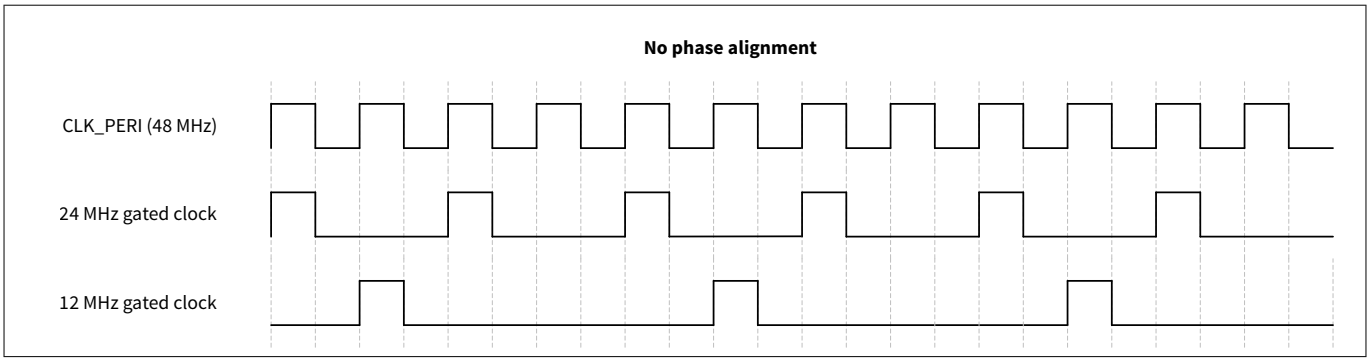


Figure 71 Non phase-aligned clock dividers

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 it s 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 dividers that 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 aligned clocks appear as follows.

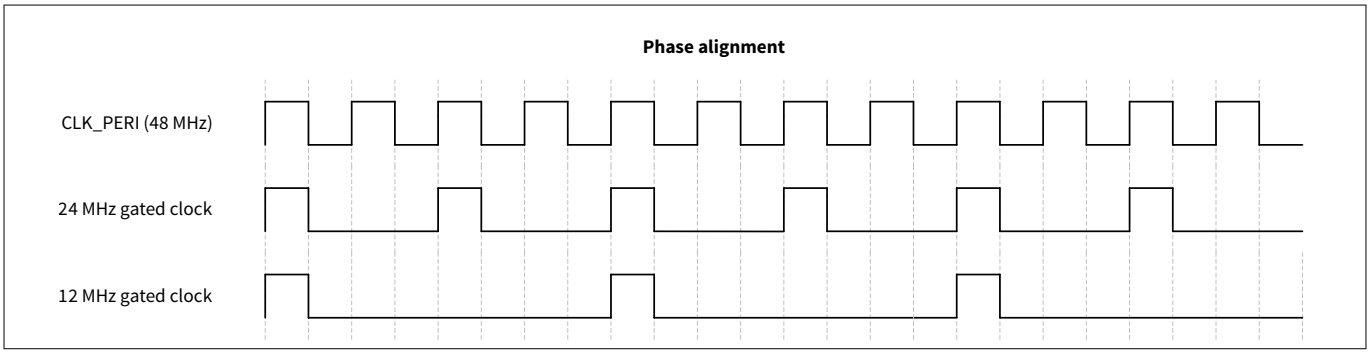


Figure 72 Phase-aligned clock dividers

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 appear as follows.

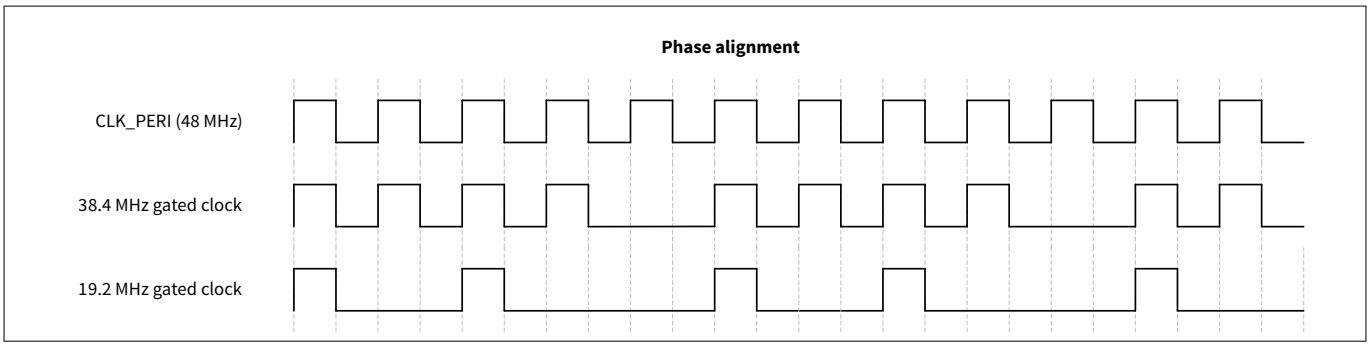


Figure 73 Phase-aligned fractional dividers

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.

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 the DeepSleep 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 the DeepSleep mode. Therefore, when transitioning from DeepSleep to Active mode, all dividers (and clock signals) are enabled and phase-aligned by design.

Phase alignment is accomplished by setting the PA_DIV_SEL and PA_DIV _TYPE bits in the DIV_CMD register before enabling the clock. For example, to align the fourth 8-bit divider to the third 16-bit divider, set DIV_SEL to '4', TYPE_SEL to '0', PA_DIV_SEL to '3', and PA_TYPE_SEL to '1'.

19.7.3 Peripheral clock divider groups

The PSC3M/P8, PSC3M/P7 MCU has seven PCLK clock divider groups, the outputs of each group are connected to a specific peripheral. For a complete list of peripherals, refer to the following clock groups table.

Peri clock group 0

- Root clock: CLK_HF[0]
- Divider count: 1
- Output count: 1
- 8 bits integer divider count: 0
- 16 bits integer divider count: 0
- 16.5 bits fractional divider count: 0
- 24.5 bits fractional divider count: 1

Table 127 Peri clock group 0: divided clock outputs

Output Number	Destination
0	Debug trace clock (debug600.clock_trace_in)

***Note**: Debug trace clock directly connects to CLK_HF[0].*

Peri clock group 1

- Root clock: HF[0]
- Divider count: 0
- Output count: 1
- 8-bit integer divider count: 0
- 16-bit integer divider count: 0
- 16.5-bit fractional divider count: 0
- 24.5-bit fractional divider count: 0

Table 128 Peri clock group 1: divided clock outputs

Output number	Destination
0	CRYPTO (cryptolite.clk_hf)

***Note**: CRYPTO peripheral directly connect to CLK_HF[0].*

Peri clock group 2

- Root clock: HF[1]
- Divider count: 5
- Output count: 5

- 8-bit integer divider count: 1
- 16-bit integer divider count: 4
- 16.5-bit fractional divider count: 0
- 24.5-bit fractional divider count: 0

Table 129 Peri clock group 2: divided clock outputs

Output number	Destination
0	IOSS (ioss.clk_hf)
1	TCPWM group 0 32-bit counter 0 (tcpwm[0].clock_counter_en[0])
2	TCPWM group 0 32-bit counter 1 (tcpwm[0].clock_counter_en[1])
3	TCPWM group 1 16-bit counter 0 (tcpwm[0].clock_counter_en[256])
4	TCPWM group 1 16-bit counter 1 (tcpwm[0].clock_counter_en[257])

***Note**: The I/O subsystem directly connects to CLK_HF[1].*

Peri clock group 3

- Root clock: HF[2]
- Divider count: 8
- Output count: 12
- 8-bit integer divider count: 1
- 16-bit integer divider count: 2
- 16.5-bit fractional divider count: 4
- 24.5-bit fractional divider count: 1

Table 130 Peri clock group 3: divided clock outputs

Output number	Destination
0	SCB0 (scb[0].clock_scb_en)
1	SCB0 (scb[0].clock_tgs_en)
2	SCB1 (scb[1].clock_scb_en)
3	SCB1 (scb[1].clock_tgs_en)
4	SCB2 (scb[2].clock_scb_en)
5	SCB2 (scb[2].clock_tgs_en)
6	SCB3 (scb[3].clock_scb_en)
7	SCB3 (scb[3].clock_tgs_en)
8	SCB4 (scb[4].clock_scb_en)
9	SCB4 (scb[4].clock_tgs_en)
10	CAN 0 channel 0 (canfd[0].clock_can_en[0])
11	CAN 0 channel 1 (canfd[0].clock_can_en[1])

Peri clock group 5

- Root clock: HF[4]
- Divider count: 3
- Output count: 4

- 8-bit integer divider count: 0
- 16-bit integer divider count: 1
- 16.5-bit fractional divider count: 2
- 24.5-bit fractional divider count: 0

Table 131 Peri clock group 5: divided clock outputs

Output number	Destination
0	SCB5 (scb[5].clock_scb_en])
1	SCB5 (scb[5].clock_tgs_en])
2	SCB6 (scb[6].clock_scb_en])
3	SCB6 (scb[6].clock_tgs_en])

19.8 Clock calibration counters

The clocking system in the PSC3M/P8, PSC3M/P7 MCU has built-in hardware calibration counters. Use these counters 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 ILO. Use the result of this measurement to trim the ILO.

There are two counters:

1. Calibration counter 1 is clocked off of calibration clock 1 (generally the high-accuracy clock) and it counts down
2. 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, determine the frequency of calibration clock 2 with the following equation:

$$Calibration\ clock\ 2\ frequency = \frac{Counter\ 2\ final\ value}{Counter\ 1\ final\ value} \times Calibration\ clock\ 1\ requency \quad (3)$$

For example, if calibration clock 1 = 8 MHz, Counter 1 = 1000, and Counter 2 = 5

$$Calibration\ clock\ 2\ frequency = \frac{5}{1000} \times 8\ MHz = 40\ KHz \quad (4)$$

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 the CLK_CAL_CNT1 register. Calibration counter 2 can be read in the CLK_CAL_CNT2 register.

When calibration counter 1 reaches 0, the CAL_COUNTER_DONE bit is set in the CLK_CAL_CNT1 register.

19.9 Clock supervision (CSV)

The clock supervision (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 low frequency.

The PSC3M/P8, PSC3M/P7 MCU clocking system has CSV_HFs, CSV_REF, CSV_LF, and CSV_ILO to monitor all high-frequency root clocks (CLK_HFs), reference clock (CLK_REF_HF), low-frequency clock (CLK_LF), and internal low-speed oscillator(ILO) (see Figure 66 for the CSV connection). The CSV has the following features:

- 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 (Register: CSV_REF_SEL). Typically, the IMO is selected (default)
- ILO is supervised both by CLK_LF and CLK_REF_HF; CLK_LF is needed for DeepSleep supervision and CLK_REF_HF is needed for accuracy while Active
- CSV_REF monitor CLK_REF_HF with CLK_ILO
- All CSV_HFs and CSV_REF are in the Active domain
- The DeepSleep domain has two CSVs (CSV_LF and CSV_ILO). CSV_LF is used to monitor the selected CLK_LF clock with ILO. CSV_ILO is used to monitor ILO with CLK_LF

CSV operation

The basic operation principle of the CSV circuit is as follows:

- The monitored clock generates a Monitor event (Period), and the reference clock generates a Lower and Upper limit
- The Monitor event is compared against a Lower or Upper limit
- An error is reported if a Monitor event ≤ Lower limit, or Monitor event > Upper limit

***Note**: Period is the monitored clock count, while Target is the reference clock count. Their time periods are the same in an ideal situation.*

The Figure 74 shows an example of the signal of CSV operation.

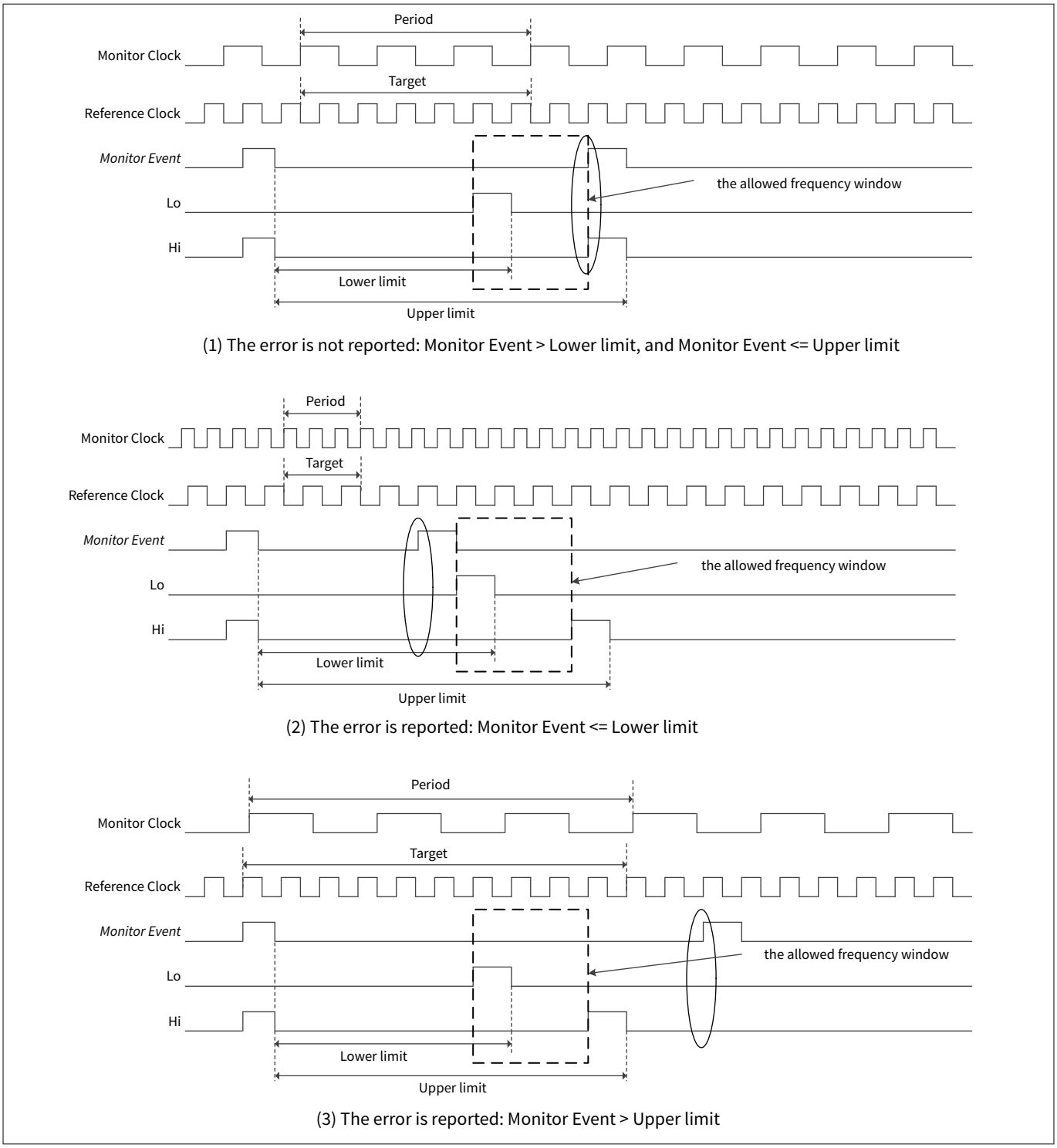


Figure 74 CSV operation signal example

The key points for CSV operation are as follows:

- Check the 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 is 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 DeepSleep mode
  - After wakeup, they 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 shutdown
  - A fault report results in an interrupt
- The DeepSleep domain CSVs operate during Active and DeepSleep, and can only report faults (no reset option)
  - A CSV error detection wakes up the system (if needed), which enables fault reporting
  - The fault report results in an interrupt (no direct interrupt from CSV)
- 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 re-started after the monitored clock is restarted
- The CSV_REF_SEL register selects 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_CSV_REF_CTL, CSV_LF_CSV_REF_CTL, CSV_ILO_CSV_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_CSV_REF_LIMIT, CSV_LF_CSV_REF_LIMIT, CSV_ILO_CSV_REF_LIMIT
  - The following registers can configure PERIOD (time period). Set the Period as -1
    CSV_HF_CSVx_MON_CTL, CSV_REF_CSV_MON_CTL, CSV_LF_CSV_MON_CTL, CSV_ILO_CSV_MON_CTL
- When CSV violation occurs, SRSS_FAULT_CSV is indicated in the IDX bit of the FAULT_STRUCT_STATUS register. The CSV fault report is captured in FAULT_STRUCT_DATA register, see the Fault monitoring chapter for more details

19.10 Registers summary

Registers related to the clock system are shown in Table 132, Table 133, Table 134, and Table 135

Table 132 Clock system registers summary in the PERI

Register	Function
PERI_PCLK_GRx_DIV_CMD	Divider command register
PERI_PCLK_GRx_CLOCK_CTLy	Divider clock control register
PERI_PCLK_GRx_DIV_8_CTLy	8-bit divider control register
PERI_PCLK_GRx_DIV_16_CTLy	16-bit divider control register
PERI_PCLK_GRx_DIV_16_5_CTLy	16.5-bit divider control register
PERI_PCLK_GRx_DIV_24_5_CTLy	24.5-bit divider control register

Table 133 Clock system registers summary in the SRSS

Register	Function
CLK_DSI_SELECTx	Clock DSI select register
CLK_PATH_SELECTx	Clock path select register
CLK_OUTPUT_FAST	Fast clock output select register
CLK_OUTPUT_SLOW	Slow clock output select register
CLK_CAL_CNT1	Clock calibration counter 1 register
CLK_CAL_CNT2	Clock calibration counter 2 register
CLK_DIRECT_SELECTx	Clock root direct select register
CLK_ROOT_SELECTx	Clock root select register
CLK_SELECT	Clock select register
CLK_IMO_CONFIG	IMO configuration register
CLK_ECO_PRESCALE	ECO prescaler configuration register
CLK_ECO_CONFIG	ECO configuration register
CLK_ECO_CONFIG2	ECO configuration register 2
CLK_ECO_STATUS	ECO status register
CLK_FLL_CONFIG	FLL configuration register
CLK_FLL_CONFIG1	FLL configuration register 1
CLK_FLL_CONFIG2	FLL configuration register 2
CLK_FLL_CONFIG3	FLL configuration register 3
CLK_FLL_CONFIG4	FLL configuration register 4
CLK_FLL_STATUS	FLL status register
CLK_ILO_CONFIG	ILO configuration register
CLK_IHO_CONFIG	IHO configuration register
CLK_DPLL_LPx_CONFIG	DPLL configuration register
CLK_DPLL_LPx_CONFIG2	DPLL configuration register 2
CLK_DPLL_LPx_CONFIG3	DPLL configuration register 3

(table continues...)

Table 133 (continued) Clock system registers summary in the SRSS

Register	Function
CLK_DPLL_LPx_CONFIG4	DPLL configuration register 4
CLK_DPLL_LPx_CONFIG5	DPLL configuration register 5
CLK_DPLL_LPx_CONFIG6	DPLL configuration register 6
CLK_DPLL_LPx_CONFIG7	DPLL configuration register 7
CLK_DPLL_LPx_STATUS	DPLL status register
CSV_HF_CSVx_REF_CTL	Clock supervision reference control register for CLK_HFx
CSV_HF_CSVx_REF_LIMIT	Clock supervision reference limits register for CLK_HFx
CSV_HF_CSVx_MON_CTL	Clock supervision monitor control register for CLK_HFx
CSV_REF_SEL	Select CSV reference clock for Active domain
CSV_REF_CSV_REF_CTL	Clock supervision reference control register for CLK_REF_HF
CSV_REF_CSV_REF_LIMIT	Clock supervision reference limits register for CLK_REF_HF
CSV_REF_CSV_MON_CTL	Clock supervision monitor control register for CLK_REF_HF
CSV_LF_CSV_REF_CTL	Clock supervision reference control register for CLK_LF
CSV_LF_CSV_REF_LIMIT	Clock supervision reference limits register for CLK_LF
CSV_LF_CSV_MON_CTL	Clock supervision monitor control register for CLK_LF
CSV_ILO_CSV_REF_CTL	Clock supervision reference control register for CLK_ILO
CSV_ILO_CSV_REF_LIMIT	Clock supervision reference limits register for CLK_ILO
CSV_ILO_CSV_MON_CTL	Clock supervision monitor control register for CLK_ILO

Table 134 Clock system registers summary in the BACKUP

Register	Function
BACKUP_CTL	BACKUP control register

Table 135 Clock system registers summary in the PWRMODE

Register	Function
PWRMODE_CLK_SELECT	Clock selection register for power mode components

20 Watchdog timer

The watchdog timer (WDT) is a hardware timer that automatically resets the device in the event of an unexpected firmware execution path. If enabled, the WDT must be serviced periodically in the firmware to avoid a reset. Otherwise, the timer elapses and generates a device reset. In addition, the WDT can be used as an interrupt source or a wake-up source in low-power modes.

The PSC3M/P8, PSC3M/P7 MCU family includes one free-running WDT and one multi-counter WDT (MCWDT). The watchdog system has four counters: the free-running WDT has one 32-bit counter, and the MCWDT has two 16-bit counters and one 32-bit counter. All 16-bit counters and the 32-bit counter can generate a watchdog device reset. All four counters can generate an interrupt on a match event.

20.1 Features

The PSC3M/P8, PSC3M/P7 MCU WDT supports these features:

One 32-bit free-running WDT with:

- ILO as the input clock source by default. The clock source can be selected from ILO and CLK_BAK. For more details on clock sources, refer to the Clocking system section
- Device reset generation if not serviced within a configurable interval
- Periodic Interrupt/wakeup generation in Active, Sleep, DeepSleep, and Hibernate power modes

Multi-counter watchdog timer (MCWDT) with:

- Device reset generation, if not serviced within a configurable interval
- LFCLK (ILO , WCO) as the input clock source
- Periodic interrupt/wake up generation in Active, Sleep, and DeepSleep power modes
- Two 16-bit and one 32-bit independent counters, which can be configured as a single 64-bit or 48-bit (with one 16-bit independent counter), or two 32-bit cascaded counters
- Window mode operation supported for 16-bit counters

20.2 Architecture

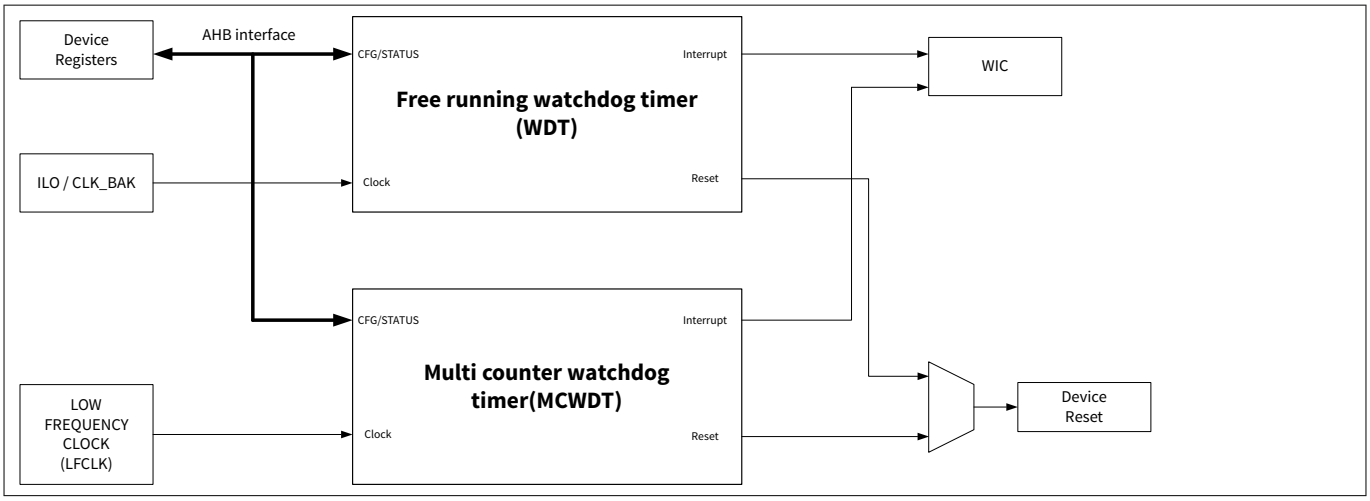


Figure 75 Watchdog timer block diagram

20.3 Basic watchdog timer

20.3.1 Overview

Figure 76 shows the functional overview of the free-running WDT. The WDT has a free-running wraparound up-counter with a maximum of 32-bit resolution. The counter is clocked by the ILO by default. The clock source can be changed using WDT_CTL.WDT_CLK_SEL bits to CLK_BAK. The timer can generate an interrupt on match and a reset event on the third unserviced interrupt. The number of bits used for a match comparison is configurable as depicted in Figure 76.

***Note**: BAK - Selected CLK_BAK source, if present. See BACKUP_CTL. This choice is not recommended for applications that rely upon the watchdog timer for safety or security. Generation of CLK_BAK is not protected by WDT_LOCK and is in a different memory region with potentially different security attributes.*

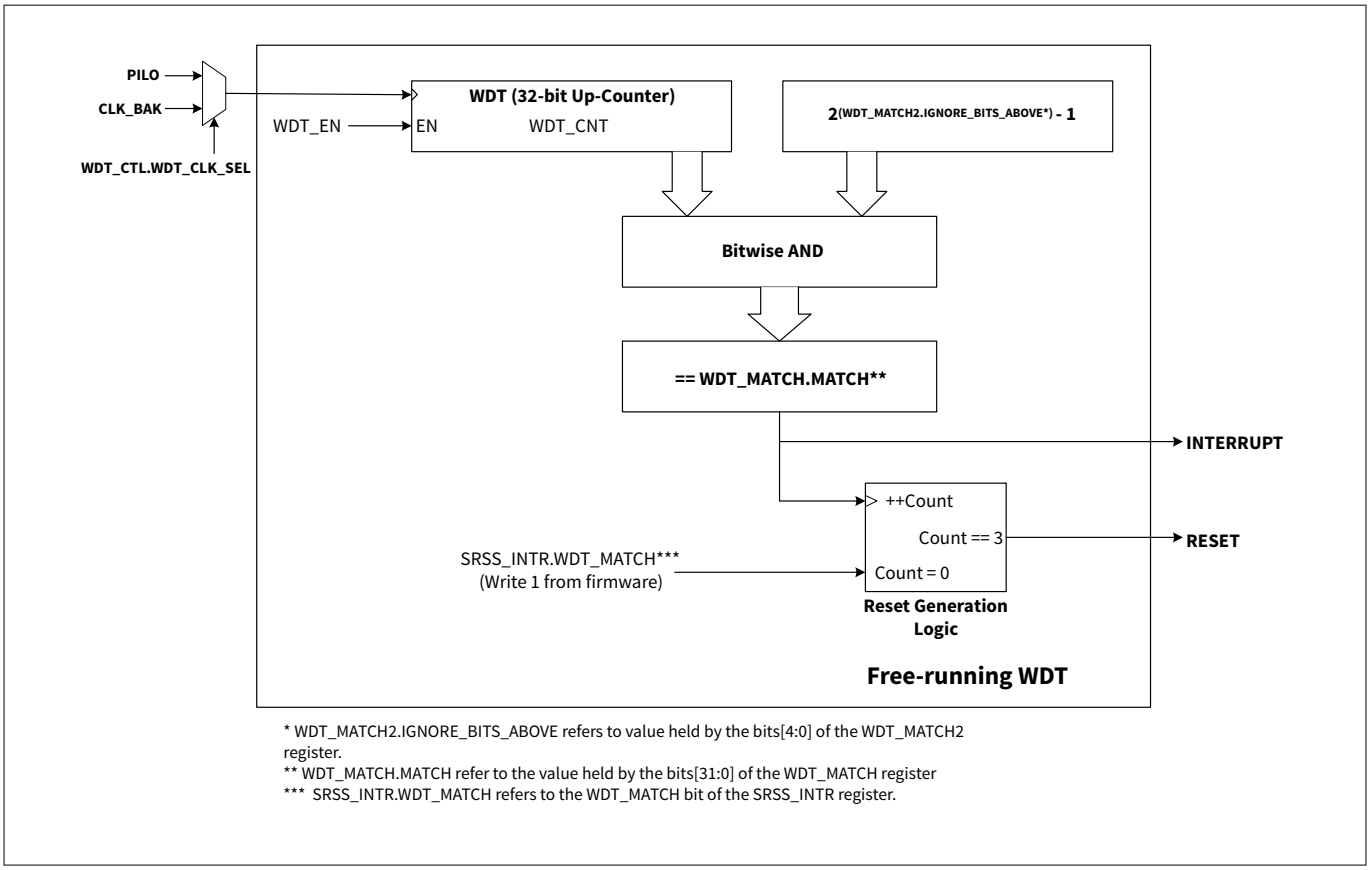


Figure 76 Free-running WDT functional diagram

When enabled, the WDT counts up on each rising edge of the ILO or CLK_BAK. When the counter value (WDT_CNT register) equals the match value stored in MATCH bits [31:0] of the WDT_MATCH register, an interrupt is generated. The match event does not reset the WDT counter and the WDT keeps counting until it reaches the 32-bit boundary (0xFFFF FFFF) at which point, it wraps around to 0 and counts up. The match interrupt is generated every time the counter value equals the match value.

The WDT_MATCH bit of the SRSS_INTR register is set whenever a WDT match interrupt occurs. This interrupt must be cleared by writing a '1' to the same bit. Clearing the interrupt resets the watchdog. If the firmware does not clear the interrupt for two consecutive occasions, the third interrupt generates a device reset.

In addition, the WDT provides an option to set the number of bits to be used for comparison. The IGNORE_BITS_ABOVE (bits [4:0] of the WDT_MATCH2 register) is used for this purpose. These bits configure the

number of LSBs to ignore from the 32-bit count value while performing the match. For example, when the value of these bits equals 4, bit 5 to bit 31 are ignored and bit 0 to bit 4 are considered for match and the WDT counter behaves similar to a 5-bit counter. Note that these bits do not reduce the counter size – the WDT_CNT register still counts from 0 to (0xFFFF FFFF) (32-bit).

The WDT can be enabled or disabled using the WDT_EN bit [0] of the WDT_CTL register. WDT_CTL[5:4] adds the ability to select the WDT clock source among the ILOs (default) and the CLK_BAK. The WDT_CTL register provides a mechanism to lock the WDT configuration registers. The WDT_LOCK bits [31:30] control the lock status of the WDT registers. These bits are special bits, which can enable the lock in a single write; to release the lock, two different writes are required. The WDT_LOCK bits protect the WDT_EN bit, WDT_MATCH register, CLK_ILO_CONFIG register, and LFCLK_SEL bits [2:0] of the CLK_SELECT register. Note that the WDT_LOCK bits are not retained in DeepSleep mode and Hibernate mode and reset to their default (LOCK) state after a wake-up. Therefore, to update any register protected by the WDT_LOCK bits after a DeepSleep wake-up or Hibernate wake-up, issue a WDT UNLOCK sequence before the register update.

Table 136 explains various registers and bitfields used to configure and use the WDT.

Table 136 Free-running WDT configuration options

Register [Bit_Pos]	Bit_Name	Description
WDT_CTL[0]	WDT_EN	Enable or disable the watchdog reset
		0: WDT disabled
		1: WDT enabled
WDT_CTL[5:4]	WDT_CLK_SEL	Select source for WDT
		0: ILO
		2: CLK_BAK - Selected clk_bak source, if present. See BACKUP_CTL. This choice is not recommended for applications that rely on the watchdog timer for safety or security, Generation of clk_bak is not protected by WDT_LOCK and is in a different memory region with potentially different security attributes.
WDT_CTL[31:30]	WDT_LOCK	Lock or unlock write access to the watchdog configuration and clock-related registers. When the bits are set, the lock is enabled.
		0: No effect
		1: Clear bit 0
		2: Clear bit 1
		3: Set both bit 0 and 1 (lock enabled)

(table continues...)

Table 136 (continued) Free-running WDT configuration options

Register [Bit_Pos]	Bit_Name	Description
		WDT locks on a reset. Since this field is not retained in DeepSleep or Hibernate mode, the WDT is locked after wake-up from these modes.
WDT_CNT[31:0]	COUNTER	Current value of WDT counter
WDT_MATCH[31:0]	MATCH	Match value to generate watchdog match event
WDT_MATCH2[4:0]	IGNORE_BITS_ABOVE	Number of LSBs to consider for comparison with the MATCH value. This bit-field has a minimum setting of 3.
SRSS_INTR[0]	WDT_MATCH	WDT interrupt request This bit is set whenever a watchdog match event happens. The WDT interrupt is cleared by writing a '1' to this bit. Software can write 1 to this bit to feed the watchdog.
SRSS_INTR_MASK[0]	WDT_MATCH	Mask for the WDT interrupt
		0: WDT interrupt is blocked
		1: WDT interrupt is forwarded to CPU

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, clear the WDT interrupt bit by 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. Note that when the debug probe is connected, the device reset is blocked and an interrupt is generated instead.

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

- Configure the watchdog reset period such that the firmware can reset the watchdog at least once during the period, even along the longest firmware delay path
- Reset (feed) the watchdog by clearing the interrupt bit regularly in the main body of the firmware code by writing a '1' to the WDT_MATCH bit in the SRSS_INTR register. Note that this does not reset the watchdog counter; it feeds only the watchdog so that it does not cause a reset for the next two match events

Do not reset the watchdog (clear interrupt) in the WDT interrupt service routine (ISR) if WDT is used as a reset source to protect the system against crashes. Therefore, do not use the WDT reset feature and ISR together.

The recommended steps to use WDT as a reset source are as follows:

1. Make sure the WDT configuration is unlocked by clearing the WDT_LOCK bits[31:30] of the WDT_CTL register. Note that clearing the bits requires two writes to the register with each write clearing one bit as explained in Table 136
2. Write the desired IGNORE_BITS_ABOVE in the WDT_MATCH2 register to set the counter resolution to be used for the match

3. Write any match value to the WDT_MATCH register. The match value does not control the period of watchdog reset as the counter is not reset on a match event. This value provides an option to control only the first interrupt interval; after that the successive interrupts' period is defined by the IGNORE_BITS_ABOVE. The watchdog match value is given by the following equation:

$$\text{Watchdog reset period} = ILO_{period} \times ((2 * 2^{(IGNORE_BITS_ABOVE+1)}) + (WDT_MATCH + 1)) \quad (5)$$

4. Set the WDT_MATCH bit in the SRSS_INTR register to clear any pending WDT interrupt
5. Enable ILO by setting the ENABLE bit [31] of the CLK_ILO_CONFIG register
6. Enable the WDT by setting the WDT_EN bit in the WDT_CTL register
7. Lock the WDT and ILO configuration by writing '3' to the WDT_LOCK bits. This also locks the LFCLK_SEL bits of the CLK_SELECT register
8. In the firmware, write '1' to the WDT_MATCH bit in the SRSS_INT register to feed (clear interrupt) the watchdog

20.3.3 Watchdog interrupt

In addition to generating a device reset, the WDT can be used to generate interrupts. Note that interrupt servicing and watchdog reset cannot be used simultaneously using the free-running WDT.

The watchdog counter can send interrupt requests to the CPU in CPU Active power modes and to the wake-up interrupt controller (WIC) in CPU Sleep and DeepSleep power modes. In addition, the watchdog can wake up the device from Hibernate power mode. It works as follows:

- CPU Active mode: In Active power mode, the WDT can send the interrupt to the CPU. The CPU acknowledges the interrupt request and executes the ISR. Clear the interrupt in the ISR
- CPU Sleep or DeepSleep mode: In this mode, the CPU is powered down. Therefore, the interrupt request from the WDT is directly sent to the WIC, which then wakes up the CPU. The CPU acknowledges the interrupt request and executes the ISR. Clear the interrupt in the ISR
- Hibernate mode: In this mode, the entire device except a few peripherals (such as WDT and LPCOMP) 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 section

Because of its free-running nature, the WDT should not be used for periodic interrupt generation. Use the Multi-counter WDT (MCWDT), instead, to generate periodic interrupts. If absolutely required, follow these steps to use the WDT as a periodic interrupt generator:

1. Unlock the WDT if this is the first update to the WDT registers after a DeepSleep or Hibernate wake-up, or a device reset
2. Write the desired IGNORE_BITS_ABOVE in the WDT_MATCH register to set the counter resolution to be used for the match
3. Write the desired match value to the WDT_MATCH register
4. Set the WDT_MATCH bit in the SRSS_INTR register to clear any pending WDT interrupt
5. Enable the WDT interrupt to CPU by setting the WDT_MATCH bit in SRSS_INTR_MASK
6. Enable the SRSS interrupt to the CPU by configuring the appropriate ISER register (see the Interrupts section for details)
7. In the ISR, unlock the WDT; clear the WDT interrupt and add the desired match value to the existing match value. By doing so, another interrupt is generated when the counter reaches the new match value (period)

20.4 Multi-counter WDT

20.4.1 Overview

The PSC3M/P8, PSC3M/P7 MCU has one MCWDT block. The MCWDT block includes two 16-bit counters (MCWDT0_WDT0 and MCWDT0_WDT1) and one 32-bit counter (MCWDT0_WDT2). You can configure these counters to work independently or in cascade (up to 64-bit). The 16-bit counters can generate an interrupt or reset the device. The 32-bit counter can only generate an interrupt. All the counters are clocked by LFCLK.

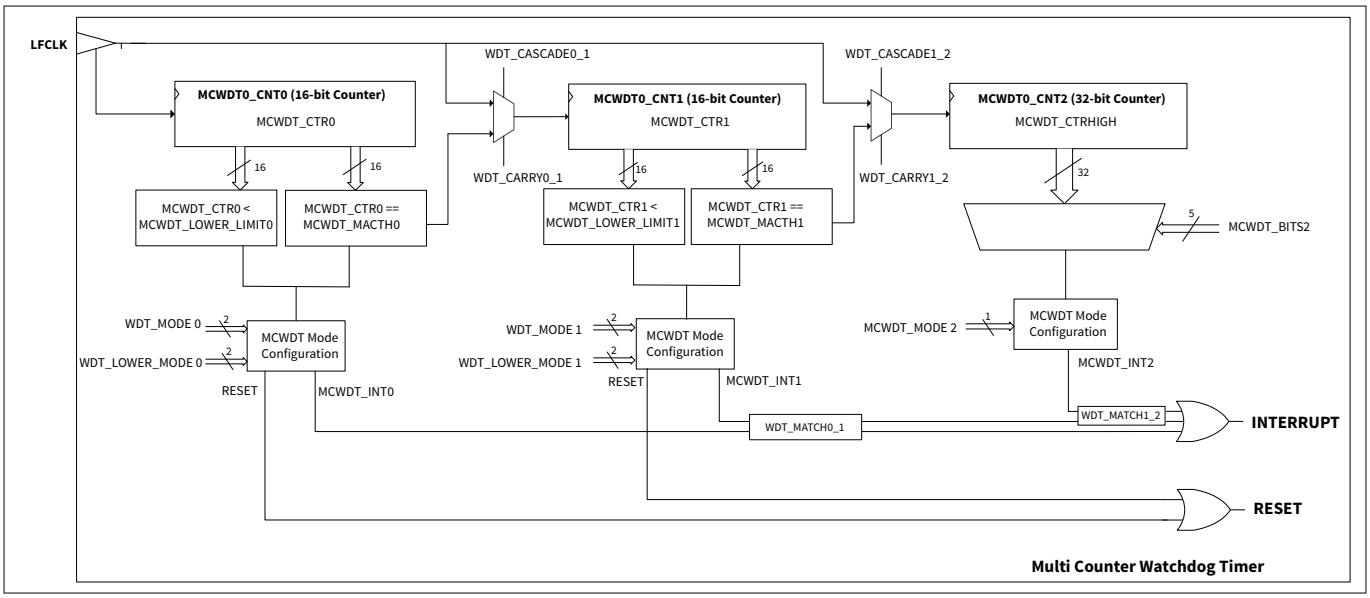


Figure 77 Multi-Counter WDT functional diagram

20.4.1.1 MCWDT0_WDT0 and MCWDT0_WDT1 counters operation

MCWDT0_WDT0 and MCWDT0_WDT1 are 16-bit up counters, which can be configured to be a 16-bit free-running counter or a counter with any 16-bit period. Use these counters to generate an interrupt or reset.

The WDT_CTR0 bits [15:0] and WDT_CTR1 bits [31:16] of the MCWDT0_CNTLOW register hold the current counter values of MCWDT0_WDT0 and MCWDT0_WDT1 respectively. The WDT_MATCH0 bits [15:0] and WDT_MATCH1 bits [31:16] of the MCWDT0_MATCH register store the match value for MCWDT0_WDT0 and MCWDT0_WDT1 respectively. The WDT_MODEx bits of the MCWDT0_CONFIG register configure the action the watchdog counter takes on a match event (WDT_MATCHx == WDT_CTRx). The MCWDT0_WDT0/WDT1 counters perform the following actions:

- Assert interrupt (WDT_INTx) on match
- Assert a device reset on match
- Assert an interrupt on match and a device reset on the third unhandled interrupt

In addition to generating reset and interrupt, configure the match event to clear the counters. This is done by setting the WDT_CLEARx bit of the MCWDT0_CONFIG register. The MCWDT0_WDT0/WDT1 counter operation is shown in Figure 78

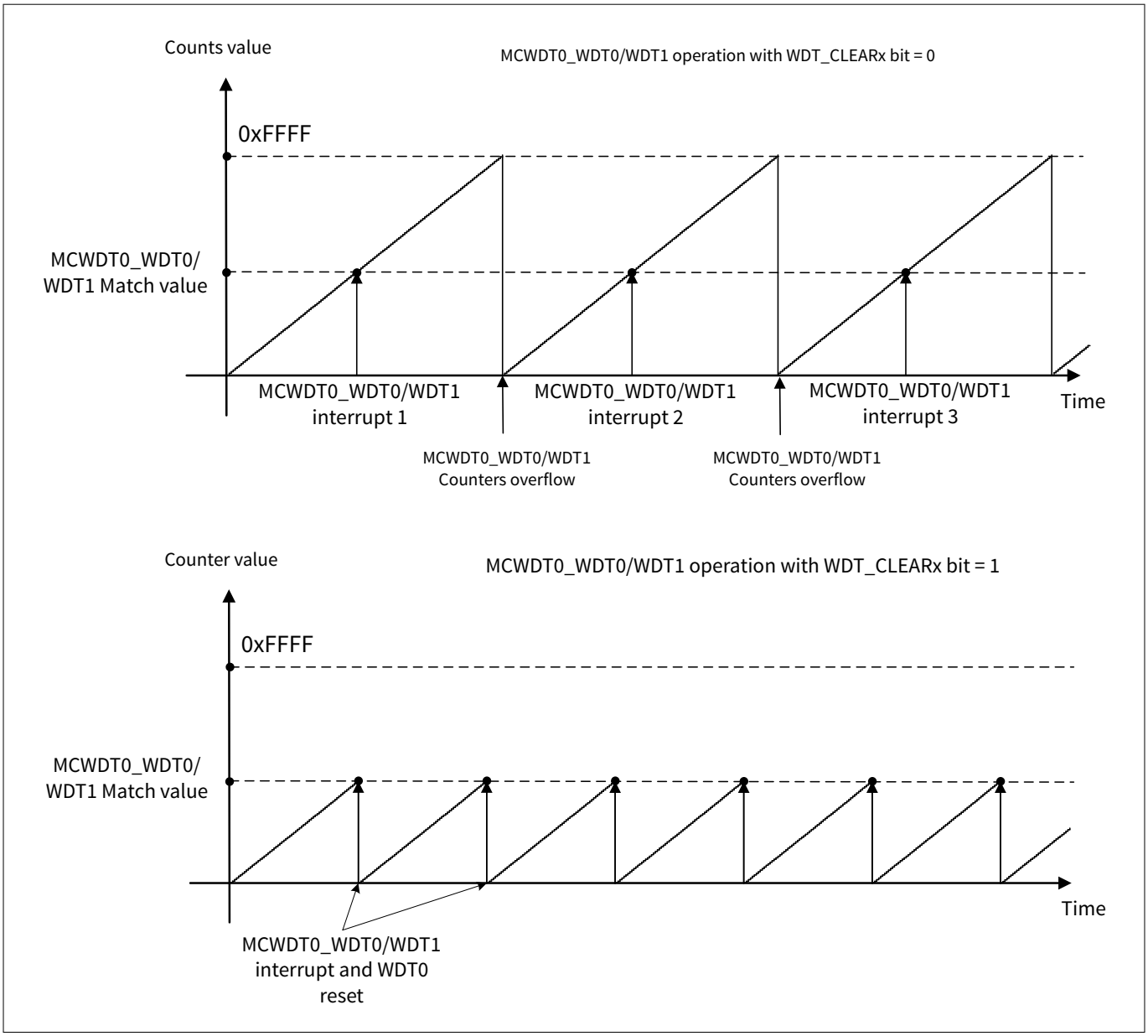


Figure 78 MCWDT0_WDT0/WDT1 operation

The PSC3M/P8, PSC3M/P7 MCU supports the window mode operation for the MCWDT0_CTR0/1 sub-counters as shown in Figure 79. In window mode, an early threshold value stored in WDT_LOWER_LIMIT0 bits [15:0] and WDT_LOWER_LIMIT1 bits [31:16] of the MCWDT0_LOWER_LIMIT register for MCWDT0_WDT0 and MCWDT0_WDT1 respectively. It can be used if the WDT_LOWER_MODE0 bit [5:4] and WDT_LOWER_MODE1 bit [13:12] is set to '2' in the MCWDT0_CONFIG register to generate a reset if the counter is serviced too early.

The Window capability only works when the counter does not overflow and wrap around. As a practical matter, it means that only WDT_MODE0/1== RESET is useful for window operation. Other WDT_MODE0/1 settings are not supported for window operation.

The Window capability works with sub-counters 0 and 1. Cascading them is also supported, in which case the lower limit registers are treated as a combined limit. The Window capability is not supported for sub-counter 2.

It must be possible to dynamically change the lower and upper limits immediately after servicing. This must not trigger any action unless the original service violated the allowed window.

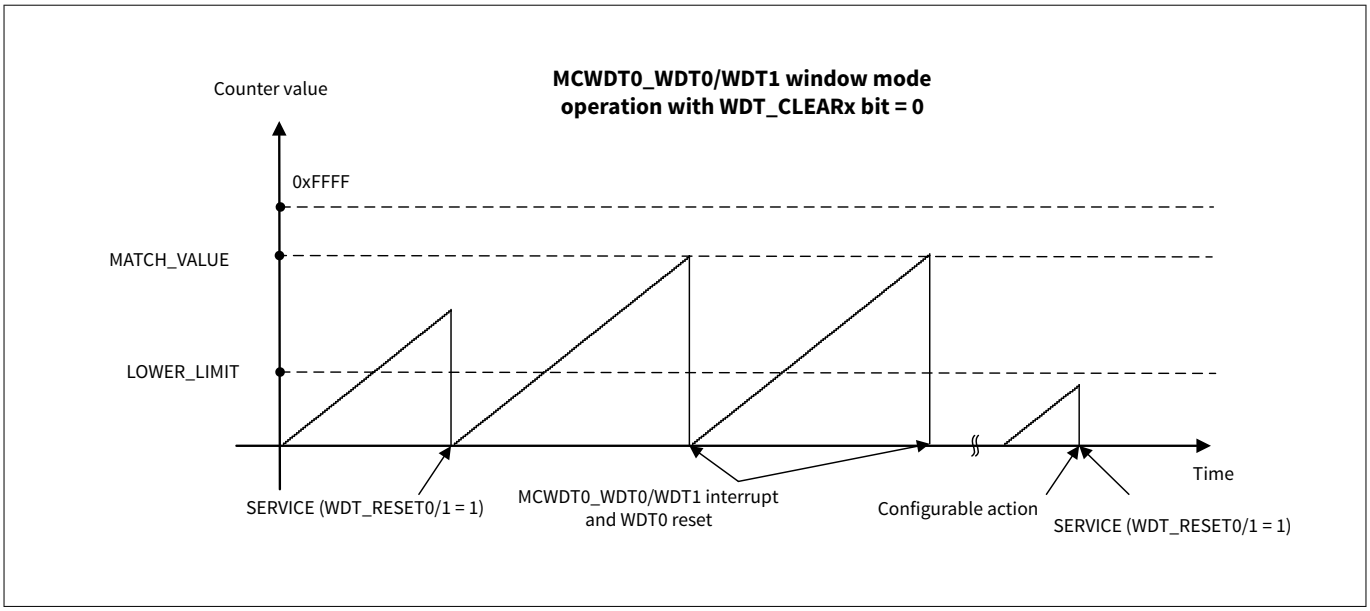


Figure 79 MCWDT0_WDT0/WDT1 window mode operation

Table 137 MCWDT0_WDT0 and MCWDT0_WDT1 configuration options

Register [Bit_Pos]	Bit_Name	Description
MCWDT0_CONFIG[1:0] MCWDT0_CONFIG[9:8]	WDT_MODE0 WDT_MODE1	Watchdog Counter Action on Match. Action is taken on the next increment after the values match (WDT_CTR0=WDT_MATCH0).
		0:Do nothing
		1: Assert interrupt (WDT_INTx)
		2: Assert device Reset
		3: Assert interrupt on match and a device reset after 3rd unhandled interrupt
MCWDT0_CONFIG[5:4] MCWDT0_CONFIG[13:12]	WDT_LOWER_MODE0 WDT_LOWER_MODE1	Watchdog Counter Action on service before lower limit. 0: Do nothing 1: Assert interrupt (MCWDT_INTx) 2: Assert WDT reset
MCWDT0_CONFIG[2] MCWDT0_CONFIG[10]	WDT_CLEAR0 WDT_CLEAR1	Clear the MCWDT0_WDT0/WDT1 counter on match. In other words, (WDT_MATCHx + 1) acts similar to a period for the MCWDT0_WDT0/ WDT1 counter. 0: Free running counter 1: Clear WDT_CTRx bits on match.

(table continues...)

Table 137 (continued) MCWDT0_WDT0 and MCWDT0_WDT1 configuration options

Register [Bit_Pos]	Bit_Name	Description
MCWDT0_CNTLOW[15:0] MCWDT0_CNTLOW[31:16]	WDT_CTR0 WDT_CTR1	Current counter values. Bits[15:0] contains the current value of the MCWDT0_WDT0 counter. Bits[31:16] contains the current value of the MCWDT0_WDT1 counter.
MCWDT0_MATCH[15:0] MCWDT0_MATCH[31:16]	WDT_MATCH0 WDT_MATCH1	Match values Changing WDT_MATCHx requires 1.5 CLK_LF cycles to come into effect. After changing WDT_MATCHx, do not enter the DeepSleep mode for at least one CLK_LF cycle to ensure theWDT updates to the new setting.
MCWDT0_LOWER_LIMIT[15:0] MCWDT0_LOWER_LIMIT[31:16]	WDT_LOWER_LIMIT0 WDT_LOWER_LIMIT1	Provides lower limit values for counters 0 and 1.

20.4.1.2 MCWDT0_WDT2 counter operation

The MCWDT0_WDT2 is a 32-bit free-running counter, which can be configured to generate an interrupt. The MCWDT0_CNTHIGH register holds the current value of the WDT2 counter. WDT2 does not support a match feature. However, it can be configured to generate an interrupt when one of the counter bits toggle. The WDT_BITS2 bits [28:24] of the MCWDT0_CONFIG register selects the bit on which the MCWDT0_WDT2 interrupt is asserted. WDT_MODE2 bit [16] of the MCWDT0_CONFIG register decides whether to assert an interrupt on bit toggle or not. Figure 80 shows the MCWDT0_WDT2 counter operation.

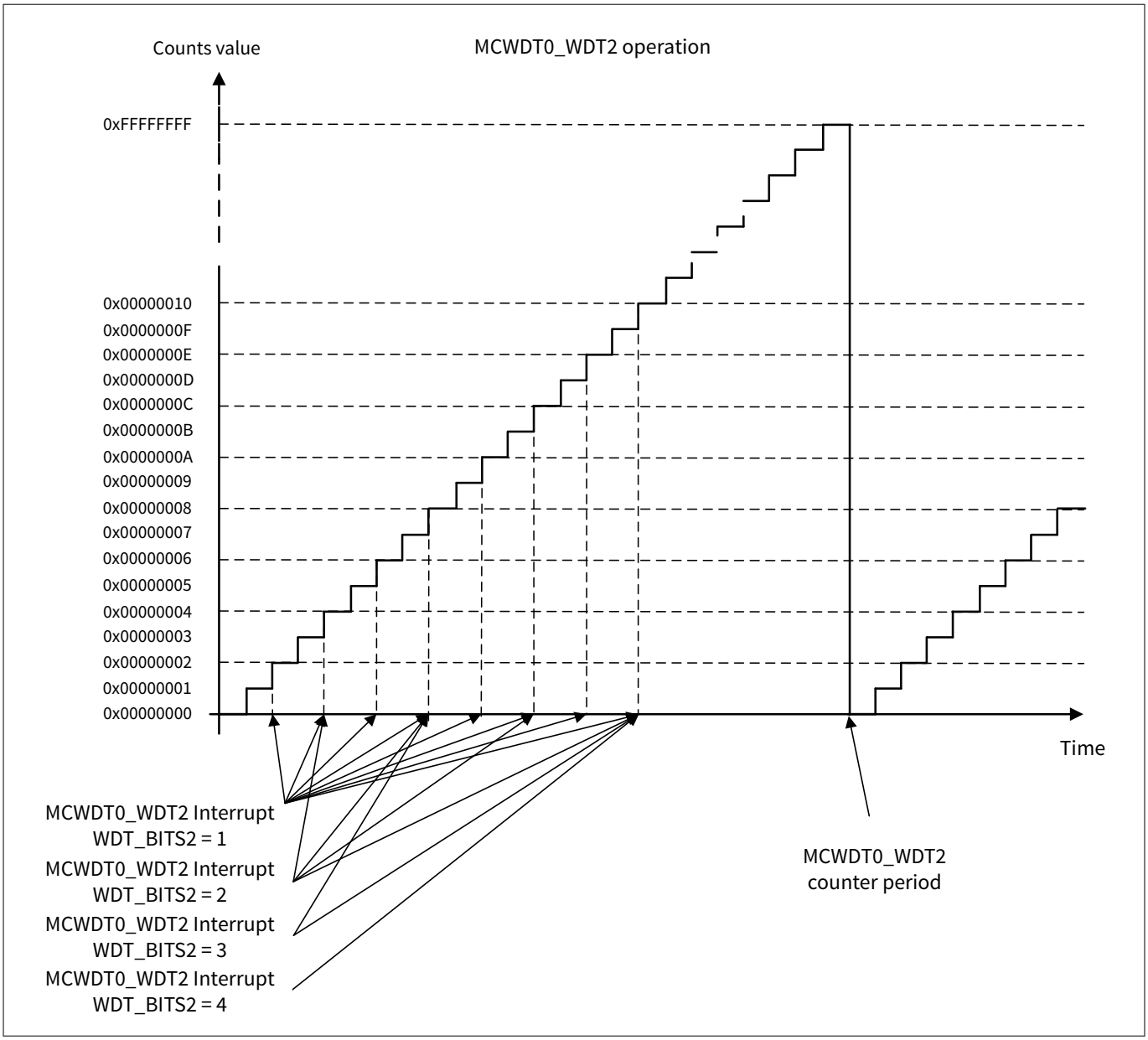


Figure 80 MCWDT0_WDT2 operation

Table 138 MCWDT0_WDT2 Configuration Options

Register [Bit_Pos]	Bit_Name	Description
MCWDT0_CONFIG[16]	WDT_MODE2	MCWDT0_WDT2 mode 0: Free-running counter 1: Free-running counter with interrupt (WDT_INTx) request generation. The interrupts are generated when the bit position specified in WDT_BITS2 toggles in the counter value held by MCWDT0_CNTHIGH register.

(table continues...)

Table 138 (continued) MCWDT0_WDT2 Configuration Options

Register [Bit_Pos]	Bit_Name	Description
MCWDT0_CONFIG[28:24]	WDT_BITS2	Bit to monitor for MCWDT0_WDT2 interrupt assertion 0: Asserts when bit [0] of MCWDT0_CNTHIGH register toggles (interrupt every tick) …… 31: Asserts when bit [31] of MCWDT0_CNTHIGH register toggles (interrupt every 2^(31) ticks)
MCWDT0_CNTHIGH[31:0]	WDT_CTR2	Current counter value of MCWDT0_WDT2

20.4.2 Enabling and disabling MCWDT

The MCWDT counters are enabled by setting the WDT_ENABLEx bit in the MCWDT0_CTL register and are disabled by clearing it. Enabling or disabling a MCWDT requires up to two LFCLK cycles to come into effect. Therefore, the WDT_ENABLEx bit value must not be changed more than once in that period and the WDT_ENABLEDx bit of the MCWDT0_CTL register can be used to monitor the enabled/disabled state of the counter.

The WDT_RESETx bit of the MCWDT0_CTL register clears the corresponding MCWDT0 counter when set in firmware. The hardware clears the bit after the counter resets. This option is useful when the MCWDT0_WDT0 or MCWDT0_WDT1 is configured to generate a device reset on a match event. In such cases, the device resets when the counter reaches the match value. Thus, setting the WDT_RESET0 or WDT_RESET1 bit resets MCWDT0_WDT0 or MCWDT0_WDT1 respectively, preventing device reset.

After the MCWDT is enabled, do not write to the MCWDT configuration (MCWDT0_CONFIG) and control (MCWDT0_CTL) registers. Accidental corruption of these registers can be prevented by setting the MCWDT_LOCK bits [31:30] of the MCWDT0_LOCK register. If the application requires updating the match value (WDT_MATCH) when the MCWDT 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 bit protects only the MCWDT0_CTL, MCWDT0_CONFIG, and MCWDT0_MATCH registers. The LFCLK select registers are protected by the free-running WDT lock bits.

Table 139 Watchdog configuration options

Register [Bit_pos]	Bit_Name	Description
MCWDT0_CTL[0]	WDT_ENABLE0	Enable MCWDT counter. May take up to two LFCLK cycles to take effect. 0: Counter is disabled (not clocked) 1: Counter is enabled (counting up)
MCWDT0_CTL[8]	WDT_ENABLE1	
MCWDT0_CTL[16]	WDT_ENABLE2	

(table continues...)

Table 139 (continued) Watchdog configuration options

Register [Bit_pos]	Bit_Name	Description
MCWDT0_CTL[1] MCWDT0_CTL[9] MCWDT0_CTL[17]	WDT_ENABLED0 WDT_ENABLED1 WDT_ENABLED2	Indicates actual state of counter x. Monitor this bit after changing the WDT_ENABLEx bit, to receive an acknowledgment of the changes.
MCWDT0_CTL[3] MCWDT0_CTL[11] MCWDT0_CTL[19]	WDT_RESET0 WDT_RESET1 WDT_RESET2	Reset the MCWDT counter x to 0. Hardware clears the bit when reset is complete. 0: Software - No action 1: Software - Resets the counter
MCWDT0_LOCK[31:30]	MCWDT_LOCK	Locks or unlocks write access to the MCWDT0_CTL, MCWDT0- _CONFIG, and MCWDT0_MATCH 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)

20.4.3 Watchdog cascade options

The cascade configuration shown in Figure 77 provides an option to increase the MCWDT counter range or maximum timeout. The WDT_CASCADE0_1 bit [3] of the MCWDT0_CONFIG register cascades MCWDT0_WDT0 and MCWDT0_WDT1 and the WDT_CASCADE1_2 bit [11] of the MCWDT0_CONFIG register cascades MCWDT0_WDT1 and MCWDT0_WDT2. Note that cascading two 16-bit counters does not provide a 32-bit counter; instead, you get a 16-bit period counter with a 16-bit prescaler. For example, when cascading MCWDT0_WDT0 and MCWDT0_WDT1, MCWDT0_WDT0 acts as a prescaler for MCWDT0_WDT1 and the prescaler value is defined by the WDT_MATCH0 bits [15:0] in the MCWDT0_MATCH register. The MCWDT0_WDT1 has a period defined by WDT_MATCH1 bits [31:16] in the MCWDT0_MATCH register. The same logic applies to MCWDT0_WDT1 and MCWDT0_WDT2 cascading.

Additionally, the carry operation of the cascaded counters can be configured on a match or roll over using the WDT_CARRY0_1 and WDT_CARRY1_2 bits of the MCWDT0_CONFIG register. The WDT_MATCH0_1 and WDT_MATCH1_2 bits can be configured to define the matching behavior of the cascaded counters. The interrupt on match can be generated when only the first counter matches or when both the cascaded counters match simultaneously.

The summary of the cascade configuration bits is given in Table 140

Table 140 Watchdog cascade options

Register [Bit_Pos]	Bit_name	Description
MCWDT0_CONFIG[3]	WDT_CASCADE0_1	Cascade Watchdog Counters 0,1. Counter 1 increments the cycle after WDT_CTR0=WDT_MATCH0. 0: Independent counters 1: Cascaded counters
MCWDT0_CONFIG[11]	WDT_CASCADE1_2	Cascade Watchdog Counters 1,2. Counter 2 increments the cycle after WDT_CTR1=WDT_MATCH1. It is allowed to cascade all three WDT counters. 0: Independent counters 1: Cascaded counters. When cascading all three counters, WDT_CLEAR1 must be 1.
MCWDT0_CONFIG[6]	WDT_CARRY0_1	Carry out behavior that applies when WDT_CASCADE0_1==1. This bit is not used when WDT_CASCADE0_1==0. 0: carry out on counter 0 match. 1: carry out on counter 0 roll-over.
MCWDT0_CONFIG[14]	WDT_CARRY1_2	Carry out behavior that applies when WDT_CASCADE1_2==1. This bit is not used when WDT_CASCADE1_2==0. 0: carry out on counter 1 match. 1: carry out on counter 1 roll-over.
MCWDT0_CONFIG[7]	WDT_MATCH1_0	Specifies matching behavior when WDT_CASCADE0_1==1. When WDT_CASCADE0_1==0, this bit is not used and match is based on counter 1 alone. 0: Match based on counter 1 alone. 1: Match based on counter 1 and counter 0 matching simultaneously.

(table continues...)

Table 140 (continued) Watchdog cascade options

Register [Bit_Pos]	Bit_name	Description
MCWDT0_CONFIG[15]	WDT_MATCH1_2	Specifies matching behavior when WDT_CASCADE1_2==1. When WDT_CASCADE1_2==0, this bit is not used and match is based on counter 2 alone. 0: Match based on counter 2 alone. 1: Match based on counter 2 and counter 1 matching simultaneously.

When using cascade (WDT_CASCADE0_1 or WDT_CASCADE1_2 set), resetting the counters when the prescaler or lower counter is at its match value with the counter configured to clear on match, results in the upper counter incrementing to 1 instead of remaining at 0. Correct this behavior by issuing a second reset to the upper counter after approximately 100 µs from the first reset. Note that the second reset is required only when the first reset is issued while the prescaler counter value is at its match value. Figure 81 illustrates the behavior when MCWDT0_WDT0 and MCWDT0_WDT1 are cascaded along with the second reset timing.

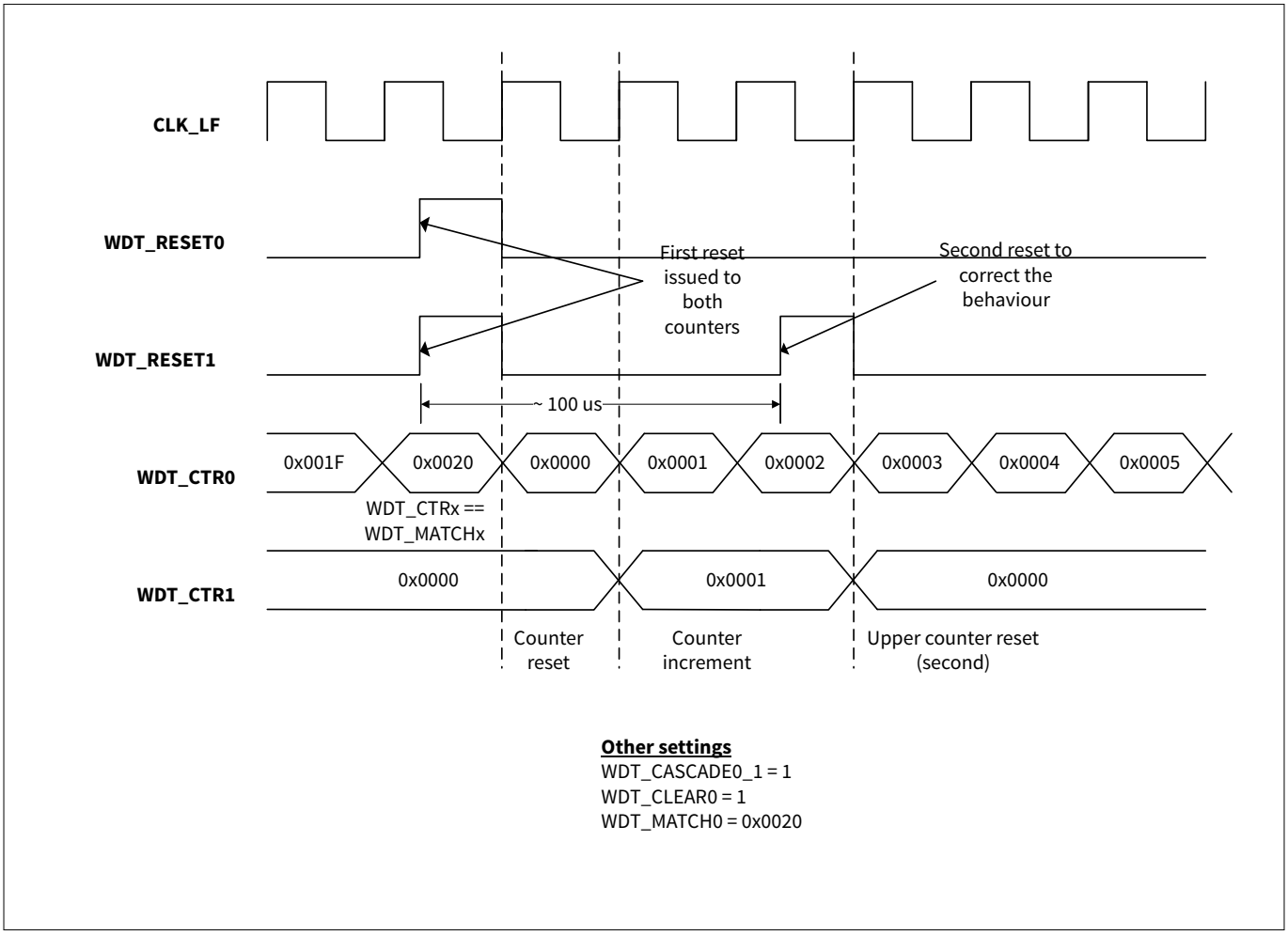


Figure 81 MCWDT reset behavior in Cascaded mode

In addition, the counters exhibit non-monotonicity in the following cascaded conditions:

- If WDT_CASCADE0_1 is set, then WDT_CTR1 does not increment the cycle after WDT_CTR0 = WDT_MATCH0

- If WDT_CASCADE1_2 is set, then WDT_CTR2 does not increment the cycle after WDT_CTR1 = WDT_MATCH1
- If both WDT_CASCADE0_1 and WDT_CASCADE1_2 are set, then WDT_CTR2 does not increment the cycle after WDT_CTR1 = WDT_MATCH1 and WDT_CTR1 does not increment the cycle after WDT_CTR0 = WDT_MATCH0

When cascading is enabled, always read the WDT_CTR1 or WDT_CTR2 counter value only when the prescaler counter (WDT_CTR0 or WDT_CTR1) value is not 0. This makes sure the upper counter is incremented after a match event in the periodic counter.

20.4.4 MCWDT reset

MCWDTx_WDT0 and MCWDTx_WDT1 can be configured to generate a device reset similar to the free-running WDT reset. Note that when the debug probe is connected, the device reset is blocked but an interrupt is generated if configured. Follow these steps to use the MCWDTx_WDT0 or MCWDTx_WDT1 counter of a MCWDTx block to generate a system reset:

1. Configure the MCWDT to generate a reset using the WDT_MODEx bits in MCWDTx_CONFIG. Configure the WDT_MODE0 or WDT_MODE1 bits in MCWDTx_CONFIG to '2' (reset on match) or '3' (interrupt on match and reset on the third unhandled interrupt)
2. Optionally, set the WDT_CLEAR0 or WDT_CLEAR1 bit in the MCWDTx_CONFIG register for MCWDTx_WDT0 or MCWDTx_WDT1 to reset the corresponding watchdog counter to '0' on a match event. Otherwise, the counters are free running. See MCWDT0_WDT0 and MCWDT0_WDT1 counters operation for details
3. Calculate the watchdog reset period such that the firmware can reset the watchdog at least once during the period, even along the longest firmware delay path. If WDT_MODEx == 2, the match value is the same as the watchdog period. If WDT_MODEx == 3, the match value is one-third of the watchdog period. Write the calculated match value to the WDT_MATCH register for MCWDTx_WDT0 or MCWDTx_WDT1. Optionally, enable cascading to increase the interval

   ***Note**: The legal value for the WDT_MATCH field is 1 to 65535*

4. If WDT_MODEx == 2, set the WDT_RESETx bit in the MCWDTx_CONFIG register to reset the WDTx counter to 0. If WDT_MODEx == 3, set the WDT_INTx bit in MCWDTx_INTR to clear any pending interrupts
5. Enable WDTx by setting the WDT_ENABLEx bit in the MCWDTx_CTL register. Wait until the WDT_ENABLEDx bit is set
6. Lock the MCWDTx configuration by setting the MCWDT_LOCK bits of the MCWDTx_CTL register
7. In the firmware, feed (reset) the watchdog as explained in step 4

Do not reset the watchdog in the WDT ISR. It is also not recommended to use the same watchdog counter to generate a system reset and interrupt. For example, if MCWDTx_WDT0 is used to generate system reset against crashes, then MCWDTx_WDT1 or MCWDTx_WDT2 should be used for periodic interrupt generation

20.4.5 MCWDT interrupt

When configured to generate an interrupt, the WDT_INTx bits of the MCWDTx_INTR register provide the status of any pending watchdog interrupts. The firmware must clear the interrupt by setting the WDT_INTx. The WDT_INTx bits of the MCWDTx_INTR_MASK register mask the corresponding WDTx interrupt of the MCWDTx block to the CPU.

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

1. Write the desired match value to the MCWDT_MATCH register for MCWDTx_WDT0/WDT1 or the WDT_BITS2 value to the MCWDTx_CONFIG register for MCWDTx_WDT2

   ***Note**: The legal value for the WDT_MATCH field is 1 to 65535*

2. Configure the WDTx to generate an interrupt using the WDT_MODEx bits in MCWDTx_CONFIG. Configure the WDT_MODE0 or WDT_MODE1 bits in MCWDTx_CONFIG for MCWDTx_WDT0 or MCWDTx_WDT1 to '1' (interrupt on match) or '3' (interrupt on match and reset on third unhandled interrupt). For MCWDTx_WDT2, set the WDT_MODE2 bit in the MCWDTx_CONFIG register
3. Set the WDT_INT bit in MCWDTx_INTR to clear any pending interrupt
4. Set the WDT_CLEAR0 or WDT_CLEAR1 bit in the MCWDTx_CONFIG register for MCWDTx_WDT0 or MCWDTx_WDT1 to reset the corresponding watchdog counter to '0' on a match event
5. Mask the WDTx interrupt to the CPU by setting the WDT_INTx bit in the MCWDTx_INTR_MASK register
6. Enable WDTx by setting the WDT_ENABLEx bit in the MCWDTx_CTL register. Wait until the WDT_ENABLEDx bit is set
7. Enable the MCWDTx interrupt to the CPU by configuring the appropriate ISER register. Refer to the Interrupts section
8. In the ISR, clear the WDTx interrupt by setting the WDT_INTx bit in the MCWDTx_INTR register

Note that interrupts from all three WDTx counters of the MCWDT block are mapped as a single interrupt to the CPU. In the interrupt service routine, the WDT_INTx 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 section. The MCWDT block can send interrupt requests to the CPU in Active power mode and to the WIC in Sleep and DeepSleep power modes. It works similar to the free-running WDT.

20.5 Reset cause detection

The RESET_WDT bit [0] in the RES_CAUSE register indicates the reset generated by the free-running WDT. The RESET_MCWDT0 bit in the RES_CAUSE register indicates the reset generated by the MCWDT0 block. These bits remain set until cleared or until a Power-On Reset (POR), Brown-Out Detection (BOD), or External Reset (XRES) occurs. All other resets leave this bit unaltered. For more details, see the Reset system section.

20.6 Registers summary

Table 141 WDT registers

Name	Description
WDT_CTL	Watchdog Counter Control Register
WDT_CNT	Watchdog Counter Count Register
WDT_MATCH	Watchdog Counter Match Register
WDT_MATCH2	Watchdog Counter Match Register2
MCWDT0_CNTLOW	Multi-counter WDT Sub-counters 0/1
MCWDT0_CNTHIGH	Multi-counter WDT Sub-counter 2
MCWDT0_MATCH	Multi-counter WDT Counter Match Register for counters 0 and 1
MCWDT0_CONFIG	Multi-counter WDT Counter Configuration, including bit toggle interrupt generation for counter 2
MCWDT0_CTL	Multi-counter WDT Counter Control
MCWDT0_LOCK	Multi-counter WDT Counter Lock Register
MCWDT0_LOWER_LIMIT	Multi-Counter WDT Lower Limit Register
MCWDT0_INTR	Multi-counter WDT Counter Interrupt Register

(table continues...)

Table 141 (continued) WDT registers

Name	Description
MCWDT0_INTR_SET	Multi-counter WDT Counter Interrupt Set Register
MCWDT0_INTR_MASK	Multi-counter WDT Counter Interrupt Mask Register
MCWDT0_INTR_MASKED	Multi-counter WDT Counter Interrupt Masked Register
CLK_SELECT	Clock Selection Register
CLK_ILO_CONFIG	ILO Configuration
SRSS_INTR	SRSS Interrupt Register
SRSS_INTR_SET	SRSS Interrupt Set Register
SRSS_INTR_MASK	SRSS Interrupt Mask Register
SRSS_INTR_MASKED	SRSS Interrupt Masked Register
RES_CAUSE	Reset Cause Observation Register

21 Backup

The Backup domain adds an “always on” functionality to PSC3M/P8, PSC3M/P7 MCUs using a separate power domain supplied by a backup supply (VBACKUP) such as a battery or supercapacitor. It contains a real-time clock (RTC) with an alarm feature, supported by a 32768-Hz watch crystal oscillator (WCO).

The Backup is not a power mode; it is a power domain with its own power supply, which can be Active during any of the device power modes.

21.1 Features

- Fully-featured RTC
  - Year/Month/Date, Day-of-Week, Hour : Minute : Second fields
  - RTC uses binary instead of Binary Coded Decimal
  - 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
- 32768 Hz WCO with calibration
- Automatic backup power switching
- Built-in supercapacitor charger
- 32 backup registers

21.2 Architecture

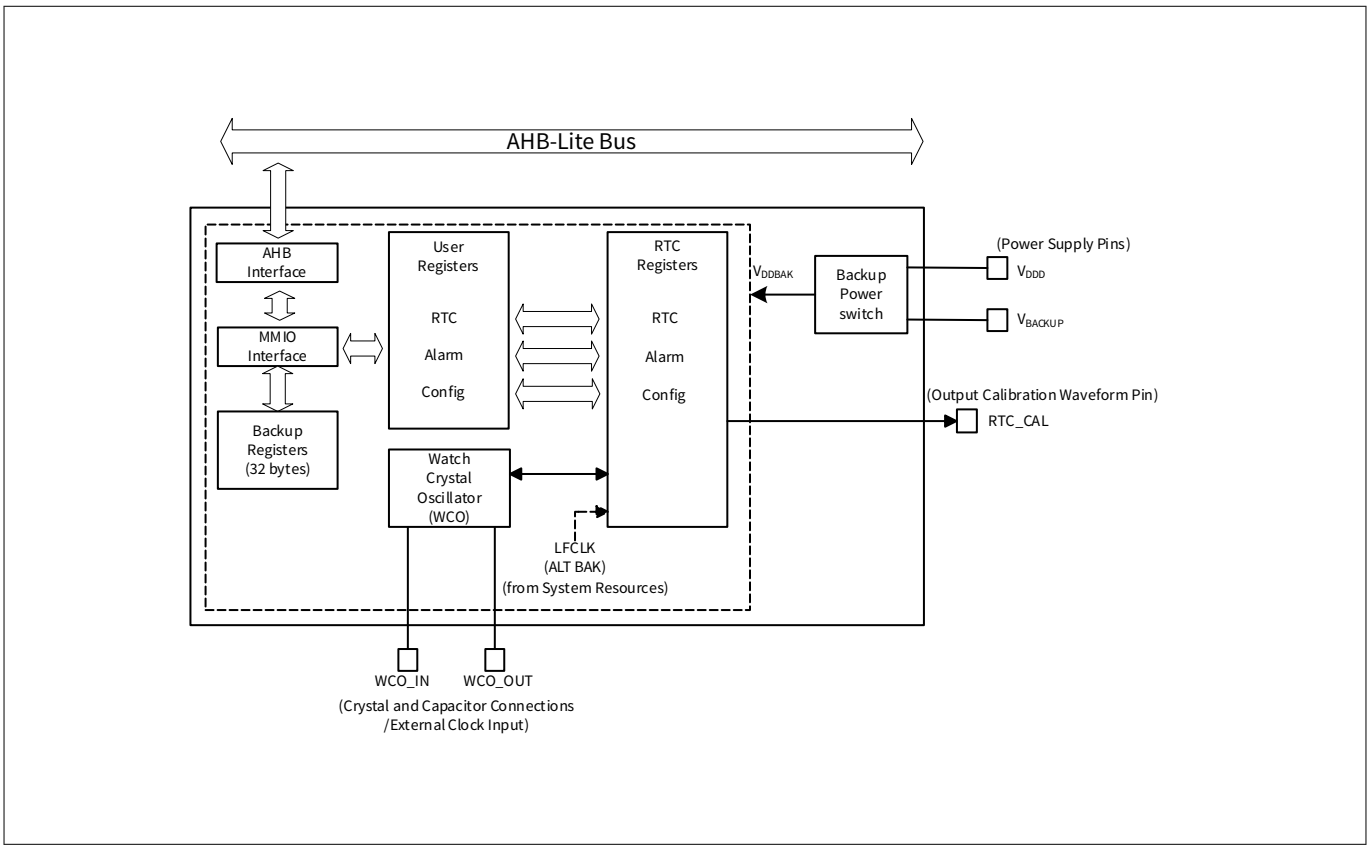


Figure 82 Block diagram

The Backup system includes an accurate WCO that can generate the clock required for the RTC with the help of an external crystal or an external clock inputs. The RTC has a programmable alarm feature, that generate interrupts to the CPU. The AHB-Lite interface provides firmware access to MMIO registers in the Backup domain.

An automatic backup power switch selects the VDDBAK supply required to run the blocks in the Backup domain either VDDD (main power) or VBACKUP (backup battery/supercapacitor power).

The domain also has Backup registers that store 32 bytes of data and retain their contents even when the main supply (VDDD) is OFF, if the backup supply (VBACKUP) is present.

21.3 Power supply

Power to the Backup system (VDDBAK) is automatically switched between VDDD (main supply) and VBACKUP (Backup domain supply). VBACKUP is typically connected to an independent supply derived from a battery or supercapacitor.

There are no VBACKUP versus VDDD sequencing restrictions for the power selector switch. Either VBACKUP or VDDD may be removed during the normal operation, and the Backup system will remain powered.

The VDDBAK_CTL bit field in the BACKUP_CTL register controls the behavior of the power selector switch. See the registers reference manual for more details on this register. The different mode options are as follows:

- VDDBAK_CTL = 0 (Default mode): Selects VDDD when the brownout detector in the system resources is enabled and no brownout situation is detected (see thePower supply and monitoring chapter for more details). Otherwise, it selects the highest supply among VDDD and VBACKUP
- VDDBAK_CTL = 1, 2, or 3: Always selects VBACKUP for debug purposes

If a supercapacitor is connected to VBACKUP, the PSC3M/P8, PSC3M/P7 MCU can charge the supercapacitor while VDDD is available. The supercapacitor charging can be enabled by writing “3C” to the EN_CHARGE_KEY bit field in the BACKUP_CTL register. Note that this feature charges the supercapacitors only and cannot safely charge a battery. Do not write this key when VBACKUP is connected to a battery. The battery charging is handled at the board level using an external circuitry.

***Note**: If VDDD and VBACKUP are connected on the PCB, the Backup domain may require an explicit reset triggered by firmware using the RESET bit field in the BACKUP_RESET register. This firmware reset is required if the VBACKUP supply was invalid during a previous power supply ramp-up or brownout event. It is not necessary to reset the Backup domain if the RES_CAUSE register indicates a non-power related reset as the cause for the reset, or if the PSC3M/P8, PSC3M/P7 MCU just woke from the Hibernate mode and the supply is assumed to be valid the entire time.*

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. The WCO can also operate without crystal, using external clock/sine wave inputs. These additional operating modes are explained later in this section. The WCO is supplied by the Backup domain and can therefore run without VDDD present
- Alternate backup clock (ALTBAK): This option allows the use of LFCLK generated by the System Resources Subsystem (SRSS) as the Backup domain clock. Note that LFCLK is not available in all device power modes or when the VDDD is removed

The clock glitches can propagate into the Backup system when LFCLK is enabled or disabled by the SRSS. In addition, LFCLK may not be as accurate as WCO depending on the actual source of LFCLK. Due to this LFCLK is not recommend for RTC applications. Also, if the WCO is intended to be the clock source then choose it directly instead of routing through LFCLK.

For more details on clocks and calibration, see the Clocking system in the reference manual.

The RTC clock source can be selected using the CLK_SEL bit field in the BACKUP_CTL register. The WCO_EN bit in the BACKUP_CTL register is used to enable or disable the WCO. If the WCO operates with an external crystal, make sure the WCO_BYPASS bit in the BACKUP_CTL register is cleared before enabling the WCO. In addition, the PRESCALER bit-field in BACKUP_CTL is configured for a prescaler value of 32768.

***Note**: The external crystal and load capacitors of proper values must be connected to WCO_IN and WCO_OUT pins. See the device datasheet for details on the component values and the electrical connections. In addition, GPIOs must be configured for WCO_OUT and WCO_IN signals. See the I/O subsystem chapter on reference manual to configure the GPIOs.*

21.4.1 WCO with external clock/sine wave input

The WCO can also operate from external clock/sine wave inputs. In these modes, WCO must be bypassed by setting the WCO_BYPASS bit in the BACKUP_CTL register before enabling the WCO. Also, the GPIOs must be configured for WCO_OUT and WCO_IN signals (in the Analog mode). The external clock/sine wave input modes, prescaler settings, and electrical connections are as follows:

- 32768 Hz External clock mode: In this mode, WCO_IN is floating and WCO_OUT is externally driven by a 32768- Hz square wave clock toggling between ground and VDDD supply levels. In this configuration, the WCO_OUT pin functions as a digital input pin for the external clock. The PRESCALER bit field in BACKUP_CTL is configured to a prescaler value 32768
- 60 Hz External clock mode: This mode can be used for deriving a clock from the 60 Hz AC mains supply. In this mode, WCO_OUT is floating and WCO_IN is driven with an external sine wave of zero DC offset, derived from the 60 Hz/120 V mains through a 100:1 capacitive divider. For example, a suitable capacitive divider is formed by connecting a 220 pF/6 V capacitor between WCO_IN and ground, and a 2.2 pF/200 V capacitor between WCO_IN and the 60 Hz/120 V mains input. The PRESCALER bit field in BACKUP_CTL is configured to a prescaler value of 60
- 50 Hz External clock mode: This mode is similar to the 60-Hz mode, and is used for 50 Hz/220 V mains standard. The capacitive divider explained previously is modified to fit this type of supply by having a 1 pF/250 V capacitor between WCO_IN and the mains input. The PRESCALER bit field in BACKUP_CTL is configured to a prescaler value of 50

21.4.2 Calibration

The absolute frequency of the clock input can be calibrated using the BACKUP_CAL_CTL register. Calibration only works when the PRESCALER bit field in BACKUP_CTL is set to 32768.

CALIB_VAL is a 6 bit field in the BACKUP_CAL_CTL register that holds the calibration value for absolute frequency (at a fixed temperature). One bit of this field translates into 128 ticks added or removed from the clock count. Therefore, each bit translates to a change of 1.085 ppm (128/(60*60*32768)), when CAL_COMP_PER_MIN is set to a default value. The positive values 0x01-0x3c (1..60) add pulses, negative values remove pulses, so giving a range of +/-65.1 ppm (limited by 60 minutes per hour, not the range of this field) when CAL_COMP_PER_MIN is set at default value. Calibration is performed hourly, starting at 59 minutes and 59 seconds, applied as 64 ticks every 30 seconds until there are 2*CALIB_VAL adjustments when CAL_COMP_PER_MIN is set to a default value.

CALIB_SIGN field in the BACKUP_CAL_CTL register controls whether the ticks are added (takes fewer clock ticks to count one second) or subtracted (takes more clock ticks to count one second).

21.5 Reset

The PSC3M/P8, PSC3M/P7 MCU reset sources monitor the device power supply such as the power-on reset (POR) and Brownout detection (BOD) that cannot reset the Backup system as long as the backup supply

(VDDBAK) is present. Moreover, the internal and external resets: watchdog timer (WDT) reset and XRES cannot reset the Backup system. The following conditions are needed for reset.

- All the power supplies are removed from the Backup domain, also known as a “cold-start”
- The firmware triggers a Backup domain reset using the RESET bit field in the BACKUP_RESET register

21.6 Real-time clock

Table 142 RTC fields

Bit field name	Number of bits	Description
RTC_SEC	6	Calendar seconds, 0-59
RTC_MIN	6	Calendar minutes, 0-59
RTC_HOUR	5	Calendar hours, value depending on 12/24 hour mode In 24 hour mode, bits RTC_HOURS[20:16] = 0-23 In 12 hour mode, bit RTC_HOURS[20]:0= AM, 1= PM, RTC_HOURS[19:16]=1-12
CTRL_12HR	1	Select 12/24 hour mode: 1=12 hr, 0=24 hr
RTC_DAY	3	Calendar day of the week, 1-7 The user defines the meaning of the values, assign 1= Monday is recommended
RTC_DATE	5	Calendar day of the Month, 1-31 Automatic leap year correction
RTC_MON	4	Calendar month, 1-12
RTC_YEAR	7	Calendar year, 0-99

The RTC supports both 12-hour format with AM/PM flag, and 24-hour format for “hours” field. The RTC also includes a “day of the week” field, which counts from 1 to 7. Programmer defines weekday to be 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 4, the month of February (Month = 2) has 29 days instead of 28. When the year reaches 2100 - the year field rolls over from 99 to 00 - the leap year correction will be wrong (2100 is flagged as a leap year which it is not); therefore, an interrupt is raised to allow the firmware to take appropriate actions. This interrupt is called the century interrupt.

The user registers containing these bit fields: BACKUP_RTC_TIME and BACKUP_RTC_DATE. See the corresponding register descriptions in the registers reference for details. 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 sets the READ bit in the BACKUP_RTC_RW register. The RTC registers is copied to user registers and frozen so that the firmware reads a coherent RTC value safely. Clear the READ bit to complete the transaction.

The READ bit cannot be set if:

- The RTC is still busy with a previous operation (i.e., the RTC_BUSY bit in the BACKUP_STATUS register is set)
- WRITE bit in the BACKUP_RTC_RW register is set

The firmware verifies the above bits are not set before the READ bit is set.

21.6.2 Writing to RTC User registers

When the WRITE bit in the BACKUP_RTC_RW register is set, data is written into the RTC user registers; otherwise, the RTC user registers are ignored. When all the RTC writes complete, the firmware clears the WRITE bit for the RTC update to take effect. After the WRITE bit is cleared, the hardware copies all the new data on one single WCO clock edge to ensure coherency to the actual RTC registers.

The WRITE bit cannot be set if:

- RTC is still busy with a previous operation (ixs.e., the RTC_BUSY bit in the BACKUP_STATUS register is set)
- READ bit in the BACKUP_RTC_RW register is set

The firmware updates the values written to the RTC fields to form a coherent legal set. The hardware does not check the validity of the written values. Writing illegal values results in an undefined behavior of the RTC.

In the middle of an RTC update with the WRITE bit set, and a brownout, reset, or entry to Deep Sleep or Hibernate mode occurs, the write operation will not complete. This is because the WRITE bit is cleared by a reset, and the RTC update is triggered only when this bit is cleared by a WRITE transaction. If the write operation is in progress (RTC_BUSY), data corruption occurs. Otherwise, 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 update only the hour field, although the seconds and minutes fields are disturbed and continues to run. In this case, an ‘Update’ flag is maintained for each RTC field. The updated fields are copied to the actual RTC when the BACKUP_RTC_RW.WRITE bit is cleared.

21.7 Alarm feature

The alarm feature allows the RTC to generate an interrupt, which is used to wake 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 is set to enable matching; if the bit is cleared, then the field will be ignored for matching.

The alarm bit fields are as follows:

Table 143 Alarm bit fields

Bit field name	Bit_Pos	Register_name	Description
ALM_SEC	5:0	BACKUP_ALMx_TIME	Alarm seconds, 0-59
ALM_SEC_EN	7	BACKUP_ALMx_TIME	Alarm second enable: 0=ignore, 1=match
ALM_MIN	13:8	BACKUP_ALMx_TIME	Alarm minutes, 0-59

(table continues...)

Table 143 (continued) Alarm bit fields

Bit field name	Bit_Pos	Register_name	Description
ALM_MIN_EN	15	BACKUP_ALMx_TIME	Alarm minutes enable: 0=ignore, 1=match
ALM_HOUR	20:16	BACKUP_ALMx_TIME	Alarm hours, value depends on 12/24 HR mode 24 HR: [4:0]=0-23 12 HR: [4]:0=AM, 1=PM, [3:0]=1-12
ALM_HOUR_EN	23	BACKUP_ALMx_TIME	Alarm hour enable: 0=ignore, 1=match
ALM_DAY	26:24	BACKUP_ALMx_TIME	Alarm day of the week, 1-7 It is up to the user to define the meaning of the values, but 1=Monday is recommended
ALM_DAY_EN	31	BACKUP_ALMx_TIME	Alarm day of the Week enable: 0=ignore, 1=match
ALM_DATE	4:0	BACKUP_ALMx_DATE	Alarm day of the Month, 1-31. Leap year corrected
ALM_DATE_EN	7	BACKUP_ALMx_DATE	Alarm day of the month enable: 0=ignore, 1=match
ALM_MON	11:8	BACKUP_ALMx_DATE	Alarm month, 1-12
ALM_MON_EN	15	BACKUP_ALMx_DATE	Alarm month enable: 0=ignore, 1=match
ALM_EN	31	BACKUP_ALMx_DATE	Master enable for alarm 1. 0: alarm 1 is disabled. Fields for date and time are ignored. 1: alarm 1 is enabled. The alarm triggers when the new date and time matches all the enabled date and time fields, which can happen more than once depending on configuration. If none of the date and time fields are enabled, then this alarm triggers once every second.

If the master enable (ALM_EN) is set, but all the alarm fields for date and time are disabled, an alarm interrupt is generated once every second. Note that there is no alarm field for year.

The PSC3M/P8, PSC3M/P7 MCU has two independent alarms. See the BACKUP_ALM1_TIME, BACKUP_ALM1_DATE, BACKUP_ALM2_TIME, and BACKUP_ALM2_DATE registers in the registers reference manual for more details.

***Note**: The alarm user registers, similar to RTC user registers, require certain special steps before read/write operations, as explained in the Reading RTC User registers and Writing to RTC User registers in the reference manual.*

Interrupts must be properly configured for the RTC to generate interrupts/wake up events. Also, to enable RTC interrupts to wake up the device from Hibernate mode, the HIB_WAKE_RTC bit in the PWR_HIB_WAKE_CTL register must be set. See the Device power modes chapter and Interrupts chapters for more details.

The BACKUP_INTR_MASK register is used to disable certain interrupts from the Backup system.

Table 144 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 (interrupt generated when the year field rolls over from 99 to 00)

21.8 Backup registers

The Backup domain has thirty two registers (BACKUP_BREG_SET0 to BACKUP_BREG_SET3). Each of these register can be used to store 1 byte of important user information/flags. These registers can retain their contents even when the main supply (VDDD) is off as long as backup supply (VBACKUP) is present. These registers can also be used to store information that must be retained when the device enters Hibernate mode.

21.9 Registers summary

Table 145 Backup registers

Register name	Description
BACKUP_CTL	Main control register (including power and clock)
BACKUP_RTC_RW	RTC read/write control register
BACKUP_STATUS	Status register
BACKUP_RTC_TIME	Calendar seconds, minutes, hours, and day of week
BACKUP_RTC_DATE	Calendar day of month, month, and year
BACKUP_ALM1_TIME	Alarm 1 seconds, minute, hours, and day of week
BACKUP_ALM1_DATE	Alarm 1 day of month, and month
BACKUP_ALM2_TIME	Alarm 2 seconds, minute, hours, and day of week
BACKUP_ALM2_DATE	Alarm 2 day of month, and month
BACKUP_INTR	Interrupt Request register
BACKUP_INTR_MASK	Interrupt Mask register

(table continues...)

Table 145 (continued) Backup registers

Register name	Description
BACKUP_INTR_MASKED	Interrupt Masked Request register
BACKUP_BREG_SET0 - BACKUP_BREG_SET3	Backup registers
BACKUP_RESET	Reset Register for the Backup domain

22 Reset system

PSC3M/P8, PSC3M/P7 supports several types of resets that ensure error-free operation during power-up and allow the device to reset based on user-supplied external hardware or internal software reset signals. The resets, which have a broad scope and are generally aligned with power domains and global power modes, 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 propagated, and reset deassertion is synchronized to each clock domain where it is used. Additionally, PSC3M/P8, PSC3M/P7 contains the hardware that has the capability to report the reset that occurs.

Furthermore, 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 the faults generated by the safety circuits, such as clock supervision, supply supervision, and multi-counter watchdog timer.

22.1 Features

The PSC3M/P8, PSC3M/P7 MCU has the following reset sources:

- Brownout detection (BOD) to reset the device if the power supply falls below the device specifications during normal operation
- External reset (XRES) to reset the device using an external input
- Watchdog timer (WDT/MCWDT) reset to reset the device if the firmware execution fails to periodically service the watchdog timer
- Software-initiated reset(SYS) to reset the device on demand using the firmware
- Test controller or debugger asserted reset(TC_DEBUG)
- Fault detection reset to reset the device if certain faults occur(FAULT_ACT)
- Clock supervision triggered reset for high-frequency clock-related errors (CSV_HF)
- Clock supervision triggered reset for reference clock-related errors (CSV_REF)
- Hibernate wakeup reset to bring the device out of the Hibernate low-power mode(HIB WAKEUP)
- Programmable XRES (PXRES) to trigger a reset equivalent to XRES from the software

22.2 Reset sources

The following sections describe the reset sources available in the PSC3M/P8, PSC3M/P7 MCU family. Table 146 includes a list of all the reset sources.

Table 146 PSC3M/P8, PSC3M/P7 MCU reset sources

Reset source	Reset condition	Availability in power modes	Observable	SRAM retention
External reset (XRES)	External reset pin asserted low	All	No	No
Brownout detect (BOD)	Brownout detected for VDDD, VCCD, and backup supply	All except Off, XRES, Hibernate	No	No
WDT reset	Watchdog timer triggered reset action	Active, Sleep, Deep Sleep	Yes	Yes[1)]

(table continues...)

Table 146 (continued) PSC3M/P8, PSC3M/P7 MCU reset sources

Reset source	Reset condition	Availability in power modes	Observable	SRAM retention
MCWDT reset	Multi-counter WDT triggered reset action	Active, Sleep, Deep Sleep	Yes	Yes[1)]
Hibernate wakeup (HIB_WAKEUP)	Exit from Hibernate mode	Hibernate	Yes (on PWR_HIBERNATE.TOKEN field)	No
Internal system reset (SYS)	System reset (generated internally by CPUSS using SYSRESETREQ)	Active, Sleep	Yes	Yes[1)]
Test/Debug reset (TC_DEBUG)	Test controller or debugger asserted reset	Active, Sleep	Yes	Yes[1)]
Fault detection active logic (FAULT_ACT)	Fault manager triggered reset	Active, Sleep	Yes	Yes[1)]
Clock supervision (CSV_HF)	Loss or frequency violation of a high-frequency clock HF0 – HF6.	Active, Sleep	Yes	CSV_HF0: NO[2)] CSV_HF1..6: YES[1)]
Clock supervision (CSV_REF)	Loss or frequency violation of the reference clock source that is used to monitor the other HF clock sources.	Active, Sleep	Yes	No
Programmable XRES (PXRES)	CPU-triggered reset with scope similar to XRES (triggered using RES_PXRES_CTL.PXRES_TRIGGER).	Active, Sleep	No	No

1) Assumes orderly shutdown handled by the processor before the reset is triggered.
2) Since system RAM is clocked by clk_hf0, a reset triggered by CSV_HF0 could lead to an unstable RAM clock and corrupted contents.

***Note**: None of these sources can reset the backup system. The backup domain is reset only when all the power supplies are removed from it (also known as a cold start) or if the firmware triggers a reset using the BACKUP_RESET register. For more details, see Backup.*

The Table 147 shows the mapping of reset sources to the destinations affected by the reset. An 'X' indicates that the block has at least some register/bits that are affected by the corresponding reset.

Table 147 PSC3M/P8, PSC3M/P7 MCU reset cause distribution

Reset type/ Source	Reset Cause bits	Data registers in FAULT structure	Backup registers	Debug logic	CM33
XRES	X	X		X	X
BOD	X	X	X[1)]	X	X
WDT		X		X[2)]	X
MCWDT		X		X[2)]	X
HIB_WAKEUP	X	X		X	X
SYS		X			X
TC_DEBUG		X		X	X
FAULT_ACT		X			X
CSV_HF		X			X
CSV_REF		X			X

1) If the backup power supply is not present.
2) If the reset source is triggered during Deep Sleep.

***Note**: Watchdog and clock supervision faults are special cases because it is possible that the fault handling system has crashed. For example, a clock supervision fault may mean that a clock glitch occurred when the clock failed. Although unlikely, this glitch could lock up the fault handling system. In this case, the fault logging system may not log the fault, but the reset can be observed.*

22.2.1 Brownout reset

Brownout detection circuits monitor the device's digital voltage supply (VDDD) and internally generated supply voltage (VCCD) and generate a reset if they fall below their voltage threshold. The device stays in reset until all brownout detectors are released. This also occurs during an initial power ramp. See the Power supply and monitoring chapter for more details.

22.2.2 WDT/MCWDT reset

The watchdog timer reset causes a reset if the WDT or multi-counter WDT (MCWDT) are not serviced by the firmware within a specified time limit. In the window mode, MCWDT generates a reset if it is serviced earlier than the specified time. See the Watchdog timer chapter for more details.

22.2.3 Software reset

Software reset is a mechanism that allows the software or a connected debugger to request a system reset. Cortex®-M33 can request a reset by writing a '1' to the TRIGGER_SOFT bit of the RES_SOFT_CTL register. This can also be done via a debugger probe.

In addition to the software reset using the RES_SOFT_CTL register, this MCU also has an option to trigger XRES from the Arm® Cortex®-M33. The XRES interrupt can be triggered by writing a '1' to the PXRES_TRIGGER bit of the RES_PXRES_CTL register. RES_PXRES_CTL is a software-accessible register that triggers a full scope reset equivalent to XRES.

22.2.4 External reset

The external reset (XRES) is a reset triggered by an external signal, causing an immediate system reset when asserted. The XRES pin is active low – a logic '1' on the pin has no effect, while a logic '0' causes a reset. The pin is pulled to logic '1' inside the device and is available as a dedicated pin. For detailed pinout information, refer to the Pinout section of the device datasheet.

The XRES 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 and other electrical characteristics are listed in the Electrical Specifications section of the device datasheet. XRES is available in all power modes but cannot reset the backup system.

22.2.5 Debugger reset

The debugger reset is asserted by the test controller or debugger. The DP_CTL_STAT register in the SWD/JTAG-DP has a reset request bit (CDBGRSTREQ), and this request is sent to the reset controller to create the debug reset signal.

The debugger reset can be identified from the RESET_TC_DBGRESET bit field of the RES_CAUSE register.

22.2.6 Fault detection reset

The fault reporting structure in PSC3M/P8, PSC3M/P7 can be configured to request a reset for user-configurable faults.

Faults that occur during Deep Sleep do not wake the system but are pending on wake. If a condition requires a wake from Deep Sleep, it should be configured as an interrupt to the WIC.

Faults generated by clock supervision and MCWDTs can indicate that the fault system may have also failed. These circuits can be configured to directly cause a reset.

22.2.7 High frequency clock error reset

The clock supervision logic initiates a reset due to the loss of a high-frequency clock or watch-crystal clock, or due to a high-frequency clock error. The RESET_CSV_HF is a 7-bit field in the RES_CAUSE2 register that can be used to identify resets caused by the loss or frequency violation of a high-frequency clock.

For more information on clocks, see the Clocking system chapter.

22.2.8 Reference clock error reset

The clock supervision logic requests a reset due to the loss or frequency violation of the reference clock source used to monitor other high-frequency clock sources. The RESET_CSV_REF field in the RES_CAUSE2 register can be used to identify resets caused by the loss or frequency violation of the reference clock source used to monitor the other high-frequency clock sources.

For more information on clocks, see the Clocking system chapter.

22.2.9 Hibernate wakeup reset

The Hibernate wakeup reset occurs when one of the Hibernate wakeup sources performs a device reset to return to the Active power mode. See the Device power modes chapter for details on Hibernate mode and 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 bitfield to differentiate hibernate wakeup from a general reset event.

22.3 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. These registers have specific status bits allocated for some of the reset sources. They record the occurrences of WDT reset, MCWDT reset, software reset, logic-protection fault reset, debugger reset, and clock-supervision resets. However, they do not record the occurrences of BOD, XRES, PXRES, or Hibernate wakeup resets. The bits in these registers are set on the occurrence of the corresponding reset and remain set after the reset until cleared by the firmware or a loss of retention such as XRES or BOD resets.

Hibernate wakeup resets can be detected by examining the TOKEN field in the PWR_HIBERNATE register as described previously. However, Hibernate wakeup resets that occur as a result of an XRES cannot be detected. The reset causes in the RES_CAUSE and RES_CAUSE2 registers are shown in Table 148.

After identifying and evaluating the reset cause, it is necessary to clear the RESET_CAUSE and RESET_CAUSE2 registers. This procedure is required to capture the next reset.

Table 148 Reset cause bits to detect reset source

Register [bit pos]	Bitfield	Number of bits	Description
RES_CAUSE[0]	RESET_WDT	1	A hardware WDT reset has occurred since the last power cycle.
RES_CAUSE[1]	RESET_ACT_FAULT	1	Fault logging system requested a reset from its Active logic.
RES_CAUSE[3]	RESET_TC_DBGRESET	1	Test controller or debugger asserted reset. Only resets debug domain.
RES_CAUSE[4]	RESET_SOFT	1	CPU requested a system reset through its TRIGGER_SOFT. This can be done via a debugger probe or from firmware.
RES_CAUSE[5]	RESET_MCWDT0	1	Multi-counter WDT reset has occurred since the last power cycle.
RES_CAUSE2[6:0]	RESET_CSV_HF	7	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	1	Clock supervision logic requested a reset due to loss or frequency violation of the reference clock source that is used to monitor the other HF clock sources.

For more information, see the RES_CAUSE and RES_CAUSE2 registers in the register's reference manual.

If these methods cannot detect the cause of the reset, then it may be an unrecorded and a non-retention reset. These resets cannot be distinguished using on-chip resources.

22.4 Registers summary

Table 149 Reset system registers list

Register	Description
RES_CAUSE	Reset cause observation register
RES_CAUSE2	Reset cause observation register 2
PWR_HIBERNATE	Hibernate power mode control register. Contains a TOKEN field that can be used to detect the Hibernate wakeup reset
RES_SOFT_CTL	Soft reset trigger register
RES_PXRES_CTL	Control register for programmable external reset (PXRES).

23 I/O subsystem

23.1 Features

The PSC3M/P8, PSC3M/P7 GPIOs have the following features:

- Drive modes
  - Strong drive
  - Resistive pull-up
  - Resistive pull-down
  - Open drain, drives low
  - Open drain, drives high
  - Resistive pull-up or pull-down
  - High impedance
- Four drive strength options
- Input threshold selection (CMOS or LVTTL)
- Individual control of input and output disables
- Hold mode for latching the previous state, which retains the I/O state during Deep Sleep and Hibernate modes
- Selectable slew rates for dV/dt-related noise control
- Support for up to two PMBus signal outputs with GPIOs featuring overvoltage tolerance (OVT) functionality
- Up to eight GPIOs multiplexed with analog inputs to provide additional analog input connections (a maximum of four of these eight GPIOs can be shared with SMBus-compliant OVT GPIOs)

23.2 Architecture

The PSC3M/P8, PSC3M/P7 MCU is equipped with analog and digital peripherals. Figure 83 shows an overview of the routing between the peripherals and pins.

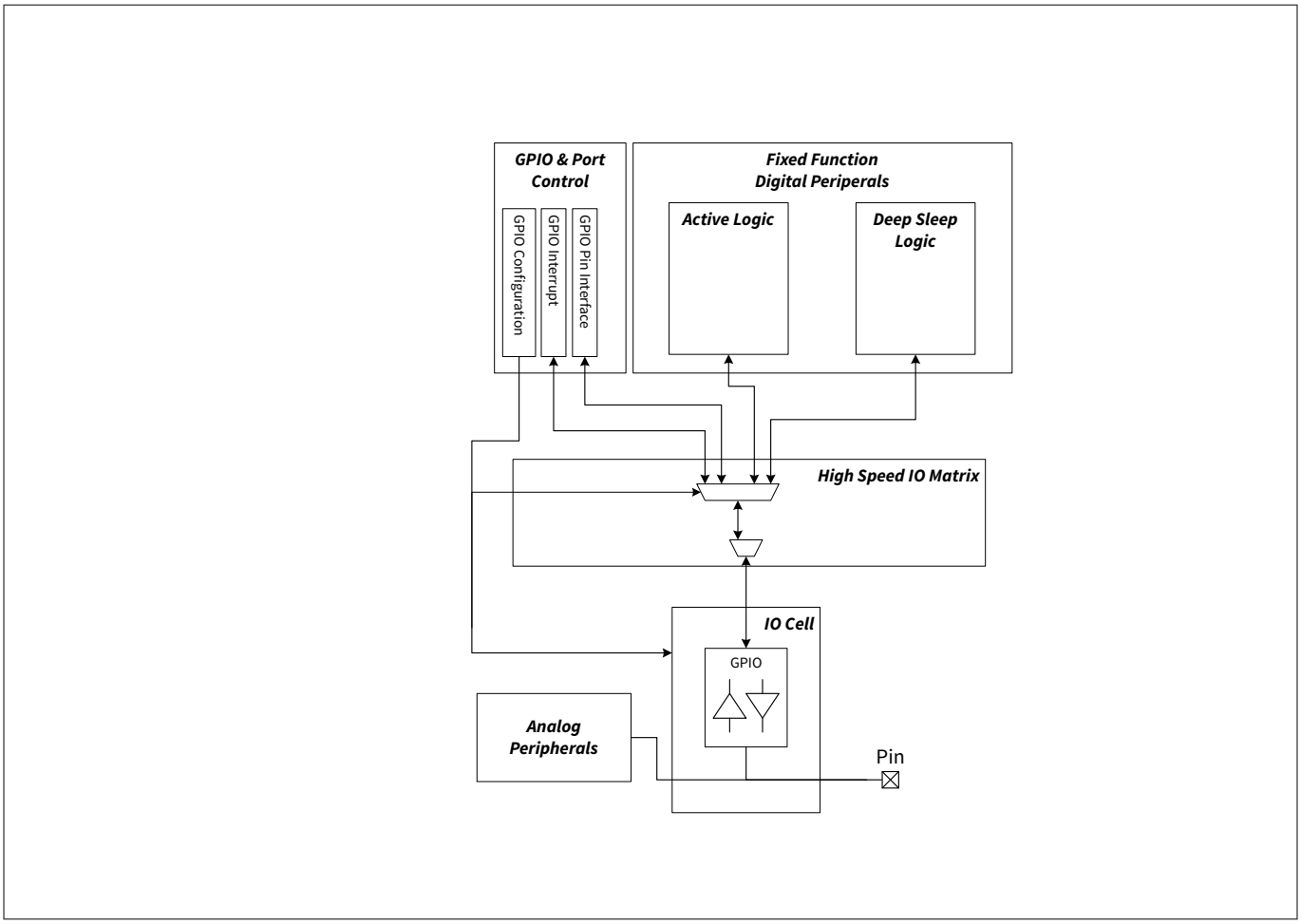


Figure 83 GPIO interface overview

GPIO pins are connected to I/O cells. These cells are equipped with an input buffer for the digital input, providing high-input impedance and a driver for the digital output signals.

The digital peripherals connect to the I/O cells via the high-speed I/O matrix (HSIOM). The HSIOM for each pin contains multiplexers to connect between the selected peripheral and the pin. Additionally, HSIOM bridges the connection between the digital system interconnect (DSI) and the pins.

23.2.1 I/O cell architecture

Figure 84 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.

The GPIO component provides the I/O cell configuration information through the 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 or 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.

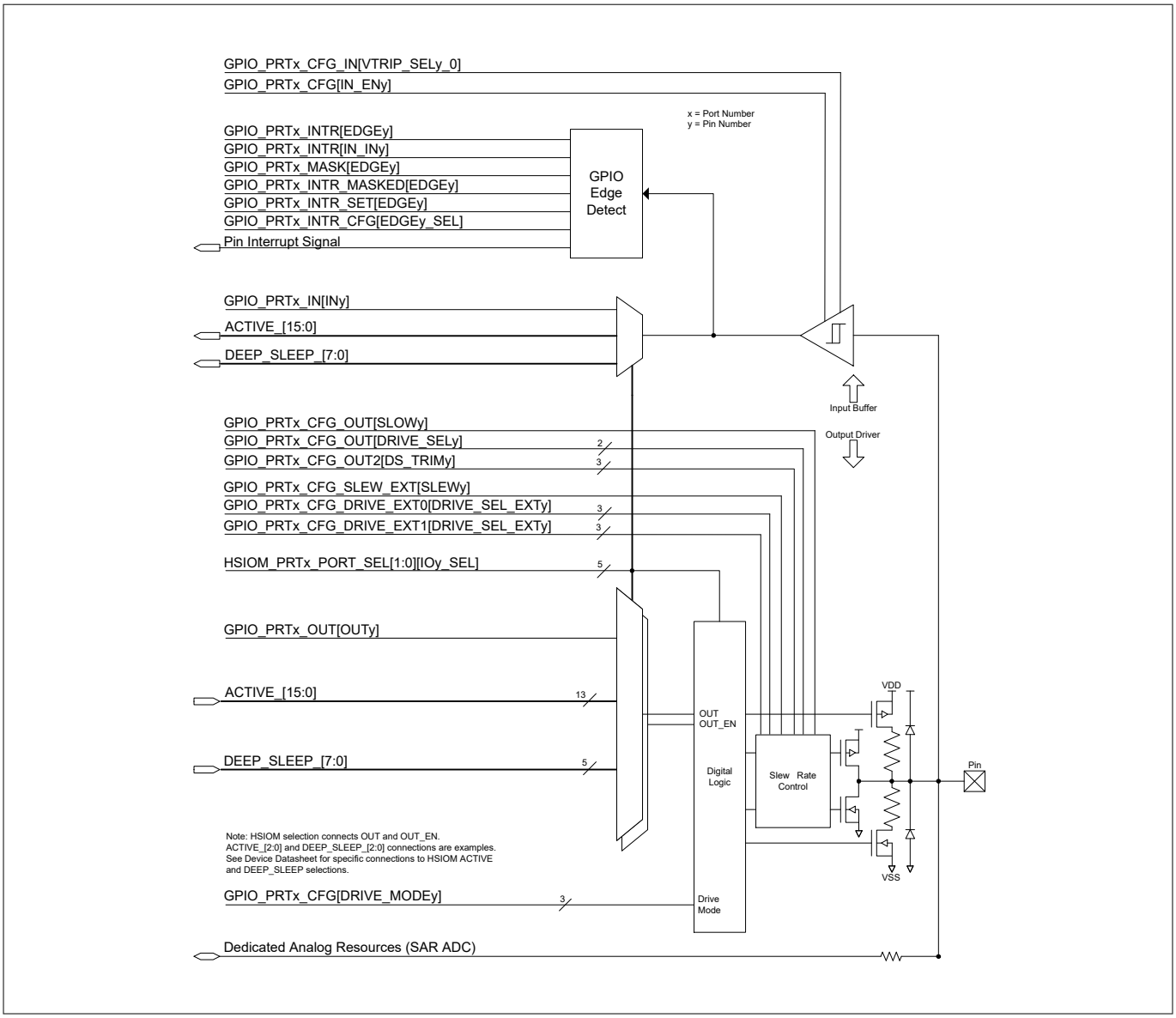


Figure 84 GPIO cell architecture

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, parallel test mode 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.

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 the control of the HSIOM register fields.

23.2.2 HSIO to map alternate functions on GPIO

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 user-selected peripheral. The HSIOM_PRTx_PORT_SEL[1:0] registers allow a single selection from up to 32 different connections to each pin as listed in Table 150.

Table 150 HSIOM connections

Alternate function 5-bit[4:0]	Name	Digital driver signal source		Digital 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 the datasheet for specific pin connectivity
9	ACT_1	Active Source OUT	Active Source OUT_EN	Active Source IN	Active functionality 1 - See the datasheet for specific pin connectivity
10	ACT_2	Active Source OUT	Active Source OUT_EN	Active Source IN	Active functionality 2 - See the datasheet for specific pin connectivity
11	ACT_3	Active Source OUT	Active Source OUT_EN	Active Source IN	Active functionality 3 - See the datasheet for specific pin connectivity

(table continues...)

Table 150 (continued) HSIOM connections

Alternate function 5-bit[4:0]	Name	Digital driver signal source		Digital 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 the 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 the 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 the 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 the datasheet for specific pin connectivity
16	ACT_4	Active Source OUT	Active Source OUT_EN	Active Source IN	Active functionality 4 - See the datasheet for specific pin connectivity
17	ACT_5	Active Source OUT	Active Source OUT_EN	Active Source IN	Active functionality 5 - See the datasheet for specific pin connectivity
18	ACT_6	Active Source OUT	Active Source OUT_EN	Active Source IN	Active functionality 6- See the datasheet for specific pin connectivity

(table continues...)

Table 150 (continued) HSIOM connections

Alternate function 5-bit[4:0]	Name	Digital driver signal source		Digital signal destination	Description
		OUT	OUT_EN		
19	ACT_7	Active Source OUT	Active Source OUT_EN	Active Source IN	Active functionality 7 - See the datasheet for specific pin connectivity
20	ACT_8	Active Source OUT	Active Source OUT_EN	Active Source IN	Active functionality 8 - See the datasheet for specific pin connectivity
21	ACT_9	Active Source OUT	Active Source OUT_EN	Active Source IN	Active functionality 9 - See the datasheet for specific pin connectivity
22	ACT_10	Active Source OUT	Active Source OUT_EN	Active Source IN	Active functionality 10 - See the datasheet for specific pin connectivity
23	ACT_11	Active Source OUT	Active Source OUT_EN	Active Source IN	Active functionality 11 - See the datasheet for specific pin connectivity
24	ACT_12	Active Source OUT	Active Source OUT_EN	Active Source IN	Active functionality 12 - See the datasheet for specific pin connectivity
25	ACT_13	Active Source OUT	Active Source OUT_EN	Active Source IN	Active functionality 13 - See the datasheet for specific pin connectivity

(table continues...)

Table 150 (continued) HSIOM connections

Alternate function 5-bit[4:0]	Name	Digital driver signal source		Digital signal destination	Description
		OUT	OUT_EN		
26	ACT_14	Active Source OUT	Active Source OUT_EN	Active Source IN	Active functionality 14 - See the datasheet for specific pin connectivity
27	ACT_15	Active Source OUT	Active Source OUT_EN	Active Source IN	Active functionality 15 - See the 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 the 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 the 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 the 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 the datasheet for specific pin connectivity

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

23.2.3 I/O state on power-up

At power-up, all GPIOs are in a high-impedance analog state, and the input buffers are disabled. During run-time, you can configure GPIOs by writing to the associated registers. However, the pins that support the debug

access port (DAP) connections (SWD lines) are always enabled as SWD lines during power-up. If these pins are left floating, there is a risk of crowbar current as the DAP connection does not provide pull-up or pull-down resistors. To avoid this, you can disable or reconfigure the DAP connection for general-purpose use through the HSIOM, but only after the device has booted and started executing code.

23.2.4 Behavior in low-power modes

Table 151 shows the status of GPIOs in low-power modes.

Table 151 Behavior in low-power modes

Low-power mode	State
Sleep	Standard GPIO, GPIO-OVT, and SIO pins are active and can be driven by most peripherals such as TCPWMs, and SCBs, which can operate in Sleep mode. Inputs buffers are active; so an interrupt on any I/O can be used to wake the CPU.
Deep Sleep	GPIO, GPIO-OVT, and SIO pins, connected to Deep Sleep domain peripherals, are functional. All other pins 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 frozen state. Pin interrupts are functional only on select IOs and can be used to wake the device. See the device datasheet for specific hibernate pin connectivity.

23.3 Functional Description

23.3.1 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 + I2C
- TTL

Use the `GPIO_PRTx_CFG_IN.VTRIP_SELy_0` bit to select CMOS and TTL buffer modes.

***Note**: The trip levels of CMOS and TTL are shown in Figure 85.*

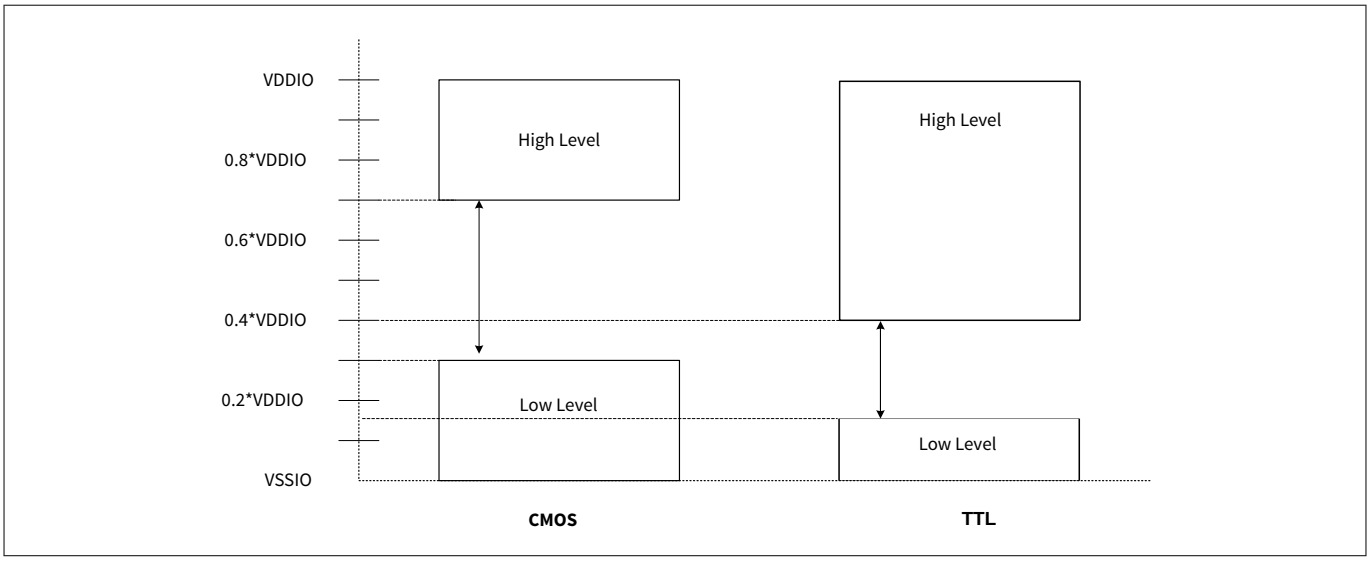


Figure 85 Input buffer mode's tripping levels

23.3.2 Digital output driver

The digital output driver drives the pins. 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 VDDIO/VDDD/VDDA or drop below VSSIO/VSSD/VSSA. For the absolute maximum and minimum GPIO voltage, see the device datasheet.

The output data register (`GPIO_PRTx_OUT`) associated with the output pin enables or disables the digital output driver in the hardware. Peripherals other than the GPIO port, directly control both the output and the output enable signals of the output buffer.

23.3.2.1 GPIO-OVT pins

Select device pins are designed to be over voltage tolerant (OVT) and can be used to interface with busses or signals that may exceed the VDDIO supply of the pin or the entire device, or when the VDDIO supply may not always be present. GPIO-OVT pins are similar to regular GPIOs but have the added feature of being overvoltage tolerant. These pins include hardware that compares the voltage on the pin to the VDDIO voltage. If the voltage on the GPIO-OVT pin exceeds VDDIO, the output driver is disabled, and the pin driver is tri-stated. This results in negligible current sinking at the pin. See the device datasheet to know which pins are over voltage tolerant.

***Note**: In over voltage conditions, the input buffer data is not valid if the external source's specification of VOH and VOL do not match the trip points of the input buffer defined by the current VDDIO voltage.*

23.3.2.2 Drive strength

The drive strength of an I/O specifies how much current an I/O pin can drive and sink while maintaining the minimum VOH and VOL levels. When the output is configured for a fast slew rate, the port drive strength can be set to one of four levels using the DRIVE_SEL field of the port output configuration register (`GPIO_PRTx_CFG_OUT`). The drive strength field determines the active portion of the output driver used and can affect the slew rate of output signals. The available drive strength options are full drive strength (default), one-half strength, one-

quarter strength, and one-eighth strength. The drive strength must be set to full drive strength when the slow slew rate bit (SLOW) is set.

The I/O port provides two advanced registers, known as the Port output Drive Select Extension Register (GPIO_PRTx_CFG_DRIVE_EXT[0-1]), which allow the drive strength of each pin in the respective port to be configured individually.

Table 152 Drive select for GPIO_STD

Drive select	DRIVE_SEL[0:1]	Description
DRIVE_SEL_THREE	11	1/8 drive strength: GPIO drives current at one-eighth of its maximum rated specification.
DRIVE_SEL_TWO	10	1/4 drive strength: GPIO drives current at one-quarter of its maximum rated specification.
DRIVE_SEL_ONE	01	1/2 drive strength: GPIO drives current at one-half of its maximum rated specification
DRIVE_SEL_ZERO	00	Full drive strength: GPIO drives current at its maximum rated specification.

23.3.2.3 Slew rate control

Some GPIO pins support fast and slow output slew rate options for strong drivers, which are configured using the SLOW bit in the Port Output Configuration register (GPIO_PRTx_CFG_SLEW_EXT). By default, this bit is cleared, and the port operates in fast slew mode. The bit can be set when a slow slew rate is required. A slower slew rate reduces EMI and crosstalk and is recommended for low-frequency signals or signals without strict timing constraints.

When configured for a fast slew rate, the drive strength can be set to one of four levels defined in Drive strength.

23.3.2.4 Drive mode

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 153 lists the drive modes using GPIO_PRTx_CFG register. It provides a comprehensive breakdown of the different modes in two distinct columns: GPIO functionality and fixed peripheral functionality.

***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 150.*

Table 153 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	-	-	-	-	-	-	-	-

(table continues...)

Table 153 (continued) 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
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	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

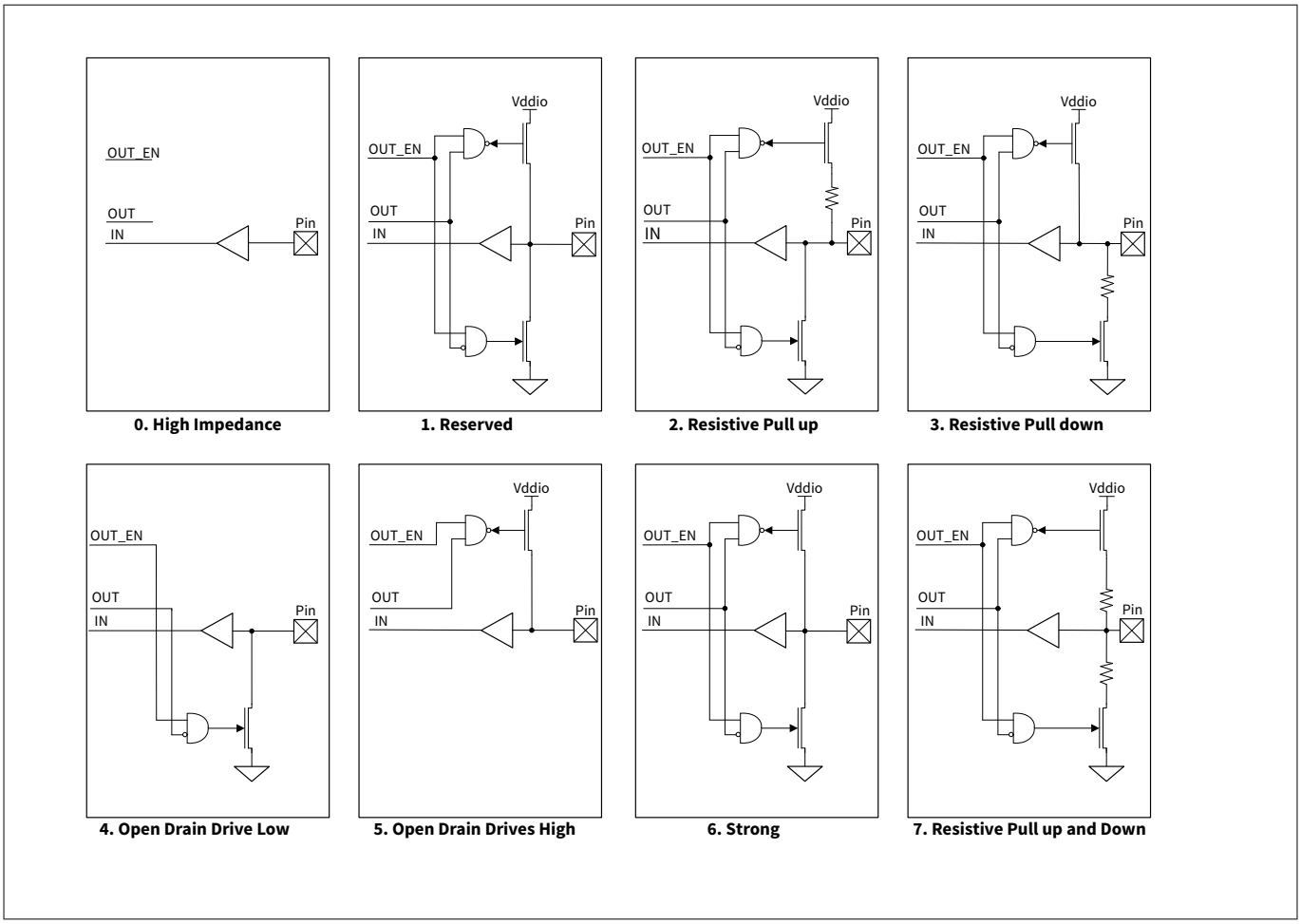


Figure 86 GPIO port drive mode block diagrams

Figure 87 is an output driver diagram that shows the pin view for fixed-function-based peripherals in all available drive modes.

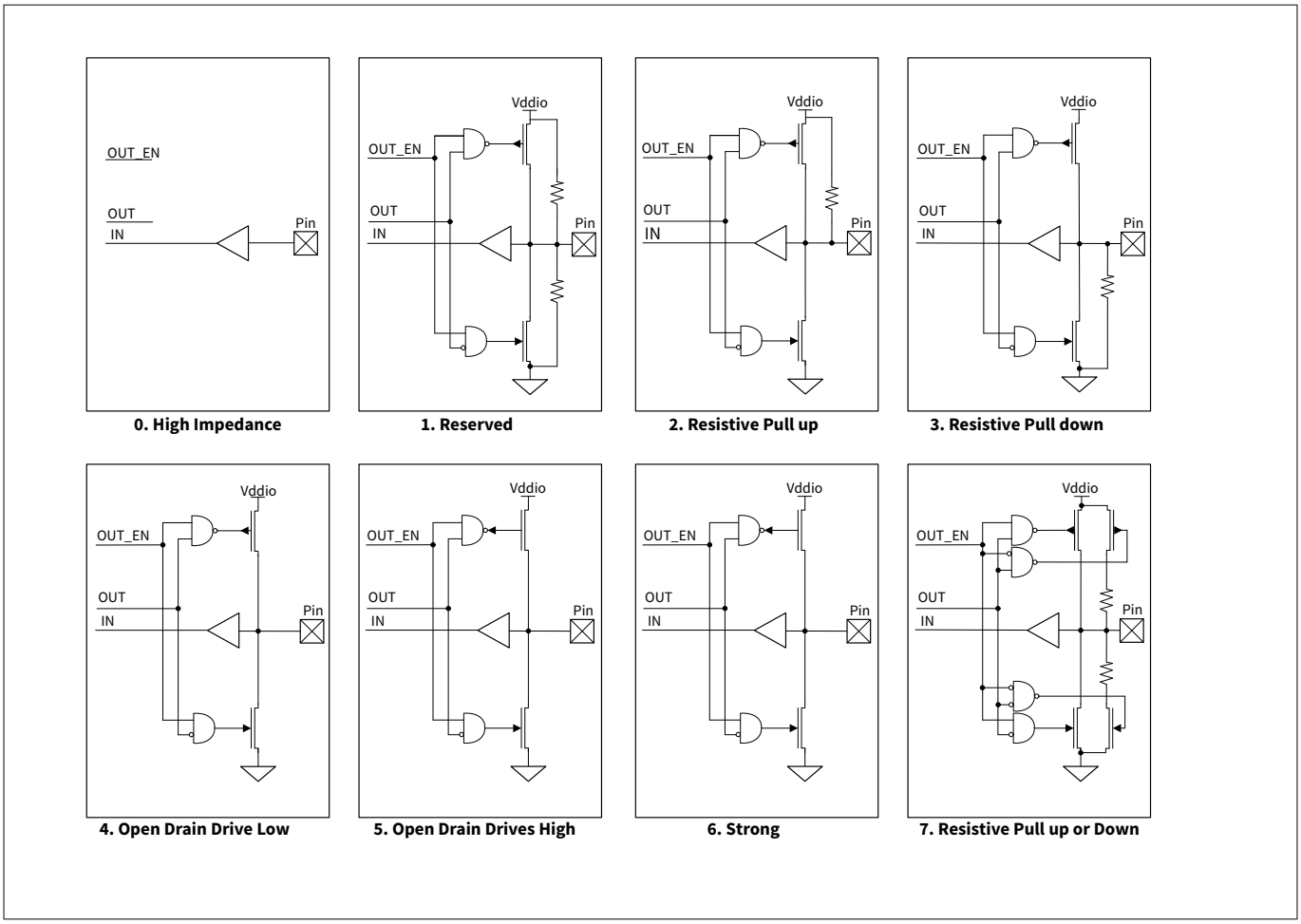


Figure 87 Fixed function peripheral I/O drive mode block diagrams

1. **High-impedance(0)**

   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

2. **Resistive pull-up mode(2) and resistive pull-down mode(3)**

   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 the MCU 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

3. **Open drain drives low(4) and open drain drives high(5)**

   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 bidirectional 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 drive low mode is driving I2C bus signal lines

4. **Strong drive(6)**

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

5. **Resistive pull-up or resistive pull-down(7)**

   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

23.3.3 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, the following three registers are provided to set, clear, and invert the output data respectively on specific pins in a port without affecting other pins: GPIO_PRTx_OUT_SET, GPIO_PRTx_OUT_CLR, and GPIO_PRTx_OUT_INV . 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.

23.3.4 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 88.

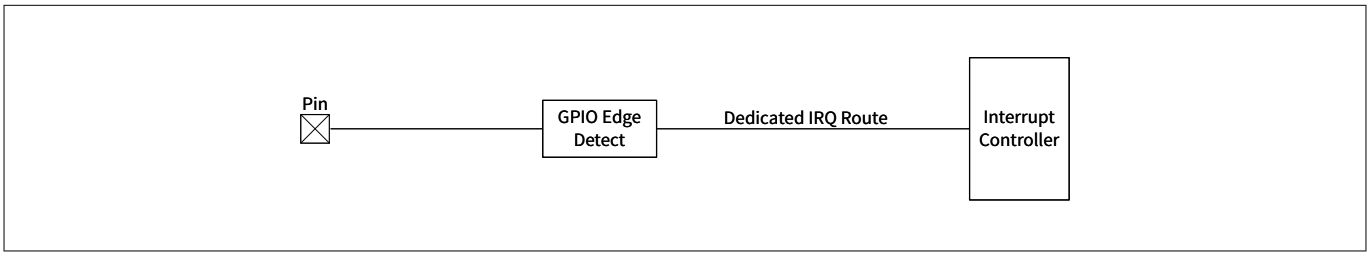


Figure 88 Interrupt signal routing

Pin signals 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 89 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 89 to form a port interrupt interrupts_gpio[i].

In addition, a 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 (such as 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 88 illustrates the interrupt functionality. Figure 89 shows the GPIO edge detect block architecture.

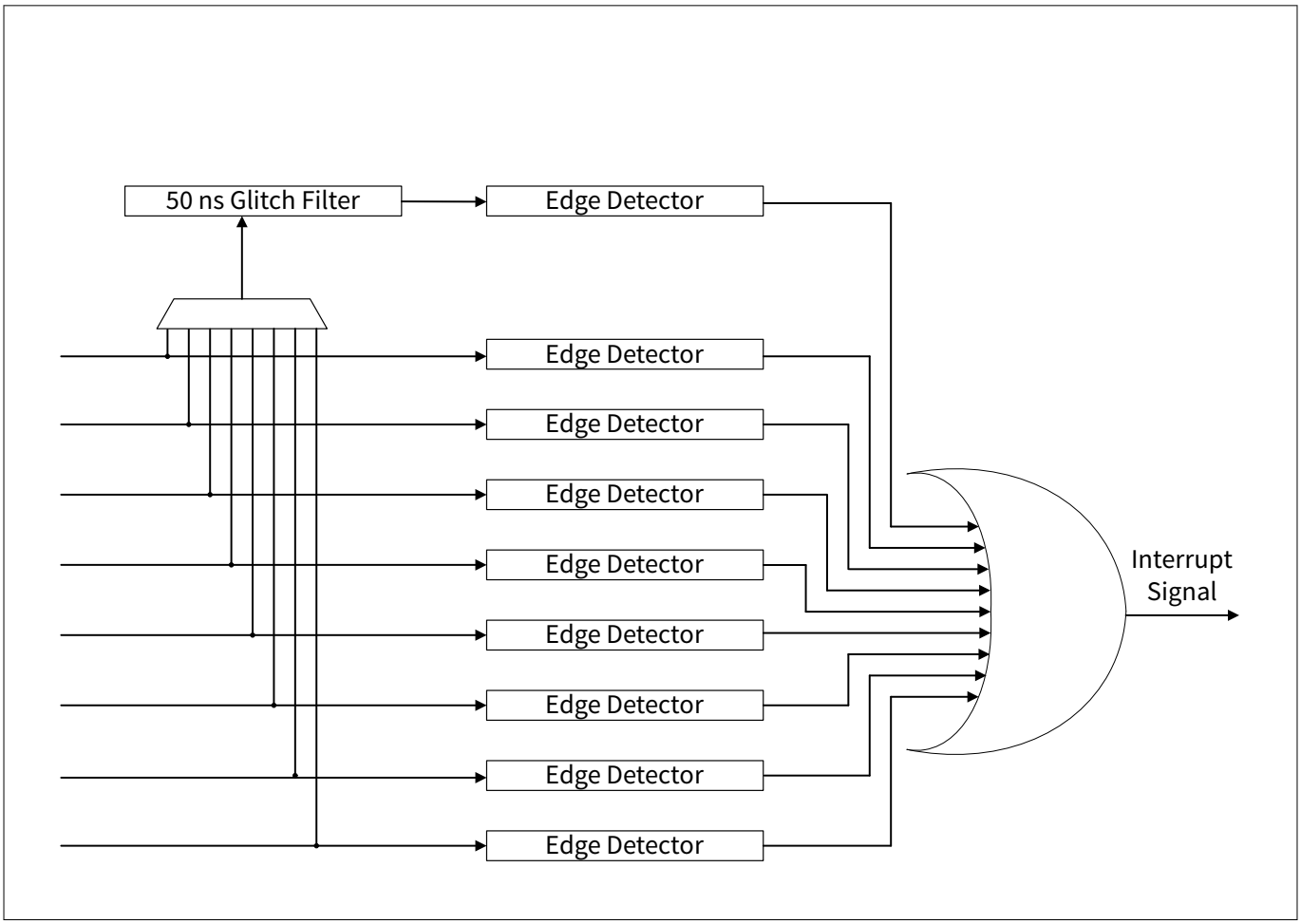


Figure 89 GPIO edge detect block architecture

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. The edge detector is configured by writing into the GPIO_PRTx_INTR_CFG.EDGEy_SEL field, as shown in Table 154.

Table 154 Edge detector configuration

EDGE_SEL	Configuration
00	Interrupt is disabled
11	Interrupt on both edges
10	Interrupt on falling edge
01	Interrupt on rising edge

Writing '1' to the corresponding status bit clears the pin edge state. Clearing the edge state status bit is a critical step in utilizing the pin's edge state. If left unchecked, this may result in repeated interrupt triggers for a single event or a lack of response for multiple events, as explained further in this section. It is advisable to only read the Port Interrupt Control status register within the corresponding interrupt service routine, as reading the register during an ongoing edge on the port may result in the edge going undetected.

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 155.

Table 155 Glitch filter input section

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, 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 or enable the forwarding of an interrupt to the CPUs. The status of an interrupt can be checked with the GPIO_VDD_INTR register.

23.3.5 Analog I/O

Analog resources, such as LPCOMP and external crystal pins for ECO and WCO, which require low-impedance routing paths, have fixed pins. These pins provide direct connections to specific analog blocks using the

AMUXBUS. 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. To configure a GPIO as an analog pin connecting to AMUXBUS, it should be configured in high-impedance analog mode with the input buffer disabled and then routed to the correct AMUXBUS using the HSIOM_PORT_SELx register.

***Note**: There is no AMUXBUS in the PPCA, for PPCA dedicated signals information, refer to Programmable Power Control Accelerator (PPCA) section.*

23.3.6 Secure GPIO

Secure GPIO provides secure configuration per IO pin. All IO pins default to non-secure. All AHB accesses associated with the pin configuration are subjected to the security attribute check as below:

1. Secure transactions can only access MMIO registers for pins that are configured as secure
2. Non-secure transactions can only access MMIO registers for pins that are configured as non-secure

Violation of the above rules results in Read Zero Write Ignore (RZWI) behavior. Therefore, software needs to ensure that the function/peripheral selected (via `HSIOM_PRTx.PORT_SEL0.IOy_SEL`) to access the IO pin has a matching security profile.

For MMIO registers common to all ports, and registers that are organized per port are exempted from this check. If a security check is required to access these MMIO registers, it must be placed in a secure region that is protected by a PPC.

23.4 Registers summary

Table 156 Registers summary

Registers	Name	Description
`GPIO_PRTx_OUT`	Port Output register	Reads and writes the output driver data for I/O pins in the port.
`GPIO_PRTx_OUT_CLR`	Port Output Clear register	Clears output data of specific I/O pins in the port.
`GPIO_PRTx_OUT_SET`	Port Output Set register	Sets output data of specific I/O pins in the port.
`GPIO_PRTx_OUT_INV`	Port Output Invert register	Inverts output data of specific I/O pins in the port.
`GPIO_PRTx_IN`	Port Input register	Reads the current pin state present on I/O pin inputs.
`GPIO_PRTx_INTR`	Port Interrupt Status register	Reads the current pin interrupt state.
`GPIO_PRTx_INTR_MASK`	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.

(table continues...)

Table 156 (continued) Registers summary

Registers	Name	Description
GPIO_PRTx_INTR_MASKED	Port Interrupt Masked Status 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	Allows firmware to set pin interrupts.
GPIO_PRTx_INTR_CFG	Port Interrupt Configuration register	Selects the edge detection type for each pin interrupt.
GPIO_PRTx_CFG	Port Configuration register	Selects the drive mode and input buffer enable for each pin.
GPIO_PRTx_CFG_IN	Port Input Configuration register	Configures the input buffer mode (CMOS or TTL) for each pin. VTRIP_SEL[7:0]_0.
GPIO_PRTx_CFG_OUT	Port Output Configuration register	Configures the port output buffer drive strength.
GPIO_PRTx_CFG_SLEW_EXT	Port Output Configuration register	Configures the port output buffer slew rate.
HSIOM_PRTx_PORT_SELy	HSIOM Port Select Register	Selects the hardware peripheral connection to I/O pins.
GPIO_INTR_CAUSEz	Interrupt Port Cause 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	Provides external power supply status.
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	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	Contains the AND-ed values of VDD_INTR and VDD_INTR_MASK Registers.
GPIO_VDD_INTR_SET	Supply Detection Interrupt Set Register	Allows a 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.
GPIO_VDD_PORSW_STATUS	Power switch status	Provides the status of the power switch.
GPIO_VDD_PORSW_CTL	Supply Power Switch control Register	Configures inrush current limit level select, threshold select, and level set for GPIO.
GPIO_SEC_INTR_CAUSE	Secure Interrupt port cause register	Provides interrupt status corresponding to ports.

24 Timer, counter, and pulse-width modulator (TCPWM)

The timer, counter, and pulse width modulator (TCPWM) block in PSC3M/P8, PSC3M/P7 MCU implements a 16- or 32-bit TCPWM functionality. TCPWM includes two counter groups where Group 0 and Group 1 each support two counters. Group0 is designed as a system timer for the main CPU and Group 1 is designed for PPCA CPU monitoring. 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. The TCPWM block works in Active and Sleep modes.

This section explains the features of the TCPWM block. About the details of implementation and operational modes, please check the Architecture and Counter function modes sections in the PPCA TCPWM chapter.

24.1 Features

The TCPWM block has the following features[3]:

- Two counter groups:
  - Group 0 and Group 1 support two counters each
  - Each counter can run in one of the following function modes:
    - Timer
    - Capture
    - Pulse-width modulation(PWM)
    - PWM with dead time (PWM_DT)
- 16- or 32-bit counter, one compare/capture (CC0), and period registers
- Counter compare, period, line_sel, and dead time registers are double-buffered
- Up, down, and up/down counting modes
- Clock pre-scaling (1x, 2x, 4x, 8x, 16x, 32x, 64x, 128x)
- Rising edge, falling edge, combined rising/falling edge detection, or pass-through on all hardware input signals to derive counter events
- The start, reload, stop, capture0, duty cycle and masking events can be generated by software
- Two output trigger signals for each counter to indicate underflow, overflow, counter/compare0 match (cc0_match), LINE_OUT, and terminal count (TC) events
- Two PWM complementary output lines for each counter. Dead time insertion is 8 bits
- Configurable option to continue/pause with the ability to enter passive state through kill polarity in debug mode
- Output state control for PWM signal during the stop or kill operation synchronous to period match
- Independent line polarity setting for PWM signal during the kill operation

24.1.1 Supported features of TCPWM block

The following table lists the counter group supported features of TCPWM.

***Note**: This table only lists the TCPWM block features of the main CPU. About the features of PPCA TCPWM blocks, please see Table 315.*

[3] See Table 157 to check which functions are supported for counter group.

Table 157 Counter group features of TCPWM block

Features[1)]	TCWPM0	
	Group 0	**Group 1**
Number of counters per TCPWM group	2	2
Counter width in number of bits per TCPWM group	32-bit	16-bit
Number of input triggers per counter only routed to one counter (TR_ONE_CNT_NR)	1	
Number of input triggers routed to all counters (TR_ALL_CNT_NR)	6	
Second capture/compare Unit (CC1) is present (GRP_CC1_PRESENT)	N/A	N/A
Advanced motor control features (AMC) are present (GRP_AMC_PRESENT)	N/A	N/A
Stepper motor control (SMC) features are present (GRP_SMC_PRESENT)	N/A	N/A
Parallel data path for CC0 and CC1 register present (GRP_DATA_IN_CC_PRESENT)	N/A	N/A
HRPWM feature extension is available for counter group (GRP_HRPWM_PRESENT)	N/A	N/A
Number of TCPWM MOTIF instances units per counter group (GRP_MOTIF_NR)	N/A	N/A

1) The feature is present; N/A: the feature is not present.

25 Controller area network flexible data-rate (CAN FD)

25.1 Overview

The CAN FD controller complies with the ISO11898-1 (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 receives 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 (CANFD[0].CLOCK_CAN_EN[0:1]) for CAN operation and system clock (SYSCPUSS/clk_hf0) for the internal block operation. The acceptance filtering is implemented by a combination of up to 192 filter elements, where each can be configured as a range, bit mask, or dedicated ID filter.

The CAN FD controller functions only in Active and Sleep power modes. In Deep Sleep mode, it is not functional but the configuration and message RAM content is fully retained.

25.1.1 Features

The CAN FD controller has the following features:

- Flexible data-rate (FD) (ISO 11898-1: 2015)
  - Up to 64 data bytes per message
  - Supports maximum 8 Mbps
- Time-Triggered (TT) communication on CAN (ISO 11898-4: 2004)
  - TTCAN protocol level 1 and level 2 completely in hardware
- AUTOSAR support
- Acceptance filtering
- Two channels; each channel includes the following:
  - Up to 64 dedicated receive buffers per channel
  - Two configurable receive FIFOs (up to 64 buffers each) with acceptance filters
  - 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 4-KB RAM
- Receive FIFO top pointer logic
  - Enables DMA access on the FIFO
- DMA for debug message and received FIFOs

25.1.2 Features not supported

- Asynchronous serial communication (ASC)
- Interrupt of bit error corrected (IR.BEC) in M_TTCAN
  - This bit is fixed at '0'

25.2 Architecture

25.2.1 Block diagram

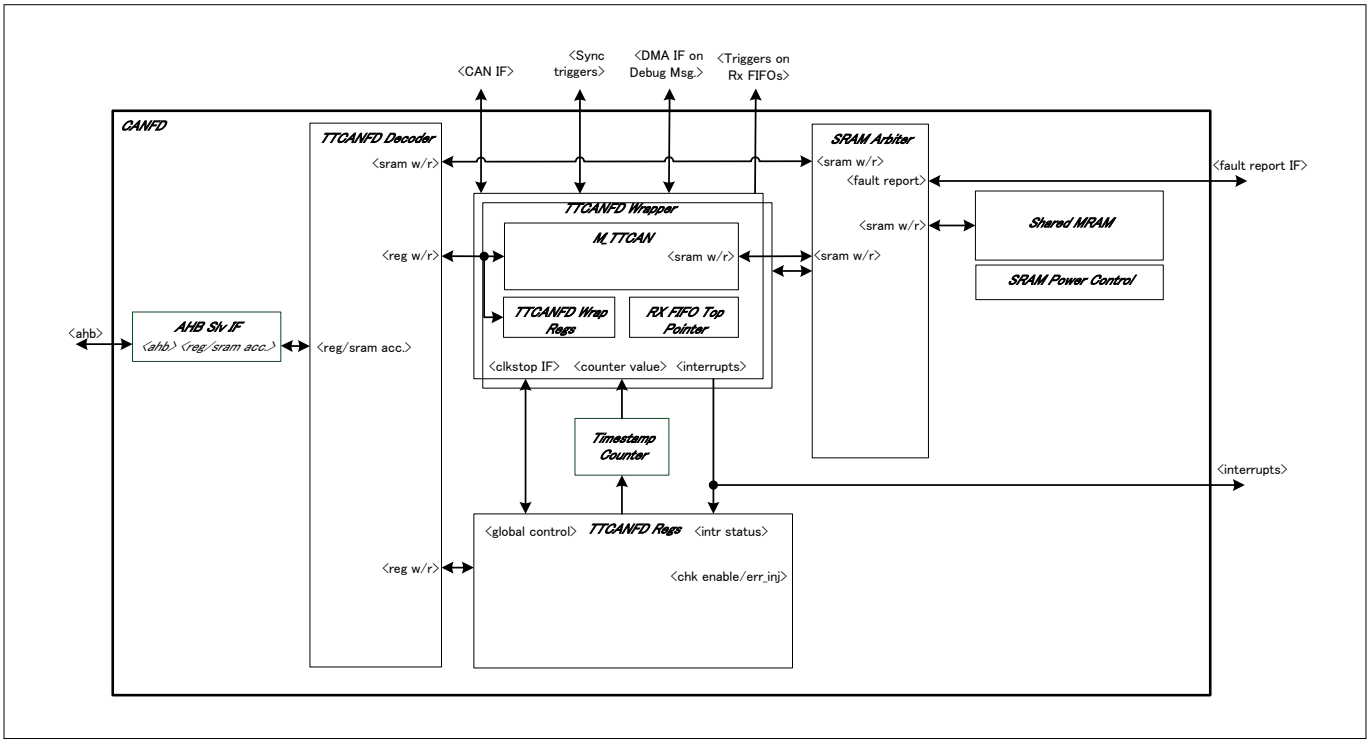


Figure 90 Block diagram

25.2.2 Dual clock sources

The TTCAN channel has two clock inputs: clk_can and clk_sys.

The `clk_can`, derived from the system peripheral clock dividers, is used for the CAN (or CAN FD) operation. It is recommended to use 20 MHz, 40 MHz, or 80 MHz.

The `clk_sys` is used for all operations such as, register accesses, SRAM accesses and so on, but in CAN operations. `clk_sys` must run equal to or faster than `clk_can`, and it is allowed to be derived from a spread-spectrum PLL to minimize EMC unlike `clk_can`.

See the Clocking system section for more details about the clock configuration.

25.2.3 Interrupt line

The M_TTCAN channel provides two interrupt lines: interrupt0 and interrupt1. Interrupts from any source within the M_TTCAN channel is routed either to interrupt0 or interrupt1 by using the ILS and TTILS registers. By default, all the interrupts are routed to interrupt0. By programming, enable interrupt line 0 (ILE.EINT0) and

enable interrupt line 1 (ILE.EINT1), the interrupt lines are enabled or disabled separately for each interrupt source.

25.3 Functional description

25.3.1 Modes of operation

25.3.1.1 Software initialization

This section describes to set or reset the initialization bit (CCCR.INIT). The CCCR.INIT bit is set as follows:

- Software or hardware reset
- Uncorrected bit error is detected in MRAM
- By choosing Bus Off

While the CCCR.INIT is set:

- Message transfer from and to the CAN bus is stopped
- The status of the CAN bus output canfd[0].ttcan_tx[0:1] is recessive (HIGH)
- The protocol error counters are unchanged

Setting CCCR.INIT does not change any Configuration register.

Resetting 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 are affected by Configuration Change Enable (CCCR.CCE) access to the Configuration registers and enabled when both bits, CCCR.INIT and CCCR.CCE, are set (write protected). CCCR.CCE is set/reset while CCCR.INIT is '1'. CCCR.CCE is automatically reset when CCCR.INIT is reset.

The following registers are reset when CCCR.CCE is set

- HPMS - High priority message status
- RXF0S - RX FIFO 0 status
- RXF1S - RX FIFO 1 status
- TXFQS - TX FIFO/queue status
- TXBRP - TX buffer request pending
- TXBTO - TX buffer transmission occurred
- TXBCF - TX buffer cancellation finished
- TXEFS - TX event FIFO status
- TTOST - TT operation status
- TTLGT - TT local and global time, only global time TTLGT.GT is reset
- TTCTC - TT cycle time and count
- TTCSM - TT cycle sync mark

In addition

- The timeout counter value (TOCV.TOC[15:0]) is preset to the value configured by the timeout period(TOCC.TOP[15:0]) when CCCR.CCE is set
- The state machines of TX and RX handlers are held in idle state while CCCR.CCE is '1'

The following registers are only writable while CCCR.CCE is '0'.

- TXBAR - TX buffer add request
- TXBCR - TX buffer cancellation request

The test mode enable (CCCR.TEST) and bus monitoring mode(CCCR.MON) can only be set by the CPU while CCCR.INIT is '1' and CCCR.CCE is '1'. Both bits can be reset at any time.

The disable automatic retransmission (CCCR.DAR) can only be set/reset while CCCR.INIT is '1' and CCCR.CCE is '1'.

Message RAM initialization

Each MRAM word should be reset by writing '0x00000000' before the CAN FD controller configuration. This prevents MRAM bit errors when reading uninitialized words, and also avoids unexpected filter element configurations in MRAM.

25.3.1.2 Normal operation

The M_TTCAN's default operating mode after the hardware reset is event-driven CAN communication without time triggers (TTOCF.OM = 00). Both CCCR.INIT and CCCR.CCE is set before the TT operation mode is changed.

When M_TTCAN is initialized and CCCR.INIT is reset to '0', 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. The automated transmission on reception of remote frames is not implemented.

25.3.1.3 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 are decoded as the 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 (RES), 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 PSR.PXE bit. When the protocol exception handling is enabled (CCCR.PXHD = 0), it causes the operation state to change from receiver (PSR.ACT =10) to integrating (PSR.ACT = 00) at the next sample point. If the protocol exception handling is disabled (CCCR.PXHD =1), the M_TTCAN treats a recessive RES bit as an error and responds with the error frame.

The CAN FD operation is enabled by programming CCCR.FDOE. If CCCR.FDOE is '1', transmission and reception of CAN FD frames is enabled. The 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 CCCR.FDOE as '0', the received frames are interpreted as classic CAN frames, which leads to the transmission of an error frame when receiving a CAN FD frame. When the CAN FD operation is disabled, no CAN FD frames are transmitted if the FDF bit of a TX buffer element is set. CCCR.FDOE and CCCR.BRSE can only be changed while CCCR.INIT and CCCR.CCE are both set.

CCCR.FDOE is '0', the setting of FDF and BRS is ignored and frames are transmitted in classic CAN format. When CCCR.FDOE = 1 and CCCR.BRSE = 0, only FDF of a TX buffer element is evaluated. When CCCR.FDOE = 1 and CCCR.BRSE = 1, transmission of CAN FD frames with bit rate switching is enabled. All the TX buffer elements with FDF and BRS bits set are transmitted in CAN FD format with the bit rate switching.

A mode change during the 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 the nodes are verified to transmit classic CAN messages that can communicate in CAN FD format and all the nodes switch to the CAN FD operation
- Wake-up messages in the CAN partial networking must be transmitted in classic CAN format
- End-of-line programming occurs if all the nodes are not CAN FD capable. The non-CAN FD nodes are held in Silent mode until the programming is completed. Then all the nodes are switchto 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 158:

Table 158 Extended 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 the CAN FD frames, the bit timing is 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 (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 (DBTP). The bit timing is switched from the data phase timing with a condition that occurs first: the CRC delimiter or when an error is detected.

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 Mbps.

In both data frame formats, CAN FD and CAN FD with bit rate switching, the value of the bit error status indicator (ESI) 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.

25.3.1.4 Transmitter delay compensation

During the data phase of a CAN FD transmission, only one node transmittes; and all the others receive. The length of the bus line has no impact. When transmitting via pin canfd[0].ttcan_tx[0:1], the M_TTCAN receives the transmitted data from its local CAN transceiver via pin canfd[0].ttcan_rx[0:1]. The received data is delayed by the transmitter delay. If the delay is greater than time segment before sample point (TSEG1), a bit error is detected. To enable a data phase bit time that is shorter than the transmitter delay, the delay compensation is introduced. Without the transmitter delay compensation, the bit rate in the data phase of a CAN FD frame is limited by the transmitter delay.

25.3.1.4.1 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 with the received data at the secondary sample point (SSP). If a bit error is detected, the transmitter reacts on the 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 the bit DBTP.TDC.

The received bit is compared with 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 canfd[0].ttcan_tx[0:1] through the transceiver to the receive input RX plus the transmitter delay compensation offset as configured by 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 to the next integer number of mtq (clk_can period).

PSR.TDCV shows the actual transmitter delay compensation value. PSR.TDCV is cleared when CCCR.INIT is set and updated at each transmission of an FD frame while DBTP.TDC is also 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 canfd[0].ttcan_tx[0:1] to canfd[0].ttcan_rx[0:1] and the configured transmitter delay compensation offset TDCR.TDCO is less than 6 bit times in the data phase
- The sum of the measured delay from canfd[0].ttcan_tx[0:1] to canfd[0].ttcan_rx[0:1] and the configured transmitter delay compensation offset TDCR.TDCO is less than or equal to 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 the receive bits at the SSPs

25.3.1.4.2 Transmitter delay compensation measurement

If transmitter delay compensation is enabled by programming 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 canfd[0].ttcan_rx[0:1] of the transmitter. The resolution of this measurement is one minimum time quanta (MTQ).

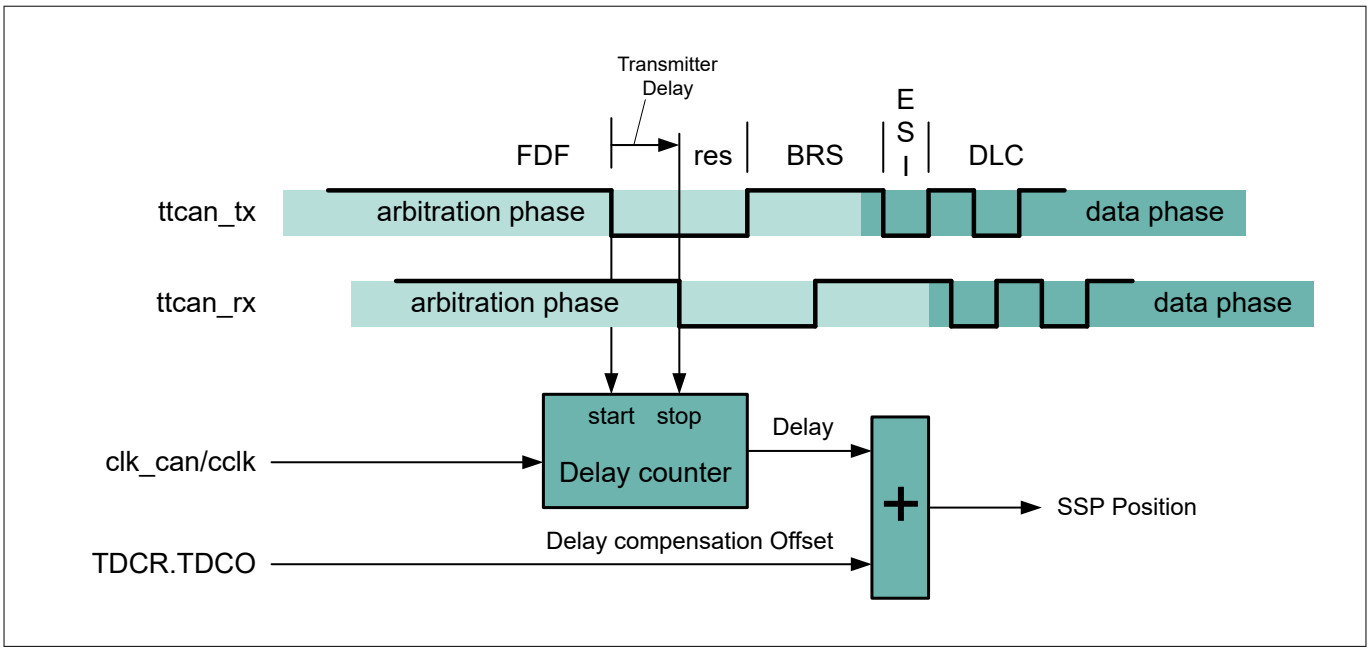


Figure 91 Transmitter delay measurements

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 is enabled by programming TDCR.TDCF. This defines a minimum value for the SSP position. Dominant edges on canfd[0].ttcan_rx[0:1], that results in an earlier SSP position are ignored for transmitter delay measurement. The measurement is stopped when the SSP position is at least TDCR.TDCF and canfd[0].ttcan_rx[0:1] is LOW.

25.3.1.5 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 (ECR.REC and ECR.TEC) are frozen while Error Logging (ECR.CEL) is active.

The CPU can set the CAN FD controller into Restricted Operation mode by setting the Restricted Operation mode bit (CCCR.ASM). CCCR.ASM can only be set by the CPU when both CCCR.CCE and CCCR.INIT are set to ‘1‘.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 MRAM in time. To leave this mode, the CPU should reset 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).*

25.3.1.6 Bus Monitoring mode

The M_TTCAN is set in Bus Monitoring mode by programming CCCR.MON to ‘1’ or when an error level S3 (TTOST.EL = 11) is entered. In this 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 the recessive state. In the Bus Monitoring mode, the TXBRP register is in the reset state.

The Bus Monitoring modeis used to analyze the traffic on a CAN bus without affecting it by the transmission of dominant bits. Figure 92 shows the connection of signals canfd[0].ttcan_tx[0:1] and canfd[0].ttcan_rx[0:1] to the M_TTCAN in Bus Monitoring mode.

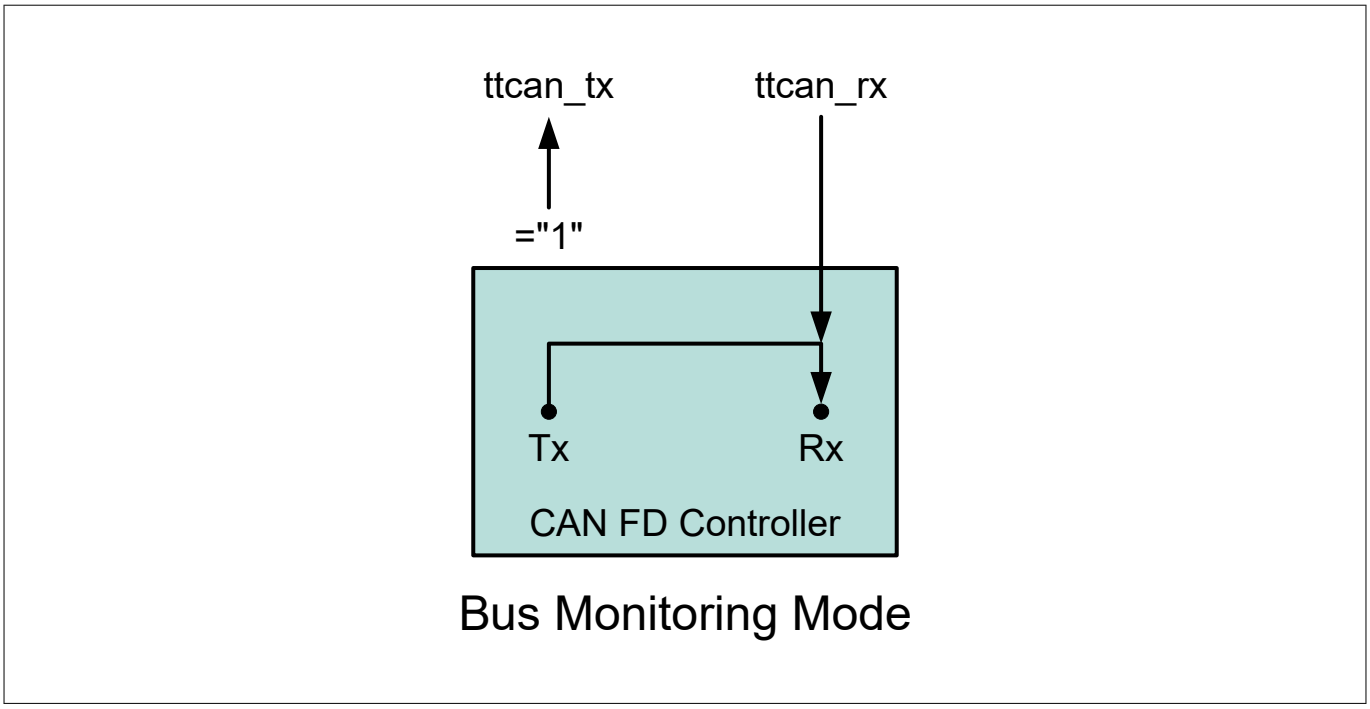


Figure 92 Pin control in Bus Monitoring mode

25.3.1.7 Disable automatic retransmission

M_TTCAN supports automatic retransmission of frames that have lost arbitration or disturbed by errors during the 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 is disabled via CCCR.DAR.

In DAR mode, all the transmissions are automatically canceled after they start on the CAN bus. The TX Request Pending bit TXBRP.TRPx is reset after the 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 TXBTO.TOx is set while the Corresponding TX Buffer Cancellation Finished bit TXBCF.CFx is not set
- Successful transmission in spite of cancellation: Both the Corresponding TX Buffer Transmission Occurred bit TXBTO.TOx and Corresponding TX Buffer Cancellation Finished bit TXBCF.CFx are set
- Arbitration lost or frame transmission disturbed: Corresponding TX Buffer Transmission Occurred bit TXBTO.TOx is not set while Corresponding TX Buffer Cancellation Finished bit TXBCF.CFx is 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).

25.3.1.8 Power Down (Sleep mode)

The M_TTCAN channel can be set to a power down mode via Clock Stop Request (CTL.STOP_REQ). As long as clock stop request is active, the STOP_REQ bit is '1'.

When all the pending transmission requests are complete, the M_TTCAN waits until the bus idle state is detected. Then, M_TTCAN sets CCCR.INIT to '1' to prevent any further CAN transfers. Now the M_TTCAN acknowledges that it is ready for power down by setting clock stop acknowledge(STATUS.STOP_ACK). Upon receiving the ACK from the channel, the hardware automatically switches off the clock to their respective channels.

To leave power down mode, the application must reset CTL.STOP_REQ. The M_TTCAN acknowledges this by resetting STATUS.STOP_ACK. Later, the application restarts the CAN communication by resetting the bit CCCR.INIT.

When the clock stop request is triggered through CTL.STOP_REQ, it must not be cleared before STATUS.STOP_ACK bit is set.

***Note**: Do not use the TTCAN CCCR.CSR register for the power down control; instead, use CTL.STOP_REQ. Similarly, use STATUS.STOP_ACK instead of CCCR.CSA.*

25.3.1.9 Test mode

To enable write access to the TEST register, Test Mode Enable bit (CCCR.TEST) is set to '1'. This allows the configuration of the test modes and test functions.

Four output functions are available for the CAN transmit pin, canfd[0].ttcan_tx[0:1], by programming 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 canfd[0].ttcan_rx[0:1] pin can be read from TEST.RX. Both functions can be used to check the CAN bus physical layer.

Due to the synchronization mechanism between the CAN clock and the host clock domain, there may be a delay of several host clock periods between writing to TEST.TX until the new configuration is visible at the output pin, canfd[0].ttcan_tx[0:1].This applies also when reading the input pin canfd[0].ttcan_rx[0:1] via TEST.RX.

***Note**: Test modes should be used for production tests or self-test only. The software control for the canfd[0].ttcan_tx[0] pin interferes with all CAN protocol functions. It is not recommended to use test modes for application.*

25.3.1.9.1 External Loop Back mode

M_TTCAN can be set in External Loop Back mode by programming TEST.LBCK to '1'. In the Loop Back mode, the M_TTCAN treats its transmitted messages as received messages and stores (if they pass acceptance filtering) into an RX buffer or an RX FIFO. Figure 93 shows the connection of signals canfd[0].ttcan_tx[0:1] and canfd[0].ttcan_rx[0:1] to the M_TTCAN in External Loop Back mode.

This mode is provided for the 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 the Loop Back mode. In this mode, M_TTCAN performs an internal feedback from the TX output to the RX input. The actual value of the canfd[0].ttcan_rx[0:1] input pin is disregarded by the M_TTCAN. The transmitted messages can be monitored at the canfd[0].ttcan_tx[0:1] pin.

25.3.1.9.2 Internal Loop Back mode

To enter the Internal Loop Back mode by programming the TEST.LBCK and CCCR.MON bits to '1'. This mode can be used for a “Hot Self-test”, meaning M_TTCAN can be tested without affecting a running CAN system connected to the `canfd[0].ttcan_tx[0:1]` and `canfd[0].ttcan_rx[0:1]` pins. In this mode, the `canfd[0].ttcan_rx[0:1]` pin is disconnected from the M_TTCAN and `canfd[0].ttcan_tx[0:1]` pin is held recessive. Figure 93 shows the connection of `canfd[0].ttcan_tx[0:1]` and `canfd[0].ttcan_rx[0:1]` to the M_TTCAN in Internal Loop Back mode.

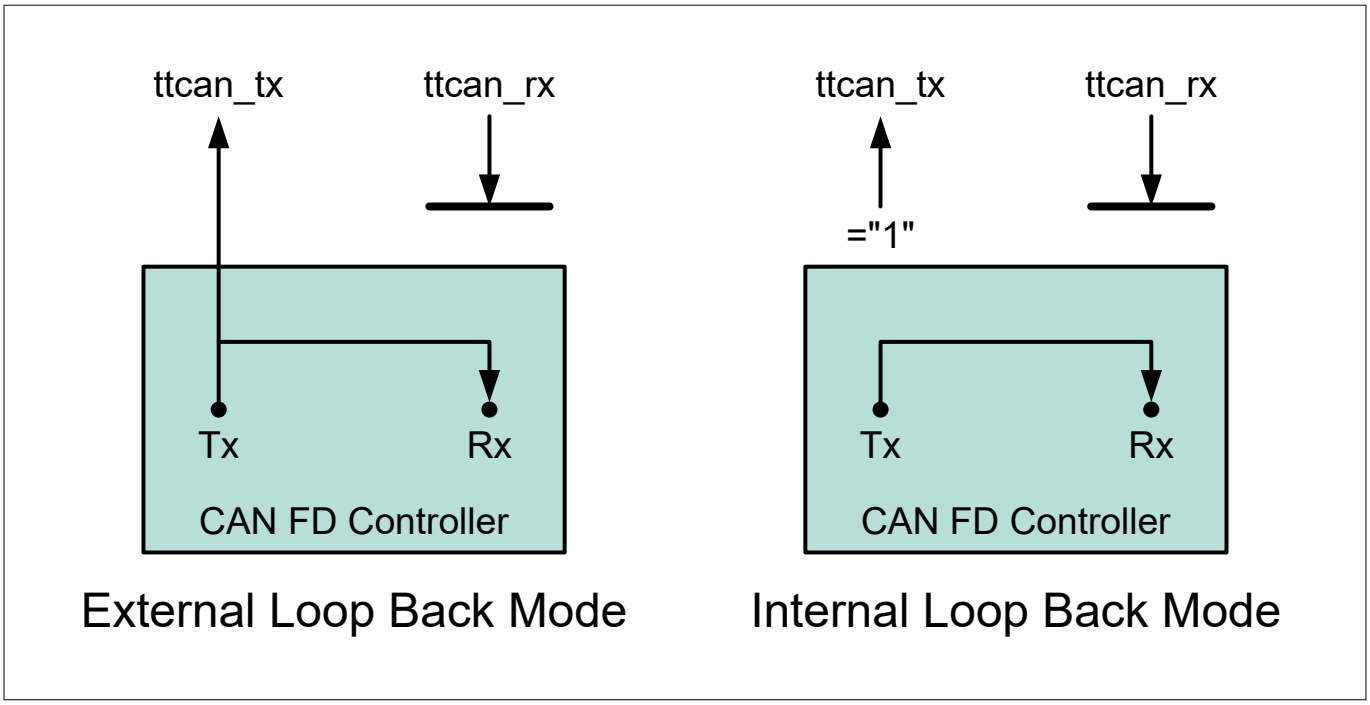


Figure 93 Pin control in Loop Back modes

25.3.1.10 Application watchdog

The application watchdog is served by reading the TTOST register. When the application watchdog is not served in time, the 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 TTOCF.AWL to '0x00'. The TT application watchdog should not be disabled in a TTCAN application program.

25.3.2 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.

The timestamp counter is configured through the TS_CTL register. The TS_CTL.ENABLED bit will enable the counter. Upon enabling, it will start incrementing according to the TS_CTL.PRESCALE [15:0]. The application can read the counter value through the TS_CNT register. Write access to the TS_CNT register will clear the 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 the internal prescaler counter is cleared. When TS_CTL.PRESCALE changes, TS_CNT should be written to reset them. This can make the internal prescaler counter follow a new value of TS_CTL.PRESCALE immediately.

The shared timestamp counter is a wrap-around counter. When the counter wraps around, IR.TSW for the M_TTCAN channel 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 TS_CNT is not retained in Deep Sleep mode whereas the TS_CTL is retained.*

25.3.3 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 a down-counter and uses the same prescaler controlled by TSCC. The TCP as the timestamp counter. A prescaler TSCC.TCP is configured to clock the timeout counter in multiples of CAN bit times (1…16). The timeout counter is configured via the register TOCC. The actual counter value can be read from TOCV.TOC.

The timeout counter can only be started while CCCR.INIT =0. It is stopped when CCCR.INIT = 1; for example, when the M_TTCAN enters Bus_Off state.

The operation mode is selected by TOCC.TOS. When operating in the Continuous mode, the counter starts when CCCR.INIT is reset. A write to TOCV presets the counter to the value configured by 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 TOCC.TOP. Down-counting is started when the first FIFO element is stored. Writing to TOCV has no effect.

When the counter reaches zero, the interrupt flag IR.TOO is set. In the Continuous mode, the counter is immediately restarted at 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 the arbitration and data fields.*

25.3.4 Receive handling

The RX handler controls the 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.

25.3.4.1 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 the acceptance filtering, each list of filters is executed from element #0 until the first match 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 (GFC)
- Standard ID Filter Configuration (SIDFC)
- Extended ID Filter Configuration (XIDFC)
- Extended ID AND Mask (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 an RX buffer
- Store received frame in RX buffer and generate pulse at filter event pin
- Reject received frame
- Set High-Priority Message interrupt flag IR.HPM
- Set High-Priority Message interrupt flag IR.HPM and store the received frame in FIFO 0 or FIFO 1

Acceptance filtering is started after the complete identifier is received. After the acceptance filtering has completed, if a matching RX buffer or RX FIFO is found, the message handler starts to write 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

The new data flag of the matching RX buffer is not set, but the RX buffer is (partly) overwritten with received data. For error types, see PSR.LEC and PSR.DLEC respectively.

RX FIFO

Put index of matching RX FIFO is not updated, but the related RX FIFO element is (partly) overwritten with received data. For error types, see PSR.LEC and 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 XIDAM before the range filter is applied
- EFT = 11: The 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 andEF1ID = 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 a Message ID filter, while SF2ID/EF2ID is used as a 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.

If 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 94 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.

It is controlled by the GFC and SIDFC message IDs, the Remote Transmission Request bit (RTR) and the Identifier Extension bit (IDE) of received frames are compared with the list of configured filter elements.

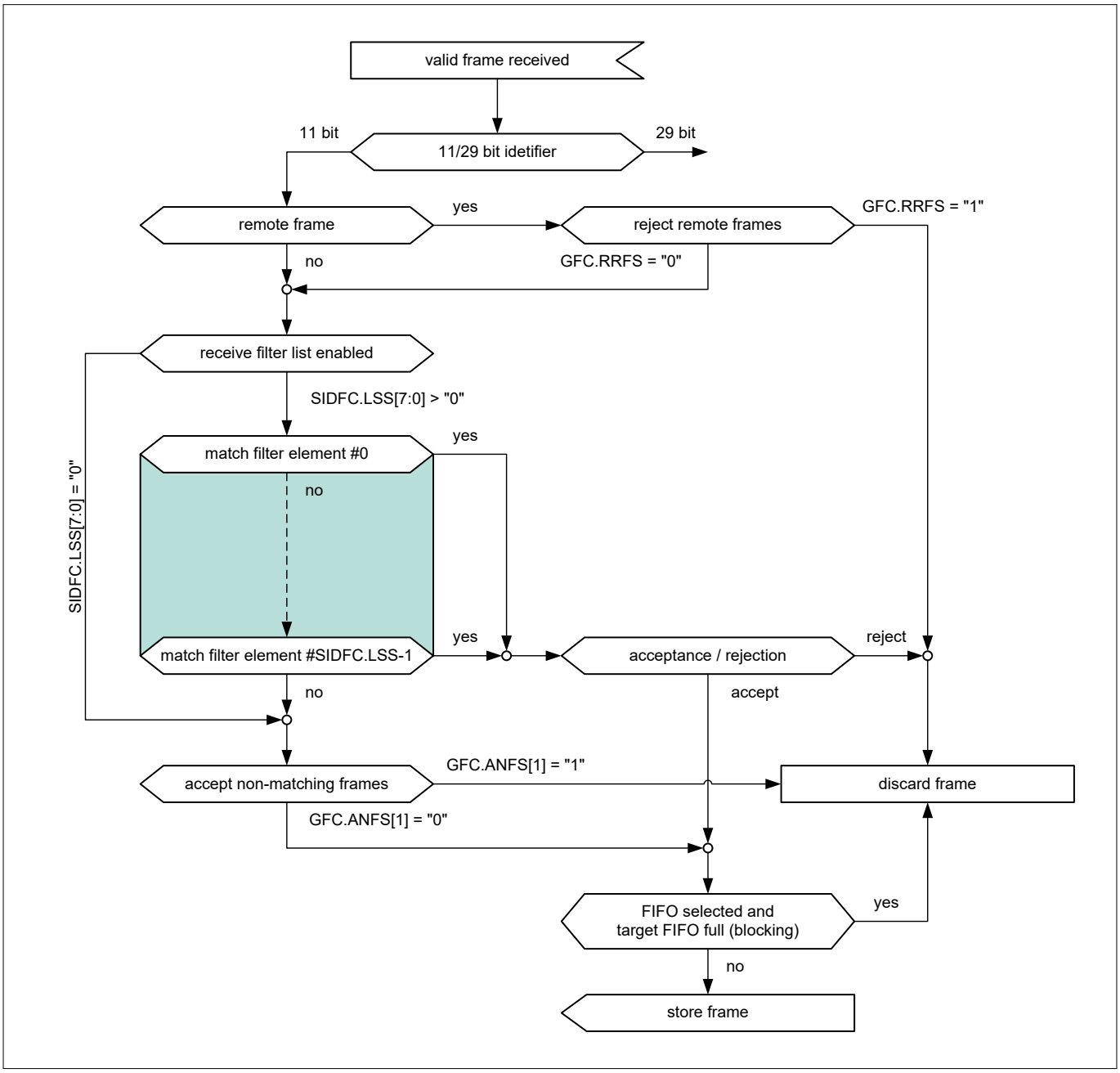


Figure 94 Standard message ID filter flow diagram

Extended message ID filtering

Figure 95 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. It is controlled by the GFC and 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.

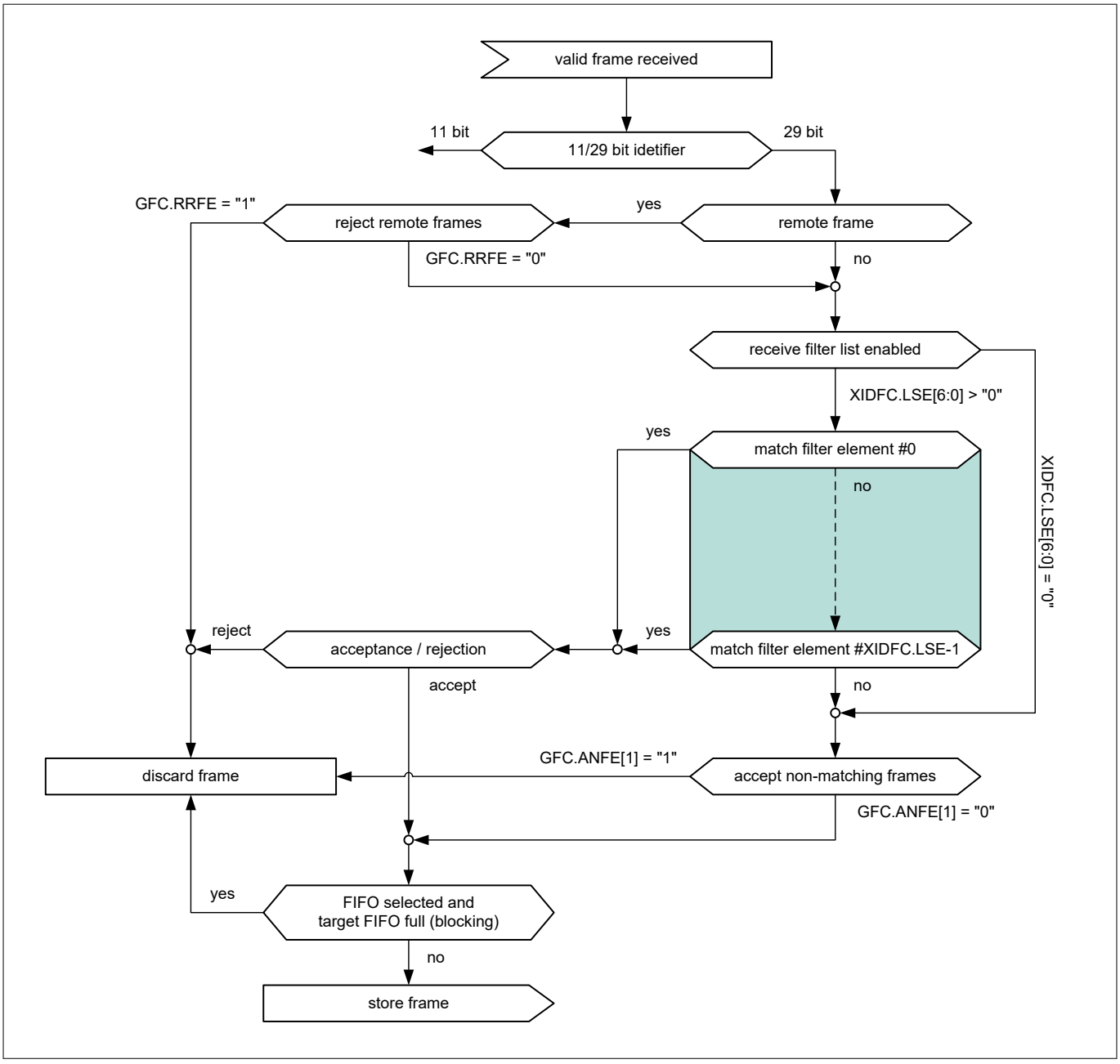


Figure 95 Extended message ID filter flow diagram

25.3.4.2 Receive FIFOs

RX FIFO 0 and RX FIFO 1 is configured to hold up to 64 elements each. The two RX FIFOs are configured via the RXF0C and RXF1C registers.

The 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 RXFnC.FnWM, interrupt flag IR.RFnW is set. When the RX FIFO Put Index reaches the RX FIFO Get Index, an RX FIFO Full condition is signaled by RXFnS.FnF. In addition, the interrupt flag IR.RFnF is set. The FIFO watermark interrupt flags can be used to trigger the DMA. DMA request for FIFO will remain set

until their respective trigger is cleared by software. Software can clear the trigger by clearing the watermark flag.

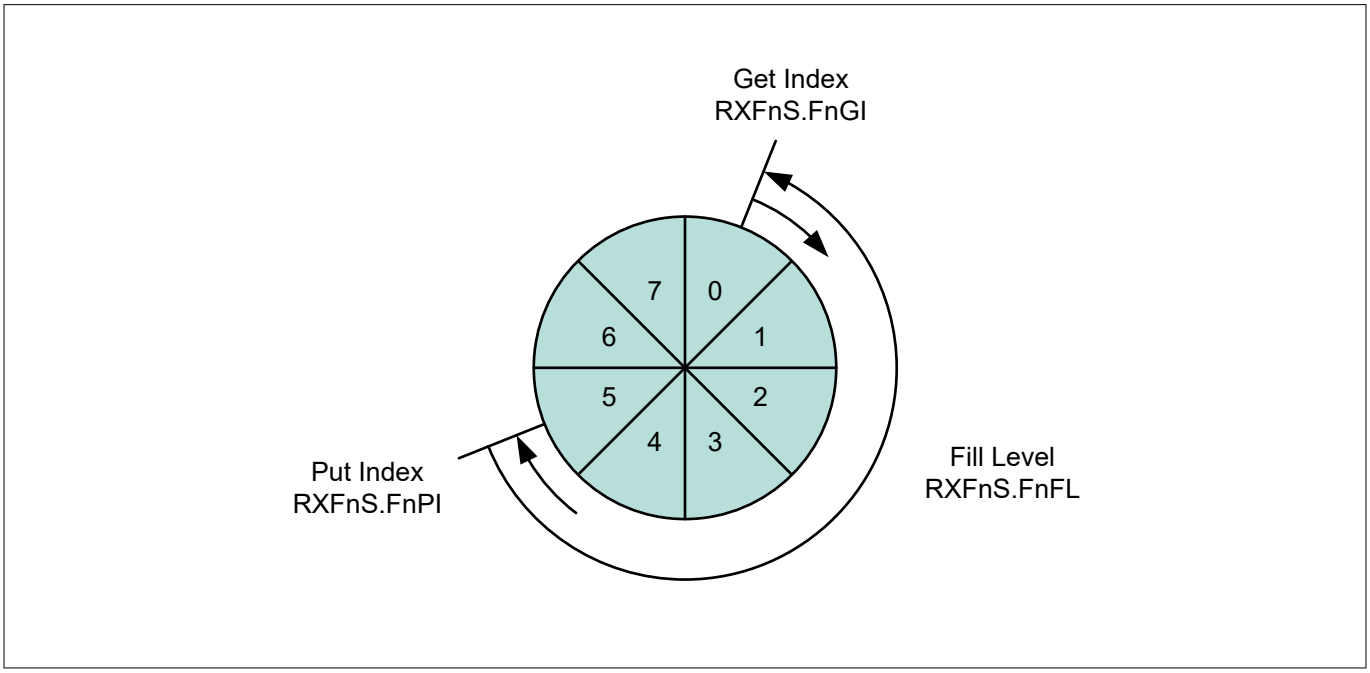


Figure 96 RX FIFO status

When reading from an RX FIFO, RX FIFO Get Index RXFnS.FnGI x FIFO element size has to be added to the corresponding RX FIFO start address RXFnC.FnSA. RXFIFO Top pointer logic is added to the CAN FD controller to make reading faster. See RX FIFO top pointer on RX FIFO top pointer.

Table 159 RX buffer/FIFO element size

RXESC.RBDS[2:0] 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

25.3.4.2.1 RX FIFO blocking mode

The RX FIFO blocking mode is configured by RXFnC.FnOM= 0. This is the default operation mode for RX FIFOs.

When an RX FIFO full condition is reached (RXFnS.FnPI = 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 RXFnS.FnF = 1. In addition, the interrupt flag 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 RXFnS.RFnL = 1. In addition, the interrupt flag IR.RFnL is set.

25.3.4.2.2 RX FIFO overwrite mode

The RX FIFO overwrite mode is configured by RXFnC.FnOM = 1. When an RX FIFO full condition (RXFnS.FnPI =RXFnS.FnGI) is signaled by 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 the overwrite mode and an RXFIFO 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 MRAM (Put Index) while the CPU is reading from the MRAM (GetIndex). In this case, inconsistent data may be read from the respective RX FIFO element. Adding an offset to the GetIndex when reading from the RX FIFO avoids this problem. The offset depends on how fast the CPU accesses the RXFIFO. Figure 97 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.

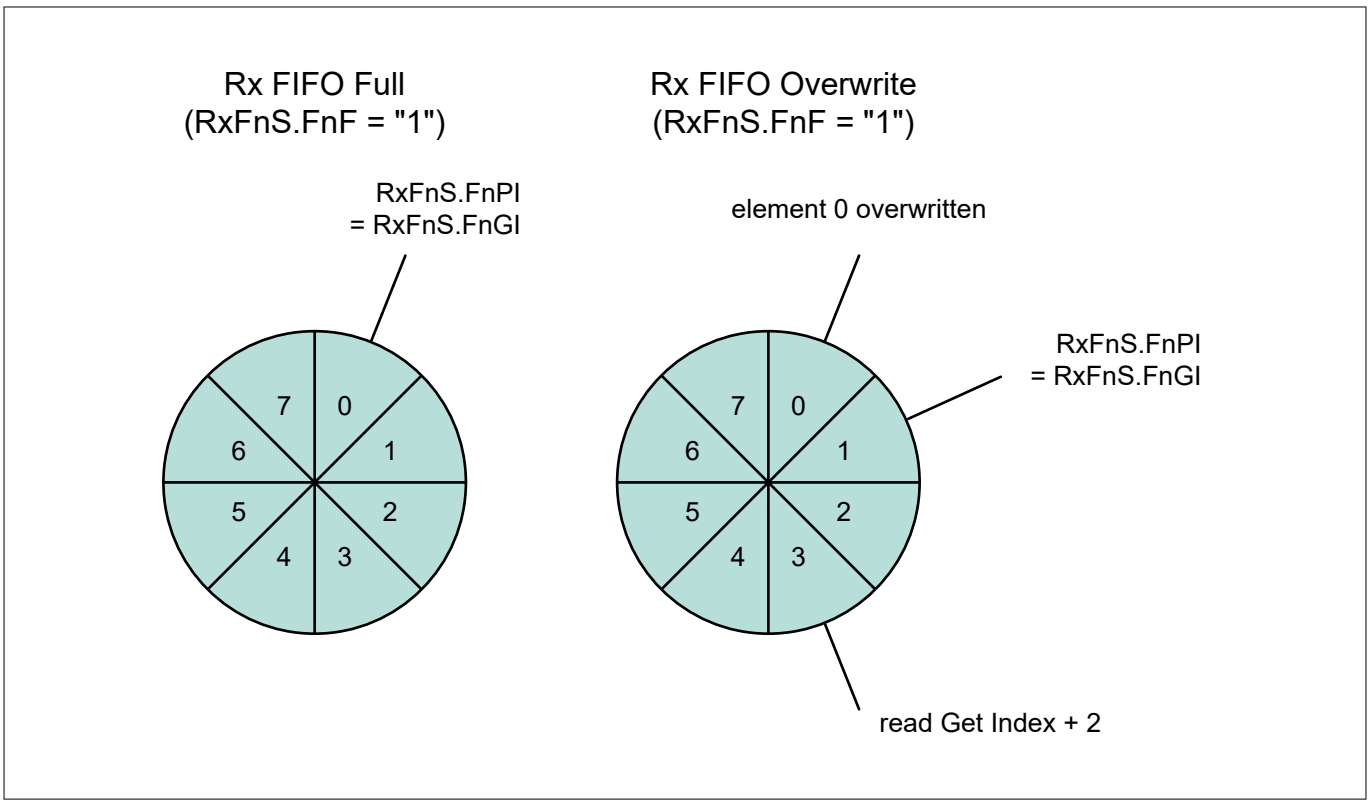


Figure 97 RX FIFO overflow handling

After reading from the RX FIFO, the number of the last element read must be written to the RX FIFO Acknowledge Index 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 (RXFnS.FnF = 0).

25.3.4.2.3 RX FIFO top pointer

M_TTCAN supports two receive FIFOs. Reading from these FIFOs requires the following steps:

- Retrieve read pointer
- Calculate correct MRAM address
- Read the data from MRAM
- Update the read pointer

To avoid these steps, RX FIFO top pointer logic is integrated in the CAN FD controller. It provides a single MMIO location (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 RXFTOP_CTL.FnTPE is set. Setting this bit enables the logic to set the FIFO top address (FnTA) and internal message word counter. The receive FIFO in the Top Status register (RXFTOPn_STAT) shows the respective FIFO top address and RXFTOPn_DATA provides the data located at the top address. See the register definitions for more details on both registers.

If RXFTOPn_DATA is read, the top pointer logic also updates the RX FIFO Acknowledge Index (RXFnA.FnA) in TTCAN channel.

***Note**: Top pointer logic is disabled when the channel is being configured (CCCR.CCE = 1). Reading RXFTOPn_DATA while the logic is disabled will return the invalid data.*

***Note**: When RX FIFO size is set to '1', do not use the RXFIFO top pointer function. Instead, read the received messages from MRAM directly.*

25.3.4.3 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 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 IR.DRX (message stored in a dedicated RX buffer) in the interrupt register is set.

Table 160 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 1	00	00 0001
2	ID message 1	00	00 0010

After the last word of a matching received message is written to the message RAM, the respective New data flag in NDAT1 and two registers are 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 is 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 resume. The following message ID filter elements causes the received message to be stored into another RX buffer, or into an RX FIFO, or the message may be rejected, depending on the filter configuration.

- Reset interrupt flag IR.DRX
- Read New Data registers
- Read messages from message RAM
- Reset New Data flags of processed messages

25.3.4.4 Debug on CAN support

The debug messages are stored into the RX buffers; three consecutive RX buffers (for example, #61, #62, and #63) are used to store debug messages A, B, and C. The format is the same for RX buffer and RX FIFO elements.

To filter the debug messages Standard/Extended Filter Elements with SFEC/EFEC = 111 is set. The 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 the 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.

25.3.4.4.1 Filtering debug messages

The 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”. Int his case the SFID1/SFID2 and EFID1/EFID2 fields have a different meaning (see Figure 94 Element and Figure 95 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 IR.DRX are set. The reception of debug messages can be monitored via RXF1S.DMS.

Table 161 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

25.3.4.4.2 Debug message handling

The debug message handling state machine assures that the debug messages are stored to three consecutive RX buffers in the correct sequence. 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 an incorrect order.

The status of the debug message handling state machine is signaled via RXF1S.DMS.

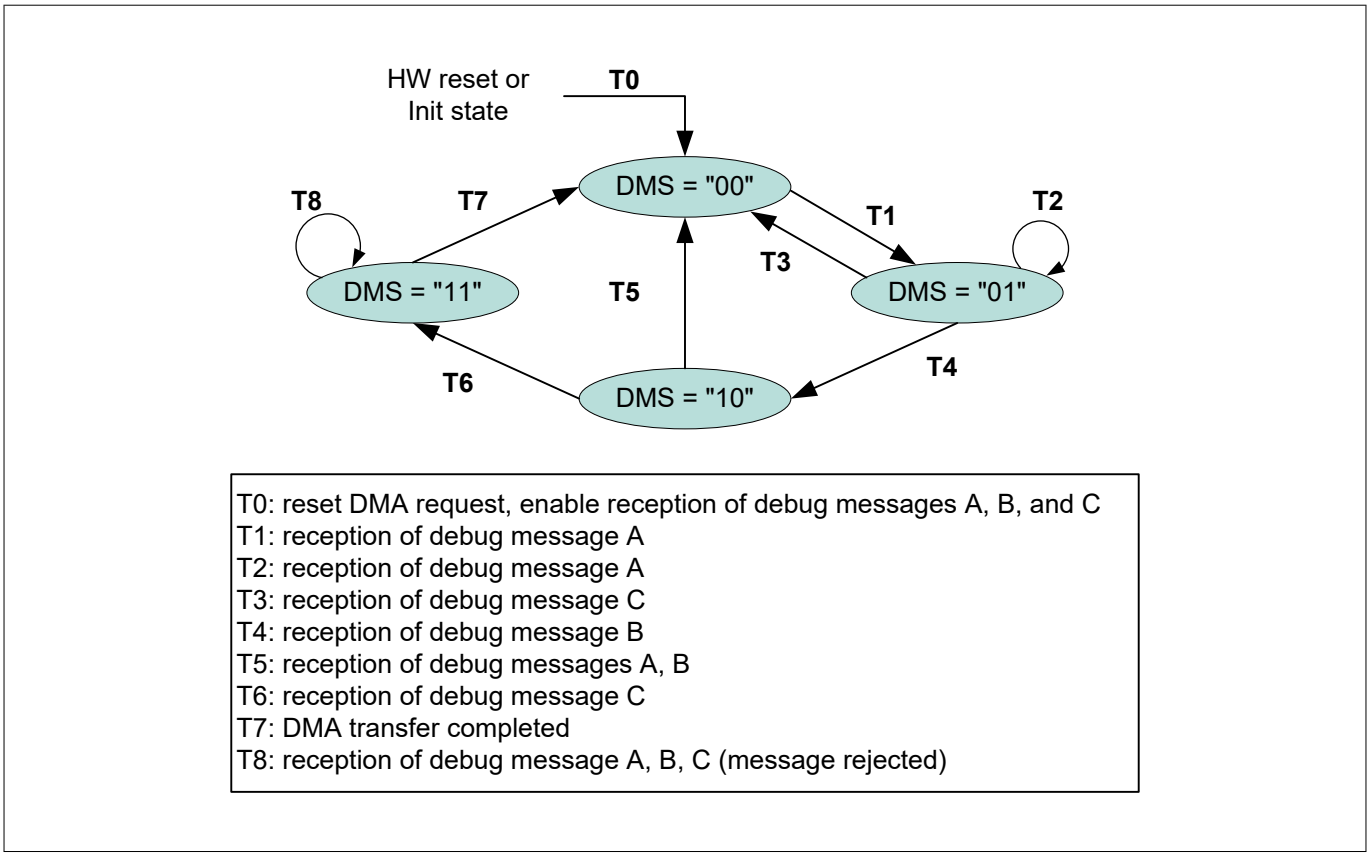


Figure 98 Debug message handling state machine

25.3.5 Transmission handling

The TX handler handles the 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) is configured separately for each TX buffer element. The TX buffer element is described in TX buffer element. Table 162 describes the possible configurations for frame transmission.

Table 162 Possible configuration for frame transmission

CCR		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 (TXBRP) is updated, or when a transmission is started.

25.3.5.1 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 ECUs 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 (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 transmitted pause disabled(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.

25.3.5.2 Dedicated TX buffers

The 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, the TX buffer with the lowest buffer number is transmitted .

If the data section is updated, a transmission is requested by an Add Request via 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.

25.3.5.2.1 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 163. 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 (TXBC.TBSA[15:2]).

Table 163 TX Buffer/FIFO/queue element size

TXESC.TBDS [2:0]	Data field [bytes]	Element size [RAM words]
000	8	4
001	12	5
010	16	6
011	20	7

(table continues...)

Table 163 (continued) TX Buffer/FIFO/queue element size

TXESC.TBDS [2:0]	Data field [bytes]	Element size [RAM words]
100	24	8
101	32	10
110	48	14
111	64	18

25.3.5.3 Transmit FIFO

The TX FIFO operation is configured by programming TXBC.TFQM to '0'. Messages stored in the TX FIFO are transmitted starting with the message referenced by the Get Index TXFQS.TFGI. After each transmission, the Get Index is incremented cyclically until the TX FIFO is empty. The TXFIFO enables transmission of messages with the same message ID from different TX buffers in the order that these messages are written to the TX FIFO. The M_TTCAN calculates the TX FIFO Free Level TXFQS.TFFL as the 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 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 (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 '1' to the 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 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 163. Therefore, the start address of the next available (free) TX FIFO buffer is calculated by adding TX FIFO/Queue Put Index TXFQS.TFQPI (0…31) × Element Size to the TX buffer Start Address TXBC.TBSA.

25.3.5.4 TX queue

The TX queue operation is configured by programming 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 queue buffers are configured with the same message ID, the queue buffer with the lowest buffer number is transmitted first.

New messages must be written to the TX buffer referenced by the Put Index TXFQS.TFQPI. An Add Request cyclically increments the Put Index to the next free TX buffer. If the TX queue is full (TXFQS.TFQF = 1), the Put Index is not valid to write any further message till one of the requested messages is sent or a pending transmission request is cancelled.

The application may use the TXBRP register instead of the Put Index and may place messages to any TX buffer without pending transmission requests.

A TX queue buffer allocates element size of 32-bit words in the message RAM as shown in Table 163. Therefore, the start address of the next available (free) TX queue buffer is calculated by adding TX FIFO/queue Put Index TXFQS.TFQPI (0…31) × Element Size to the TX buffer start address TXBC.TBSA.

25.3.5.5 Mixed dedicated TX buffers/TX FIFO

In this case, the TX buffers section in the MRAM is subdivided into a set of dedicated TX buffers and a TX FIFO. The number of dedicated TX buffers is configured by TXBC.NDTB. The number of TX buffers assigned to the TX FIFO is configured by TXBC.TFQS. If TXBC.TFQS is programmed to '0', only the dedicated TX buffers are used.

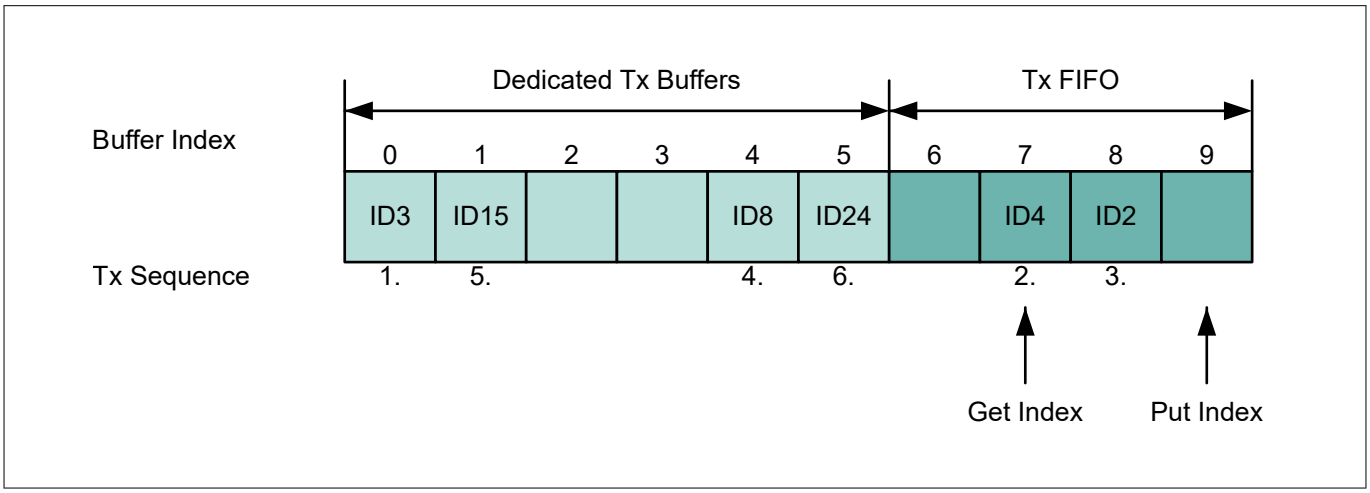


Figure 99 Example of mixed configuration dedicated TX buffers/TX FIFO

TX prioritization:

- Scan dedicated TX buffers and oldest pending TX FIFO buffer (referenced by TXFS.TFGI)
- Scan dedicated TX buffers and oldest pending TX FIFO buffer (referenced by TXFS.TFGI)

25.3.5.6 Mixed dedicated TX buffers/TX queue

In this case, the TX buffers section in the MRAM is subdivided into a set of dedicated TX buffers and a TX queue. The number of dedicated TX buffers is configured by TXBC.NDTB. The number of TX queue buffers is configured by TXBC.TFQS. In the case TXBC.TFQS is programmed to'0', only dedicated TX buffers are used.

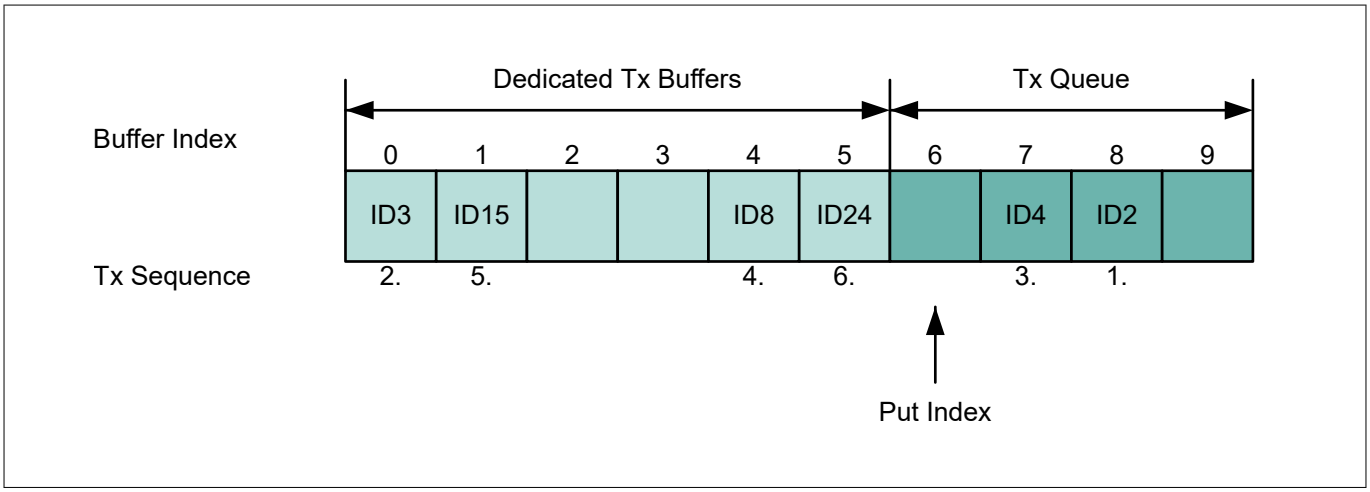


Figure 100 Example of mixed configuration dedicated TX buffers/TX queue

TX prioritization:

- Scan all TX buffers with the activated transmission request
- TX buffer with the lowest message ID gets the highest priority and is transmitted next

25.3.5.7 Transmit cancellation

M_TTCAN supports transmit cancellation. This feature is especially intended for gateway applications and AUTOSAR-based applications. To cancel a requested transmission from a dedicated TX buffer or a TX queue buffer the host must write '1' to the corresponding bit position (number of TX buffers) of the TXBCR register. The transmit cancellation is not intended for TX FIFO operation.

Successful cancellation is signaled by setting the corresponding bit of the TXBCF register to '1'.

In case the transmit cancellation is requested while transmission from a TX buffer is ongoing, the corresponding TXBRP bit remains set as long as the transmission is in progress. If the transmission was successful, the corresponding TXBTO and TXBCF bits are set. Otherwise, it is not repeated and only the corresponding 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.*

25.3.5.8 Transmit event handling

To support TX event handling, M_TTCAN has implemented a TX event FIFO. After M_TTCAN has transmits a message on the CAN bus, the message ID and time stamp are stored in a TX event FIFO element. To link a TX event to the 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; i.e., a TX buffer holds only the message to be transmitted, while the transmit status is stored separately in the TX event FIFO. This has a 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 IR.TEFF, no further elements are written to the TX event FIFO until one element is read and the TX event FIFO Get Index is incremented. In case the TX event occurs while the TX event FIFO is full, it is discarded and interrupt flag IR.TEFL is set.

To avoid the TX event FIFO overflow, use the TX event FIFO watermark. When the TX event FIFO fill level reaches the TX event FIFO watermark configured by TXEFC.EFWM, interrupt flag IR.TEFW is set.

When reading from the TX event FIFO, the TX event FIFO Get Index TXEFS.EFGI must be added twice to the TX event FIFO start address TXEFC.EFSA.

25.3.6 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 RXFTOPn_DATA is read, the Acknowledge Index (RXFnA.FnA) is updates 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, update the Acknowledge Index uses one of the following two use cases:

- When only a single element is read from the FIFO (pointed 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 FIFOs GetIndex

Note that the high-priority message from one of the two RX FIFOs, since the CPU has free access to the M_TTCAN's message RAM. In this case, the FIFO acknowledge index should is not written because this will set the Get Index to a wrong position and alters the FIFOs fill level with a loss in older FIFO elements.

***Note**: The application must ensure that a valid value is written to the FIFO acknowledge index. M_TTCAN does not check for erroneous values.*

25.3.7 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.

25.3.7.1 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 101: 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.

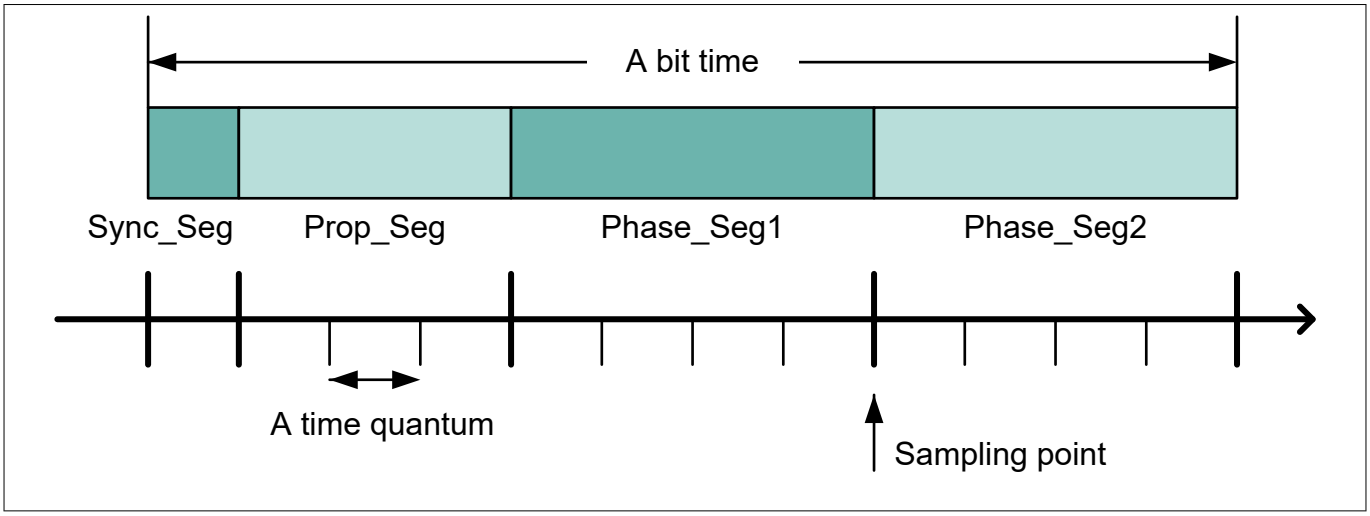


Figure 101 Bit time construction

Each segment consists of a programmable number of time quanta, which is a multiple of the time quantum that is defined by clk_can and a prescaler. The values and prescalers used to define these parameters differ for the nominal and data bit times, and are configured by Nominal Bit Timing and Prescaler register (NBTP) and Data Bit Timing and Prescaler register (DBTP) as shown in Table 164.

Table 164 Bit time parameters

Parameter	Description
Time quantum tq (nominal) and tqd (data)	Time quantum is derived by multiplying the basic unit time quanta (the clk_can period) with the respective prescaler. The time quantum is configured by the CAN FD controller as nominal: tq = (NBTP.NBRP[8:0] +1) × clk_can period data: tqd = (DBTP.DBRP[4:0] + 1) × clk_can 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 = NBTP.NTSEG1[7:0] + 1 data: Prop_Seg + Phase_Seg1 = DBTP.DTSEG1[4:0] + 1
Phase_Seg1	Phase_Seg1 is used to compensate for edge phase errors before the sampling point. It is lengthened by the resynchronization jump width. The sum of Prop_Seg and Phase_Seg1 is configured by the CAN FD controller as nominal: NBTP.NTSEG1[7:0] + 1 data: DBTP.DTSEG1[4:0] + 1
Phase_Seg2	Phase_Seg2 is used to compensate for edge phase errors after the sampling point. It is shortened by the resynchronization jump width. Phase_Seg2 is configured by the CAN FD controller as nominal: NBTP.NTSEG2[6:0] + 1 data: 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: NBTP.NSJW[6:0] + 1 data: 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 + (NBTP.NTSEG1[7:0] + 1) + (NBTP.NTSEG2[6:0] + 1)] × [(NBTP.NBRP[8:0] +1) × clk_can period]

Data bit time

= [1 + (DBTP.DTSEG1[4:0] + 1) + (DBTP.DTSEG2[3:0] + 1)] × [(DBTP.DBRP[4:0] +1) × clk_can period]

***Note**: The information processing time (IPT) of the CAN FD controller is '0'; 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.*

25.3.7.2 CAN bit rate

The bit rate is the inverse of bit time; therefore, the nominal bit rate is

1 / [{1 + (NBTP.NTSEG1[7:0] + 1) + (NBTP.NTSEG2[6:0] + 1)} × {(NBTP.NBRP[8:0] +1) × clk_can period}]

and the data bit rate is

1 / [{1 + (DBTP.DTSEG1[4:0] + 1) + (DBTP.DTSEG2[3:0] + 1)} × {(DBTP.DBRP[4:0] +1) × clk_can period}]

From these formulas, see that the bit rates of the CAN FD controller depends on the CAN clock (clk_can) period, and the range each parameter can be configured. The following tables list examples of the configurable bit rates at varying CAN clock frequencies. Empty boxes indicate that the desired bit rate are not configured at the specified input CAN clock frequency.

CAN clock frequency	8MHz		10MHz		16MHz		20MHz		32MHz		40MHz	
configuration / nominal bit rate	Number of tqs per bit time	NBTP.NBRP + 1	Number of tqs per bit time	NBTP.NBRP + 1	Number of tqs per bit time	NBTP.NBRP + 1	Number of tqs per bit time	NBTP.NBRP + 1	Number of tqs per bit time	NBTP.NBRP + 1	Number of tqs per bit time	NBTP.NBRP + 1
125Kbps	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
250Kbps	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
500Kbps	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
1Mbps	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 102 Example configuration for nominal bit rates

CAN clock frequency	8MHz		10MHz		16MHz		20MHz		32MHz		40MHz	
configuration / data bit rate	Number of tqds per bit time	DBTP.DBRP + 1	Number of tqds per bit time	DBTP.DBRP + 1	Number of tqds per bit time	DBTP.DBRP + 1	Number of tqds per bit time	DBTP.DBRP + 1	Number of tqds per bit time	DBTP.DBRP + 1	Number of tqds per bit time	DBTP.DBRP + 1
500Kbps	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
1Mbps	8tqd	1	10tqd	1	16tqd 8tqd	1 2	20tqd 10tqd	1 2	32tqd 16tqd 8tqd	1 2 4	40tqd 20tqd 10tqd	1 2 4
2Mbps	-	-	-	-	8tqd	1	10tqd	1	16tqd 8tqd	1 2	20tqd 10tqd	1 2
4Mbps	-	-	-	-	-	-	-	-	8tqd	1	10tqd	1
5Mbps	-	-	-	-	-	-	-	-	-	-	8tqd	1
8Mbps	-	-	-	-	-	-	-	-	-	-	5tqd	1

Figure 103 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.*

25.4 Message RAM (MRAM)

The MRAM stores RX/TX messages and filter configurations.

***Note**:*
- *The MRAM should be '0' before configuration of the CAN FD controller to prevent bit errors when reading uninitialized words and to avoid unexpected filter element configurations in the MRAM*
- *Unused MRAM is not used for general purposes*

25.4.1 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 MRAM (note that the number of words that can be used is limited by the size of the actual MRAM). It is not necessary to configure each of the sections listed in Figure 104, nor is there any restriction with respect to the sequence of the sections.

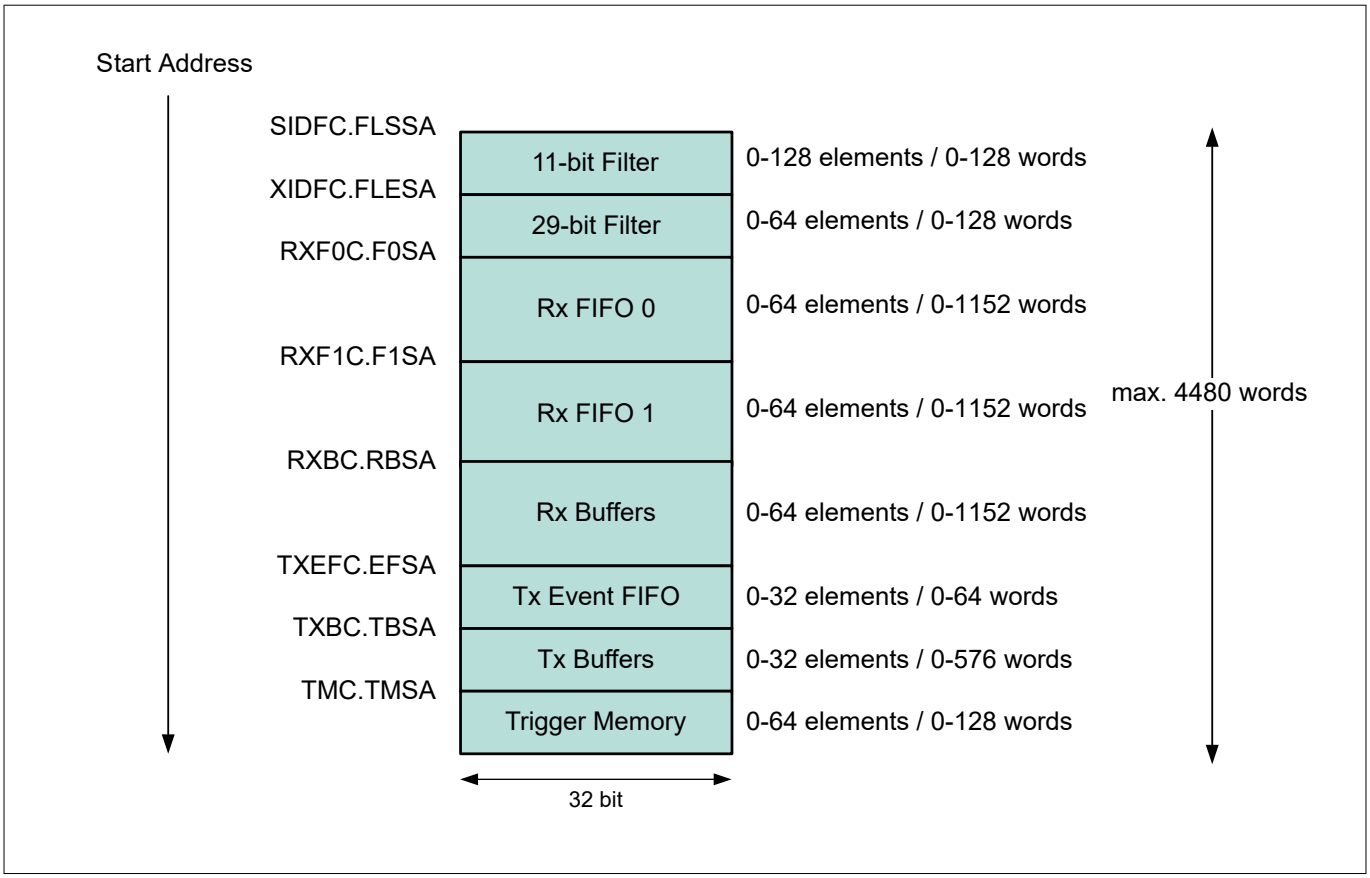


Figure 104 Message RAM configuration

The CAN FD controller addresses the message RAM in 32-bit words, not in 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*
- *The MRAM is accessible by both M_TTCAN and CPU. Dynamic round-robin scheme is implemented to allocate access*

25.4.2 RX buffer and FIFO element

An RX buffer and FIFO element is a block of 32-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 is configured in the message RAM. Each RX FIFO section is configured to store up to 64 received messages. The structure of an RX buffer and FIFO element is shown in Figure 105. The element size is configured to store CAN FD messages with up to 64 bytes data field via register RXESC (RX buffer/ FIFO element size configuration).

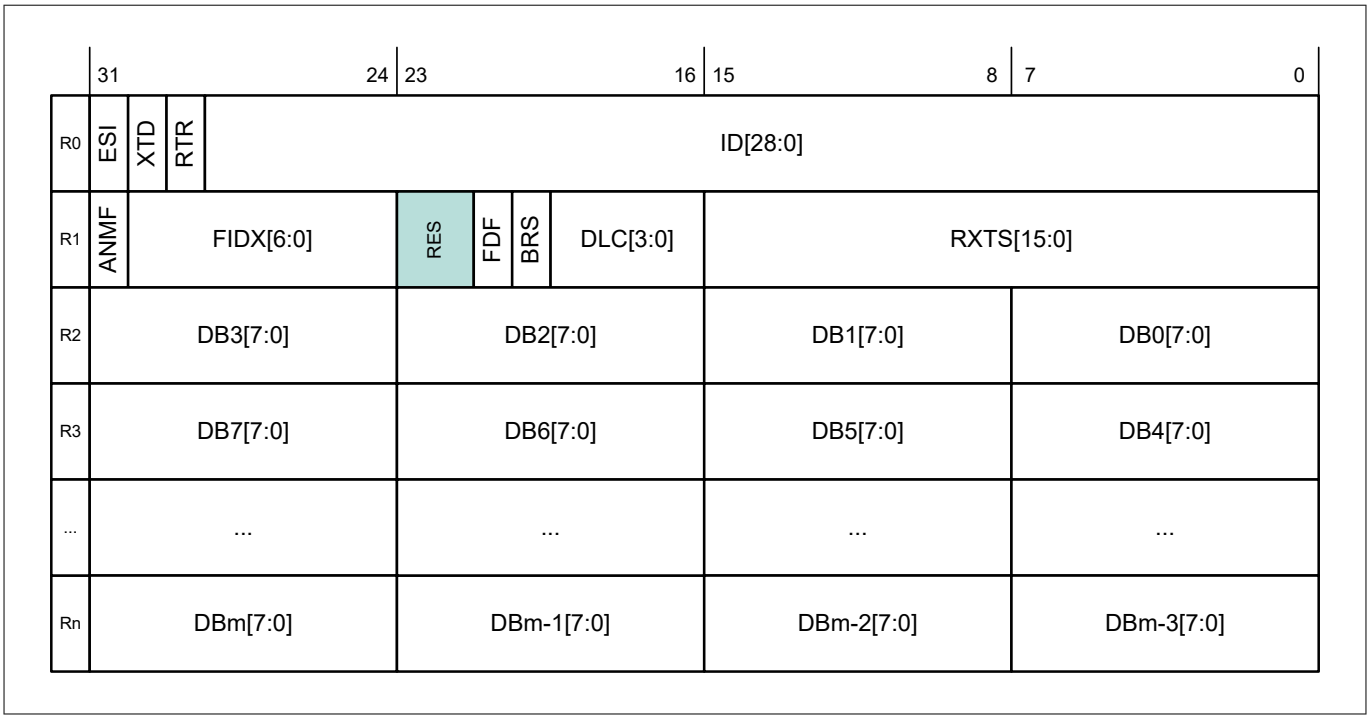


Figure 105 RX buffer and FIFO

Table 165 R0 [bit31] ESI: error state indicator

Bit	Description
0	Transmitting node is error active
1	Transmitting node is error passive

Signals to the CPU whether the received frame has a standard or extended identifier.

Table 166 R0 [bit30] XTD: extended identifier

Bit	Description
0	11-bit standard identifier
1	29-bit extended identifier

Signals to the CPU whether the received frame is a data frame or a remote frame.

Table 167 R0 [bit29] RTR: remote transmission request

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 remote request substitution (RRS) bit replaces bit remote transmission request (RTR).*

R0 [bit28:0] ID[28:0]: Identifier

A standard or extended identifier depending on bit XTD. A standard identifier is stored into ID[28:18].

Table 168 R1 [bit31] ANMF: accepted non-matching frame

Bit	Description
0	Received frame matching filter index FIDX
1	Received frame did not match any RX filter element

Acceptance of non-matching frames may be enabled via GFC.ANFS[1:0] (Accept non-matching frames standard) and GFC.ANFE[1:0] (Accept non-matching frames extended).

Table 169 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(SIDFC.LSS - 1 resp. XIDFC.LSE - 1)

R1 [bit23:22] Reserved: Reserved bits

When writing, always write '0'. The read value is undefined.

Table 170 R1 [bit21] FDF: extended data length

Bit	Description
0	Classic CAN frame format
1	CAN FD frame format

Table 171 R1 [bit20] BRS: bit rate switch

Bit	Description
0	Frame received without bit rate switching
1	Frame received with bit rate switching

Table 172 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 158 for details

R1 [bit15:0] RXTS[15:0]: RX timestamp

The timestamp counter value captured on start of frame reception. Resolution depending on configuration of the Shared Timestamp Counter Pre-scaler TS_CTL.PRESCALE[15:0].

Register name	Data bits	Data byte
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

Register name	Data bits	Data byte
R3 [bit31:24]	DB7[7:0] :	Data Byte 3
R3 [bit23:16]	DB6[7:0] :	Data Byte 2
R3 [bit15:8]	DB5[7:0] :	Data Byte 1
R3 [bit7:0]	DB4[7:0] :	Data Byte 0
- - -	- - -	- - -
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 m3

***Note**:*
- *Depending on the configuration of the element size (defined by RX buffer/FIFO element size configuration (RXESC)), Rn will vary from n = 3 to 17*
- *m is a function of n, m = (n - 1) × 4 - 1*
- *The number of valid data bytes are defined by the Data Length Code*

25.4.3 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 is 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 TXBC.TFQS[5:0] (Transmit FIFO/queue size) and TXBC.NDTB[5:0] (Number of dedicated transmit buffer). The element size is configured to store CAN FD messages with up to 64 bytes data field via register TX buffer element size configuration (TXESC).

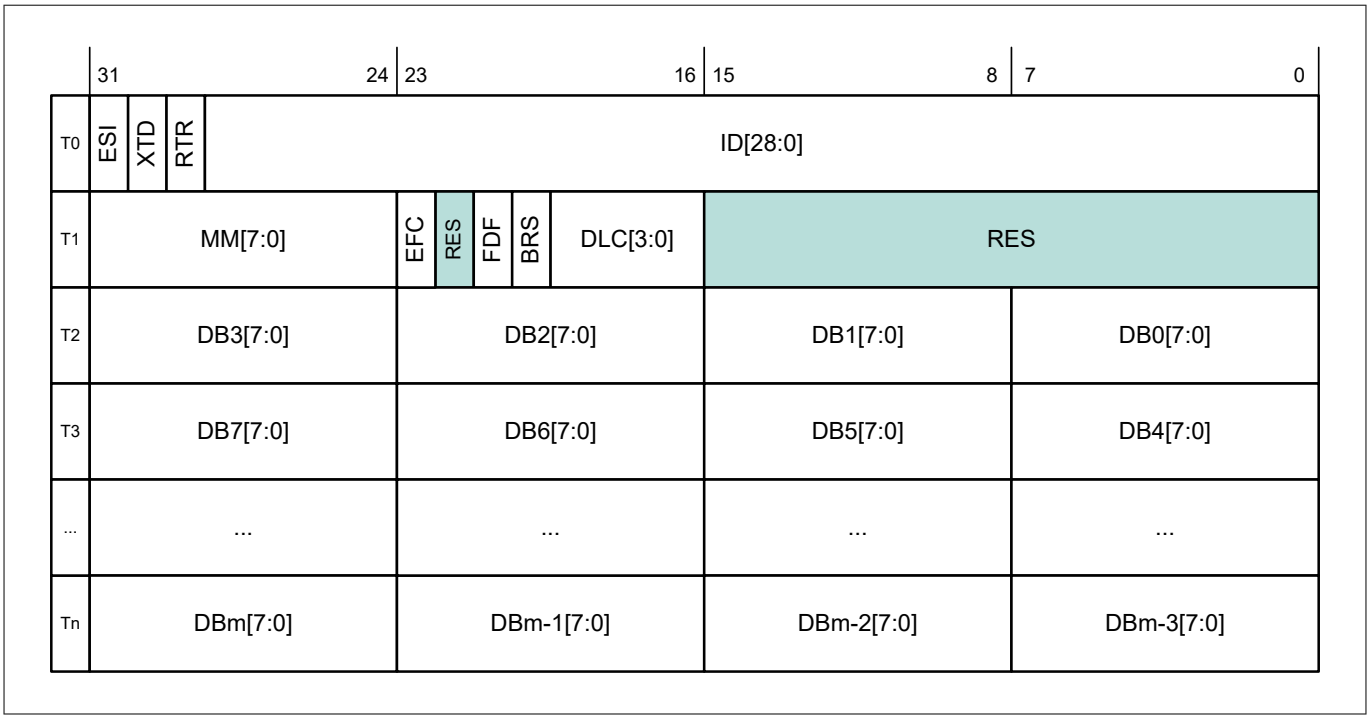


Figure 106 TX buffer element

Table 173 R0 [bit31] ESI: Error state indicator

Bit	Description
0	ESI bit in CAN FD format depends only on 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.*

Table 174 R0 [bit30] XTD: extended identifier

Bit	Description
0	11-bit standard identifier
1	29-bit extended identifier

Signals to the CPU whether the received frame has a standard or extended identifier.

Table 175 R0 [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 ISO11898-1, even if FD Operation Enable (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

Table 176 T1 [bit23] EFC: Event FIFO control

Bit	Description
0	Does not store TX events
1	Store TX events

T1 [bit22] Reserved: Reserved bit

When writing, always write '0'. The read value is undefined.

Table 177 T1 [bit21] FDF: FD format

Bit	Description
0	Frame transmitted in classic CAN format
1	Frame transmitted in CAN FD format

Table 178 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 CCCR.FDOE = 1. Bit BRS is only evaluated when in addition CCCR.BRSE = 1. See Table 162 for details on bits FDF and BRS.*

Table 179 T1 [bit19:16] DLC[3:0]: data length code

DLC[3:0]	Description
0-8	Classic CAN + CAN FD: transmit frame has0-8 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 3
T3 [bit23:16]	DB6[7:0] :	Data Byte 2
T3 [bit15:8]	DB5[7:0] :	Data Byte 1
T3 [bit7:0]	DB4[7:0] :	Data Byte 0
- - -	- - -	- - -
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 m3

***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*

25.4.4 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. The status information about the TX event FIFO can be obtained from register TX event FIFO status (TXEFS).

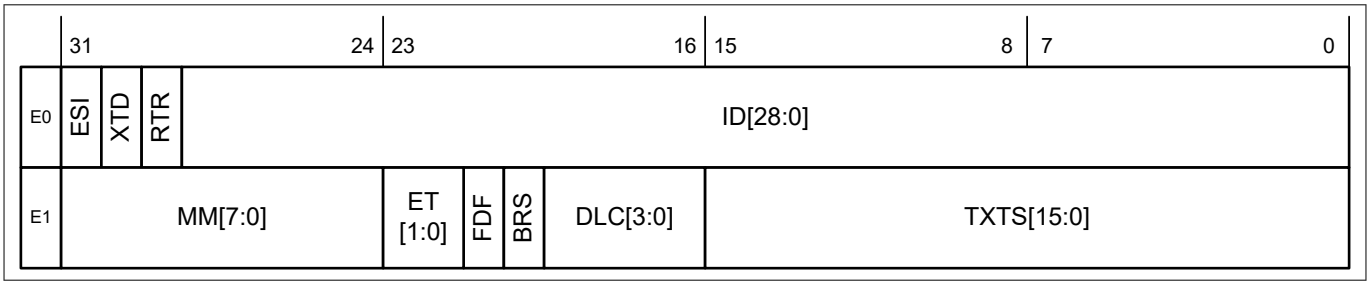


Figure 107 TX event FIFO element

Table 180 E0 [bit31] ESI: error state indicator

Bit	Description
0	Transmitting node is error active
1	Transmitting node is error passive

Table 181 E0 [bit30] XTD: Extended identifier

Bit	Description
0	11-bit standard identifier
1	29-bit extended identifier

Table 182 E0 [bit29] RTR: eemote 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

Table 183 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

Table 184 E1 [bit21] FDF: FD format

Bit	Description
0	Classic CAN frame format
1	CAN FD frame format (new DLC-coding and CRC)

Table 185 E1 [bit20] BRS: bit rate switching

Bit	Description
0	Frame transmitted without bit rate switching
1	Frame transmitted with bit rate switching

Table 186 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 12/16/20/24/32/48/64data bytes transmitted. See Table 158for details

E1 [bit15:0] TXTS[15:0]: TX timestamp

The timestamp counter value captured on start-of-frame transmission. The resolution depends on the configuration of the Shared Timestamp Counter Pre-scaler TS_CTL.PRESCALE[15:0].

25.4.5 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 the Filter List Standard Start Address (SIDFC.FLSSA[15:2]) +index of the filter element (0 to 127).

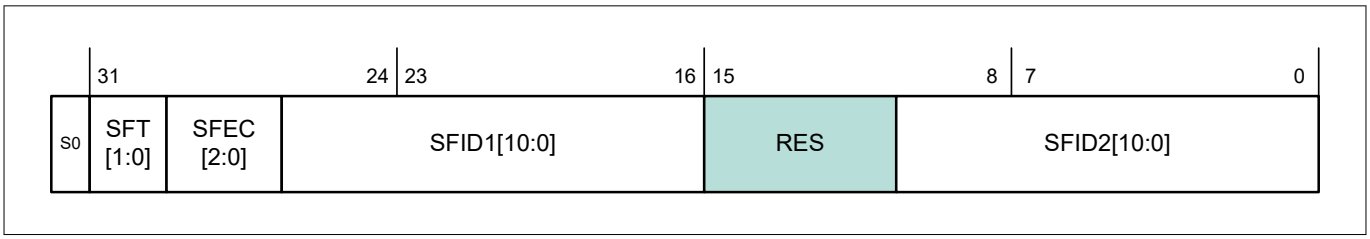


Figure 108 Standard message ID filter

Table 187 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
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 IR.HPM (High Priority Message) and, if enabled, an interrupt is generated. In this case register HPMS (High Priority Message Status) is updated with the status of the priority match.

Table 188 SFEC[2:0]: set priority and store in RX FIFO

SFC[2:0]	Description
000	Disable filter element
001	Store in RX FIFO 0 if filter matches
100	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 11

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.

Table 189 SFID[10:9]] store message into a dedicated RX buffer

SFID[10:9]	Description
00	Store message into a dedicated RX buffer
01	Debug message A
10	Debug message B
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 RXBC.RBSA[15:2] to store a matching message.

***Note**: The debug message is used to debug on CAN feature.*

25.4.6 Extended message ID filter element

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-bitextended IDs. When accessing an extended message ID filter element, its address is Filter list extended start address (XIDFC.FLESA[15:2]) + 2× index of the filter element (0 to 63).

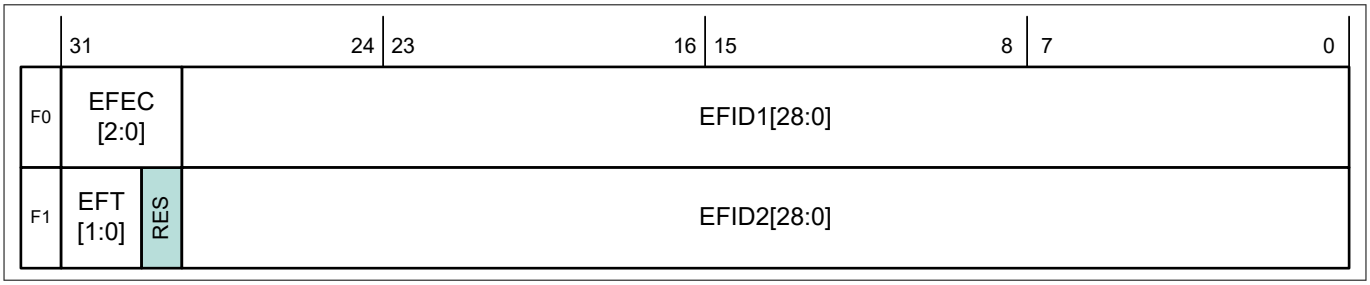


Figure 109 Extended message ID filter

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 IR.HPM (High Priority Message) and, if enabled, an interrupt is generated. In this case High Priority Message Status register (HPMS) is updated with the status of the priority match.

Table 190 Extended filter element

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 110

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

Table 191 Extended filter type

EFT[2:0]	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.

Table 192 Extended filter type

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.

RXBC.RBSA[15:2] to store a matching message.

***Note**: The debug message is used to debug on CAN feature.*

25.4.7 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 (TTTMC). TMSA plus the index of the trigger memory element (0…63).

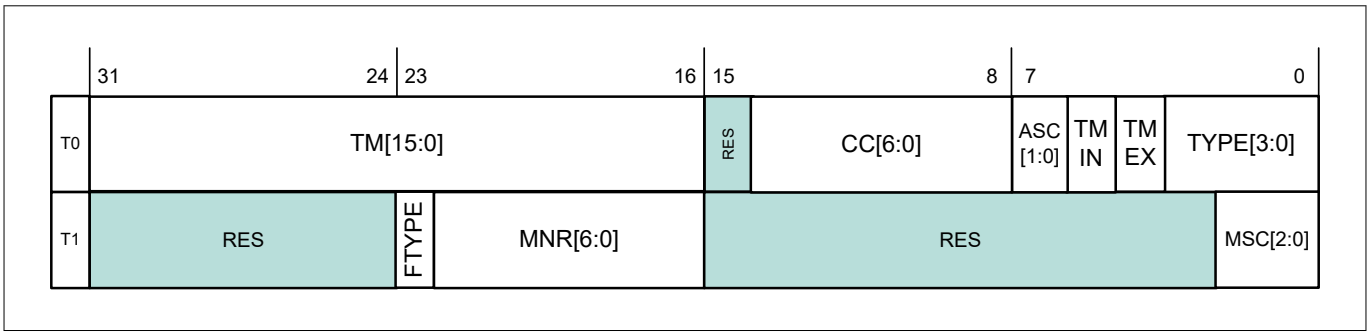


Figure 110 Trigger memory element

T0 Bit 31:16 TM[15:0]: Time Mark

Cycle time for which the trigger becomes active.

Table 193 T0 Bit 14:8 CC[6:0]: cycle code

CC[6:0]	Description
000000x	Valid for all cycles
000001c	Valid every second cycle at cycle count mod2= c
00001cc	Valid every fourth cycle at cycle count mod4 =cc
0001ccc	Valid every eighth cycle at cycle count mod8 =ccc
001cccc	Valid every sixteenth cycle at cycle countmod16 = cccc
01ccccc	Valid every thirty-second cycle at cycle countmod32 = ccccc
1cccccc	Valid every sixty-fourth cycle at cycle countmod64 = cccccc

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.

Table 194 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.*

Table 195 T0 Bit 5 TMIN: Time Mark Event Internal

TMIN	Description
0	No action
1	TTIR.TTMI is set when trigger memory element becomes active

Table 196 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_Continous - starts continuous transmission in an exclusive time window
0100	Tx_Trigger_Arbitration - starts a transmission in an arbitrating time window

(table continues...)

Table 196 (continued) T0 Bit 3:0 TYPE[3:0]: Trigger type

TYPE [3:0]	Description
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 config 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*

Table 197 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 to127.

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 TYPETx_Ref_Trigger. In this case they become active at the time mark modified by the actual Reference Trigger Offset (TTOST.RTO)*

25.4.8 RAM watchdog (RWD)

The RAM watchdog monitors the READY output of the MRAM. A MRAM access starts the Message RAM watchdog counter with the value configured by RWD.WDC. The counter is reloaded with RWD.WDC when the MRAM signals successful completion by activating its READY output. In case there is no response from the MRAM until the counter has counted down to zero, the counter stops and interrupt flag IR.WDI is set. The RWD counter is clocked by the host clock(clk_sys). See the registers reference manual for more information on the RWD register.

25.5 Time trigger CAN

25.5.1 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 is extended by other data up to the sum of eight CAN data bytes. All the 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. The reserved bits are transmitted as logical 0 and are ignored by the receivers. The reference message is configured using the 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.

Level 1

Level 1 operation is configured via TTOCF.OM = 01 and 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 199. Cycle_Count is optional.

Table 198 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]					

Level 2

Level 2 operation is configured via TTOCF.OM = 10 and TTOCF.GEN.

The information related to the reference message is stored in the first four data bytes as shown in Table 199. 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 199 First four Bytes of Level 1 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]							

Level 0

Level 0 operation is configured via 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 200. 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 '0'.

Table 200 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]							

25.5.2 TTCAN configuration

25.5.2.1 TTCAN timing

The network time unit (NTU) is the unit for the measurement of time. 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 the TTCAN Level 0 and Level 2, the NTU is a fraction of the physical second.

The NTU is the base for the local time. The integer part of the local time (16-bit value) is incremented once for each NTU. The cycle time and the global time are both derived from the local time. The fractional part (3-bit value) of the local time, cycle time, and the global time are not readable.

In the TTCAN Level 0 and Level 2, the length of the NTU is defined by the time unit ratio (TUR). The TUR is a non\u0002integer number is given by TUR = TURNA.NAV/TURCF.DC. The length of the NTU is determined by NTU = CAN Clock Period × TUR.

The TUR numerator configuration (NC) is an 18-bit number, TURCF.NCL[15:0] is programmed in the range 0x0000-0xFFFF. TURCF.NCH[17:16] is hard-wired to '0b01'. When the number '0xnnn'n is written to TURCF.NCL[15:0], TURNA.NAV starts with the value 0x10000 + 0x0nnnn = 0x1nnnn. The TUR denominator configuration (TURCF.DC) is a 14-bit number. The TURCF.DC is programmed in the range 0x0001 -0x3FFF; 0x0000 is an illegal value.

In Level 1, NC must be ≥ 4 × TURCF.DC. In Level 0 and Level 2 NC must be ≥ 8 × TURCF.DC to allow the 3-bit resolution for the internal fractional part of the NTU.

A hardware reset presets TURCF.DC to 0x1000 and TURCF.NCL to 0x10000, resulting in an NTU consisting of 16 CAN clock periods. The local time and application watchdog are not started before either the CCCR.INIT is reset or TURCF.ELT is set. The TURCF.ELT is not set before the NTU is configured. Setting TURCF.ELT to '1' also locks the write access to register TURCF.

At start-up TURNA.NAV is updated from NC (= TURCF.NCL +0x10000) when TURCF.ELT is set. In the TTCAN Level 1 there is no drift compensation. TURNA.NAV does not change during operation, it always equals to NC.

In TTCAN Level 0 and Level 2, there are two possibilities for TURNA.NAV to change. When operating as time slave or backup time master, and when TTOCF.ECC is set, 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 TTOCF.EECS is set, the host updates the TURCF.NCL. When the host sets TTOCN.ECS, TURNA.NAV will be updated from the new value of NC at the next reference message. The status flag TTOST.WECS is set when TTOCN.ECS is set and is cleared when TURNA.NAV is updated. The TURCF.NCL is write-locked while TTOST.WECS is set.

In TTCAN Level 0 and Level 2, the clock calibration process adapts TURNA.NAV in the range of the synchronization deviation limit (SDL) of NC ± 2 (TTOCF.LDSDL+5). TURCF.NCL should be programmed to the largest applicable numerical value to achieve the best accuracy in the calculation of TURNA.NAV.

The synchronization deviation (SD) is the difference between NC and TURNA.NAV (SD = |NC - TURNA.NAV|). It is limited by the SDL, which is configured by its dual logarithm

TTOCF.LDSDL (SDL = 2 (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 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, TTIR.GTE is set, and TTOSC.QCS is reset; if Disc_Bit = 1, TTIR.GTD is set. The TUR configuration examples are shown in Table 201

Table 201 TUR configuration examples

TUR	8	10	24	50	510	12500	32.5	100/12	529/17
NC	0x1FFF8	0x1FFFE	0x1FFF8	0x1FFEA	0x1FFFE	0x1FFE0	0x1FFE0	0x19000	0x10880
TURCF.DC	0x3FFF	0x3333	0x1555	0x0A3D	0x0101	0x0001	0xFC0	0x3000	0x0880

TTOCN.ECS schedules NC for activation by the next reference message. TTOCN.SGT schedules TTGTP.TP for activation by the next reference message. Setting of TTOCN.ECS and TTOCN.SGT requires TTOCF.EECS to be set (external clock synchronization enabled) while the M_TTCAN is the 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, else all CAN bus activity is stopped. The Application Watchdog Limit 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 TTOST. TTOST.AWE indicates whether the watchdog is served in time. In case the application failed to serve the application watchdog, the interrupt flag TTIR.AW is set. For software development, the application watchdog may be disabled by programming TTOCF.AWL to '0x00' (see Application watchdog).

25.5.2.2 Message scheduling

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 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. TTRMC.RID is used to recognize the reference messages. TTRMC.RMPS is not relevant for time slaves.

The initial reference trigger offset TTOCF.IRTO is a 7-bit value that defines (in NTUs) the duration, 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 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 the maximum clock drift.

TTOCF.OM decides whether the node operates in TTCAN Level 0, Level 1, or Level 2. In one network, all potential time masters operates 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 TTOCF.OM is the last step in the setup. When TTOCF.OM = 00 (event-driven CAN communication), the M_TTCAN operates according to ISO 11898-1:2015, without time triggers. When 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 TTOCF.OM = 10 (Level 2), the M_TTCAN operates according to ISO 11898-4, including

the event\u0002synchronized start of a basic cycle. When TTOCF.OM = 11 (Level 0), the M_TTCAN operates as event-driven CAN but maintains a calibrated global time base similar to Level 2.

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.

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**: 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 TTMLM.ENTT will result in either a TX count underflow (TTIR.TXU = 1 and TTOST.EL = 01, severity 1) or in a TX count overflow (TTIR.TXO = 1 and TTOST.EL = 10, severity 2).*

***Note**: In case the first reference message seen by a node does not have Cycle_Count zero, this node finishes its first matrix cycle with 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.*

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 TTMLM.CCM would be '7'. 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.

TTMLM.TXEW specifies the length of the TX enable window in the NTUs. The TX enable window is the period at the beginning of a time window where a transmission is 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. TTMLM.TXEW is chosen based on the network's synchronization quality and the relation between the length of the time windows and the length of messages.

25.5.2.3 Trigger memory

The trigger memory is part of the MRAM. 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 elementfor more information).

The time mark defines the cycle time in which a trigger becomes active. The trigger elements in the trigger memory are 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 this point, cycle time restarts. 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.

25.5.2.3.1 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 (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\u0002synchronized time-triggered operation mode. In that mode, Tx_Ref_Trigger is ignored when the M_TTCAN synchronization state is In_Gap (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 (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 (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 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 (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 the 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 the 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 (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.

25.5.2.3.2 Restrictions for the node's trigger list

Two triggers are not active at the same cycle time and cycle count, but triggers that are active in different basic cycles (different cycle code) shares 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 are placed after Tx_Trigger_Merged.

The triggers placed after the Watch_Trigger (or the Watch_Trigger_Gap when TTOST.SYS = 10) will never become active. The watch triggers themselves will not become active when the reference messages are transmitted on time.

All the unused trigger memory words (after the Watch_Trigger or after the Watch_Trigger_Gap when 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 is 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 TTOCF.IRTO. If error level 2 is reached (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 are not active until the reference messages comes on time.

There are inter-dependencies 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

25.5.2.3.3 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 202.

Table 202 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

(table continues...)

Table 202 (continued) System matrix node A

Cycle count	Time mark1	Time mark2	Time mark3	Time mark4	Time mark5	Time mark6	Time Mark7
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 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').

For example, when CC is 0b0010011 (repeat factor: 16, first basic cycle: 3) and maximum cycle count of 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 the 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 203 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	0000100	Tx_Trigger_Single	7
1	Mark1	1000000	Rx_Trigger	3
2	Mark1	1000011	Tx_Trigger_Single	7
3	Mark3	1000001	Tx_Trigger_Merged	2
4	Mark3	1000011	Rx_Trigger	5
5	Mark4	1000001	Tx_Trigger_Arbitration	4
6	Mark5	1000100	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.

25.5.2.3.4 Detection of configuration errors

A configuration error is signaled via TTOST.EL = 11 (severity 3) when:

- The FSE comes to a trigger in the list with a CC that matches the current cycle count but with a time mark 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 CC that matches the current cycle count but is it not a Tx_Trigger_Merged , Tx_Trigger_Arbitration nor a Time_Base_Trigger nor an Rx_Trigger
- The FSE of a node with 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 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 TTOST.RTO causes a reversal of their sequential order measured in the cycle time

25.5.2.4 TTCAN schedule initialization

The synchronization to the M_TTCAN message schedule starts when CCCR.INIT is reset. The M_TTCAN operates time-triggered (TTOCF.GEN = 0) or external event\u0002synchronized time-triggered (TTOCF.GEN = 1). All the nodes start with cycle time zero at the beginning of the trigger list with TTOST.SYS = 00 (out-of-synchronization); no transmission is enabled with a reference message as an exception. The nodes in external event-synchronized time\u0002triggered operation mode ignores Tx_Ref_Trigger and Watch_Trigger and uses Tx_Ref_Trigger_Gap and Watch_Trigger_Gap, until the first reference message decides whether a gap is active.

25.5.2.4.1 Time slaves

After the 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 TTIR.IWT is set, the FSE is frozen, and the cycle time becomes invalid. However, the node still takes 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 (TTOST.EL = 11, severity 3), set interrupt flag TTIR.WT, switches off the CAN bus output, and enters the bus monitoring mode (CCCR.MON set to '1'). In the Bus Monitoring mode, it receive messages, but cannot send any dominant bits and therefore, it is not acknowledged.

***Note**: To leave the fatal error state, the host must set CCCR.INIT to '1'. After reset of CCCR.INIT, the node restarts TTCAN communication.*

When no error is encountered during synchronization, the first reference message sets TTOST.SYS = 01 (synchronizing), the second sets the TTCAN synchronization state (depending on its Next_is_Gap bit) to TTOST.SYS = 11 (In_Schedule) or TTOST.SYS = 10 (In_Gap), enabling all Tx_Triggers and Rx_Triggers.

25.5.2.4.2 Potential time master

After the 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 the Watch_Trigger and Watch_Trigger_Gap when it does not receive any message or transmit the reference message before reaching the Watch_Triggers (the reason assumed is that all other nodes

are in the reset or configuration and does not acknowledge). When it reaches Init_Watch_Trigger, the attempted transmission is aborted, the interrupt flag TTIR.IWT is set, the FSE is frozen, and the cycle time will become invalid, but the node will still take part in the CAN bus communication (to acknowledge or send error flags). Resetting TTIR.IWT will reenable the transmission of reference messages until the next time the 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 (TTOST.EL = 11, severity 3), set interrupt flag TTIR.WT, switch off its CAN bus output, and enter the bus monitoring mode (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 TTOST.SYS = 01 (synchronizing), the second sets the TTCAN synchronization state (depending on its Next_is_Gap bit) to TTOST.SYS = 11 (In_Schedule) or TTOST.SYS = 10 (In_Gap), enabling all Tx_Triggers and Rx_Triggers.

A potential time master is the current time master (TTOST.MS = 11) when it is the transmitter of the last reference message; otherwise, it is the backup time master (TTOST.MS = 10).

When all potential time masters finishes the configuration, the node with the highest time master priority in the network becomes the current time master.

25.5.3 TTCAN gap control

All functions related to gap control apply only when the M_TTCAN is operated in external event-synchronized time-triggered mode (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 a 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 TTOCN.NIG = 1. The Next_is_Gap bit is transmitted as '1' with the next reference message. A gap can also be initiated under hardware control when the application program writes TTOCN.GCS = 1. When a reference message is started and TTOCN.GCS is set, Next_is_Gap = 1 will be set.

As soon as that reference message is completed, the TTOST.WFE bit will announce the gap to the time between the master and replicas. 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, TTOST.GSI will be set when the last basic cycle has finished and the gap time starts. In nodes that are time slaves, the TTOST.GSI bit will remain at '0'.

When a potential time master is in synchronization state In_Gap (TTOST.SYS = 10), it has four options to as follows:

- Under software control by writing TTOCN.FGP = 1
- Under hardware control (TTOCN.GCS = 1), TTOCN.FGP is automatically 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 TTOCN.TMG = 1, the next register time mark interrupt (TTIR.RTMI = 1) will set 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 TTOCN.FGP after the gap time has started will start the transmission of a reference message immediately and synchronizes the message schedule. When 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, the bit Next_is_Gap = 1 in the reference message will be ignored, and the bits TTOCN.NIG, TTOCN.FGP, and TTOCN.TMG will be considered.

25.5.4 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 the stop watch source via TTOCN.SWS. When TTOCN.SWS is not equal to ‘00’ and TT Interrupt Register flag TTIR.SWE is ‘0’, the actual value of the time selected by TTOCN.SWS will be copied into TTCPT.SWV on the next rising/falling edge (as configured via TTOCN.SWP) on stop watch trigger. This will set the interrupt flag TTIR.SWE. After the application program has read TTCPT.SWV, it may enable the next stop watch event by resetting TTIR.SWE to '0'.

25.5.5 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 in 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 111 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 the TTCSM register. 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).

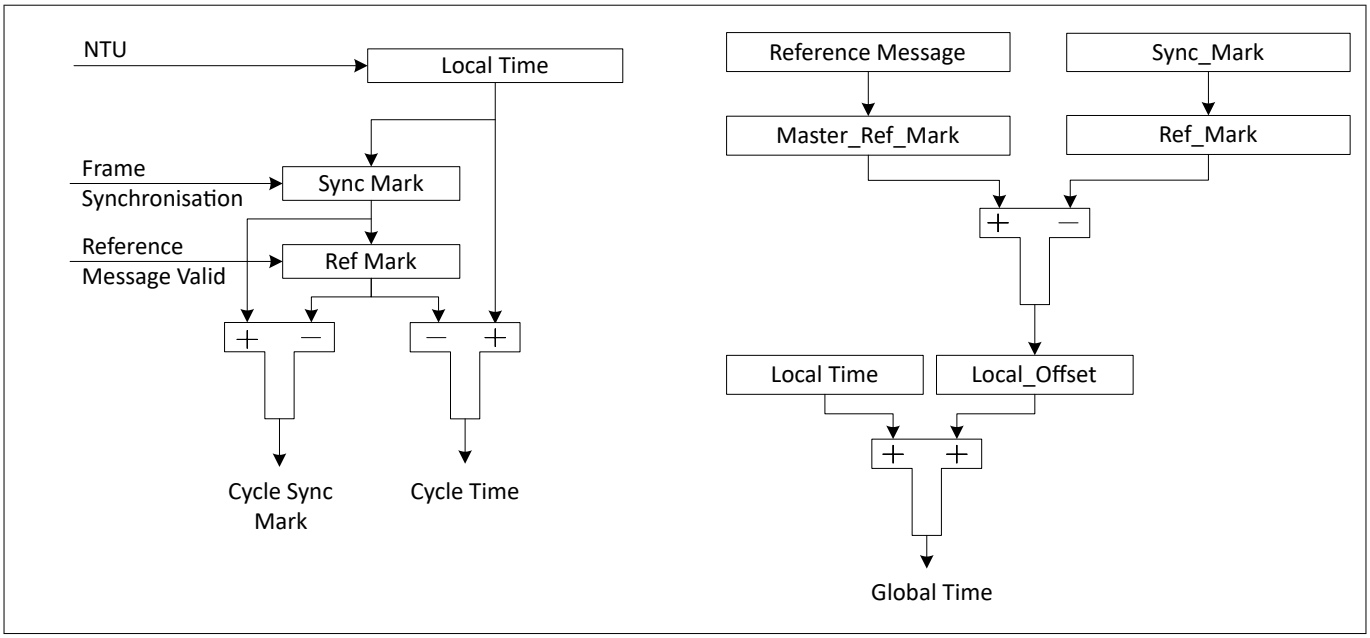


Figure 111 Cycle time and global time synchronization

The cycle time that can be read from 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 is read from 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 111). 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. An exception of global time discontinuity, it is provided to the application program by the TTLGT register and smoothed by a low-pass filtering to have a continuous monotonic value.

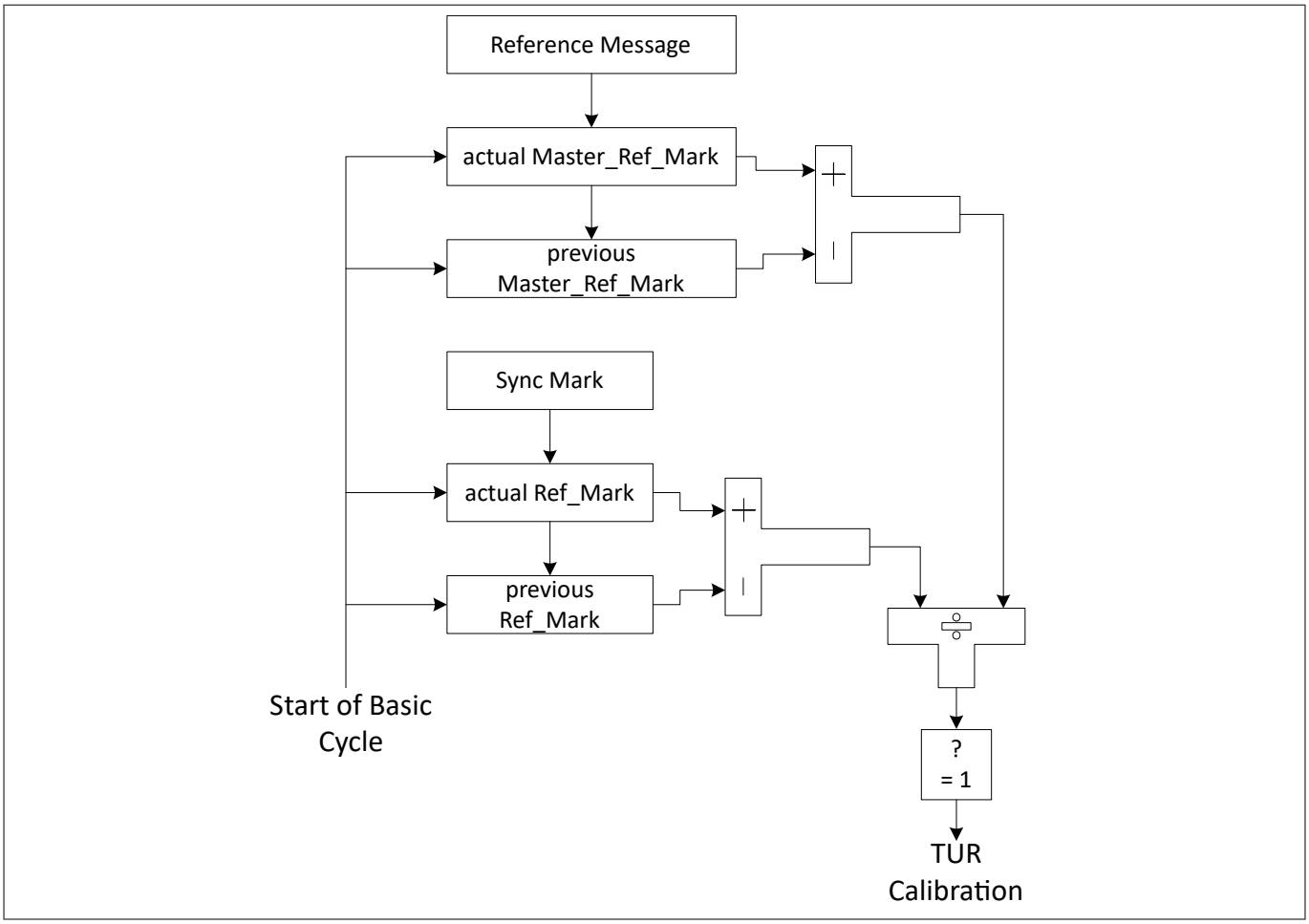


Figure 112 TTCAN Level 0 and Level 2 drift compensation

Figure 112 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 TURNA.NAV is calculated. If the synchronization deviation (SD) = |NC – TURNA.NAV| ≤ SDL, the new value for TURNA.NAV takes effect. Otherwise, the automatic drift compensation is suspended.

In TTCAN Level 0 and Level 2, TTOST.QCS indicates whether the automatic drift compensation is active or suspended. In TTCAN Level 1, 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 the bit 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 TURCF.NCL (TURCF.DC cannot be updated during operation). The new value takes effect by writing TTOCN.ECS to '1'.

The global time phase is adjusted by first writing the phase offset into the TT Global Time Preset register (TTGTP). The new value takes effect by writing TTOCN.SGT to '1'. The first reference message transmitted after the global time phase adjustment will have the Disc_Bit set to '1'.

TTOST.QGTP shows whether the node's global time is in phase with the time master's global time. TTOST.QGTP is permanently '0' in TTCAN Level 1 and when the SDL is exceeded in TTCAN Level 0,2 (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 TTOST.QCS = 0.

25.5.6 TTCAN error level

The ISO 11898-4 specifies four levels of error severity:

Table 204 Levels of error severity

Level	Severity	Description
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 mas\u0002ters still transmit reference messages with the Reference Trigger Offset TTOST.RTO set to the maximum value of 127.
S3	Severe error	Notification of application. All CAN bus operations are stopped; transmission of dominant bits is not allowed, and CCCR.MON is set. The S3 error condition remains active until the application updates the configuration (set 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 TTIR.ELC. The error level is monitored by TTOST.EL.

The M_TTCAN signals the following error conditions as required by ISO 11898-4:

Table 205 Error condition

Error condition	Description
Config_Error (S3)	Sets error level 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 TTOST.EL to ‘11’ when a watch trigger is reached because the reference message is missing.
Application_Watchdog (S3)	Sets error level TTOST.EL to ‘11’ when the application fails to serve the application watchdog. The application watchdog is configured via TTOCF.AWL. It is served by reading the TTOST register. When the watchdog is not served in time, bit TTOST.AWE and interrupt flag TTIR.AW are set, all TTCAN communication is stopped, and the M_TTCAN is set into bus monitoring mode (CCCR.MON set to '1').

(table continues...)

Table 205 (continued) Error condition

Error condition	Description
CAN_Bus_Off (S3)	Entering CAN_Bus_Off state sets the error level TTOST.EL to‘11’. CAN_Bus_Off state is signaled by PSR.BO = 1 and CCCR.INIT = 1.
Scheduling_Error_2 (S2)	Sets error level TTOST.EL to ‘10’ if the MSC of one Tx_Trigger has reached 7. In addition, the interrupt flag TTIR.SE2 is set. 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 TTOST.EL to ‘10’ when the TX count is equal or higher than the expected number of Tx_Triggers TTMLM.ENTT and a Tx_Trigger event occurs. In addition, the interrupt flag TTIR.TXO is set. TTOST.EL is reset to 00 when the TX count is no more than TTMLM.ENTT at the start of a new matrix cycle.
Scheduling_Error_1 (S1)	Sets the error level 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 TTIR.SE1 is set. If within one matrix cycle none of these conditions is valid, TTOST.EL is reset to 00.
Tx_Underflow (S1)	Sets error level TTOST.EL to ‘01’ when the TX count is less than the expected number of Tx_Triggers TTMLM.ENTT at the start of a new matrix cycle. In addition, interrupt flag TTIR.TXU is set. TTOST.EL is reset to 00 when the TX count is at least TTMLM.ENTT at the start of a new matrix.

25.5.7 TTCAN message handling

25.5.7.1 Reference message

For potential time masters, the identifier of the reference message is configured via 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 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 206 Number of data bytes transmitted with a reference messages

TTRMC.RMPS	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 TXBRP.TRP0 of message buffer 0 must be set (see Table 206). If 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 TTRMC.RID is used.

25.5.7.2 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 the 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 the filters used for Rx_Triggers are placed at the beginning of the list. It is not recommended to use an Rx_Trigger for the reference message.

25.5.7.3 Message transmission

For time-triggered message transmission, the M_TTCAN supplies 32 dedicated TX buffers (see TTCAN Configuration on page 379). A TX FIFO or TX queue is not available when the M_TTCAN is configured for time-triggered operation (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, the host proceeds as follows:

Tx_Trigger_Single/Tx_Trigger_Merged/Tx_Trigger_Arbitration:

- Check whether the previous transmission is 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 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 is disabled (M_TTCAN in Error Level S2 or the host is 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 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 TXBCF before it starts updating.

The TX handler will transfer a message from the message RAM to the 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 transmitting message may not be updated and its 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.

25.5.7.3.1 Transmission in exclusive time windows

A transmission starts 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 to 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, 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.

25.5.7.3.2 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.

25.5.7.3.3 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 a immediate sequence, and 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.

25.5.8 TTCAN interrupt and error handling

The TT Interrupt Register TTIR consists of four segments. Each interrupt can be enabled separately by the corresponding bit in the TT Interrupt Enable register 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. The error conditions require a reconfiguration of the M_TTCAN module before the communication is restarted which is an exception for IWT.

The second segment consists of the 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 TTOCN.TMC (cycle, local, or global) equals the time mark TTTMK.TM. It can also be used to finish a gap.

The fourth segment consists of the flags SOG, CSM, SMC, and SBC. These flags provide a means to synchronize the application program to the communication schedule.

25.5.9 Level 0

TTCAN Level 0 is not part of ISO11898-4. This operation mode makes the hardware, that in TTCAN Level 2 maintains the calibrated global time base, also available for event\u0002driven CAN according to ISO 11898-1:2015.

Level 0 operation is configured via TTOCF.OM = 11. In this mode, M_TTCAN operates in event-driven CAN communication; there is no fixed schedule. The configuration of TTOCF.GEN is ignored. External event-synchronized operation is not available in Level 0. A synchronized time base is maintained by the transmission of reference messages.

In Level 0 the trigger memory is not active and need not be configured. The time mark interrupt flag (TTIR.TTMI) is set when the cycle time has reached TTOCF.IRTO × 0x200. It reminds the host to set a transmission request for message buffer 0. The Watch_Trigger interrupt flag (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 (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 TTRMC. For the transmission of reference messages only message buffer 0 may be used. The node transmits reference messages any time that the host sets a transmission request for message buffer 0; there is no reference trigger offset.

Level 0 operation is configured via:

- TTRMC

- TTOCF except EVTP, AWL, GEN
- TTMLM except ENTT, TXEW
- TURCF

Level 0 operation is controlled via:

- TTOCN except NIG, TMG, FGP, GCS, TTMIE
- TTGTP
- TTTMK
- TTIR excluding bits CER, AW, IWT, SE2, SE1, TXO, TXU, SOG (no function)
- TTIR – the following bits have changed function:
  - TTMI not defined by trigger memory - activated at cycle time TTOCF.IRTO × 0x200
  - WT not defined by trigger memory - activated at cycle time 0xFF00

Level 0 operation is signaled via:

- TTOST excluding bits AWE, WFE, GSI, GFI, RTO (no function)

25.5.9.1 Synchronizing

Figure 113 describes the states and state transitions in TTCAN Level 0 operation. Level 0 has no In_Gap state.

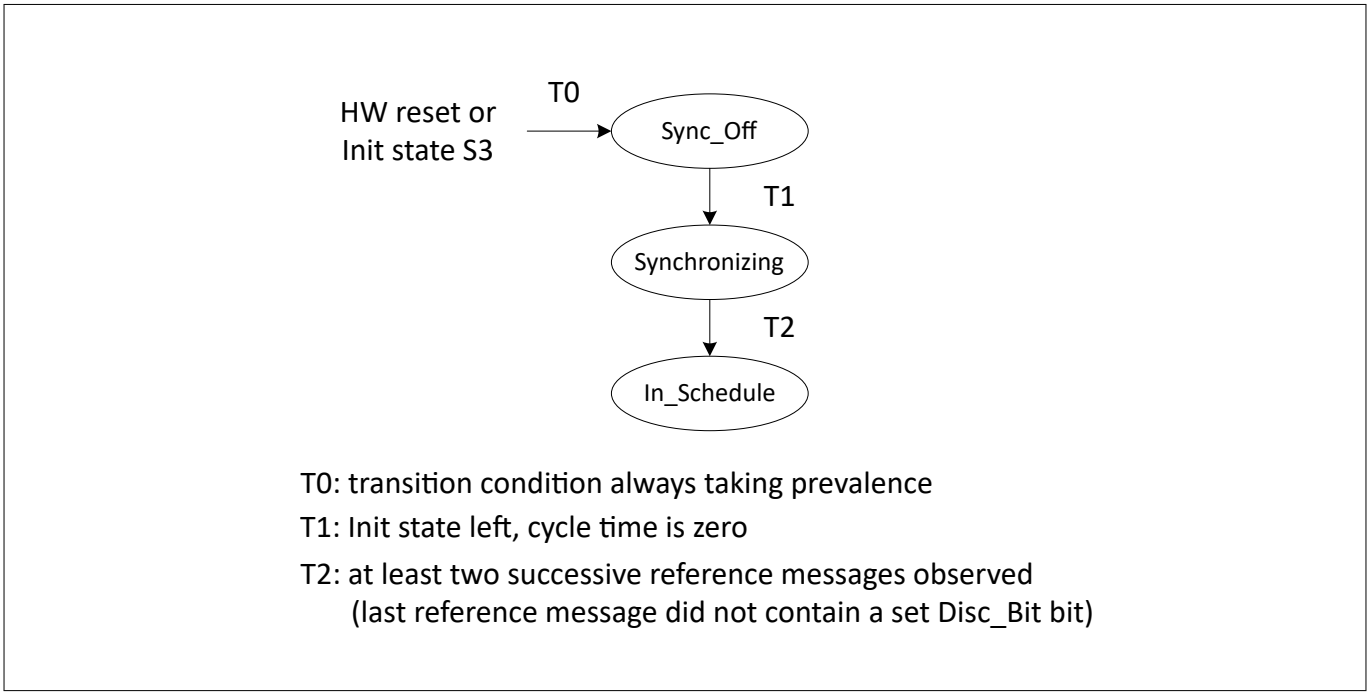


Figure 113 Level 0 Schedule Synchronization State Machine

25.5.9.2 Handling error levels

During the Level 0 operation the following error conditions occurs:

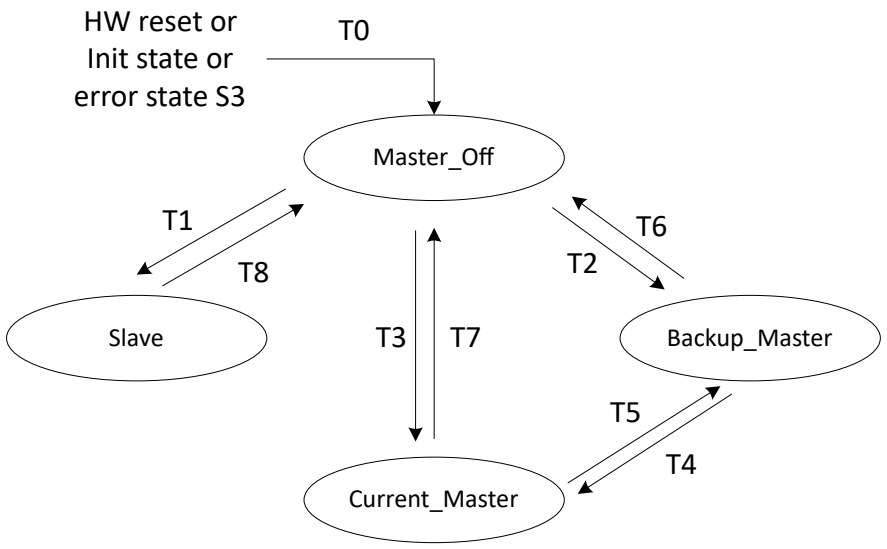
- Watch_Trigger_Reached (S3), reached cycle time 0xFF00
- CAN_Bus_Off (S3)

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 TTOST.SYS and TTOST.MS are reset, and interrupt flags TTIR.GTE and TTIR.GTD are set.

When S3 (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.

25.5.9.3 Master slave relation

Figure 114 describes the master slave relation in TTCAN Level 0. In case of an S3 error, the M_TTCAN returns to state aster_Off



T0: transition condition always taking prevalence
T1: reference message observed when not potential master
T2: reference message observed with master priority != own master priority, error state != S3
T3: reference message observed with master priority = own master priority, error state != S3
T4: reference message observed with own master priority
T5: reference message observed with master priority higher than own master priority
T6: error state S3
T7: error state S3
T8: error state S3

Figure 114 Level 0 Master to Slave Relation

25.5.10 Synchronization to external time schedule

This feature is 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 (TTOST.MS = 11).

External synchronization is controlled by the TTOCN.ESCN bit. If TTOCN.ESCN is set, at the rising edge of the internal event trigger pin, the M_TTCAN compares its actual cycle time with the target phase value configured by TTGTP.CTP.

Before setting 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 on page 384). When the host sets TTOCN.ESCN, TTOST.SPL is set.

If the difference between the cycle time and the target phase value TTGTP.CTP at the trigger is greater than nine NTU, the phase lock bit TTOST.SPL is reset, and the interrupt flag TTIR.CSM is set. TTOST.SPL is also reset (and TTIR.CSM is set), when another node becomes time master.

If both TTOST.SPL and TTOCN.ESCN are set, and if the difference between the cycle time and the target phase value TTGTP.CTP is less or equal nine NTU, the phase lock bit TTOST.SPL remains set, and the measured difference is used as the 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 comparison of the actual cycle time with TTGTP.CTP. All further edges until the beginning of the next basic cycle are ignored.*

25.6 Setup procedures

This section provides example procedures for configurations of the M_TTCAN channel.

25.6.1 General program flow

This is a general flow to configure the M_TTCAN module.

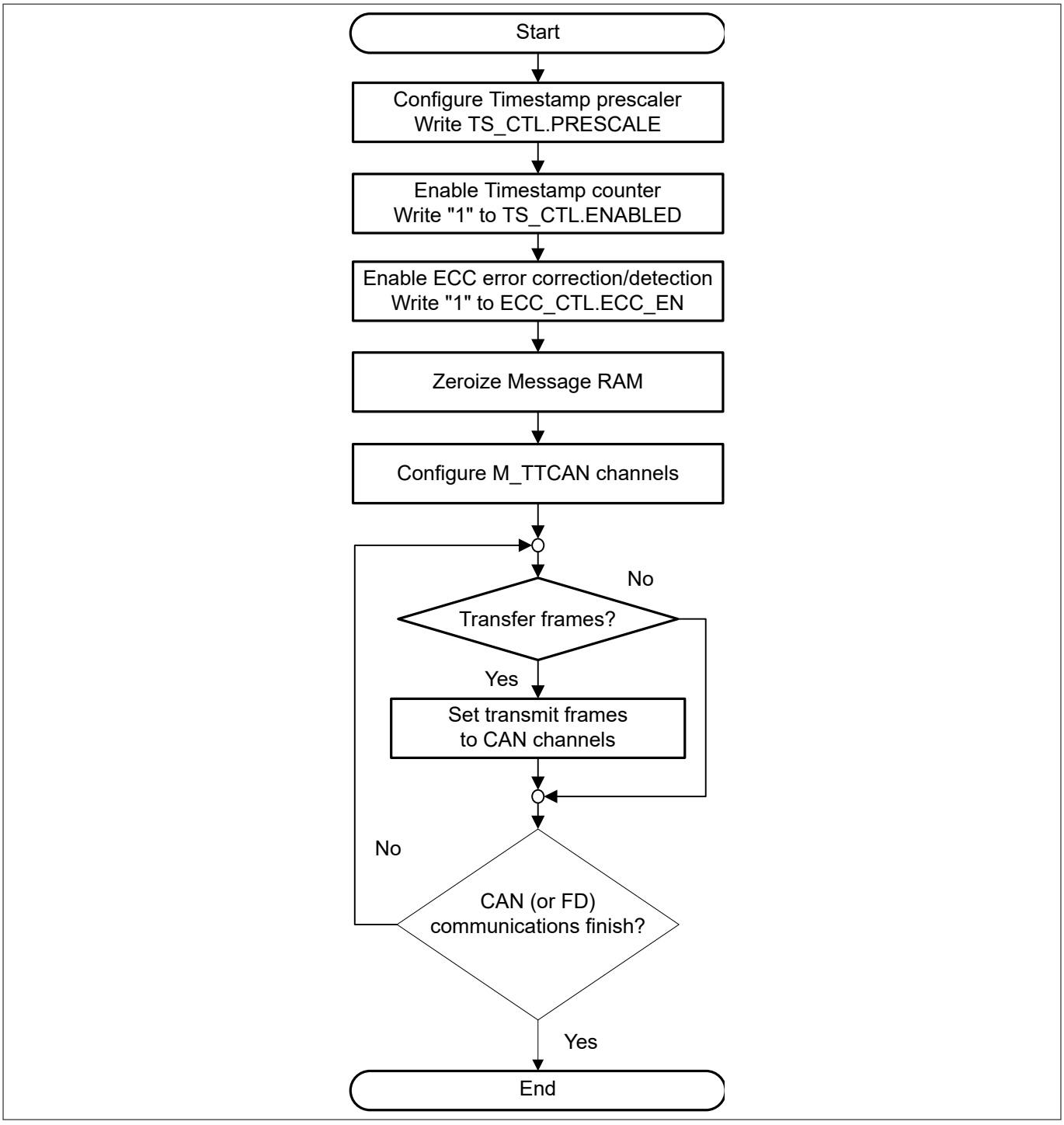


Figure 115 General program flow

25.6.2 Clock stop request

To save power, the application stops to provide the clock to the unused M_TTCAN channel by the steps shown in Figure #.

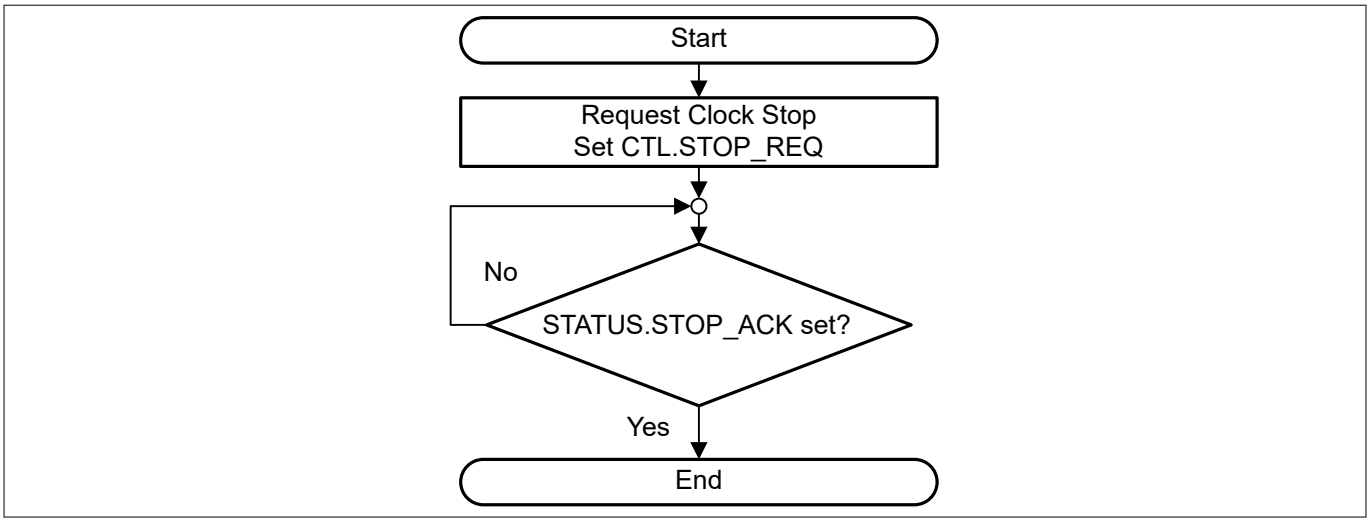


Figure 116 Clock stop request procedure

To resume the providing clock, the CTL.STOP_REQ bit is reset.

25.6.3 MRAM OFF operation

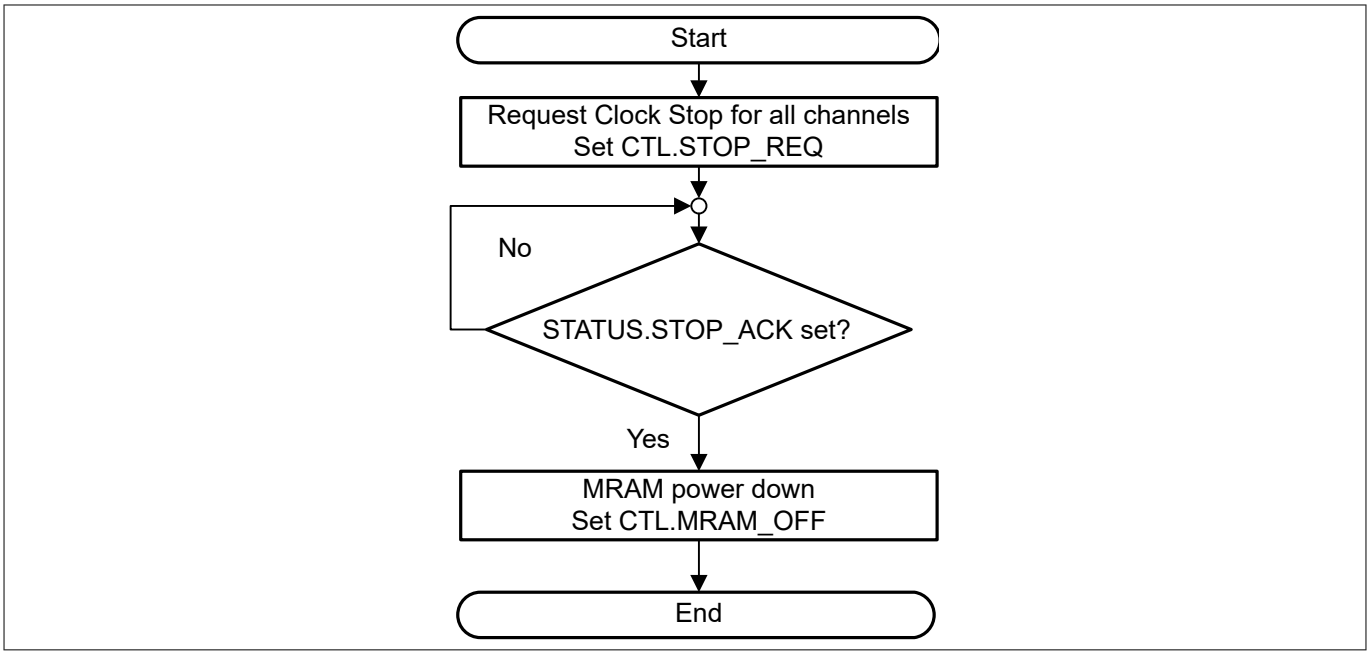


Figure 117 Message RAM OFF operation

25.6.4 MRAM ON operation

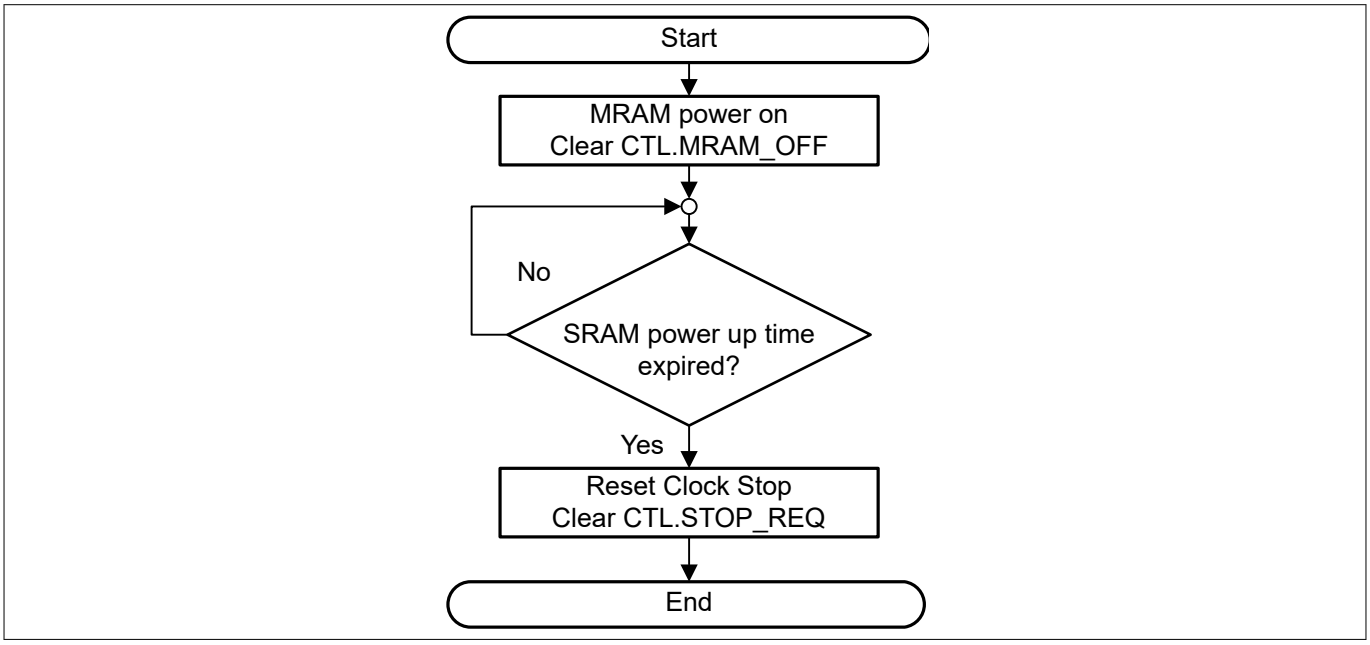


Figure 118 Message RAM On procedure

After switching MRAM ON again, software needs to allow a certain power-up time before MRAM 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.

25.6.5 Procedures specific to M_TTCAN channel

This section describes sample procedure for the M_TTCAN channel. Figure 119 shows the general program flow.

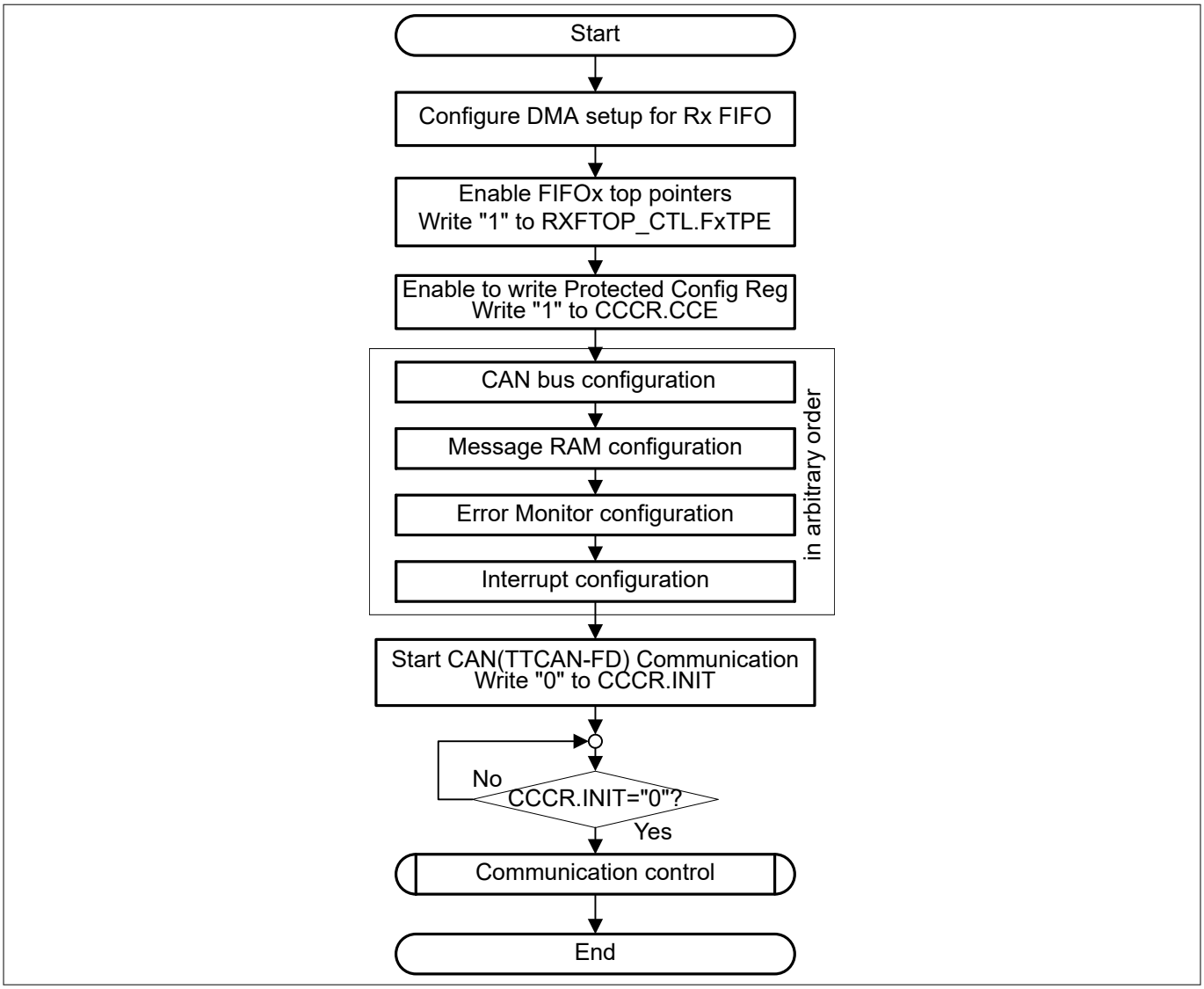


Figure 119 Configuration sequence specific to channel

25.6.6 CAN Bus configuration

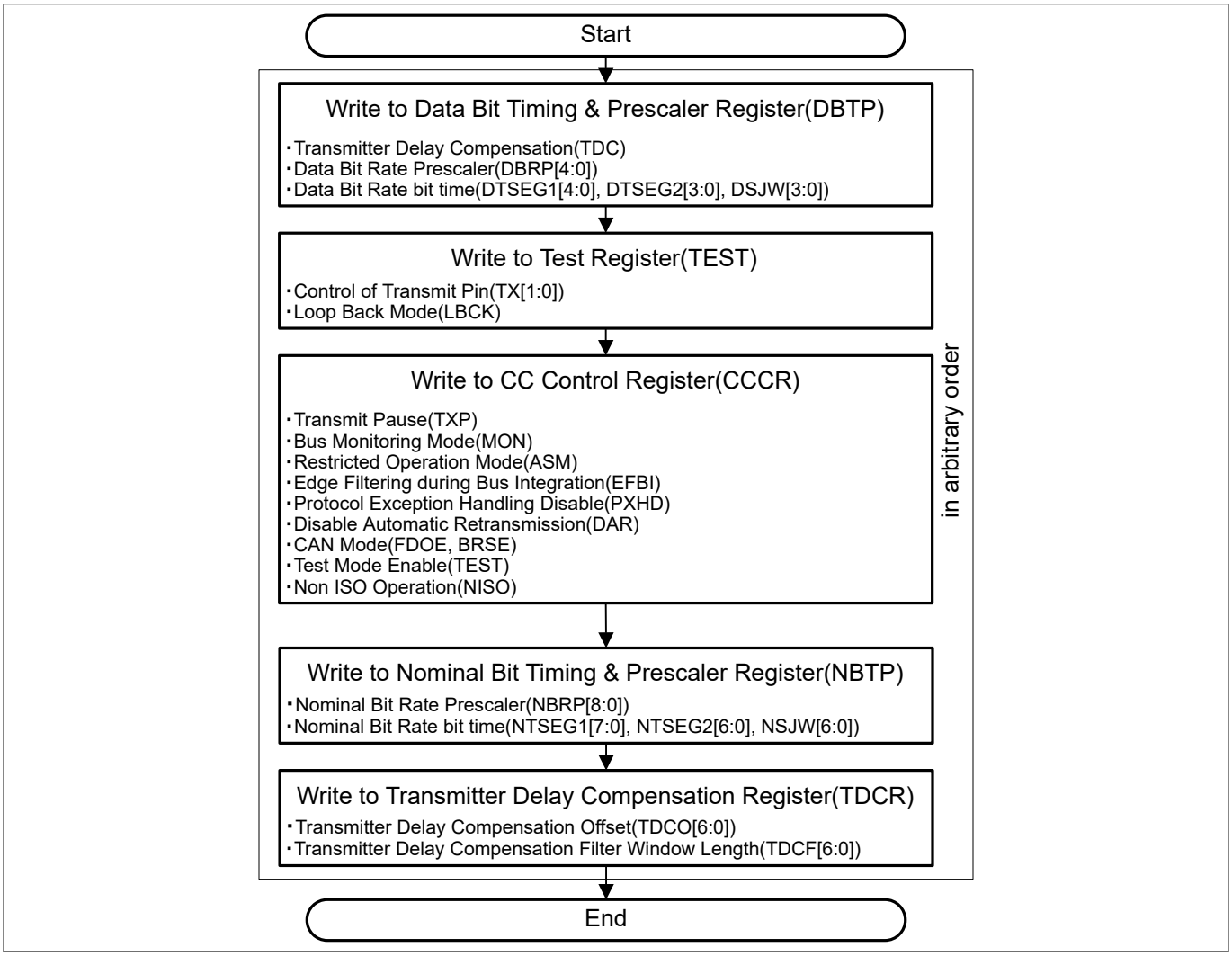


Figure 120 Configuration required for CAN bus

25.6.7 Message RAM configuration

Figure 121 shows an overview of the message RAM configuration.

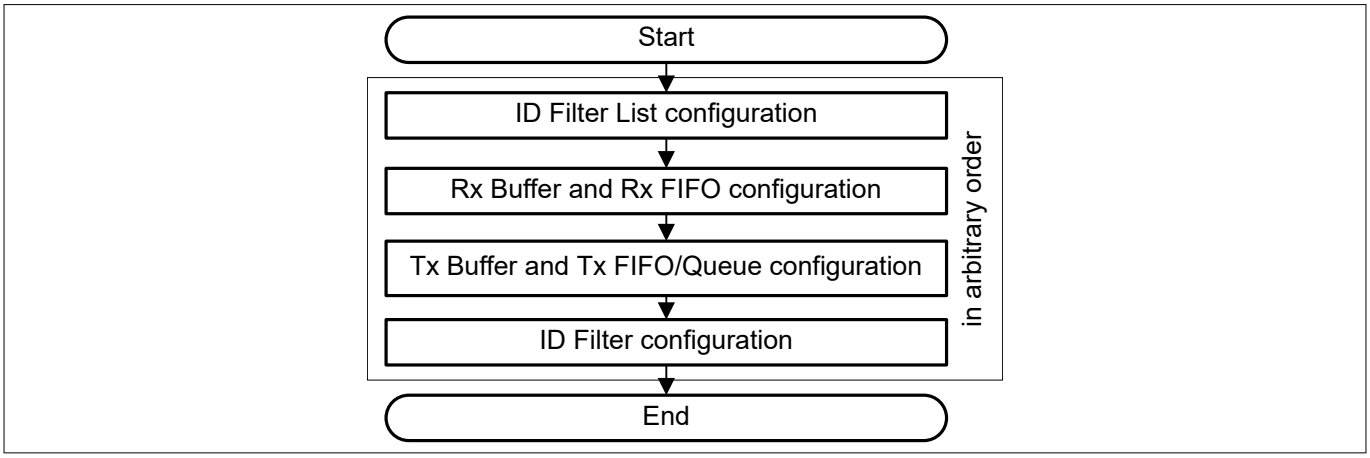


Figure 121 Message RAM configuration overview

Each configuration mentioned in the overview is detailed in the following figures.

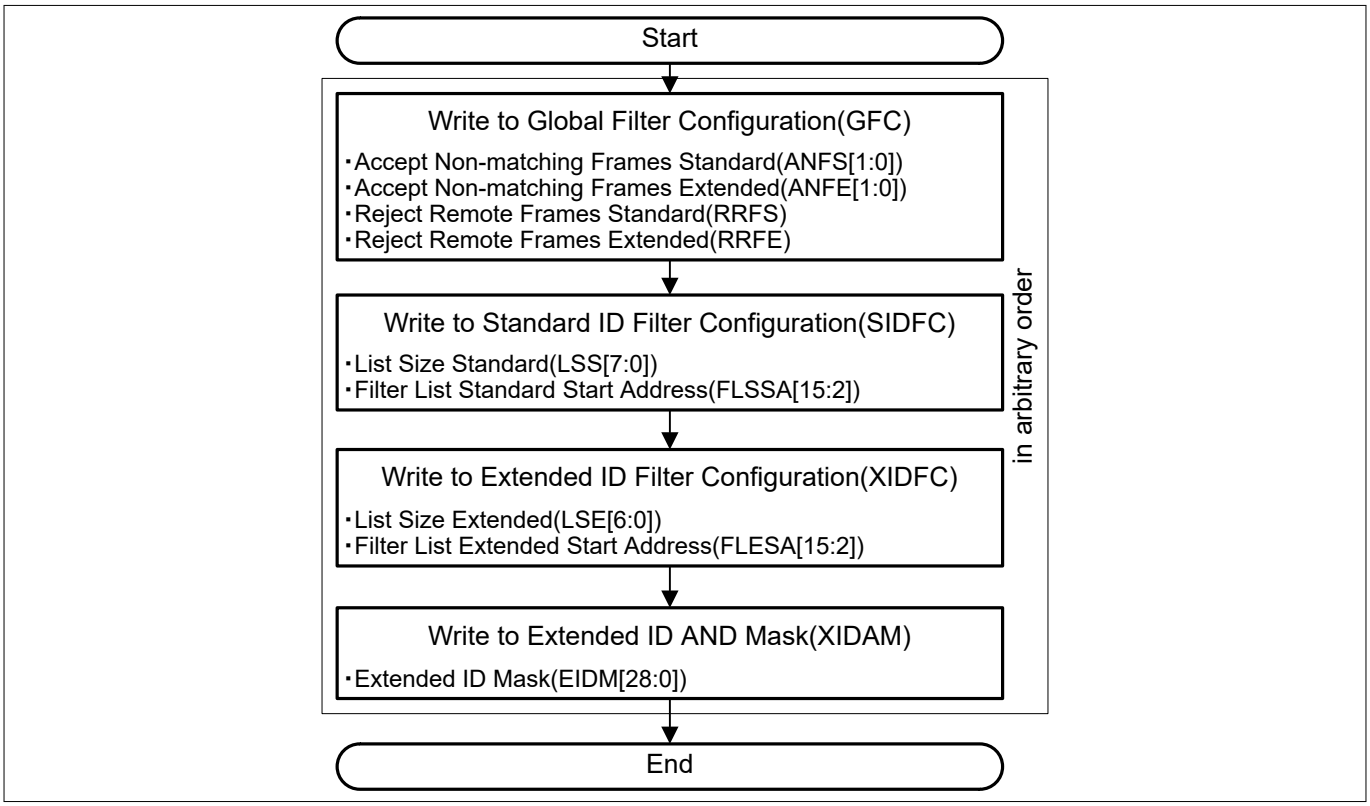


Figure 122 ID filter list configuration

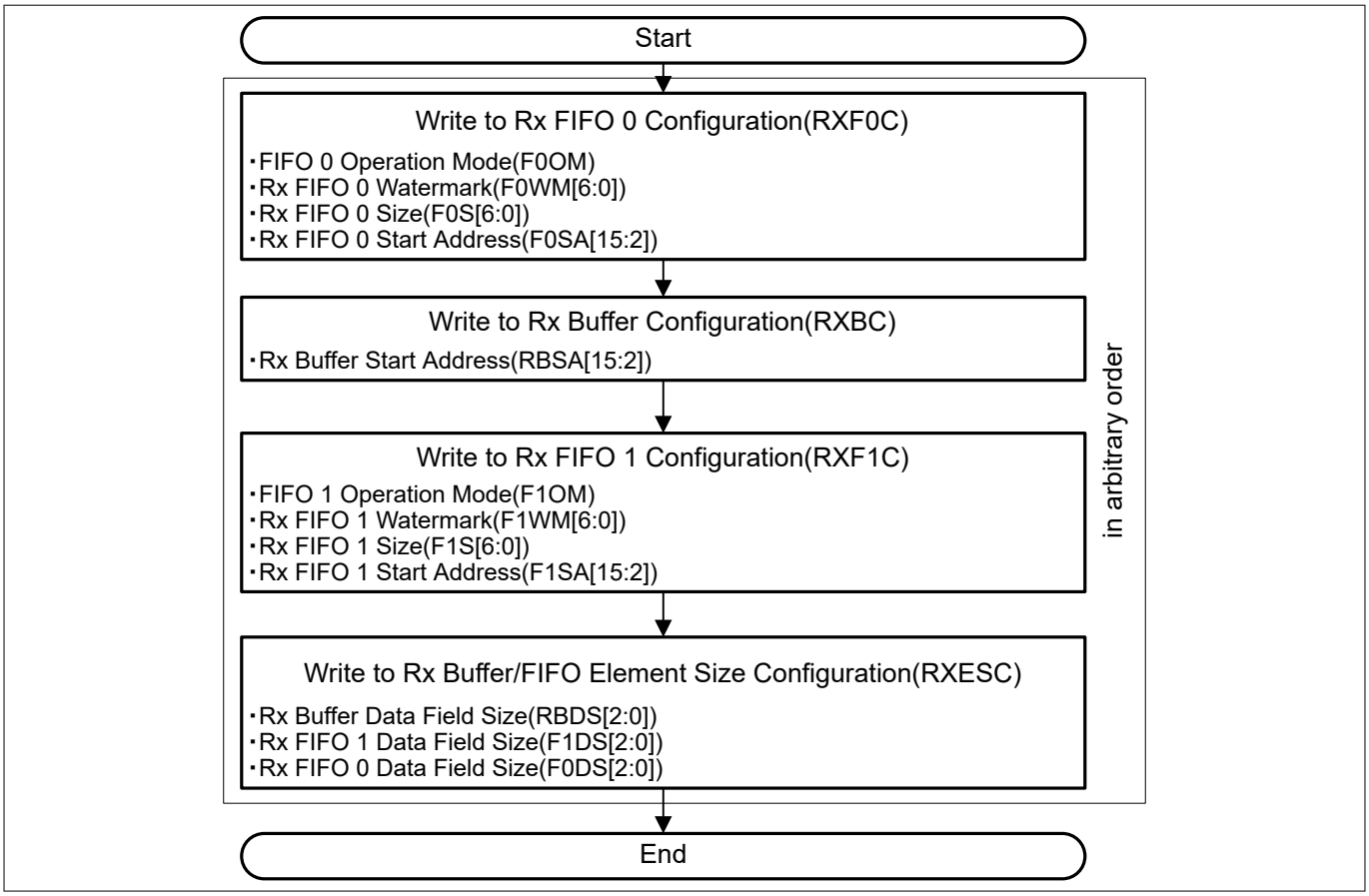


Figure 123 RX FIFO and RX buffer configuration

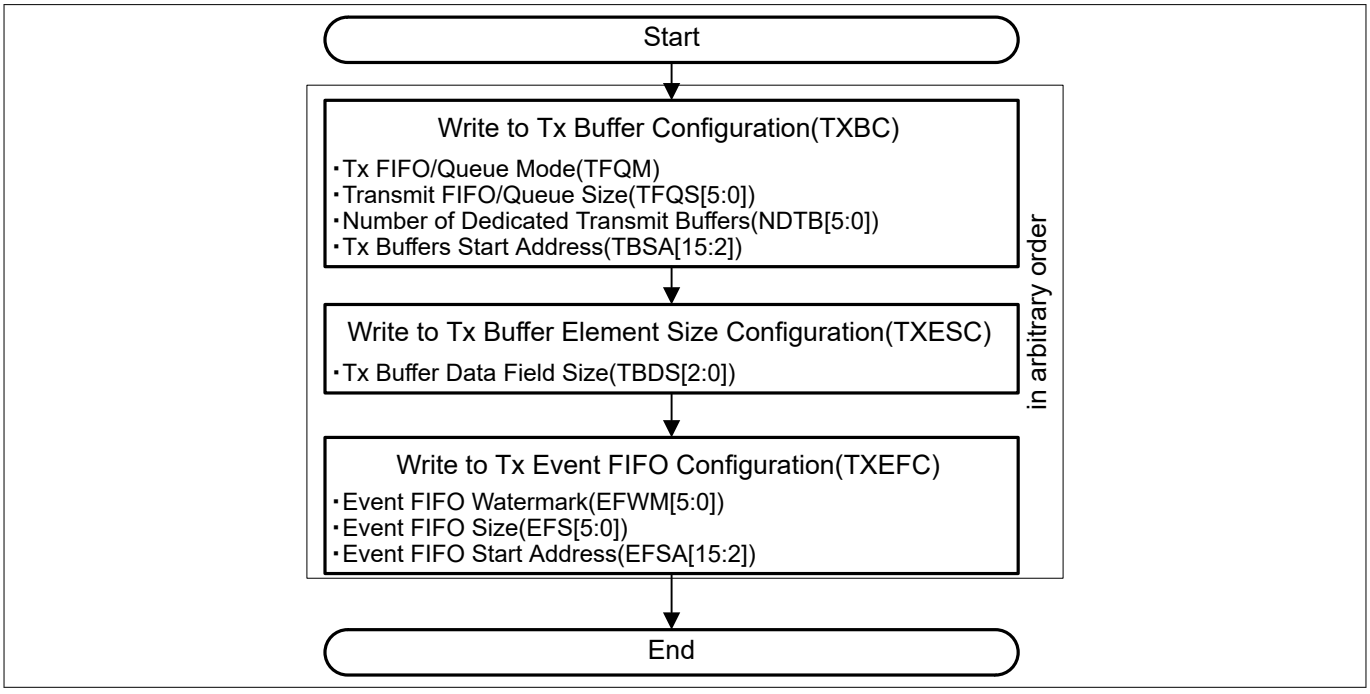


Figure 124 TX Buffer and TX FIFO/Queue configuration

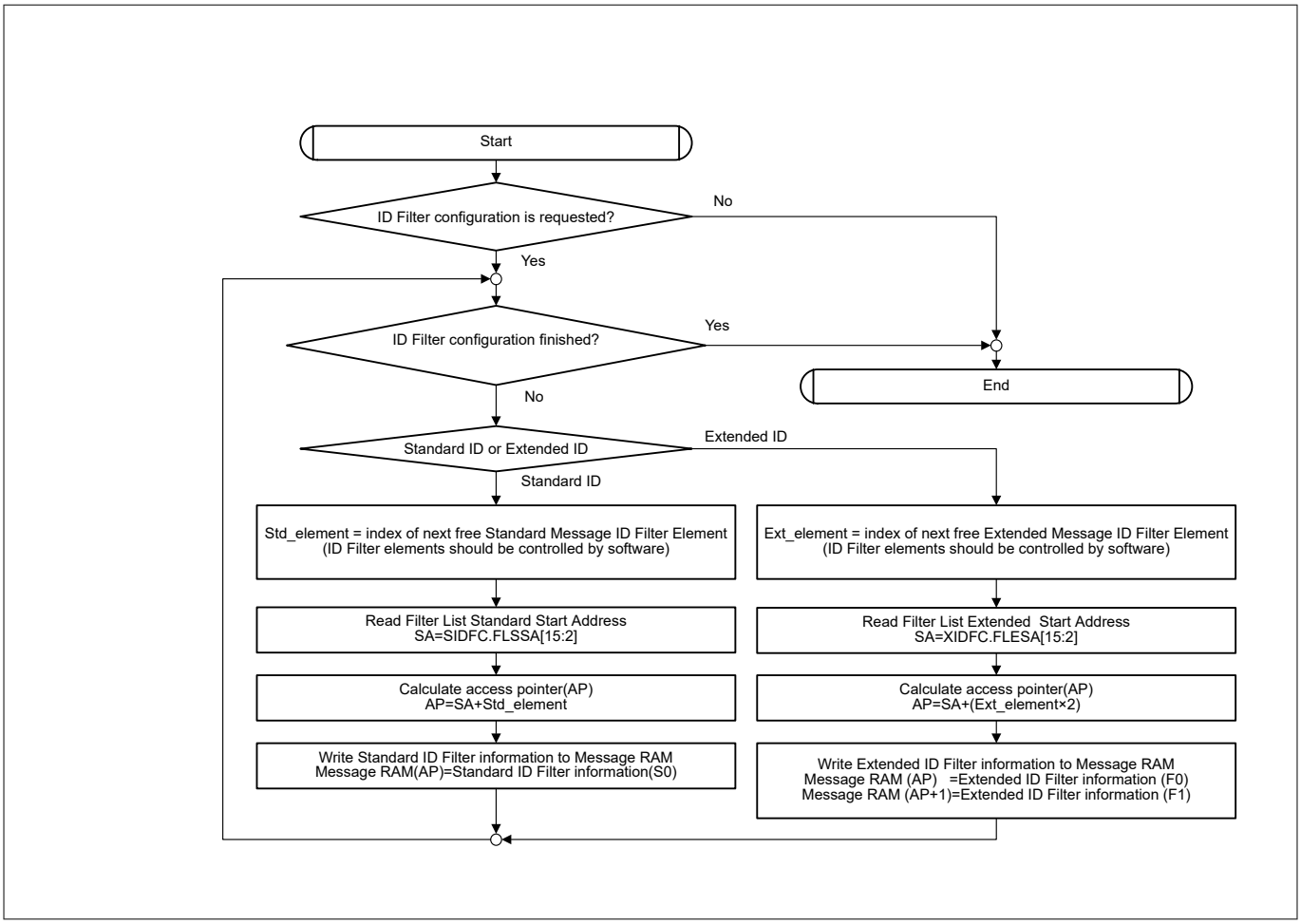


Figure 125 ID filter configuration

25.6.8 Interrupt configuration

Figure 126 shows the general interrupt configuration flow.

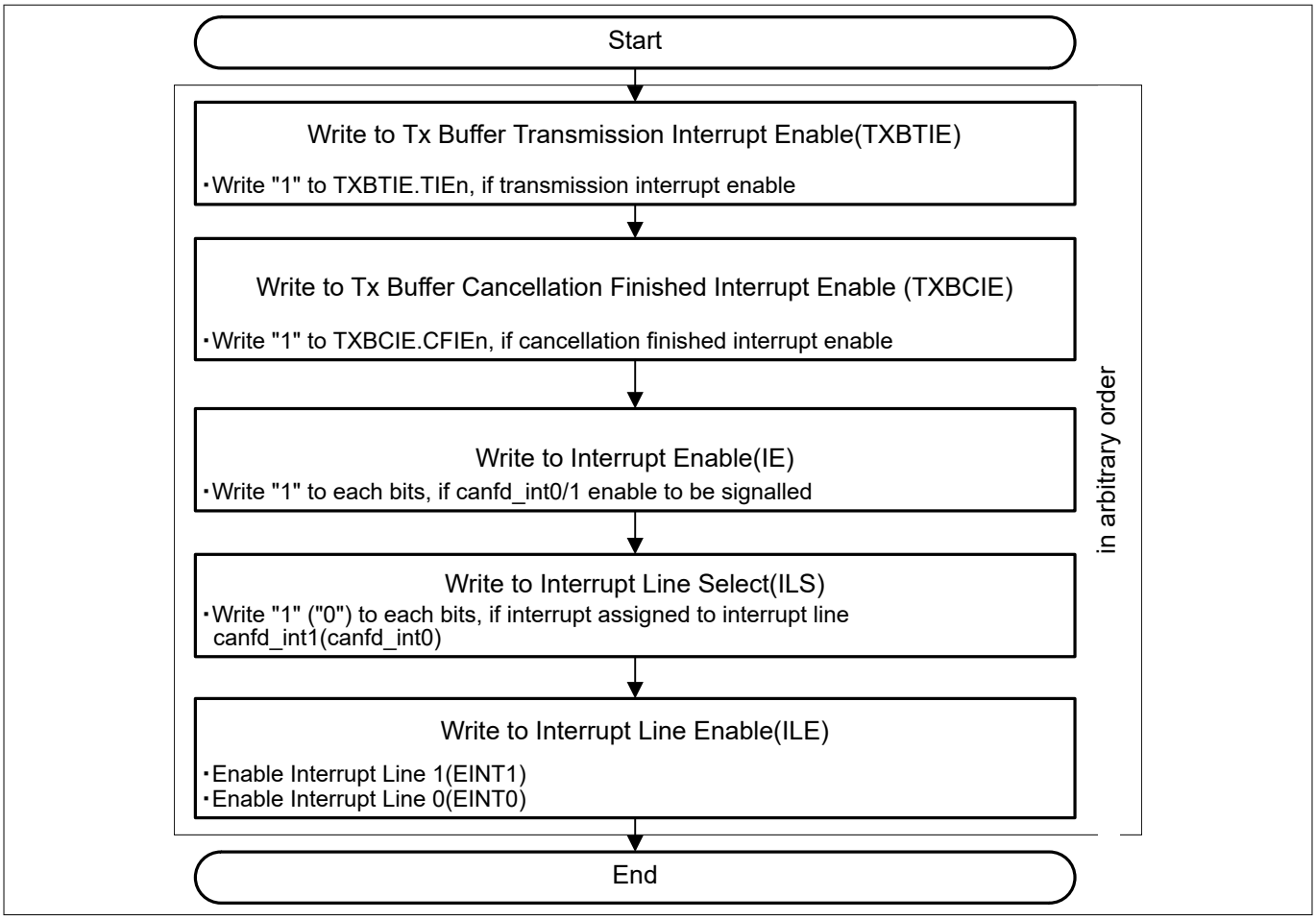


Figure 126 Interrupt configuration

25.6.9 Transmit frame configuration

Figure 127 shows the transmit frame configuration.

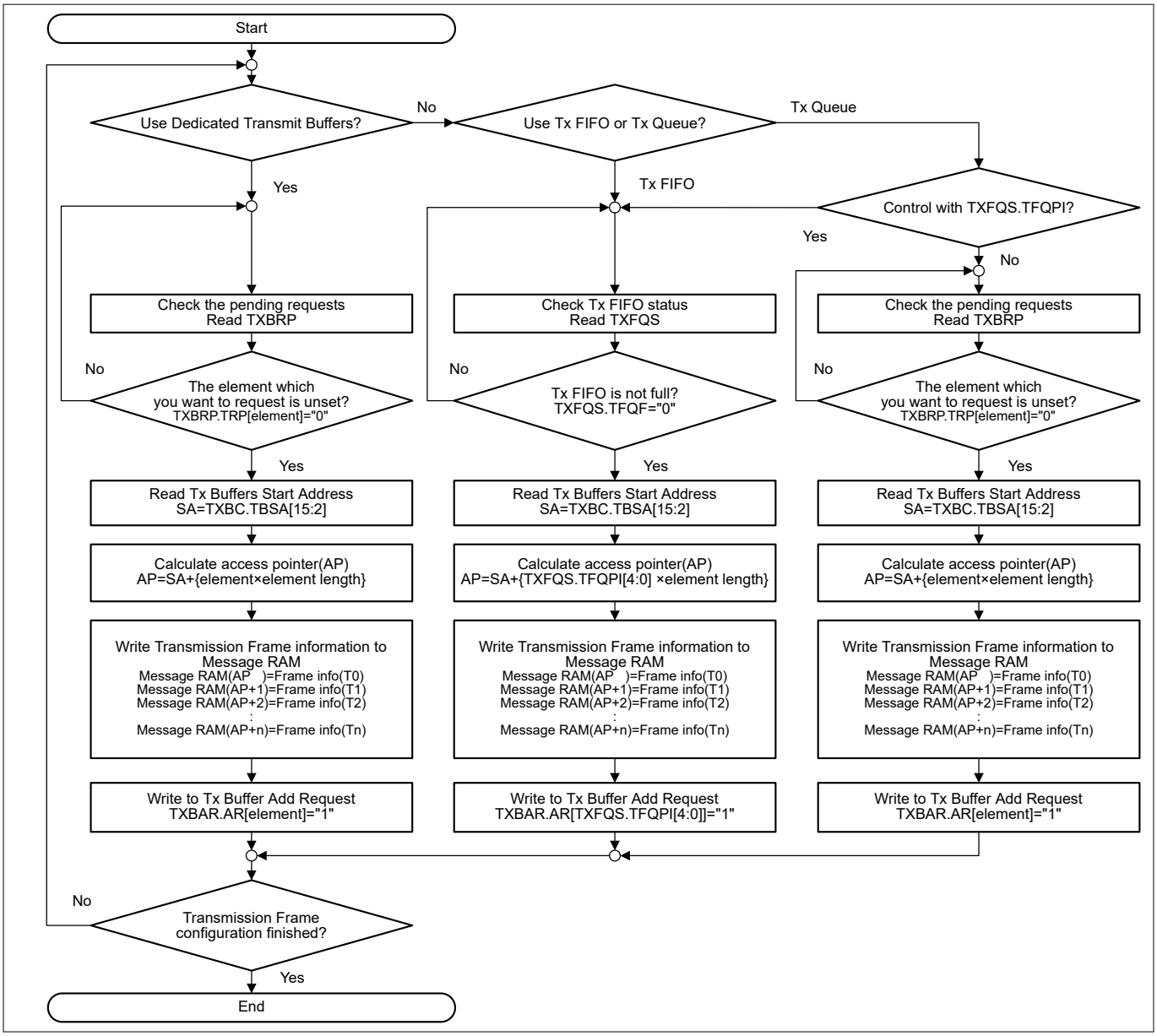


Figure 127 Transmit frame configuration

25.6.10 Interrupt handling

Figure 128 shows a general interrupt handling flow chart.

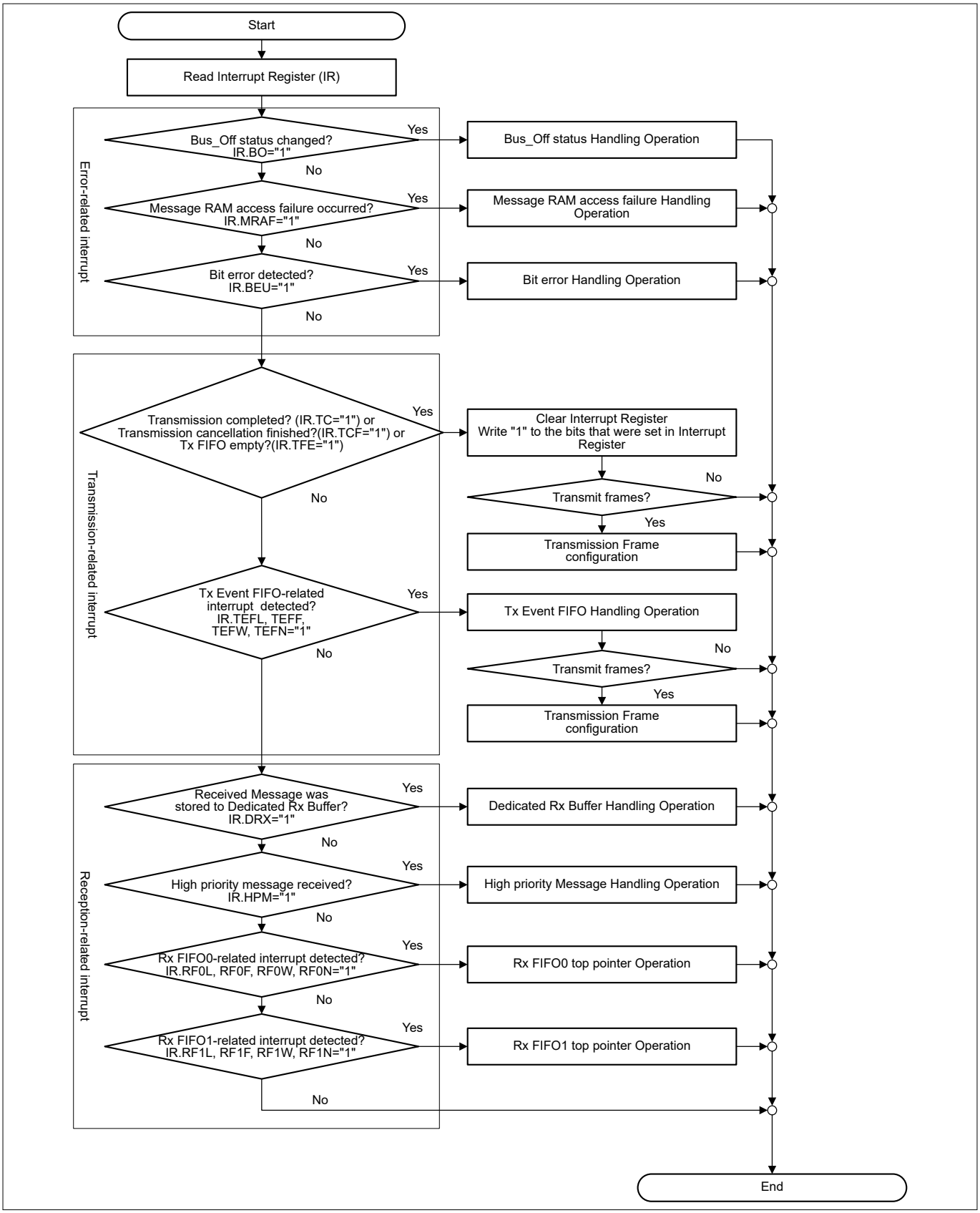


Figure 128 Interrupt handling

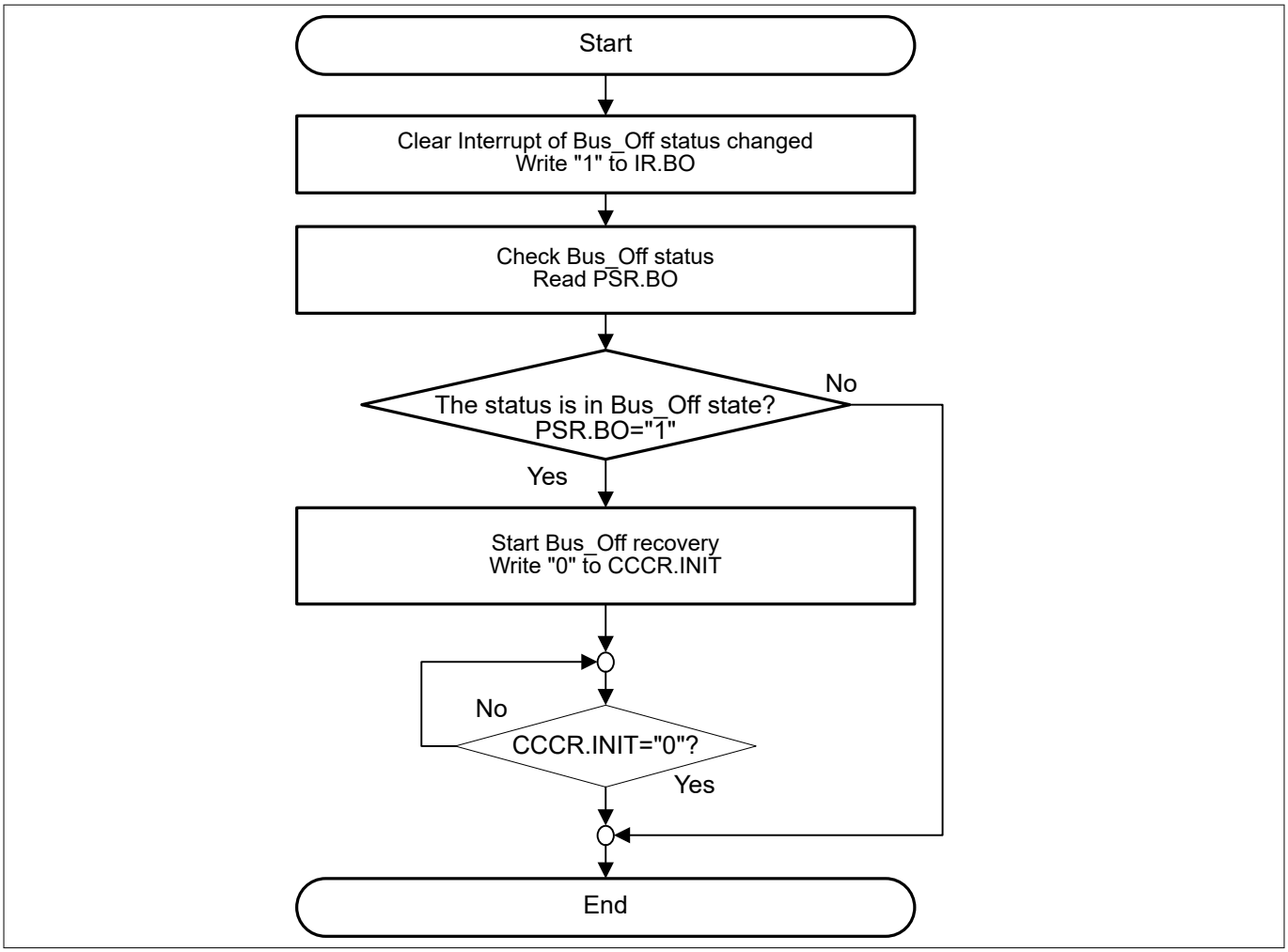


Figure 129 Bus OFF error handling

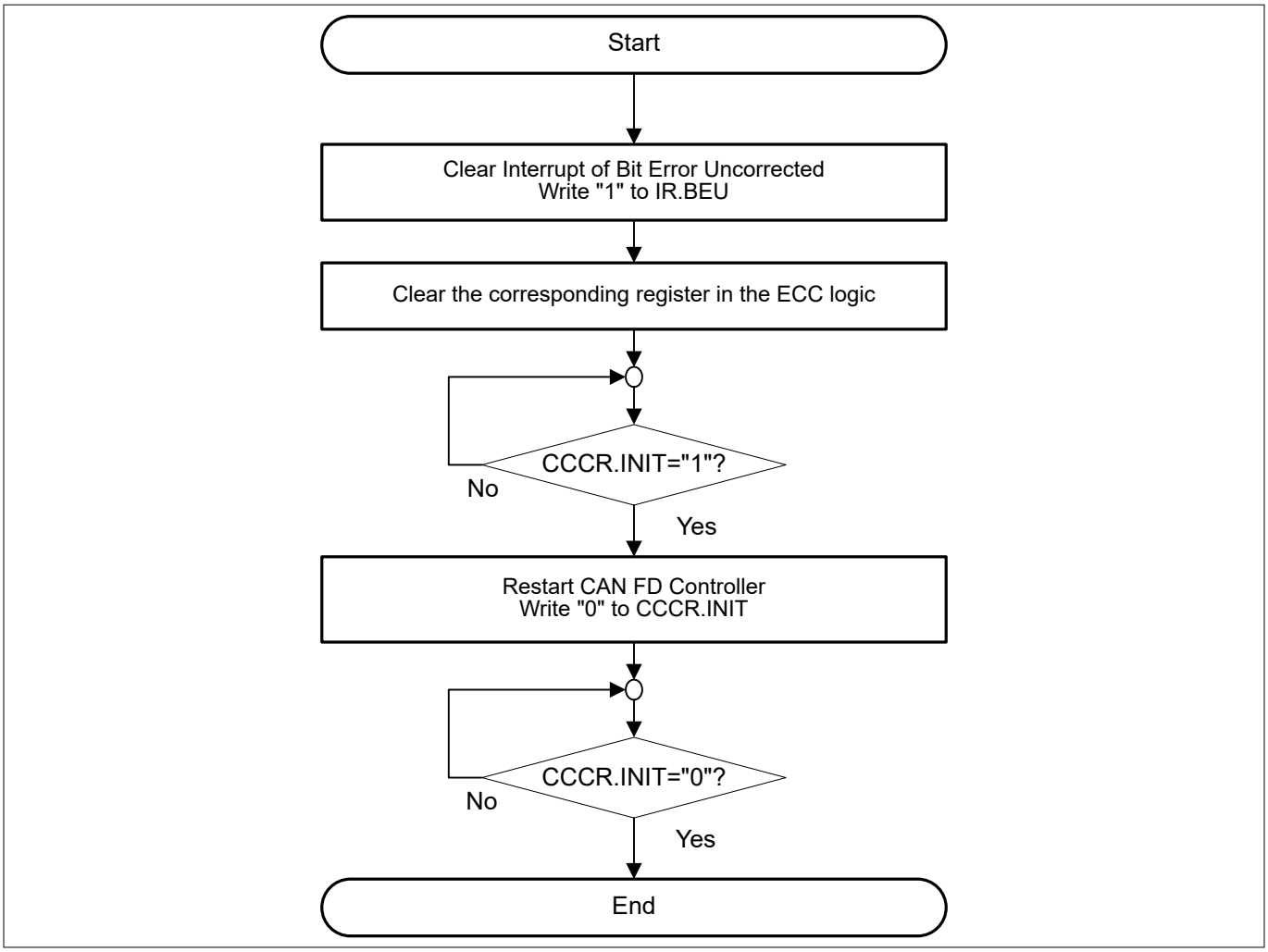


Figure 130 Message RAM access failure handling

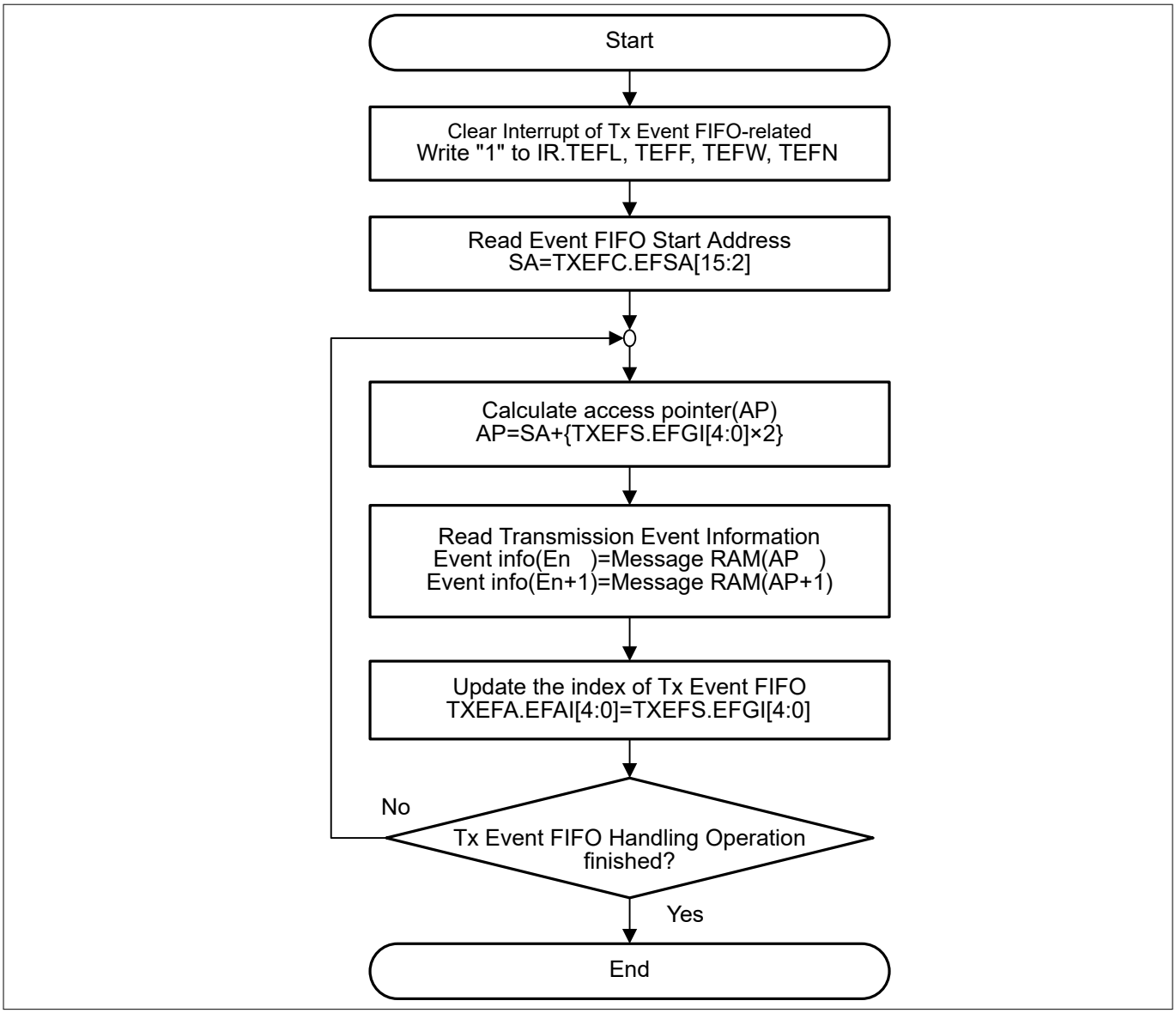


Figure 131 TX Event FIFO handling

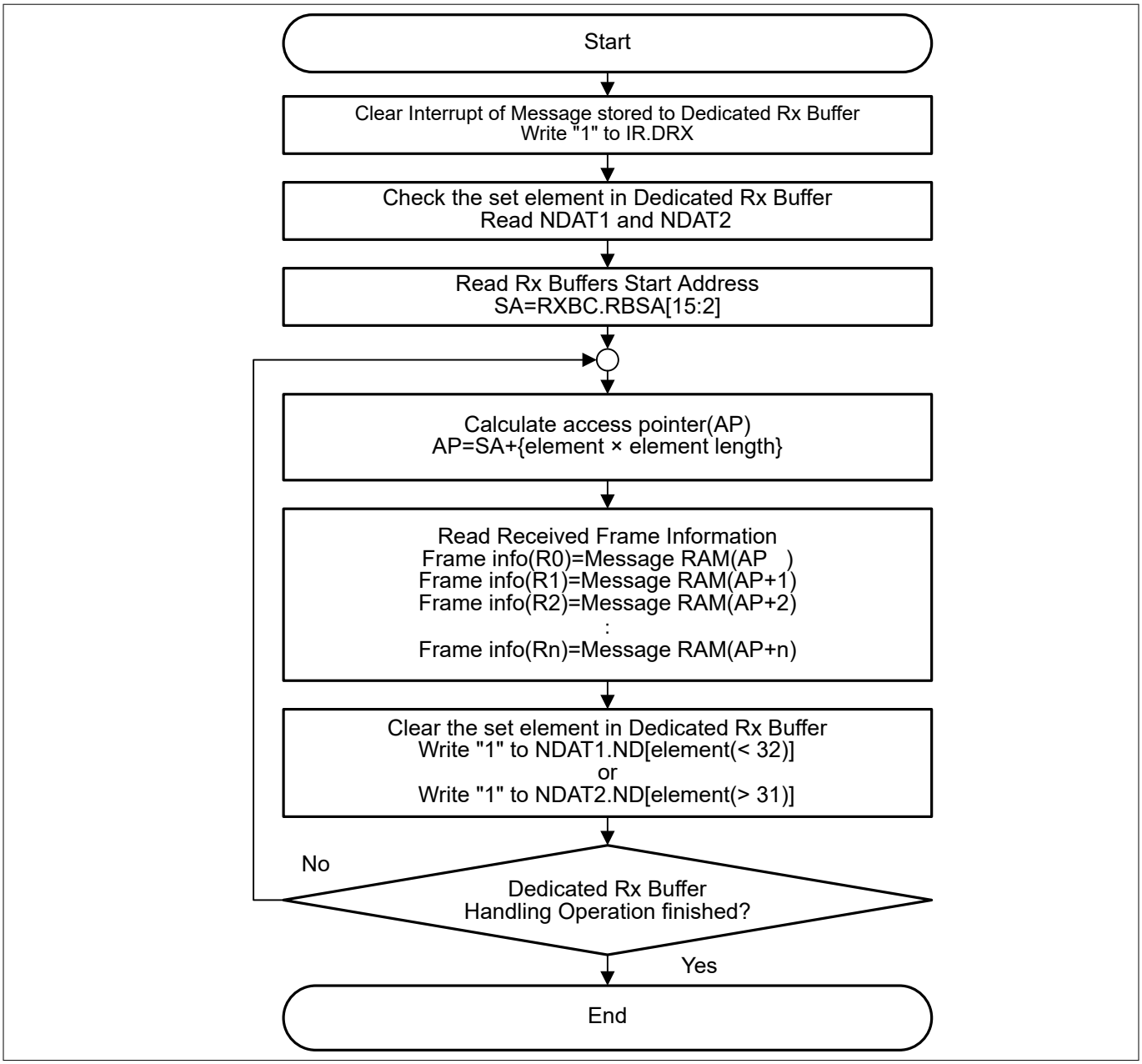


Figure 132 Dedicated RX buffer handling

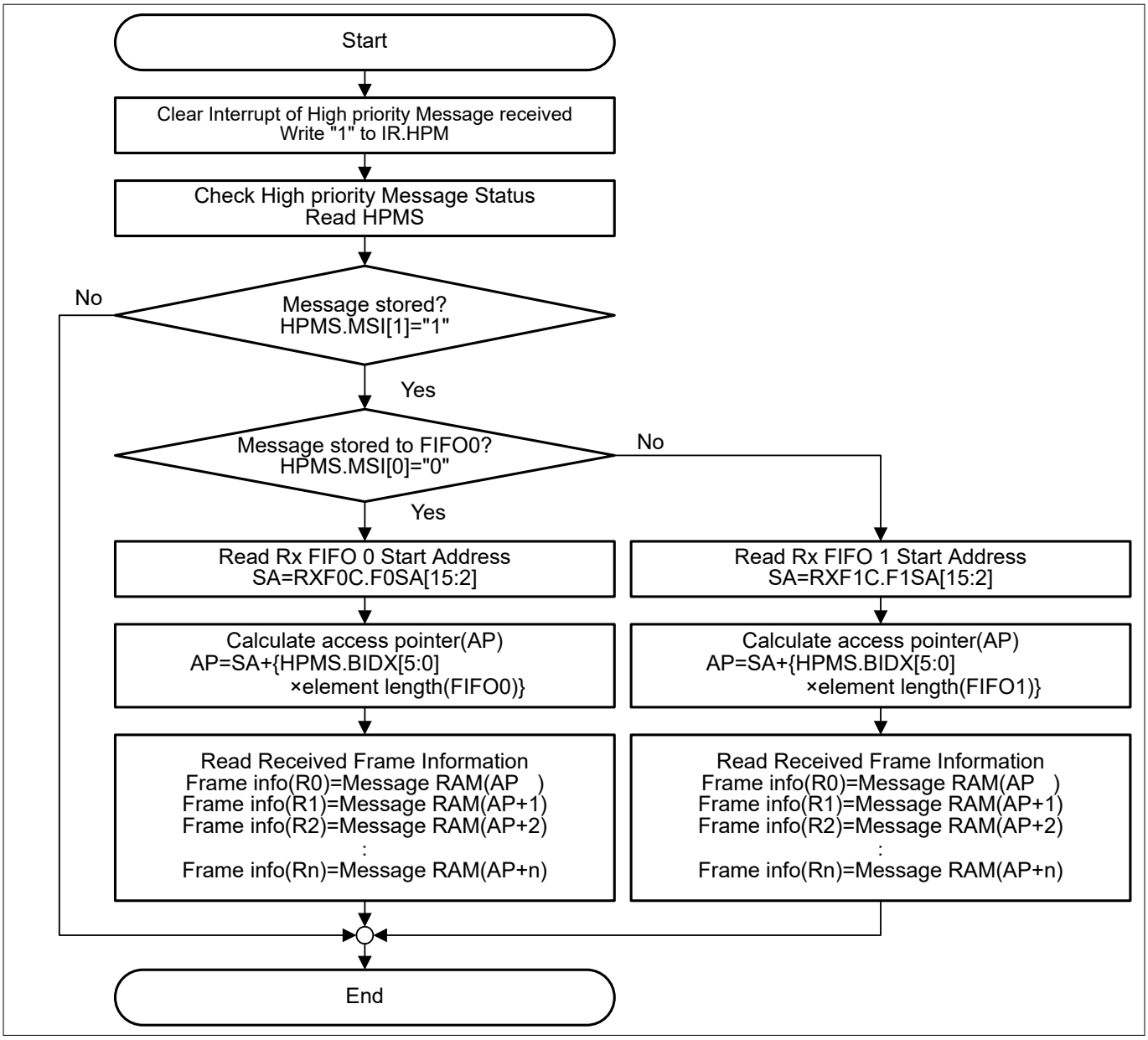


Figure 133 High-priority message handling

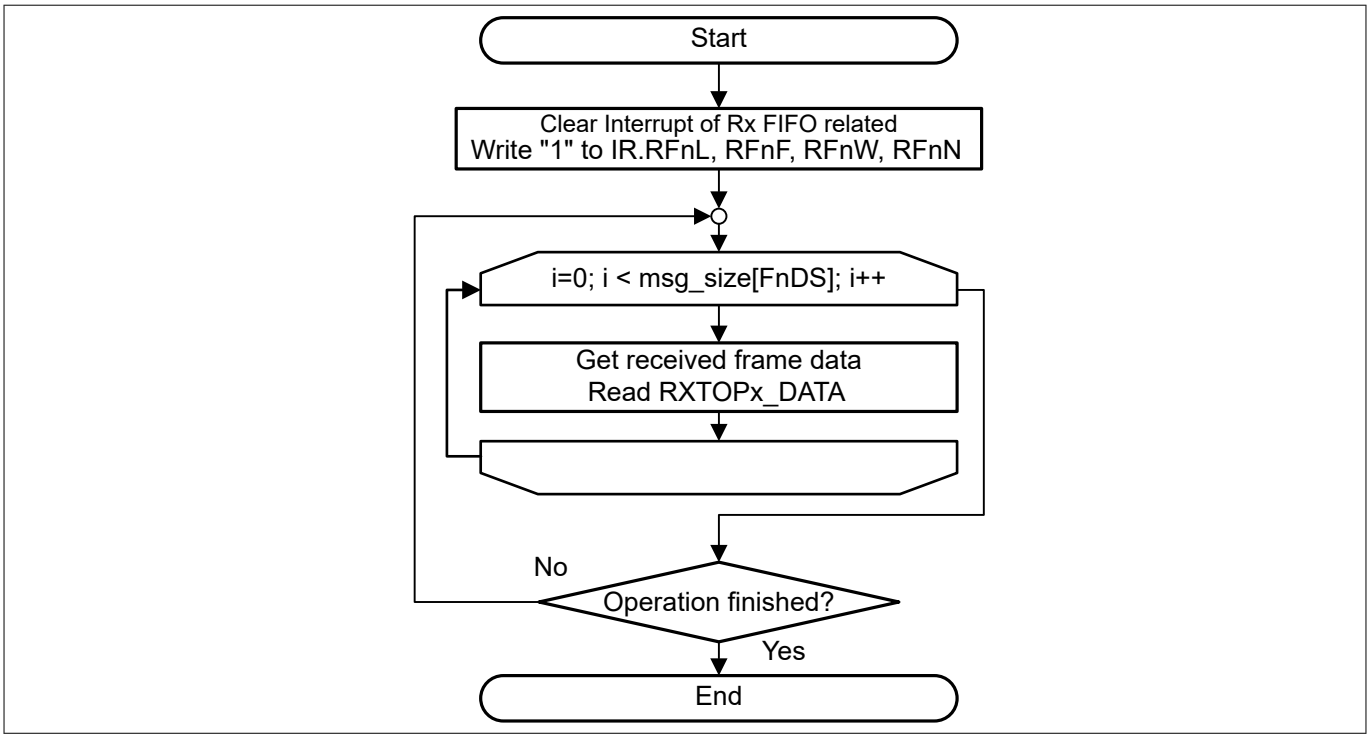


Figure 134 RX FIFO top pointer handling

25.7 Registers summary

For more details on the registers, see the PSOC™ Control C3M/P8, C3M/P7 registers RM.

26 Serial Communication Block (SCB)

Each SCB supports the following three serial communication protocols: Serial Peripheral Interface (SPI), universal asynchronous receiver transmitter (UART), and inter-integrated circuit (I2C). An SCB supports only one of the protocols at any given time. PSC3M/P8, PSC3M/P7 MCUs have several SCBs. One of them supports only I2C slave mode and SPI slave mode in the system Deep Sleep power mode.

26.1 Features

The SCB supports the following features:

- Standard SPI master and slave functionality with Motorola, Texas Instruments, and National Semiconductor protocols
- Standard UART transmitter and receiver function with SmartCard interface (ISO7816), IrDA protocol, and local interconnect network (LIN) submodes
- Supports UART half-duplex
- UART flow control (CTS input and RTS output)
- Internal clock input with programmable oversampling
- Standard I2C master and slave functionality
- Each SCB has a 256 byte FIFO for receive (Rx) and transmit (Tx)
- EZ mode for SPI and I2C slaves; allows operation without CPU intervention
- CMD_RESP mode for SPI and I2C slaves; allows operation without CPU intervention and is available only on system Deep Sleep-capable SCB
- System Deep Sleep mode of operation for SPI and I2C slaves (using external clocking), available only on system Deep Sleep-capable SCB
- System Deep Sleep wake-up on I2C slave address match or SPI slave selection, available only on system Deep Sleep-capable SCB
- SPI Master/slave support till 25 MHz, Fast SPI support on two SCBs upto 50 MHz
- Trigger outputs for connection to DMA
- Multiple interrupt sources to indicate the status of FIFOs and transfers
- Local loop-back control
- Timer added for timeout generation to support SMBUS/PMBUS
- Multiple slave address added to support SMBUS/PMBUS

26.2 Architecture

26.2.1 Trigger interface

The trigger interface provides the status information on the TX FIFO (“tr_tx_req”) and RX FIFO (“tr_rx_req”).

- **DMA trigger signals**: “tr_tx_req” and “tr_rx_req” signals are typically connected to the DW controller
  - “tr_tx_req” indicates that the TX FIFO can accept a data element for transmission
  - “tr_rx_req” indicates that the RX FIFO can provide a (received) data element
- **tr_i2c_scl_filtered signals**: The “tr_i2c_scl_filtered” signal is connected to the SCL input analog filter output. This removes the glitches
- **Serial protocol interfaces**: UART, I2C, and SPI interface signals connect to the High-Speed I/O Matrix (HSIOM) in the IO Sub-System (IOSS)

26.2.2 Clocking modes

An SCB can be clocked either by an internal clock provided by the peripheral clock dividers (clk_scb), or it can be clocked by an external master.

- UART, SPI master, and I2C master modes must use clk_scb
- Only the 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 SCB CTRL register:

- EC_AM_MODE indicates whether SPI slave selection or I2C address matching is internally (‘0’) or externally (‘1’) clocked
- EC_OP_MODE indicates whether the rest of the protocol operation (besides SPI slave selection and I2C address matching) is internally (‘0’) or externally (‘1’) clocked

***Note**: FIFO mode is NOT supported with externally clocked operation (SCBx_CTRL.EC_AM_MODE is ‘0’ or ‘1’); however, data transfer must be done with internal clocking (SCBx_CTRL.EC_OP_MODE is ‘0’).*

- 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), SCBx_CTRL.EC_ACCESS should be set to ‘0’.

Table 207 provides an overview of the clocking and buffer modes supported for each communication mode.

Table 207 Clock mode compatibility

	Internally clocked (IC)			Externally clocked (EC) (Deep Sleep SCB only)		
	FIFO	**EZ**	**CMD_RESP**	**FIFO**	**EZ**	**CMD_RESP**
I2C master	Yes	No	No	No	No	No
I2C slave	Yes	Yes	No	No[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	No[2)]	Yes	Yes
UART transmitter	Yes	No	No	No	No	No
UART receiver	Yes	No	No	No	No	No

1) In Deep Sleep, I2C slave FIFO mode the external-clocked logic can handle slave address matching. It then triggers an interrupt to wake up the CPU.
2) In Deep Sleep, SPI slave FIFO mode the external-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 applies only to the Deep Sleep-capable SCB.

Table 208 I2C slave and SPI 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
CMD_RESP mode	No	No	Yes

26.2.3 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 sections on each protocol provide more details.

- Masters can only use FIFO mode
- I2C and SPI slaves can use all three modes

***Note**: CMD_RESP Mode is available only on the system Deep Sleep-capable SCB.*

- UART only uses FIFO mode

26.2.3.1 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 × 128 elements or 16 bits × 64 elements; this is done by setting the `SCBx_CTRL.MEM_WIDTH` register.

FIFO operation mode is available only in the CPU Active and system Sleep modes. However, the I2C address or SPI slave select can be used to wake the device from system Deep Sleep on the system Deep Sleep-capable SCB. The status is provided for both the RX and TX FIFOs. There are multiple interrupt sources available, which indicate the status of the FIFOs, such as full or empty; see SCB interrupts.

26.2.3.2 EZ mode

In EZ mode, the RAM is used as a single 256-byte buffer. Both external and internal 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.

EZ mode is available in CPU Active and system Deep Sleep power modes.

26.2.3.3 CMD RESP mode

Command Response (CMD_RESP) mode is similar to the 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 system Deep Sleep-capable SCB.

CMD_RESP mode operation is available in the CPU Active and system Deep Sleep power modes.

26.2.4 Block operation

26.2.4.1 Block enable and recovery

The SCB block (CTRL.ENABLED) can be enabled or disabled by setting the register value to '1' or '0', respectively. To recover the SCB block from unexpected states, it can be reenabled with the `CTRL.ENABLED` register to perform a soft reset of the hardware block and registers. All registers in an ACTIVE power domain are reset when `CTRL.ENABLED` is '0'.

26.2.4.2 Initialization flow after reset

Do the following to initialize the SCB:

1. Program the protocol-specific information using the `SPI_CTRL`, `UART_CTRL` (and `UART_TX_CTRL` and `UART_RX_CTRL`) or `I2C_CTRL` registers. This includes the selection of a submode, master/slave functionality, and transmitter/receiver functionality when applicable
2. Program the generic transmitter (TX_CTRL) and receiver (RX_CTRL) information. This includes enabling the transmitter and receiver functionality
3. Program the transmitter FIFO (TX_FIFO_CTRL) and receiver FIFO (RX_FIFO_CTRL) information
4. Program the CTRL register to enable the SCB. Select the specific operation mode and oversampling factor

26.2.5 Power modes

The SCBs can operate in CPU Active and CPU Sleep power modes. Only SCB[0] can operate in the System Deep Sleep power mode. For more information, see the Clocking modes section.

CPU Active to System Deep Sleep power mode transition

Master mode: The master controls the communication, which requests a low-power state or wake-up. In both cases the communication, transition and low-power state are synced.

Slave mode: Transitioning to System Deep Sleep when SCB is configured as Slave and has no control over the external transmitter and in System Deep Sleep, an external Master starts transmitting. It is up to customer software or upper communication protocol to ensure that communication frames are synced with low-power states and the required transition period.

For slave mode, several boundary conditions need to be considered. On an upper level communication protocol, there is usually a command, which requests a low-power state or wake-up. In both cases the communication, transition and low-power state are synced and the limitation is not relevant.

If there is no upper level protocol and power states are asynchronous to the master communication, it is up to the customer code how the expected behavior on both sides can be achieved. For example, for read access from the master, the slave is aware that the data is still in the FIFO. Write access from the master, during transition between power states might be lost.

***Notes**:*

1. *During the CPU Active to System Deep Sleep transition in an externally-clocked mode, any transactions that are sent to the slave during the transition period are lost*
2. *During the CPU Active to System Deep Sleep transition in an internally-clocked mode, there is no automatic clock source switching and no transfers are available in System Deep Sleep mode*

26.3 Serial peripheral interface (SPI)

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

26.3.1 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 three 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) (for Motorola mode only)
  - Inter-data frame deselect time (select inactive to select active) (for Motorola mode only)
- Parity support (odd and even parity) (for Motorola mode only)
- 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
- Fast SPI operation up to 50 MHz (SCB[5] and SCB[6] only)

26.3.2 General description

Figure 135 illustrates an example of a SPI master with three slaves.

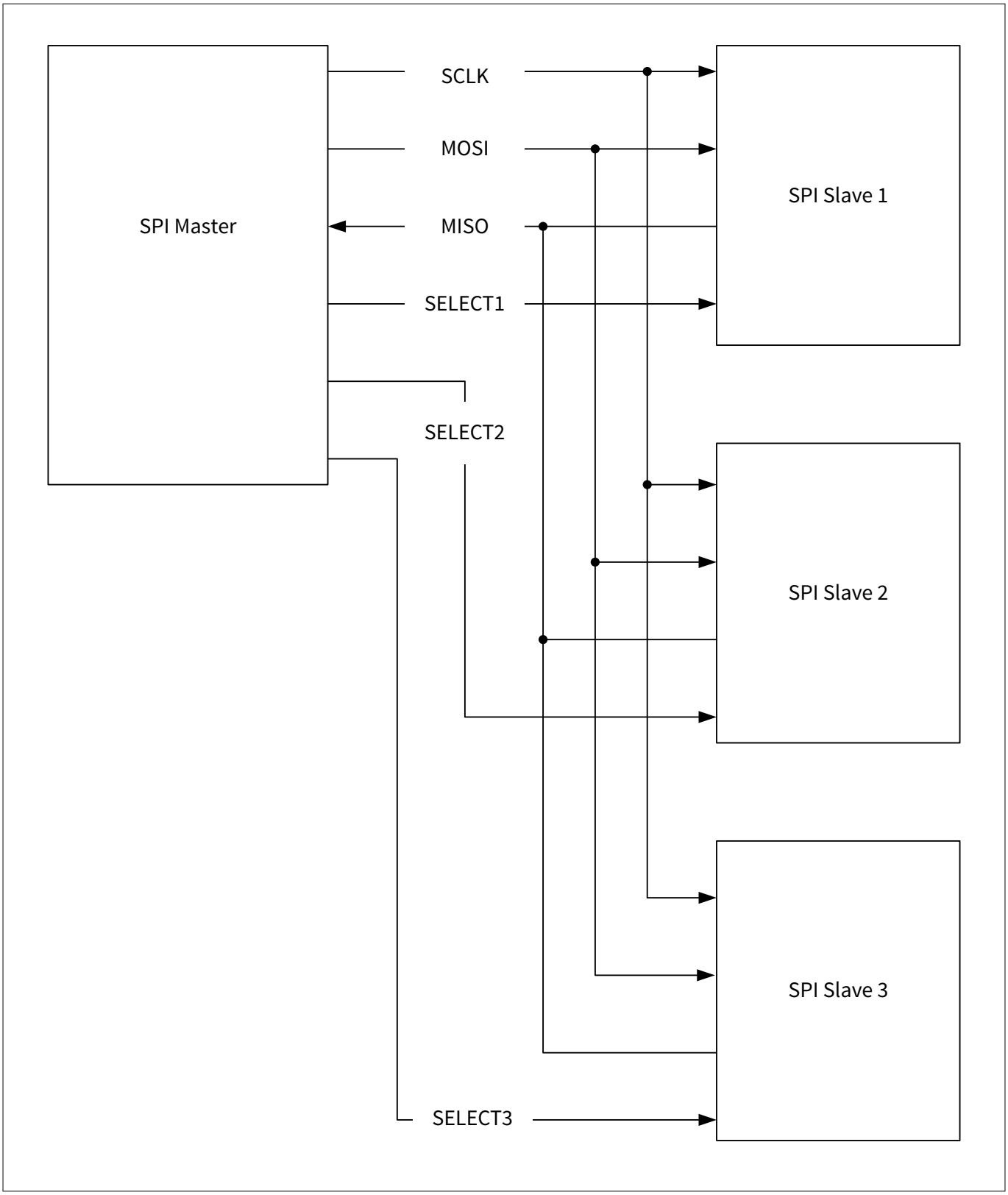


Figure 135 SPI example

A standard SPI interface consists of the following four signals:

- 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 the SELECT line. Then, it drives the 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 the data on the MISO line, which is captured by the master

The data frame size can be configured to any value in the range of 4 bits to 32 bits. The serial data can be transmitted either most significant bit (MSb) first or the 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 an IOSS/GPIO configuration

26.3.3 SPI modes of operation

26.3.3.1 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.

The 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 136 illustrates the driving and capturing of the MOSI/MISO data as a function of SCBx_SPI_CTRL.CPOL and SCBx_SPI_CTRL.CPHA.

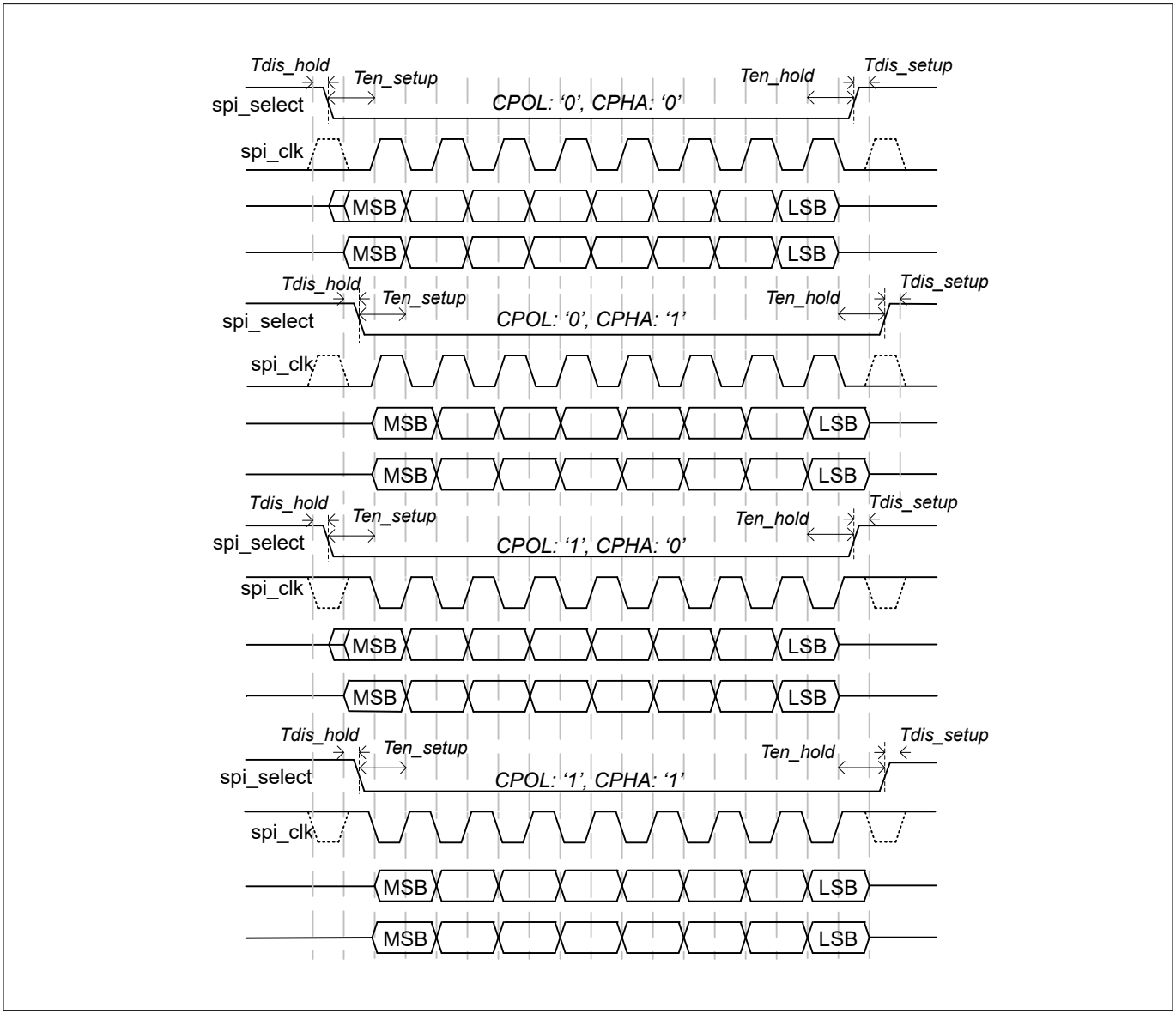


Figure 136 SPI Motorola, four modes

Figure 137 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').

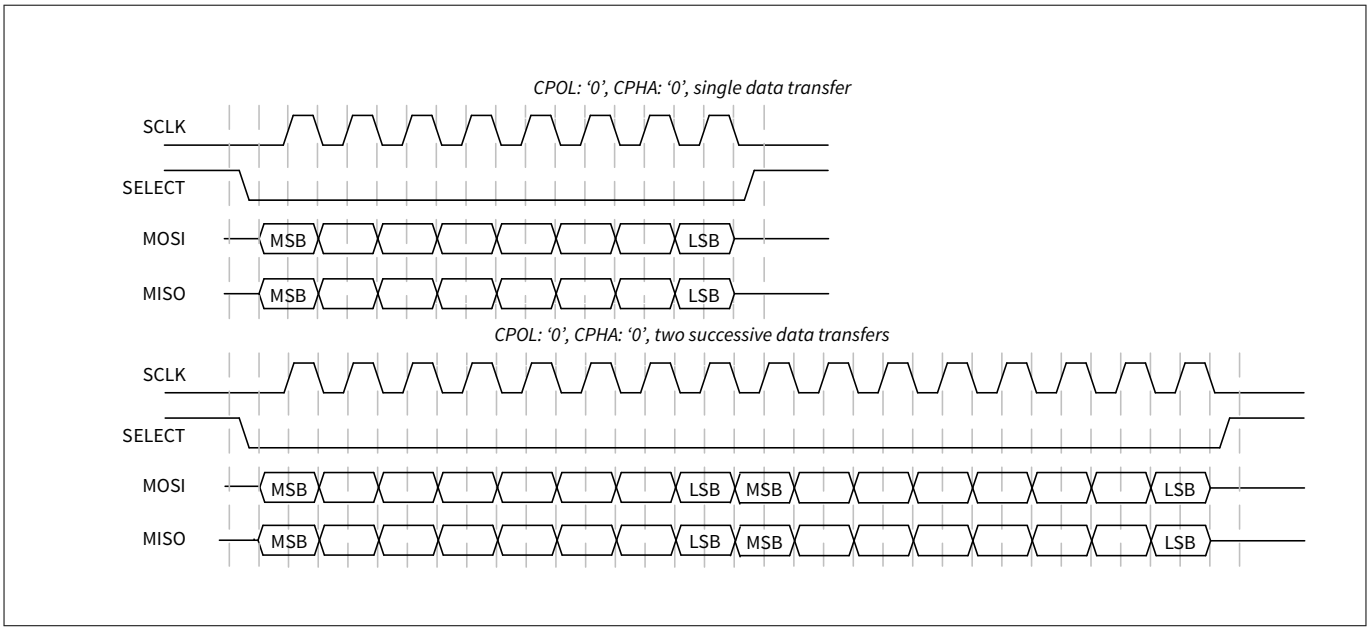


Figure 137 SPI Motorola data transfer example

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_SPI_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 the Enabling and initializing SPI section

For more information on these registers, see the PSOC™ Control C3M/P8, C3M/P7 registers reference manual.

26.3.3.2 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, the slave devices do not have to track 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 138 illustrates a single 8-bit data transfer and two successive 8-bit data transfers. The SELECT pulse precedes the first data bit. Note that how the SELECT pulse of the second data transfer coincides with the last data bit of the first data transfer.

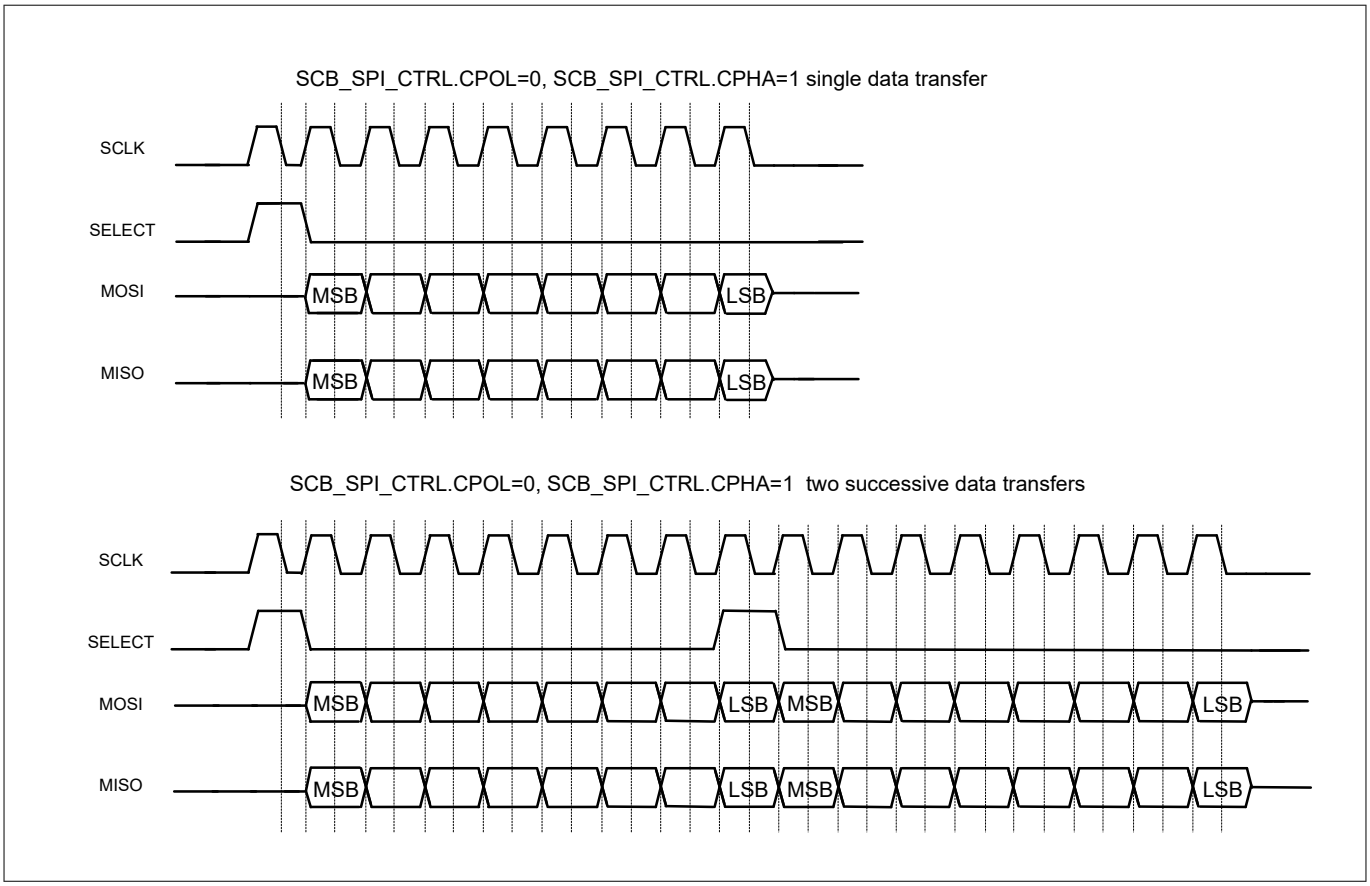


Figure 138 SPI TI data transfer example

Figure 139 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.

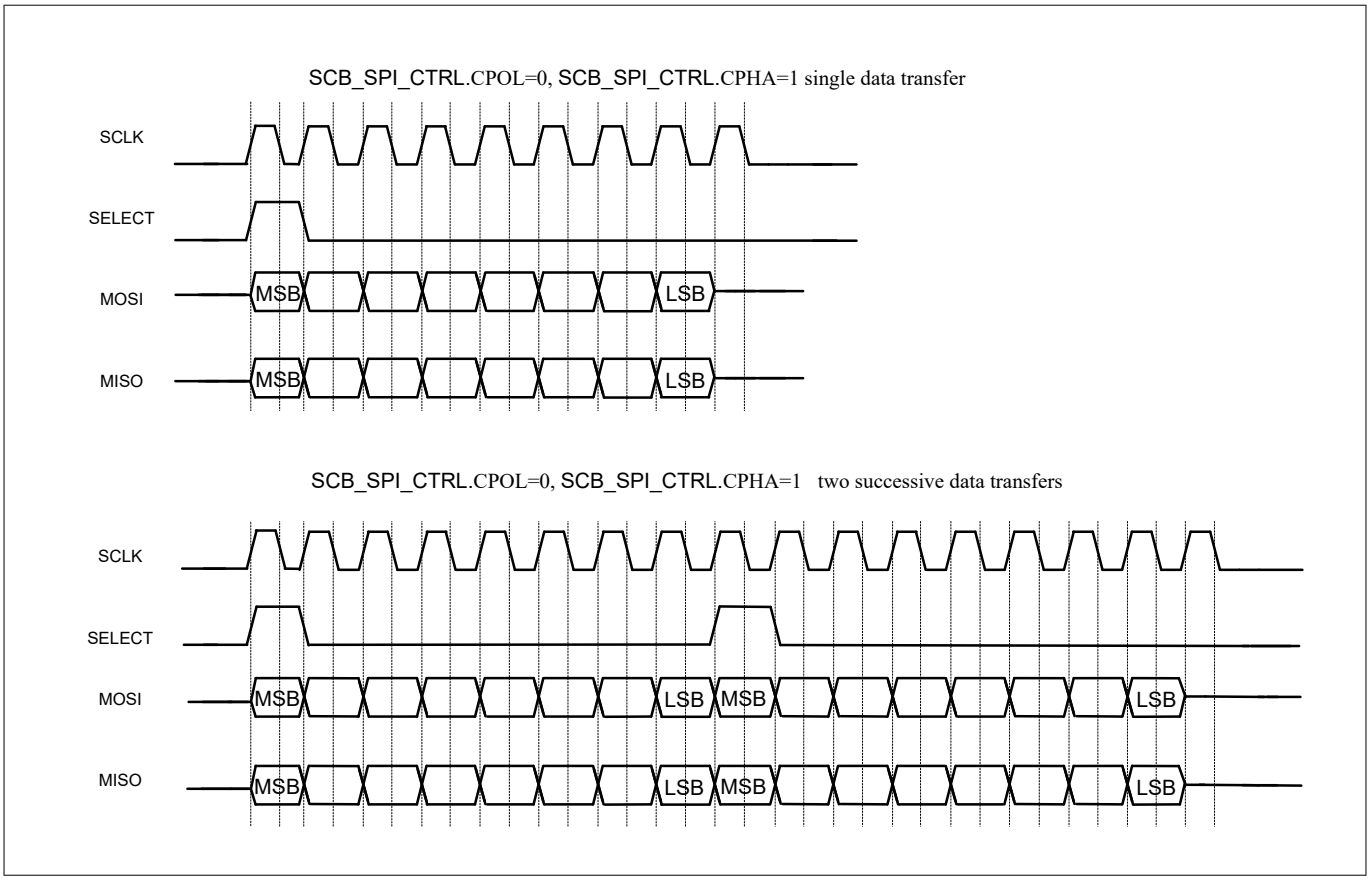


Figure 139 SPI TI data transfer example 2

Configuring SCB for SPI TI mode

To configure SCB for the 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_SPI_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 the next frame and '0' otherwise)
4. Follow steps 2 to 4 mentioned in the Enabling and initializing SPI section

For more information on these registers, see the PSOC™ Control C3M/P8, C3M/P7 registers reference manual.

26.3.3.3 National semiconductors SPI

The National Semiconductors' SPI protocol is a half-duplex protocol. In this protocol, transmission and reception do not occur at the same time, but take turns and transmission happens before reception. 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 140 illustrates a single data transfer and two successive data transfers. In both cases, the transfer size of the transmission data is 8 bits and the transfer size of the reception data is 4 bits.

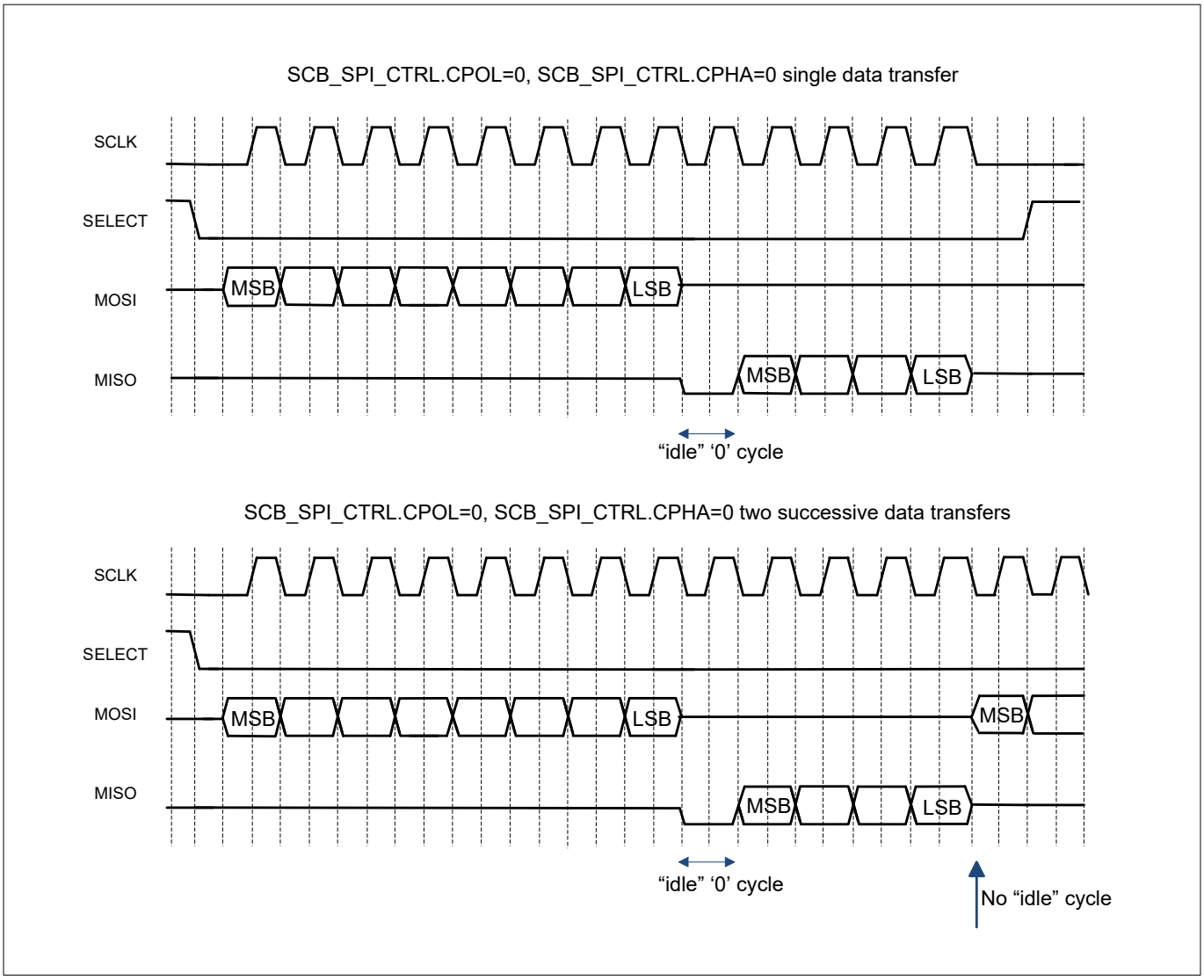


Figure 140 SPI NS data transfer example

Configuring SCB for SPI NS mode

To configure SCB for the 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_SPI_CTRL.MODE` register
3. Set the `SCBx_SPI_CTRL.CPOL` and `SCBx_SPI_CTRL.CPHA` bit to '0'
4. Follow steps 2 to 4 mentioned in the Enabling and initializing SPI section

For more information on these registers, see the PSOC™ Control C3M/P8, C3M/P7 registers reference manual.

26.3.4 SPI buffer modes

SPI can operate in the following three different buffer modes:

- FIFO mode
- EZ mode
- CMD_RESP mode

The buffer is used in different ways in each of these modes. The following subsections explain each of these buffer modes in detail.

26.3.4.1 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.

The FIFO mode is available only in CPU Active and Sleep power modes, and not in the system Deep Sleep mode.

Transmit and receive FIFOs allow write and read access. 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 the 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 DW 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.

System Deep Sleep to CPU Active transition

SCBx_CTRL.EC_AM_MODE = 1, SCBx_CTRL.EC_OP_MODE = 0, FIFO Mode.

MISO transmits 0xFF until the internally clocked logic takes over and the CPU writes to TX FIFO. Data on MOSI is ignored 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 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 system Deep Sleep to CPU Active. Another option is to disable CLK_SCB before going to system 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.

26.3.4.2 EZSPI mode

The 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 PSC3M/P8, PSC3M/P7, 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 8 bits 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 the following 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
- 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. During the reception of the command byte, the slave transmits (on MISO) either 0xfe (ready) or 0xff (busy). When disabled or reset, the slave transmits (on MISO) 0xff (busy). The byte value can be used by the master to determine whether the slave is ready to accept the SPI request. The slave then drives a reply byte on the MISO line to indicate whether 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. During the reception of the command byte, the slave transmits (on MISO) either 0xfe (ready) or 0xff (busy). When disabled or reset, the slave transmits (on MISO) 0xff (busy). The byte value can be used by the master to determine whether the slave is ready to accept the SPI request. 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. During the reception of the command byte, the slave transmits (on MISO) either 0xfe (ready) or 0xff (busy). When disabled or reset, the slave transmits (on MISO) 0xff (busy). The byte value can be used by the master to determine whether the slave is ready to accept the SPI request. The slave then drives a reply byte on the MISO line to indicate whether 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 141 illustrates the write of the EZ address, write to a memory array, and read from a memory array operation in the EZSPI protocol.

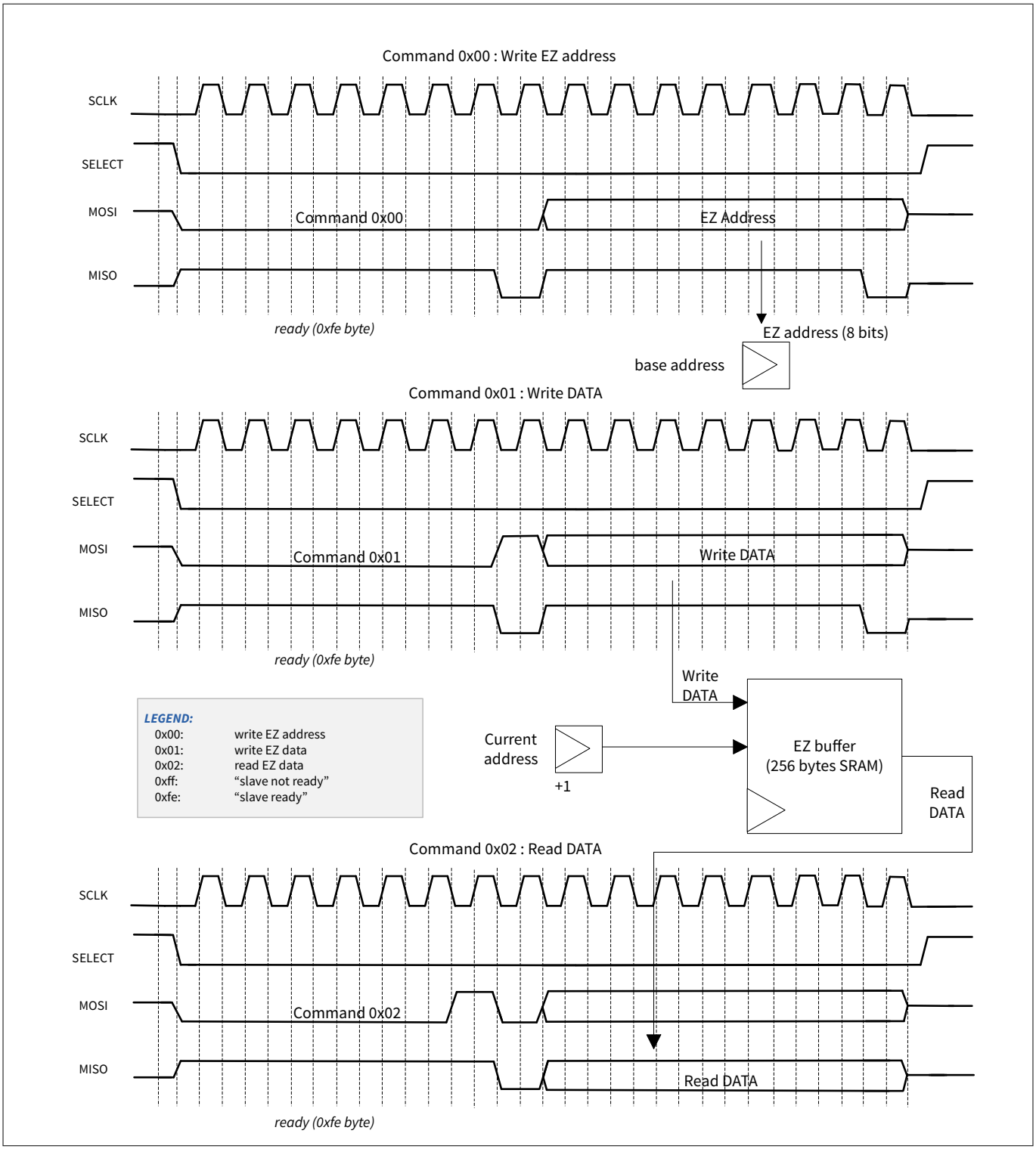


Figure 141 EZSPI example

Configuring SCB for EZSPI mode

By default, the SCB is configured for a 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 the Motorola SPI section
3. Follow steps 2 to 4 mentioned in the Enabling and initializing SPI section

For more information on these registers, see the PSOC™ Control C3M/P8, C3M/P7 registers reference manual.

System Deep Sleep to CPU Active transition

- 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 system 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
- EC_AM_MODE = 1, SCBx_CTRL.EC_OP_MODE = 1, EZ Mode - When transitioning from system Deep Sleep to CPU Active mode, there is no guarantee that the internal clock will be at the correct frequency due to the 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

***Note**: Only SCB[0] is Deep Sleep-capable in slave mode.*

26.3.4.3 Command-response mode

The command-response mode is defined only for an SPI slave. In the PSC3M/P8, PSC3M/P7 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 is 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 deselection), 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. The command-response mode operation is available in CPU Active and System 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 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. 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 different value (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.

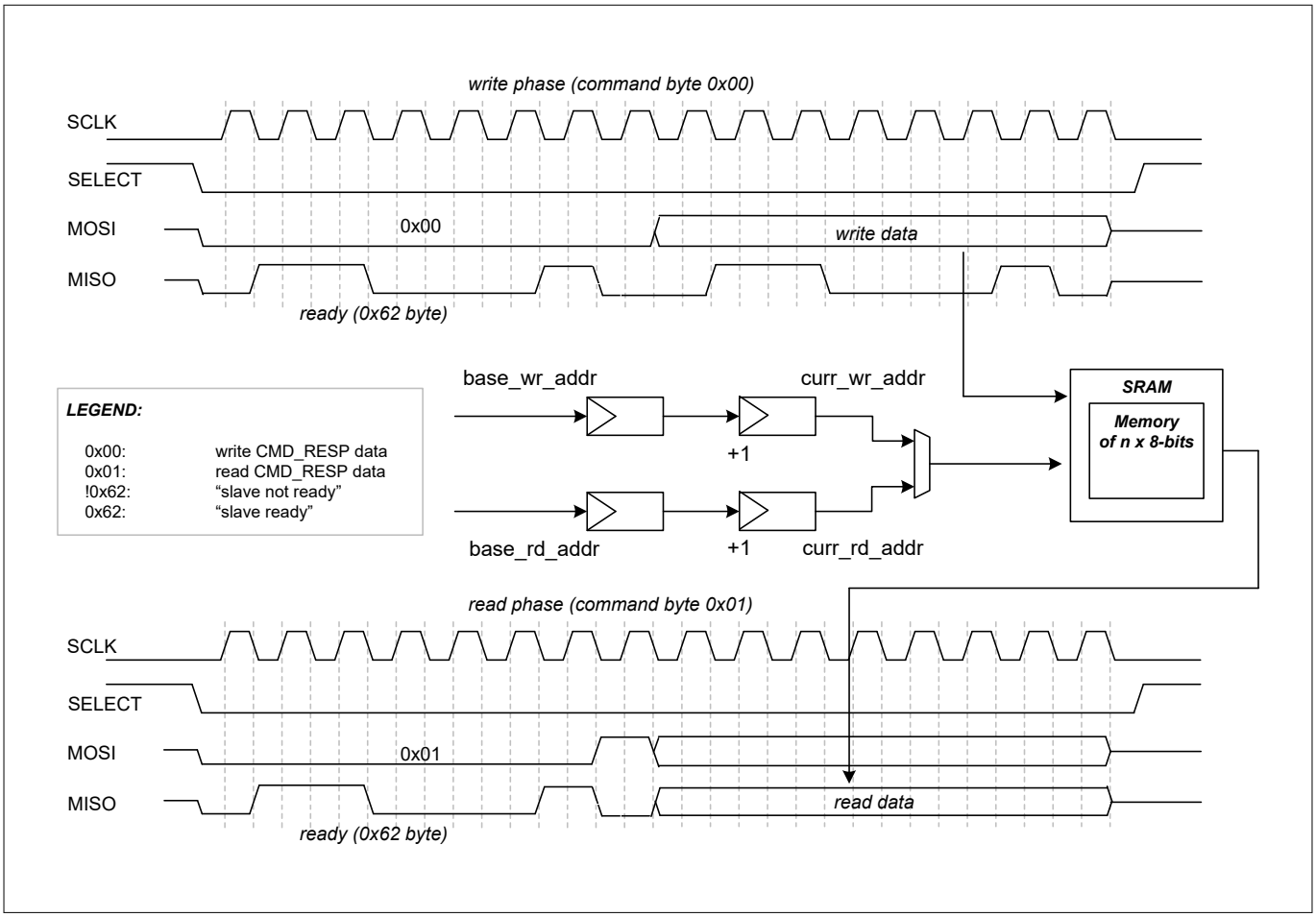


Figure 142 Command-response mode example

***Note**: A slave's base addresses are updated by the CPU and not by the master.*

System Deep Sleep to CPU Active transition

`SCBx_CTRL.EC_AM_MODE` = 1, `SCBx_CTRL.EC_OP_MODE` = 1, CMD_RESP mode.

When transitioning from System Deep Sleep to CPU Active mode, there is no guarantee that the internal clock is 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 the Motorola SPI section
3. Follow steps 2 to 4 mentioned in the Enabling and initializing SPI section

For more information on these registers, see the PSOC™ Control C3M/P8, C3M/P7 registers reference manual.

26.3.5 Clocking and oversampling

26.3.5.1 Clock modes

The SCB SPI supports both internally- and externally clocked operation modes. The SCBx_CTRL.EC_AM_MODE and SCBx_CTRL.EC_OP_MODE registers 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 an external master (SPI SCLK).

***Note**: In the PSC3M/P8, PSC3M/P7 MCU, only the Deep Sleep-capable SCB supports the 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 section.

The SCBx_CTRL.EC_AM_MODE and SCBx_CTRL.EC_OP_MODE can be configured as follows:

- EC_AM_MODE is '0' and SCBx_CTRL.EC_OP_MODE is '0': Use this configuration when only the 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
- EC_AM_MODE is '1' and SCBx_CTRL.EC_OP_MODE is '0': Use this configuration when both CPU Active and System 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 System Deep Sleep functionality interrupt brings the system into CPU Active power mode.

- - FIFO mode: Supported. The slave (MISO) transmits 0xFF until the CPU is awoken and the TX FIFO is Any data on the MOSI line will be dropped until CLK_SCB is enabled (see System Deep Sleep to CPU Active Transition for more details)
  - EZ mode: Supported. In System Deep Sleep power mode, the slave (MISO) transmits a busy (0xFF) byte during the reception of the command byte. In CPU 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
- EC_AM_MODE is '1' and SCBx_CTRL.EC_OP_MODE is '1'. Use this mode when both CPU Active and System Deep Sleep functionality are required. When the slave is selected, SCBx_INTR_SPI_EC.WAKE_UP is set to '1'. The associated System Deep Sleep functionality interrupt brings the system into CPU 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 1 SPI mode 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	No[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 the CPU Active and System Deep Sleep power modes.

In CPU Active system power mode, the memory buffer requires arbitration between the 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 access). The external interface logic takes two serial interfaces clock/bit periods for SPI. During this period, the internal logic is denied service to the memory buffer. The PSC3M/P8, PSC3M/P7 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 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 nonfailing 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 access) and SCBx_INTR_RX.BLOCKED field to '1' (for read access). 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.

26.3.5.2 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.

26.3.5.3 Oversampling and bit-rate

SPI master mode

The SPI master does not support the externally clocked mode. In the internally clocked mode, the logic operates under the 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 specifies the oversampling. The oversampling rate is calculated as the value in the SCBx_CTRL.OVS register + 1. In SPI master mode, the valid range is from 4 to 16. For SPI master with MOSI only (transmission only), the valid range is from 2 to 16. The bit rate is calculated as follows:

Bit rate = Input clock/SCBx_CTRL.OVS

Therefore, with an input clock of 100 MHz, the maximum bit rate is 25 Mbps with MISO, or 50 Mbps without MISO.

***Note**: For using a bit rate of 50 Mbps, ensure that the following parameters are set to avoid potential metastable conditions:*

- *CLK_SCB: 200 MHz*
- *OVS: 3*
- *MEDIAN: 1*

The above values indicate how fast the SCB hardware can run SCLK. However, it does not indicate whether the master can correctly receive data from a slave at those speeds. To determine that, calculate the path delay of MISO using the following equation:

$1/2 \times t_{SCLK} \geq t_{SCLK_PCB_D} + t_{DSO} + t_{SCLK_PCB_D} + t_{DSI}$

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
- $t_{SCLK_PCB_D}$ is the MISO PCB delay from slave to master t_{DSI} is the master setup time

Most slave datasheets list t_{DSO}. It may have a different name, so that look for the MISO output valid after the SCLK edge.

Most master datasheets also list t_{DSI} or master setup time. Calculate $t_{SCLK_PCB_D}$ and $t_{SCLK_PCB_D}$ based on specific PCB geometries.

If the desired speed is not achieved after doing these calculations, then consider using the MISO late sample feature of the SCB. This feature tells the SCB to sample the incoming MISO signal on the next edge of SCLK, and hence enables one-half SCLK cycle more timing margin.

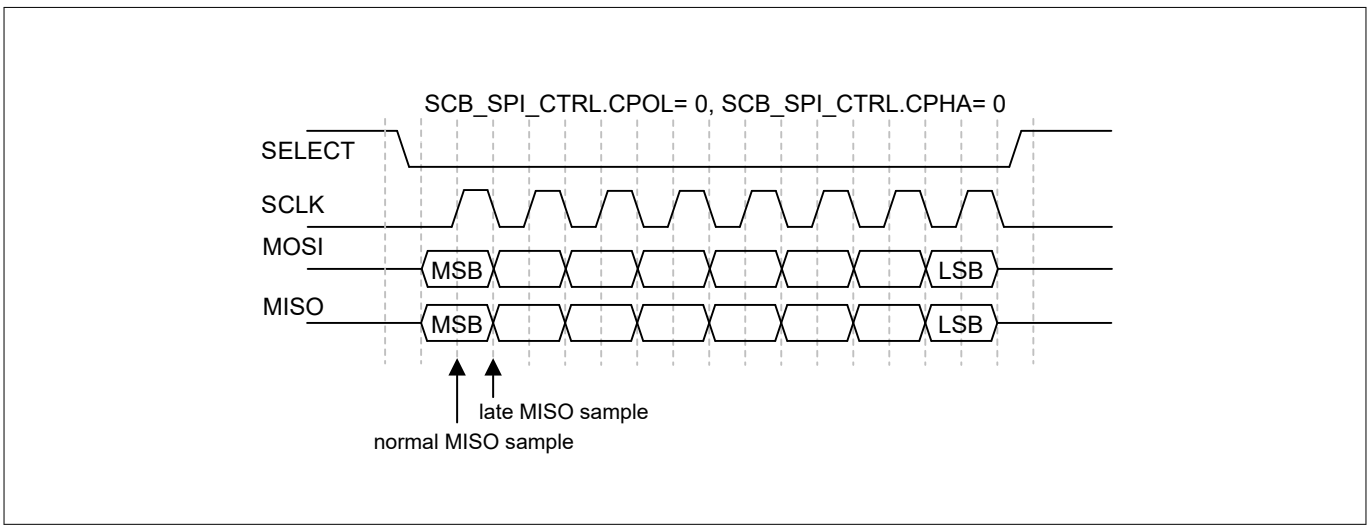


Figure 143 MISO sampling timing

This changes the equation to:

$t_{SCLK} \geq t_{SCLK_PCB_D} + t_{DSO} + t_{SCLK_PCB_D} + t_{DSI}$

Because the late sample helps in achieving better timing, leave it enabled all the time.

The t_{DSI} specification in the device datasheet assumes that the late sample is enabled.

***Note**: The SCBx_SPI_CTRL.LATE_MISO_SAMPLE is set to '1' by default.*

SPI slave mode

In the SPI slave mode, the SCBx_CTRL.OVS register is not used. The data rate is determined by the earlier equations. The late MISO sample is determined by the external master and not by SCBx_SPI_CTRL.LATE_MISO_SAMPLE.

For PSC3M/P8, PSC3M/P7 MCUs, tDSO is provided in the device datasheet. For the internally-clocked mode, it is proportional to the frequency of the internal clock. For example, it may be:

TDSO_EXT = TDSO_INT + 3 × Tclk_scb

TDSO_EXT, output data valid after the SCLK driving edge (external SPI Master clock)

TDSO_INT, when the external SPI Master does not have LATE_SAMPLE

***Note**: For using a bit rate of 50 Mbps, ensure that the following parameters are set to avoid potential metastable conditions:*

- *CLK_SCB: 200 MHz*
- *OVS: 4*
- *MEDIAN: 1*

26.3.6 SPI master SELECT output timing control

The SPI master SELECT output signal's “select” timing is a variable. This applies to the following:

- 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
  - When SCBx_SPI_CTRL.CPHA = 0: 0.75 or 1.75 SPI clock cycles (for SPI slaves requiring a longer setup time)
  - When SCBx_SPI_CTRL.CPHA = 1: 0.25 or 1.25 SPI clock cycles (for SPI slaves requiring a longer setup time)
- SELECT hold time (SCBx_SPI_CTRL.SSEL_HOLD_DEL register)
  - When SCBx_SPI_CTRL.CPHA = 0: 0.25 or 1.25 SPI clock cycles (for SPI slaves requiring a longer setup time)
  - When SCBx_SPI_CTRL.CPHA = 1: 0.75 or 1.75 SPI clock cycles (for SPI slaves requiring a longer setup time)

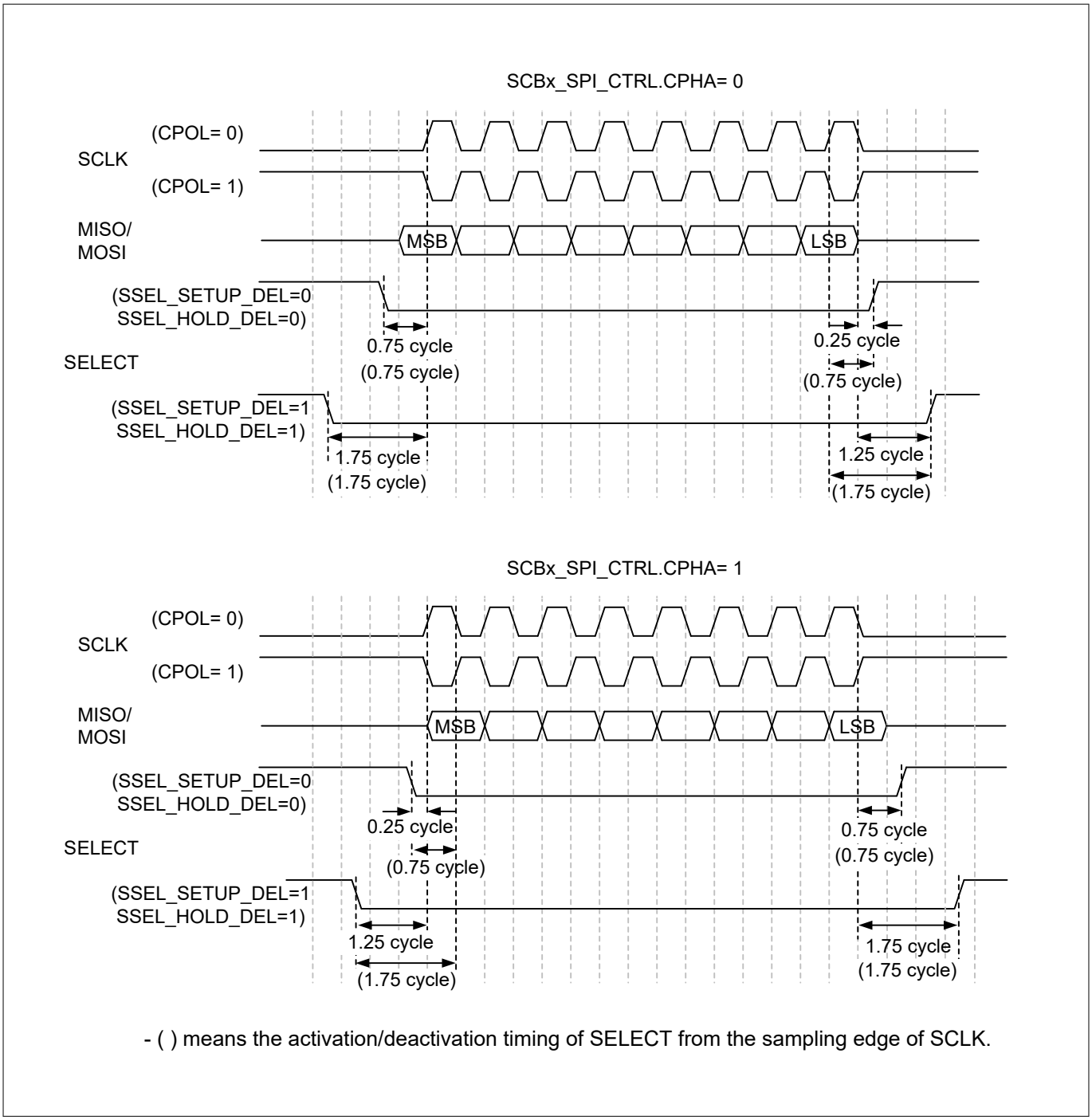


Figure 144 SELECT setup/hold delay

- INTER-FRAME deselect time (`SCBx_SPI_CTRL.SSEL_INTER_FRAME_DEL` register):
  - 1.5 SPI clock cycles or 2.5 SPI clock cycles (for SPI slaves requiring a longer deselect time)

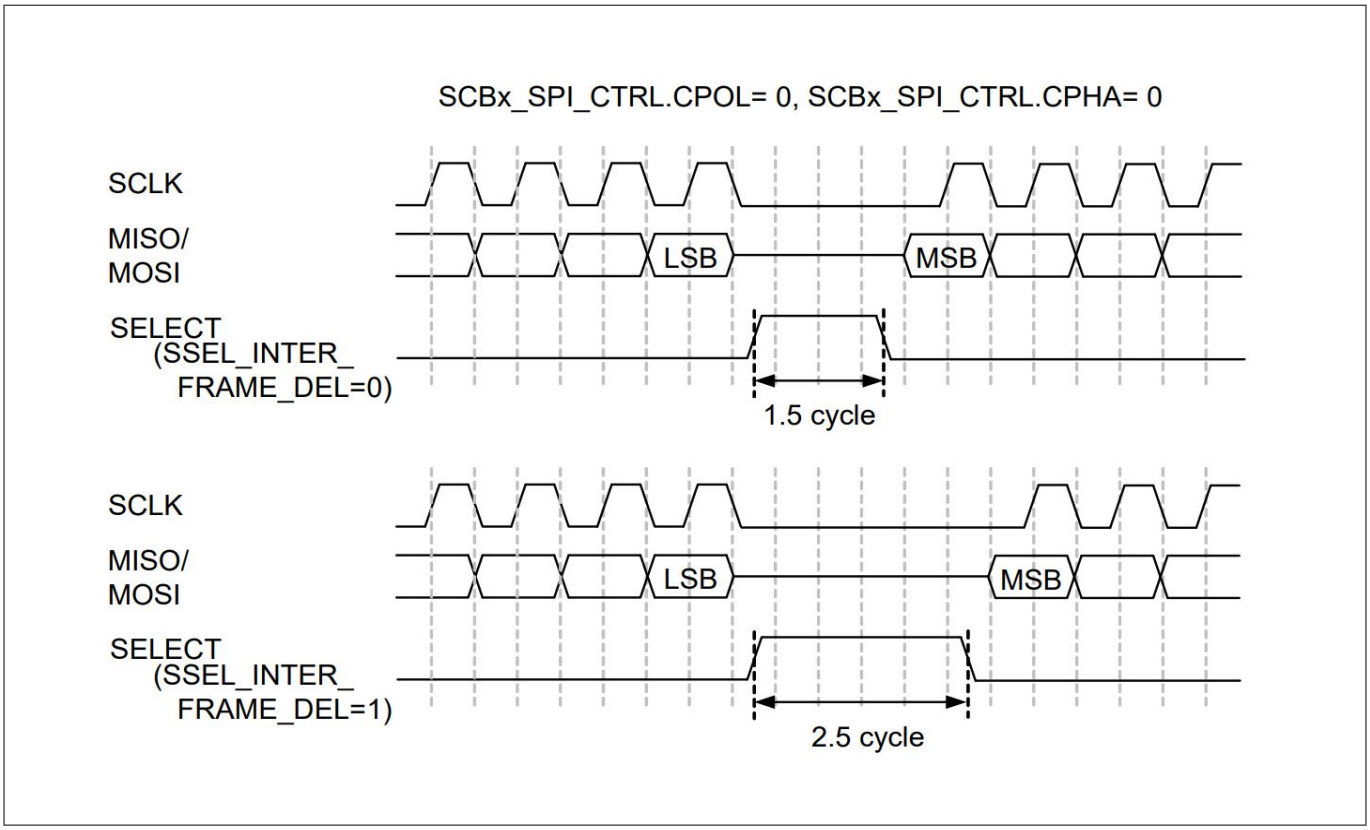


Figure 145 SELECT INTER-FRAME deselect time

26.3.7 SPI parity functionality

The features of the parity functionality feature in SPI mode are as follows:

- Applies to the SPI master and SPI slave with internally-clocked operation
- 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
- Can be configured to be enabled or disabled using `SCBx_SPI_TX_CTRL.PARITY_ENABLED` and `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 `DROP_ON_PARITY_ERROR` register. Additionally, the `PARITY_ERROR` bit in the `SCBx_INTR_RX` is set
- Supports even and odd parity (`SCBx_SPI_TX_CTRL.PARITY`, `SCBx_SPI_RX_CTRL.PARITY`)

26.3.8 Loop-back

In SPI Master mode, the 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.

26.3.9 Enabling and initializing SPI

Do the following to program the SPI:

1. Program the protocol-specific information using the `SCBx_SPI_CTRL` register. This includes selecting the sub-modes 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 data frame width (SSEL_CONTINUOUS), and SELECT polarity (SSEL_POLARITY 0-3). 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
   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 the SCBx_TX_FIFO_CTRL/SCBx_RX_FIFO_CTRL registers. Enable the following only for the 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, do not change the control bits. Make changes 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 is effective 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)

26.3.10 Fast SPI Master/Slave support

SCB[5] and SCB[6] are capable of Fast SPI operation of up to 50 MHz. The modes supported are SPI Master, SPI slave with internal clocking, and SPI slave with external clocking.

For Fast SPI operation, the clk_peri needs to operate at a frequency of 150-200 MHz.

26.3.11 I/O pad connection

26.3.11.1 SPI master

In the SPI master mode, the SCB provides data transmit and data receive functionality. Figure 146 and Table 209 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 the High-Z state, do one of the following:

- Use GPIO to drive the output to idle level
- Use Pull-up or Pull-down drive modes with an internal pull-up/pull-down resistor (fixed resistance number)
- 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 the 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).

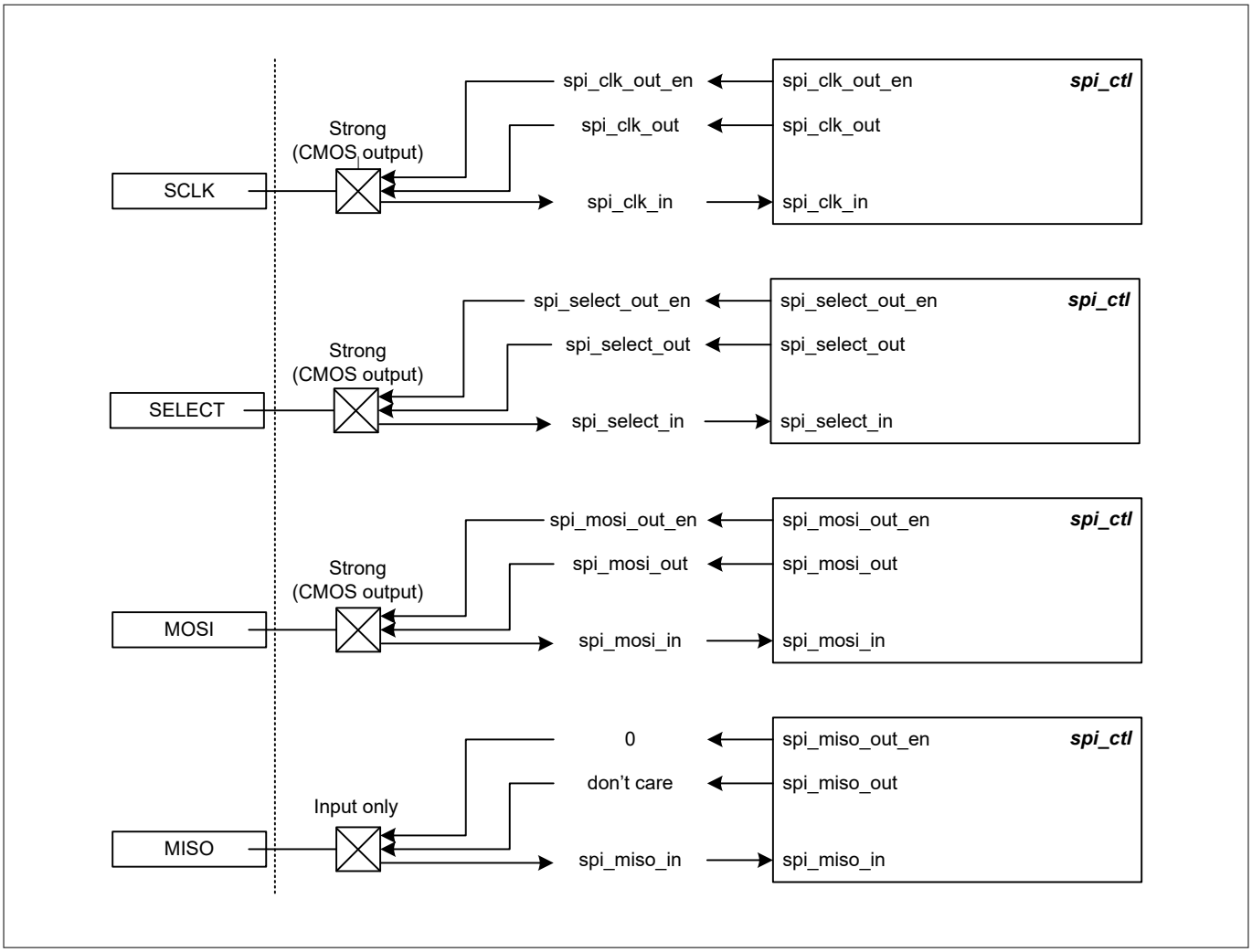


Figure 146 SPI master I/O pad connections

Table 209 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 spi_miso_out spi_miso_out_en	Receive a data element

26.3.11.2 SPI slave

In the SPI slave mode, the SCB provides data transmit and data receive functionality. Figure 13 and Table 3 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'. Therefore, the output will be High-Z. To avoid the High-Z state, do one of the following:

- Use GPIO to drive the output to idle level
- Use Pull-Up or Pull-Down drive modes with an internal pull-up/pull-down resistor (fixed resistance number)
- 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 the half-duplex mode.

- The drive mode of the MISO output can be Open Drain Drives Low only
- Add an external pull-up resistor on the MISO line

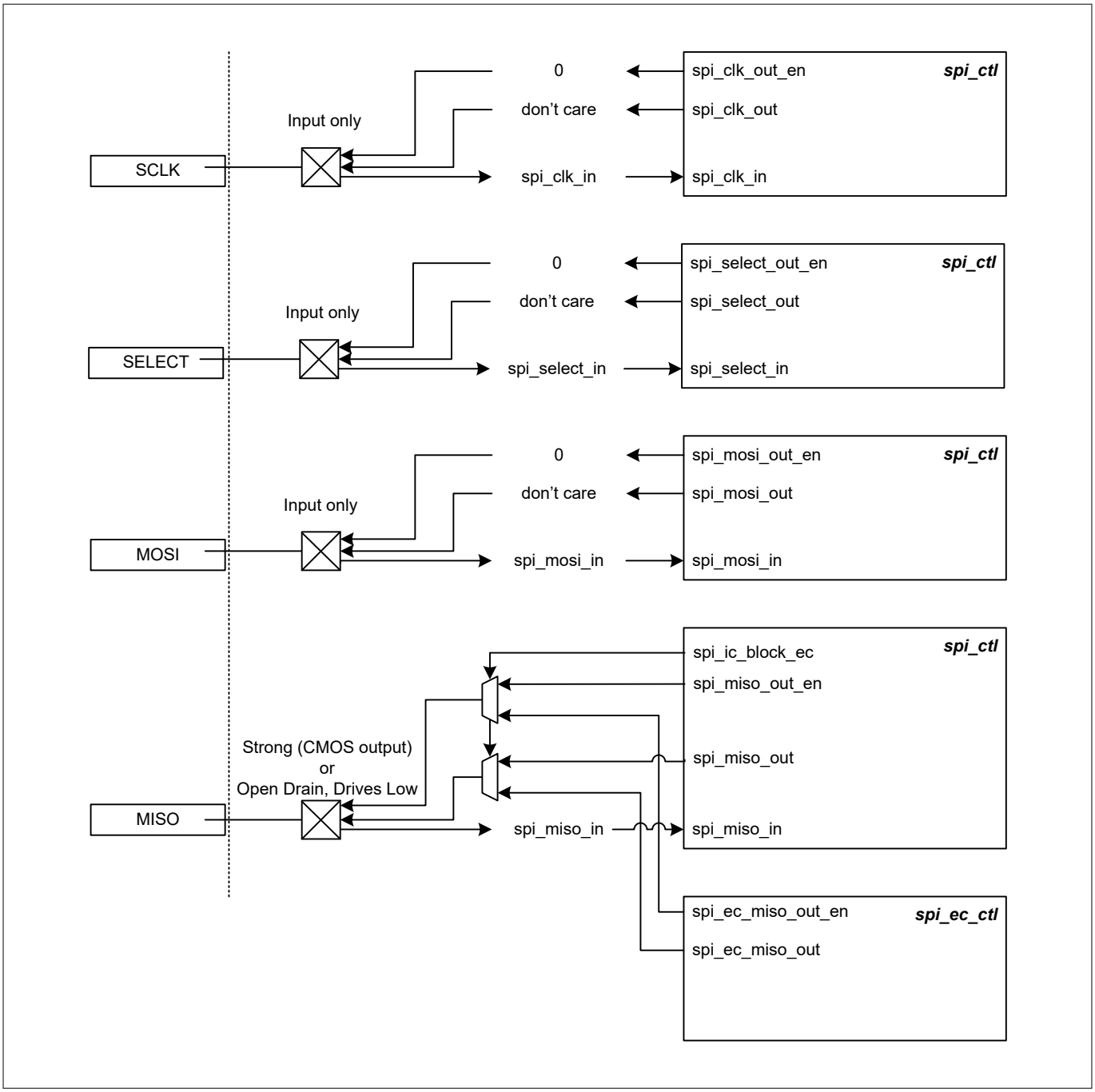


Figure 147 SPI slave I/O pad connections

Table 210 SPI slave I/O signal description

I/O pads	Drive mode	On-chip I/O signals	Usage
SCLK	Input only	spi_clk_out_en spi_clk_out spi_clk_in	Receive a clock signal
SELECT	Input only	spi_select_out_en spi_select_out spi_select_in	Receive a select signal

(table continues...)

Table 210 (continued) SPI slave I/O signal description

I/O pads	Drive mode	On-chip I/O signals	Usage
MOSI	Input only	spi_mosi_out_en spi_mosi_out spi_mosi_in	Receive a data element
MISO	Strong (CMOS output), or open drain drives low	spi_miso_out_en spi_miso_out spi_miso_in	Transmit a data element

26.3.12 Registers summary

The SPI interface is controlled using a set of 32-bit control and status registers. For more information on these registers, see the PSOC™ Control C3M/P8, C3M/P7 registers RM.

26.4 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 applies only to the TX functionality.

- Clear to Send (CTS): This is an input signal to the transmitter. If active, it means that the receiver has signaled to the transmitter that it is ready to receive data and now the transmitter can send the data
- Request to Send (RTS): This is an output signal from the receiver. If active, it indicates that the receiver is ready to receive data
- The RTS of one device is connected to the CTS of the other and vice-versa. When a device is ready to receive data (receiver), it activates it's RTS which in turn activates the CTS of the other device (transmitter) signaling it to send the data

***Note**: The flow control is only available in the UART standard protocol mode and is not available in the following modes - UART Local Interconnect Network (LIN), SmartCard (ISO7816), Infrared Data Association (IrDA), and UART multiprocessor mode.*

26.4.1 Features

The SCB supports the following features:

- Supports UART protocol
  - Standard UART (full duplex, half duplex)
  - Multi-processor mode
- SmartCard (ISO7816) reader
- IrDA
- Supports Local Interconnect Network (LIN)
  - Break detection

- Baud rate detection
- Collision detection (ability to detect that a driven bit value is not reflected on the bus, indicating that another component is driving the same bus)

- 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 7
- Parity support (odd and even parity)
- Programmable oversampling
- Start skipping
- Hardware flow control

26.4.2 General description

Figure 148 illustrates UART TX and RX transfers.

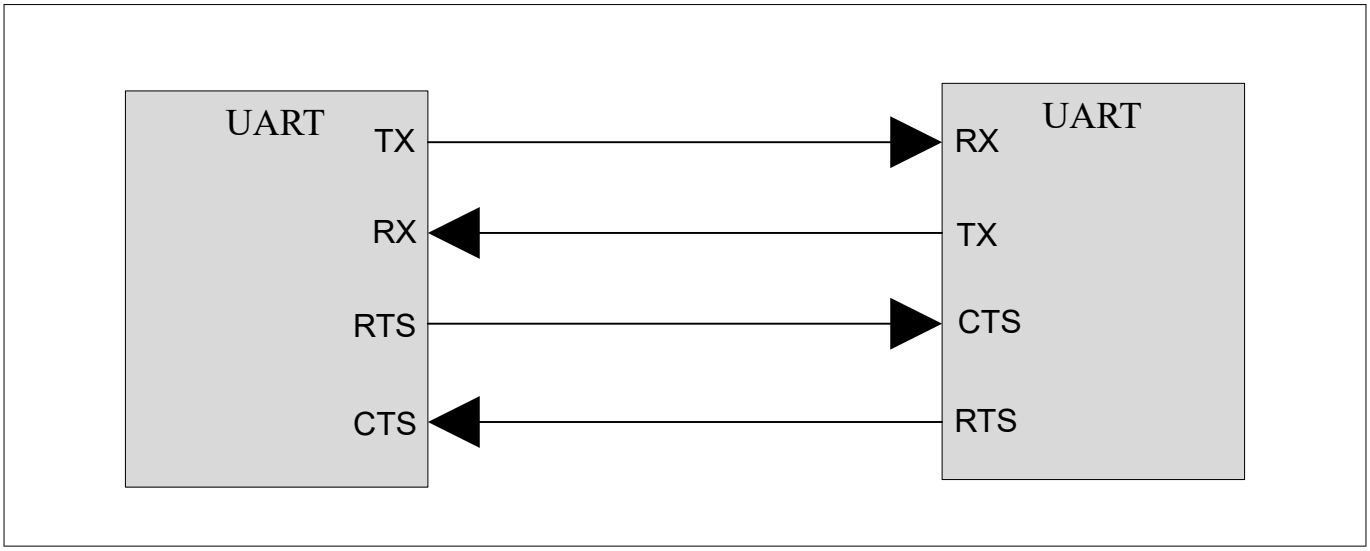


Figure 148 UART Example

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, the data frame width for both transmit (TX) and receive (RX) can be configured to any value in the range of 4 to 9 bits. This does not include start, stop, and parity bits. The number of stop bits can be in the range of 1 to 7. 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 the IrDA UART mode, the parity bit is automatically disabled.

***Note**: The UART interface does not support an external clocking operation. Hence, the UART operates only in the Active and Sleep system power modes. The UART also supports only the FIFO buffer mode.*

26.4.3 UART modes of operation

26.4.3.1 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 must 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 the 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 the transmitter and receiver, and is typically in the range of 1 to 3-bit transfer periods.

Figure 149 illustrates the UART protocol.

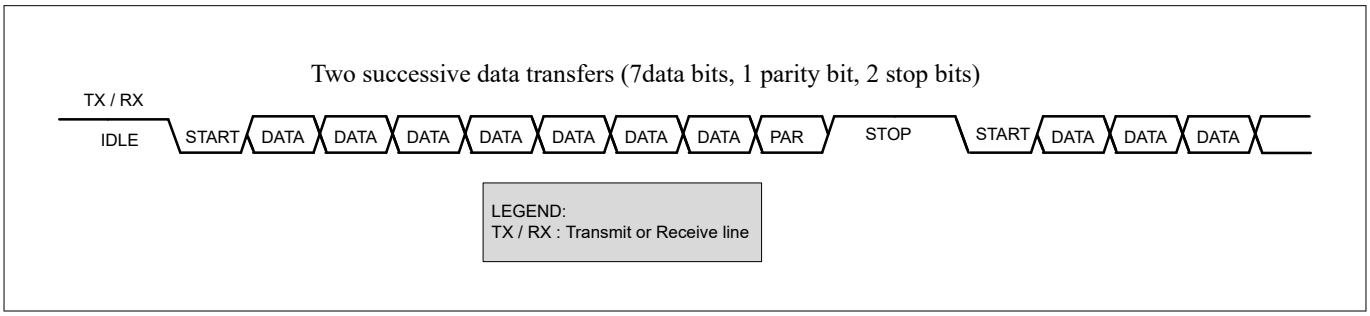


Figure 149 UART standard protocol example

The receiver oversamples the incoming signal, the value of the sample point in the middle of the bit transfer period (on the receiver's clock) is used. This is illustrated by Figure 150. Note the slight drift between transmitter and receiver clock.

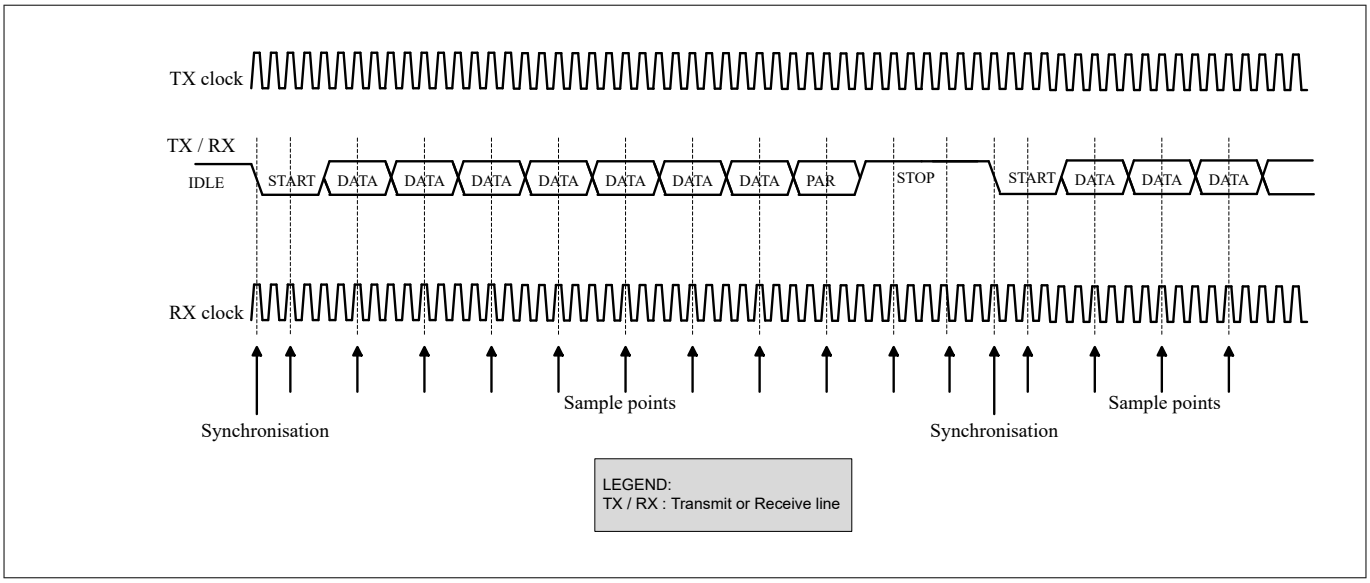


Figure 150 UART standard protocol oversampling (RX clock faster than TX clock)

Alternatively, three samples around the middle of the bit transfer period (on the receiver's clock) are used for a majority vote to reduce the noise sensitivity and to increase accuracy; this is enabled by enabling the MEDIAN filter in the SCBx_RX_CTRL register.

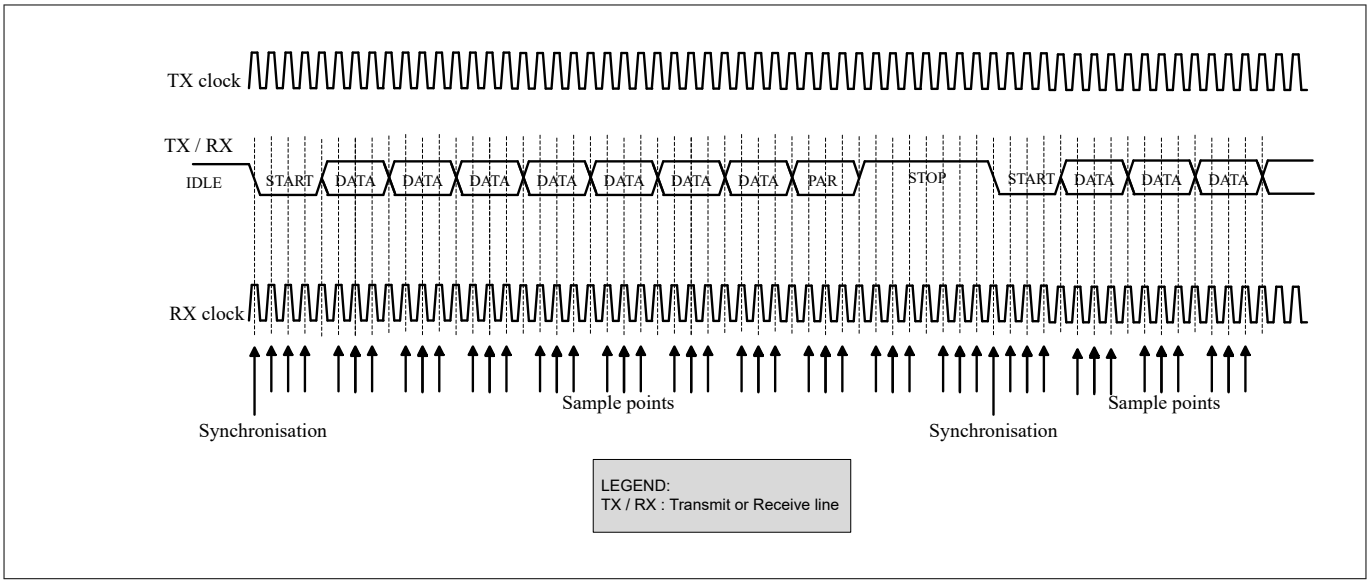


Figure 151 UART standard protocol oversampling (RX clock equal to TX clock)

26.4.3.1.1 Parity

This functionality adds a parity bit to the data frame and is used to identify single-bit data frame errors. The parity bit is always directly after 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
  - TX functionality: SCBx_UART_TX_CTRL.PARITY and SCBx_UART_TX_CTRL.PARITY_ENABLED
  - RX 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 the 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 Figure 152 illustrates the parity functionality (8-bit data frame).

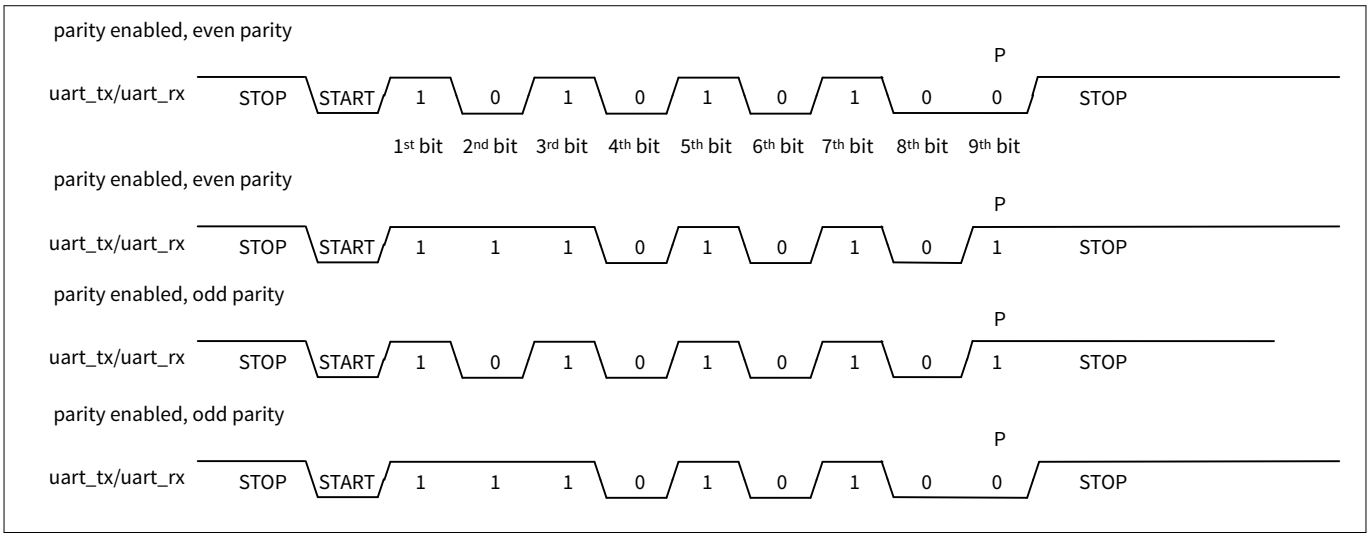


Figure 152 UART parity examples

26.4.3.1.2 Start skipping

Start skipping applies only 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 to wake up from the System Deep Sleep UART RX functionality. The intended process is described as follows:

1. Before entering System Deep Sleep power mode, the UART RX functionality is disabled, and the IOSS/GPIO is programmed to set an interrupt cause to '1' when UART_RX has a '1' to '0' transition (START bit)
2. While in System Deep Sleep, the UART RX functionality is not functional
3. The IOSS/GPIO interrupt is activated on the START bit and the system transitions from System Deep Sleep to CPU Active power mode. This transition takes up to roughly 20 µs
4. The CPU enables the UART RX functionality, with SCBx_UART_RX_CTRL.SKIP_START set to '1'
5. The UART receiver synchronizes the data frame receipt on the next '0' to '1' transition. If the UART RX functionality is enabled in time, this is transition from the START bit to the 1st 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 153 illustrates the process.

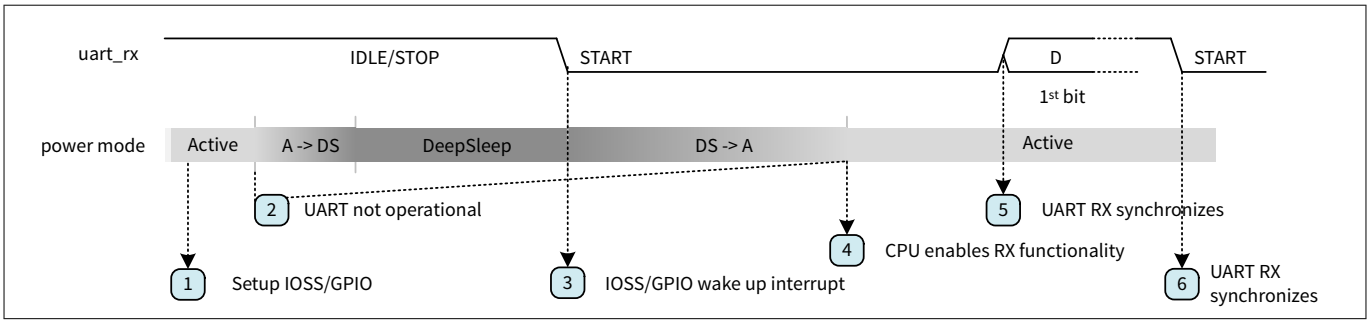


Figure 153 UART Start Skip and Wakeup from System Deep Sleep

Note that the above process works only for lower baud rates. The System Deep Sleep to CPU Active power mode transition and CPU enabling the UART receive functionality should take less than 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, the firmware takes some time to finish the wake-up 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 time, first send a "dummy" byte to the device to wake it up before sending real UART data. In this case, the SKIP_START bit can be left as 0.

This feature can work, only if the GPIO interrupt is triggered on falling edge of the START bit (not other data bits). Otherwise, the UART will receive corrupted data bytes. It depends on higher level protocols to detect the data corruption and to tell the UART when to sleep safely.

26.4.3.1.3 Half-duplex mode

The UART Half Duplex connection is supported only in Standard UART sub mode (CTRL.MODE is UART and UART_CTRL.MODE is UART_STD). From the protocol point of view, all features of standard UART sub mode are supported except the parallel transmission and reception, which is caused by the single transmission line. To enable the half-duplex pad connection, UART_RX_CTRL.HDRXEN needs to be set to '1'. Set UART_RX_CTRL.HDRXEN='0' for full-duplex communication.

Because the single transmission line is not permanently driven, a weak pull-up resistor to VDDIO needs to be connected either chip internally or externally. This keeps the logic level high during idle time. The transmission pad can be configured to either push-pull or open drain mode, depending on the upper level protocol and transmission speed. The output driver of the pad is only enabled during transmission.

The signal input of the pad needs to be permanently enabled.

While LIN protocol is enabled, the receiving path is also enabled during transmission.

The following figure illustrates the half-duplex mode.

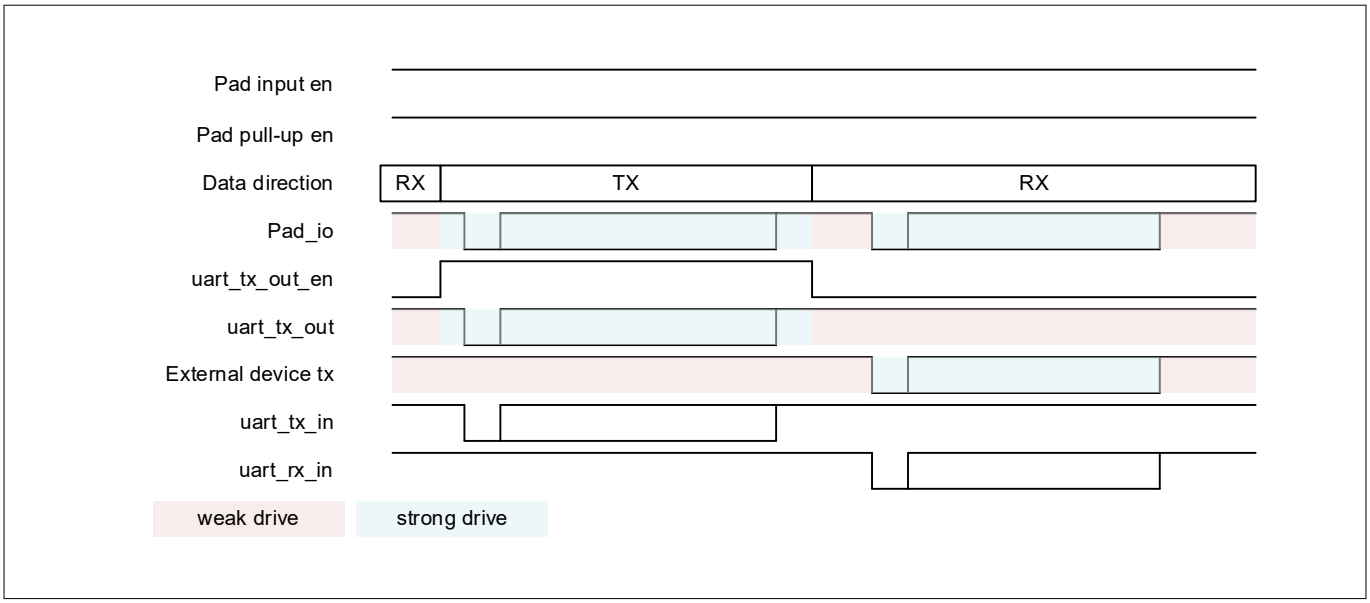


Figure 154 UART Standard Protocol in Half-duplex mode

26.4.3.1.4 Break detection

The UART mode (SCBx_CTRL.MODE is UART) and standard UART sub mode (SCBx_UART_CTRL.MODE is UART_STD) support break detection. Break detection detects when UART_RX is low ’0’ for more than SCBx_UART_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 period(s) exist for clock inaccuracies between transmitter and receiver.

For example, for an 8-bit data frame size, with parity support, the maximum number of low/’0’ bit periods are 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).

- Break detection only applies to RX 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
- Break detection is part of the LIN protocol
- When a break is detected, the SCBx_INTR_RX.BREAK_DETECT interrupt cause is set to ‘1’

Figure 155 illustrates a regular data frame and break frame (8 bits data frame, parity support, and a break width of 12-bit periods).

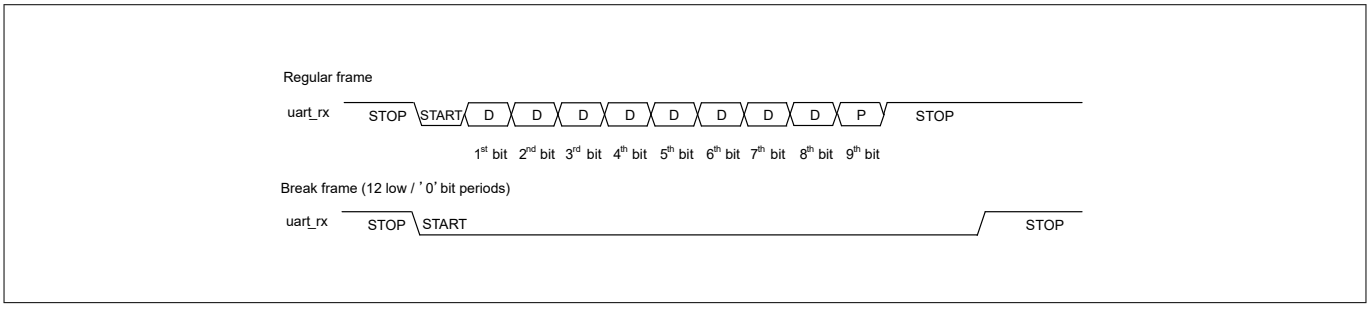


Figure 155 UART – Regular Frame and Break Frame

When SCBx_UART_RX_CTRL.BREAK_LEVEL is set to 1, idle line detection can be performed. For example, after successive transfer of several UART data frames, idle (high) level longer than normal data frame length (start+8 data+1 parity +1 stop) means the end of this successive transfer.

26.4.3.1.5 UART flow control

The standard UART mode supports flow control. Modem flow control controls the pace at which the transmitter transfers data to the receiver. Flow control only applies to TX functionality. 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 the UART CTS line is active and its TX FIFO is not empty.

- 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: Request 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 bitfields SCBx_UART_FLOW_CTRL.RTS_POLARITY and SCBx_UART_FLOW_CTRL.CTS_POLARITY. Figure 156 gives an overview of the flow control functionality.

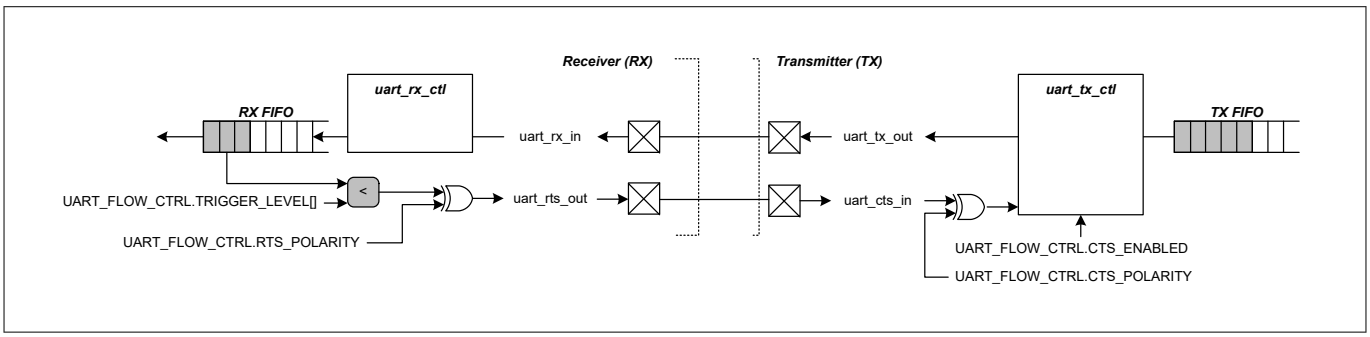


Figure 156 UART flow control connection

26.4.3.1.6 Multi-processor mode

The UART_MP (multi-processor) mode is defined with single-master-multi-slave topology, as Figure 157 shows. This mode is also known as the UART 9-bit protocol because the data field is 9 bits wide. UART_MP is part of Standard UART mode.

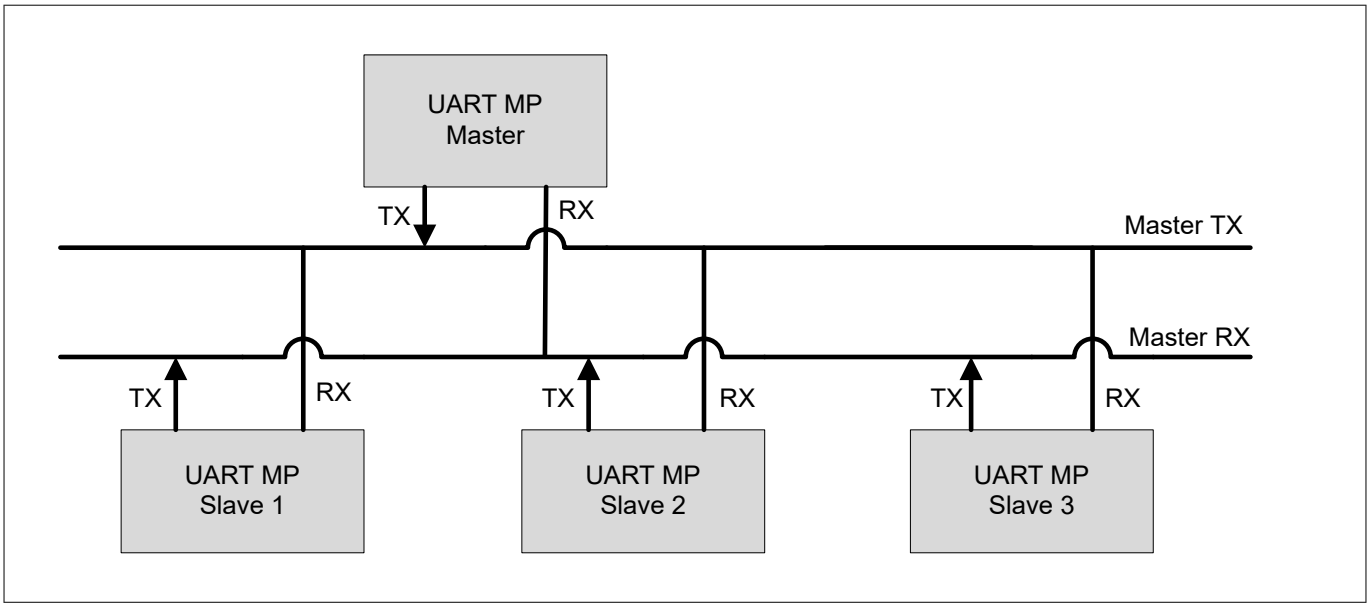


Figure 157 UART MP mode bus connections

The main properties of the 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 158
- Parity bit is disabled

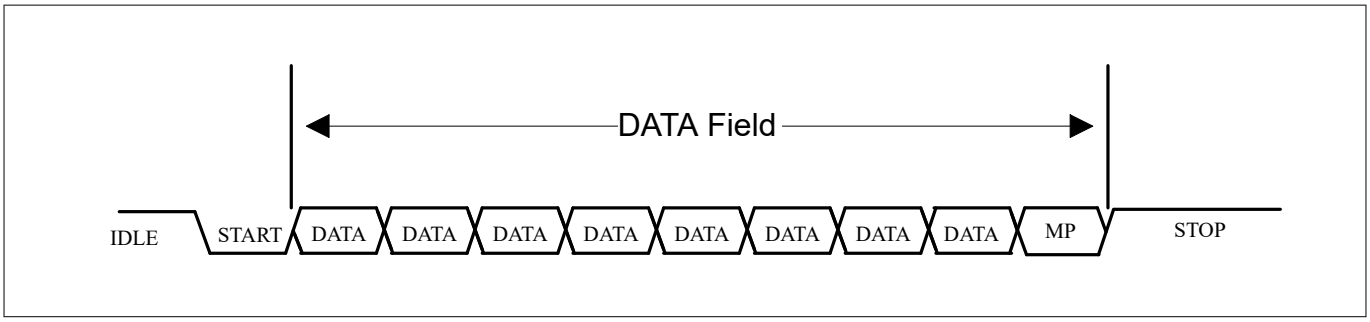


Figure 158 UART MP address and data frame

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

Configuring the SCB as a Standard UART Interface

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

1. Configure the SCB as UART interface by writing '10b' to the MODE field (bits [25:24]) of the SCBx_CTRL register
2. Configure the UART interface to operate as a Standard protocol by writing '00' to the MODE field (bits [25:24]) of the SCBx_UART_CTRL register

3. To enable the UART MP Mode or UART LIN Mode, write ‘1’ to the MP_MODE (bit 10) or LIN_MODE (bit 12) respectively of the SCBx_UART_RX_CTRL register
4. Follow the steps 2 to 4 described in Enabling and initializing UART

26.4.3.2 SmartCard (ISO 7816)

ISO7816 is an asynchronous serial interface, defined with single-master-single slave topology. ISO7816 defines both Reader (master) and Card (slave) functionality. For more information, refer to the ISO7816 Specification.

Both master (reader) and slave (card) functions are supported by the SCB. This block provides the basic physical layer support with asynchronous character transmission. The 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.

In a SmartCard transfer, the transmitter drives the start bit and data bits (and optionally a parity bit). After these bits, it enters its stop period by releasing the bus, which results in the line being ‘1’ (the value of a stop bit). After the half-bit transfer period into the stop period, the receiver may drive a NACK on the line (a value of ‘0’) for a one to two bit transfer period. This NACK is observed by the transmitter, which reacts by extending its stop period by a one-bit transfer period. For this protocol to work, the stop period should be at least two bits of 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 tri-state 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 159 illustrates the SmartCard protocol.

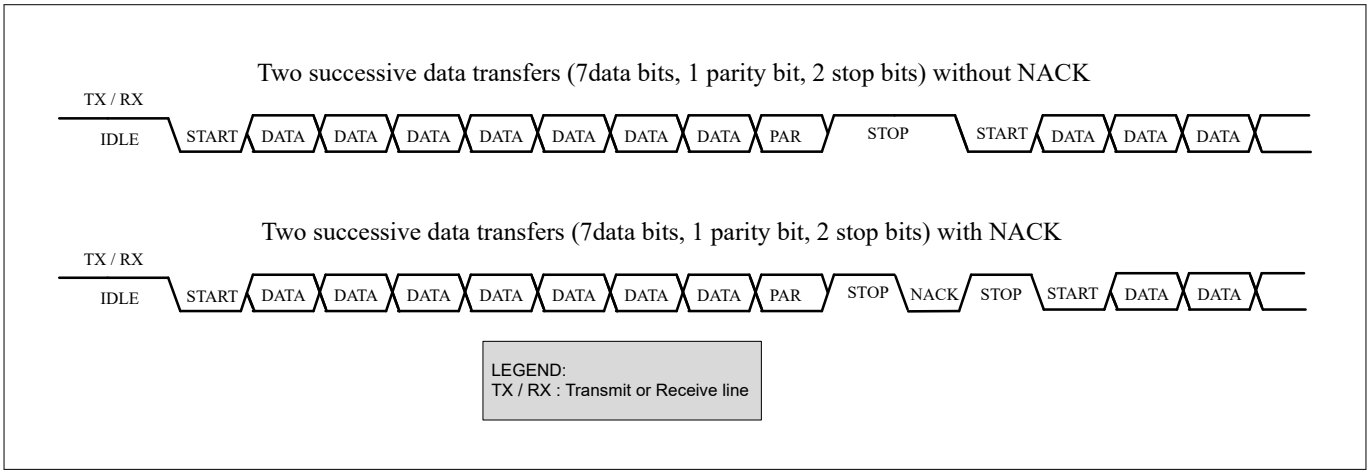


Figure 159 SmartCard example

The communication baud rate while using the SmartCard is given as:

Baud rate = Fscb_clk/Oversample

Where Fscb_clk is the "scb_clk" clock frequency and Oversample is the oversample factor.

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 UART registers.

1. Configure the SCB as UART interface by writing ‘10’ to the MODE (bits [25:24]) of the SCB_CTRL register
2. Configure the UART interface to operate as a SmartCard protocol by writing ‘01’ to the MODE (bits [25:24]) of the SCB_UART_CTRL register
3. Follow steps 2 to 4 described in Enabling and initializing UART

26.4.3.3 IrDA

The SCB supports the IrDA protocol for data rates of up to 115.2 kbps using the UART interface. It supports only the basic physical layer of the 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 UART signaling. At the transmitter, bits are modulated, and 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 to 16 times the selected baud rate, by configuring the SCB_OVS field of the SCB_CTRL register. In addition, the PSC3M/P8, PSC3M/P7 MCU supports a low-power IrDA receiver mode, which allows it to detect pulses with a minimum width of 1.41 µs.

Bit rates less than 115.2 kbps can be achieved by configuring the clk_scb frequency. Additional allowable rates are 2.4 kbps, 9.6 kbps, 19.2 kbps, 38.4 kbps, and 57.6 kbps. Figure 160 shows how a UART transfer is IrDA modulated.

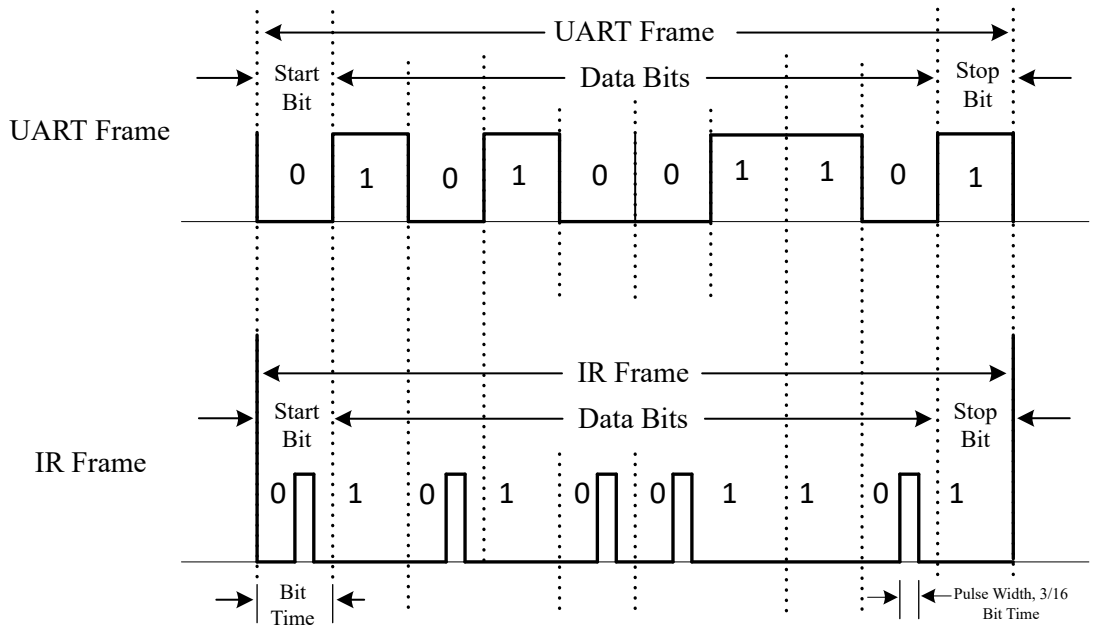


Figure 160 IrDA example

Configuring the SCB as a 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 UART registers.

1. Configure the SCB as a UART interface by writing '10b' to the MODE (bits [25:24]) of the SCB_CTRL register
2. Configure the UART interface to operate as an IrDA protocol by writing '10b' to the MODE (bits [25:24]) of the SCB_UART_CTRL register
3. Enable the Median filter on the input interface line by writing '1' to MEDIAN (bit 9) of the SCB_RX_CTRL register
4. Configure the SCB as described in Enabling and initializing UART

26.4.3.4 UART local interconnect network (LIN) mode

The LIN protocol is supported by the SCB as part of the standard UART. LIN is designed with a single-master-multi-slave topology. There is one master node and multiple slave nodes on the LIN bus. The SCB UART supports only LIN slave functionality. The LIN specification defines both physical layer (layer 1) and data link layer (layer 2). Figure 161 illustrates the UART_LIN and LIN transceiver.

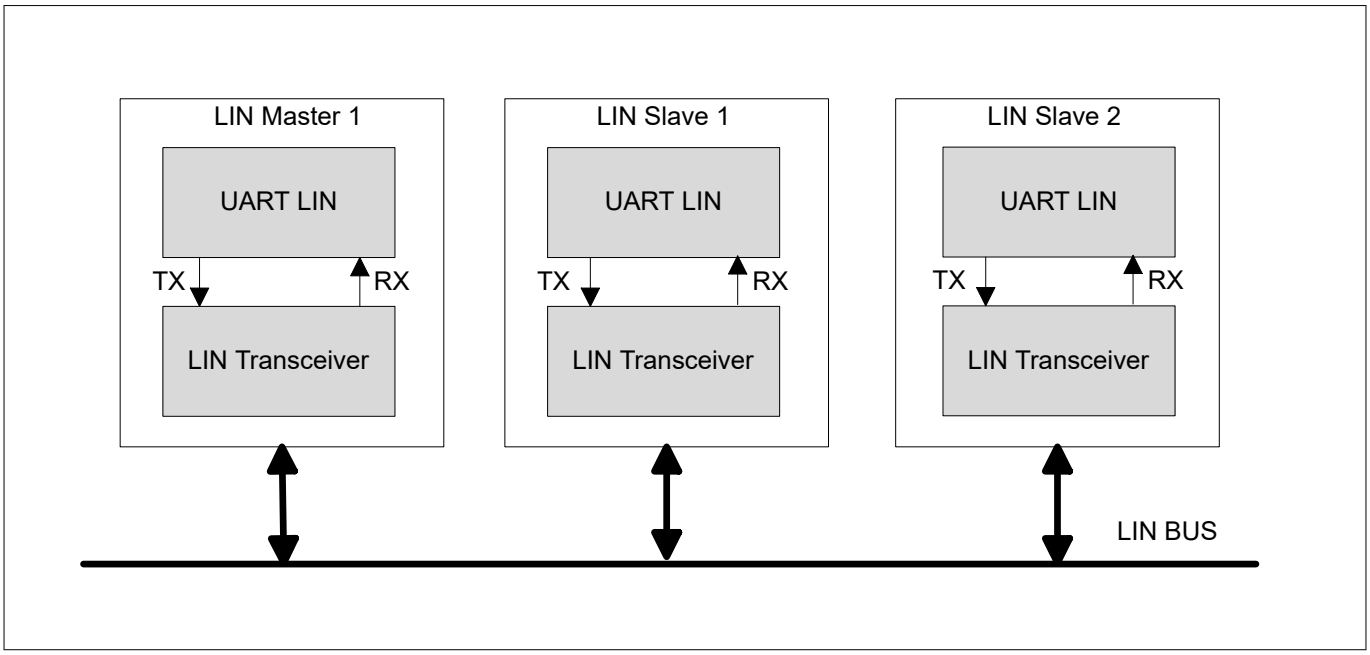


Figure 161 UART_LIN and LIN transceiver

LIN protocol defines two tasks:

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

The master node supports master task and slave task; the slave node supports only slave task, as shown in Figure 162.

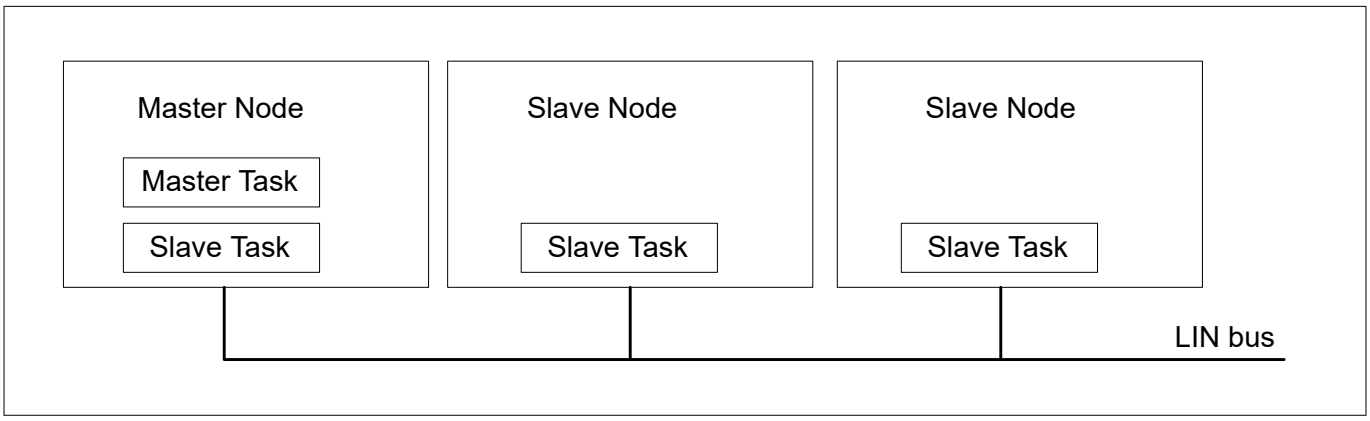


Figure 162 LIN bus nodes and tasks

LIN frame structure

LIN is based on the transmission of frames at predetermined moments of time. A frame is divided into header and response fields, as shown in Figure 163.

- The header field consists of:
  - Break field (at least 13-bit periods with the value '0')

- Sync field (a 0x55 byte frame). A sync field can be used to synchronize the clock of the slave task with that of the master task
  - Identifier field (a frame specifying a specific slave)
- The response field consists of data and checksum

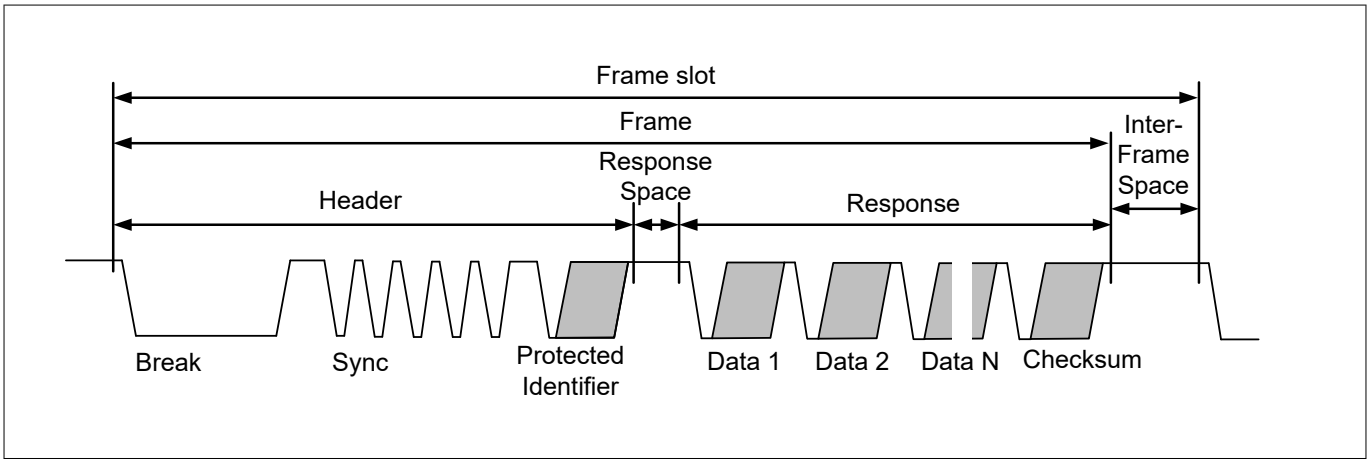


Figure 163 LIN frame structure

In LIN protocol communication, the least significant bit (LSb) of the data is sent first and the most significant bit (MSb) last. The start bit is encoded as zero and the stop bit is encoded as one. The following sections describe all the byte fields in the LIN frame.

Break field

Every new frame starts with a break field, which is always generated by the master. The break field has a logical zero with a minimum of 13-bit times and followed by a break delimiter. The break field structure is as shown in the Figure 164.

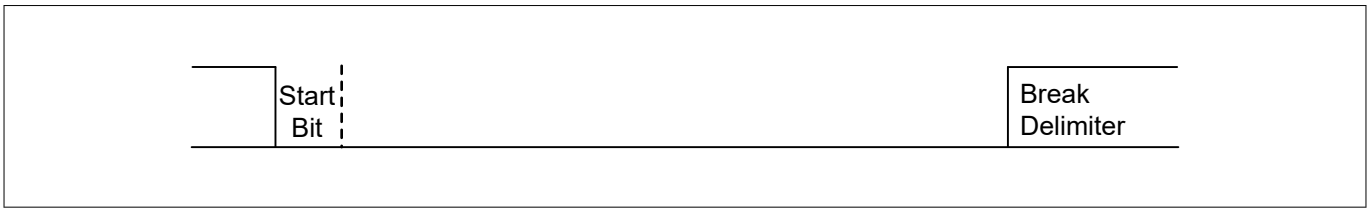


Figure 164 LIN break field

Sync field

This is the second field transmitted by the master in the header field; its value is 0x55. A sync field can be used to synchronize the clock of the slave task with that of the master task for automatic baud rate detection. Figure 165 shows the LIN sync field structure.

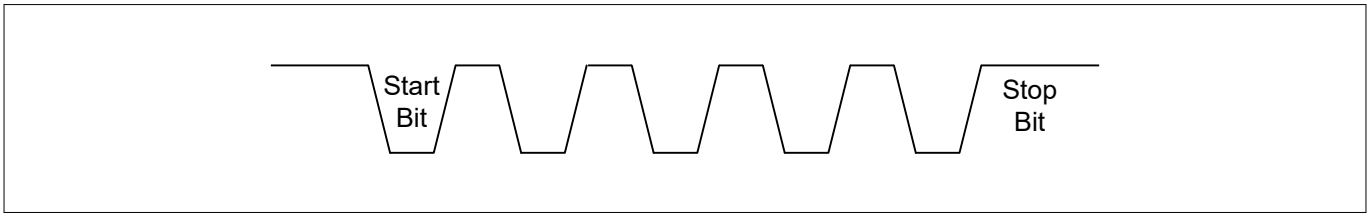


Figure 165 LIN sync field

Protected Identifier (PID) field

A protected identifier field consists of two subfields: the frame identifier (bits 0-5) and the parity (bit 6 and bit 7). The PID field structure is shown in Figure 166.

- Frame identifier: The frame identifiers are divided into three categories -
  - Values 0 to 59 (0x3B) are used for signal carrying frames
  - 60 (0x3C) and 61 (0x3D) are used to carry diagnostic and configuration data
  - 62 (0x3E) and 63 (0x3F) are reserved for future protocol enhancements
- Parity: Frame identifier bits are used to calculate the parity

***Note**: The parity calculation of PID (protected identifier field) will also be handled by a firmware and the SCB block's UART will take the PID as the common data byte.*

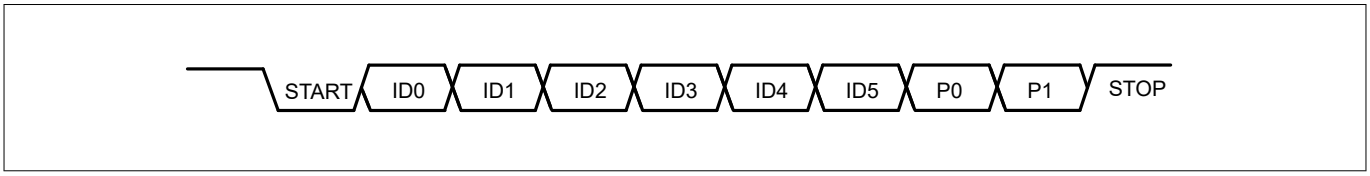


Figure 166 PID field

Data

In LIN, every frame can carry a minimum of 1 byte and a maximum of 8 bytes of data. Here, the LSb of the data byte is sent first and the MSb of the data byte is sent last.

Checksum

The checksum is the last byte field in the LIN frame. It is calculated by inverting the 8-bit sum along with a carryover of all data bytes only or the 8-bit sum with the carryover of all data bytes and the PID field. There are two types of checksums in LIN frames:

- Classic checksum: Calculated over all the data bytes only (used in LIN 1.x slaves)
- Enhanced checksum: Calculated over all the data bytes along with the protected identifier (used in LIN 2.x slaves)

***Note**: The checksum calculation is handled by the firmware, and the SCB block's UART takes it as a common data byte.*

26.4.4 Clocking and oversampling

The UART protocol is implemented using clk_scb as an oversampled multiple of the baud rate. For example, to implement a 100-kHz UART, clk_scb could be set to 1 MHz and the oversample factor set to '10'. The oversampling is set using the SCB_CTRL.OVS register field. The oversampling value is SCB_CTRL.OVS + 1. In the UART standard submode (including LIN) and the SmartCard submode, the valid range for the OVS field is [7,15].

In the UART transmit IrDA submode, this field indirectly specifies the oversampling. Oversampling determines the interface clock per bit cycle and the width of the pulse. This submode has only one valid OVS value–16 (which is a value of '0' in the OVS field of the SCB_CTRL register); the pulse width is roughly 3/16 of the bit period (for all bit rates).

In the UART receive IrDA submode (1.2, 2.4, 9.6, 19.2, 38.4, 57.6, and 115.2 kbps), this field indirectly specifies the 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 than or equal to two SCB input clock cycles are guaranteed to be detected by the receiver.

Pulse widths less than two clock cycles and greater than or equal to 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 SCB_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. See the registers list for more details on the OVS values for different baud rates.

26.4.5 Enabling and initializing UART

Do the following to program the UART:

1. Program protocol-specific information using the UART_TX_CTRL, UART_RX_CTRL, and 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 SCB_TX_CTRL and SCB_RX_CTRL
   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 SCB_TX_FIFO_CTRL and SCB_RX_FIFO_CTRL registers, respectively.
   a. Set the trigger level
   b. Clear the transmitter and receiver FIFO and Shift registers
4. Enable the block (write a '1' to the ENABLE bit of the SCB_CTRL register). After the block is enabled, do not change the control bits. 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 reenabled

Note that reenabling the block causes reinitialization and the associated state is lost (for example, FIFO content).

26.4.6 I/O pad connection

The following sections explains the I/O pad connection of UART

26.4.6.1 UART mode

26.4.6.1.1 Standard UART mode

Figure 167, Figure 168, Figure 169, and Table 211 list the use of the I/O pads for the Standard UART mode.

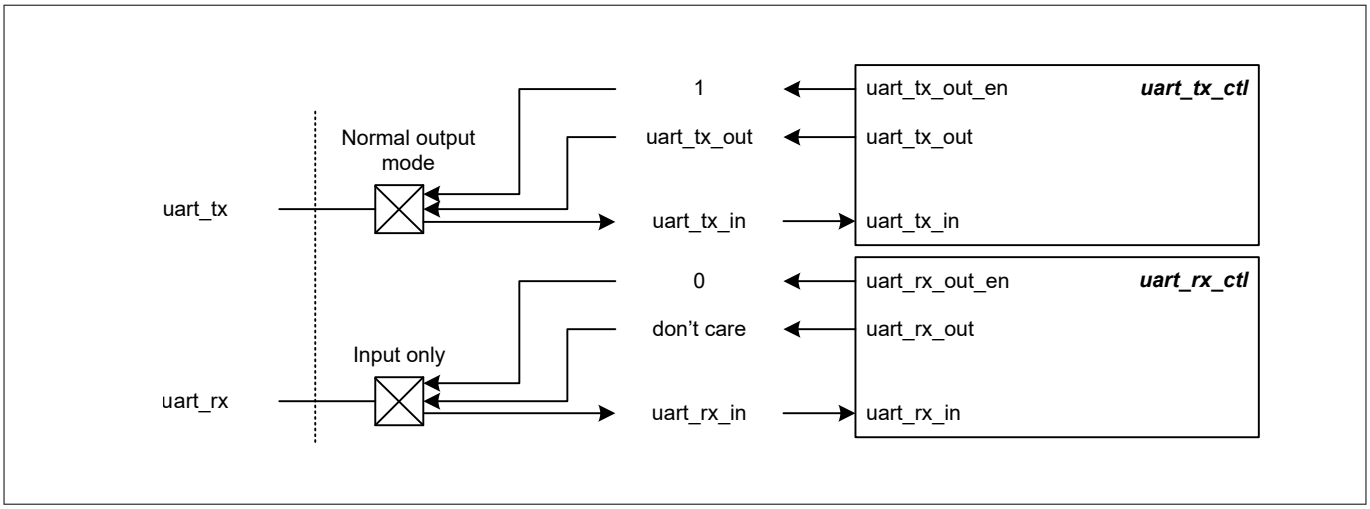


Figure 167 Standard UART mode I/O pad connections

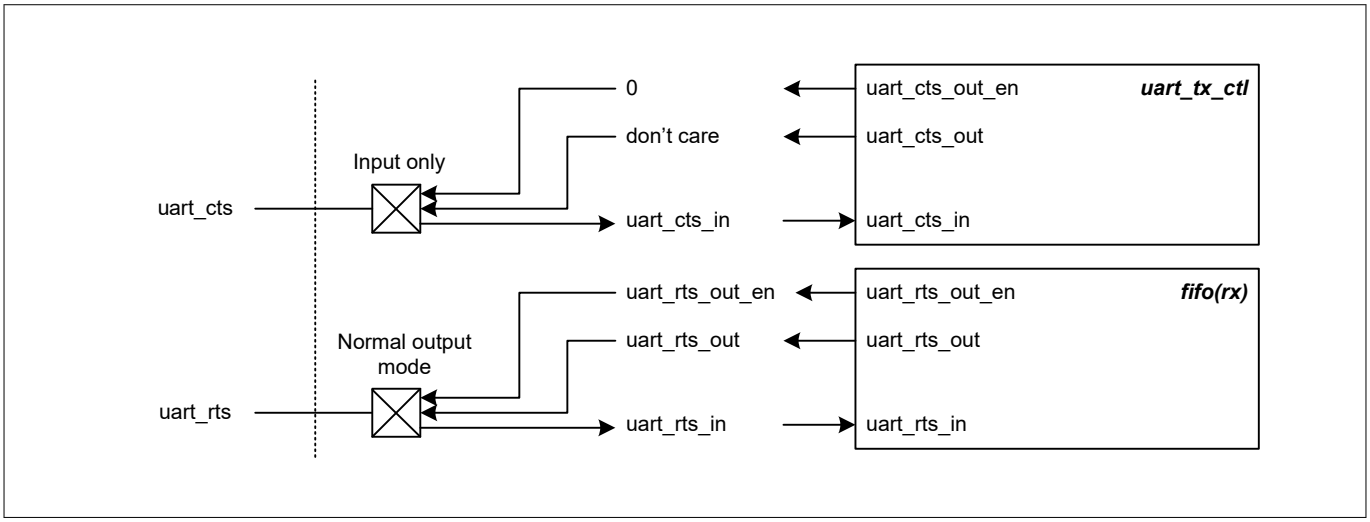


Figure 168 Standard UART mode, flow control I/O pad connections

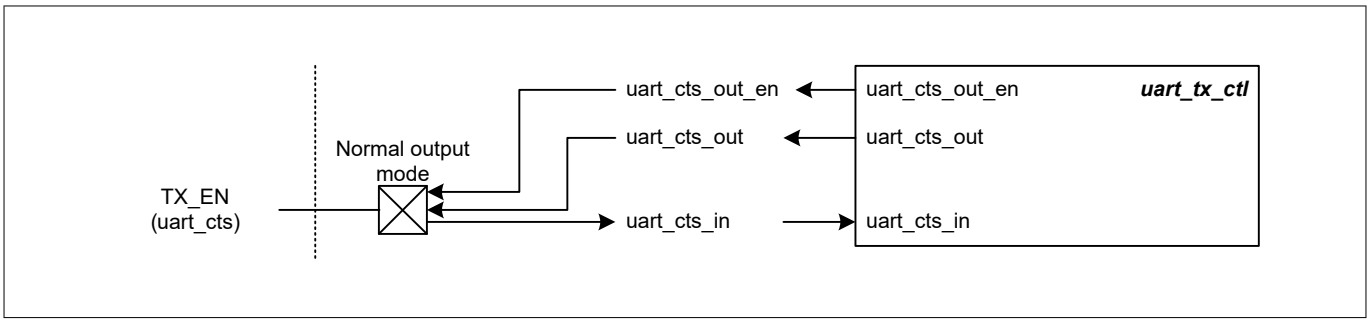


Figure 169 Standard UART mode, CTS reused as TX_EN for RS485

Table 211 UART I/O pad connection usage

I/O pads	Drive mode	On-chip I/O signals	Usage
uart_tx	Normal output mode	uart_tx_out_en uart_tx_out	Transmit a data element

(table continues...)

Table 211 (continued) UART I/O pad connection usage

I/O pads	Drive mode	On-chip I/O signals	Usage
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	Normal output mode	uart_rts_out_en uart_rts_out	Indicate SCB readiness to receive data
TX_EN(uart_cts)	Normal output mode	uart_cts_out_en uart_cts_out	Indicate SCB readiness to transmit data

26.4.6.1.2 Half-duplex mode

For half-duplex mode the transmitting and receiving pad is identical. The arbitration who is driving the line needs to be done on protocol level. An internal or external pull-up is required for half-duplex operation. Figure 170 and Table 212 list the use of the I/O pads for the half-duplex mode.

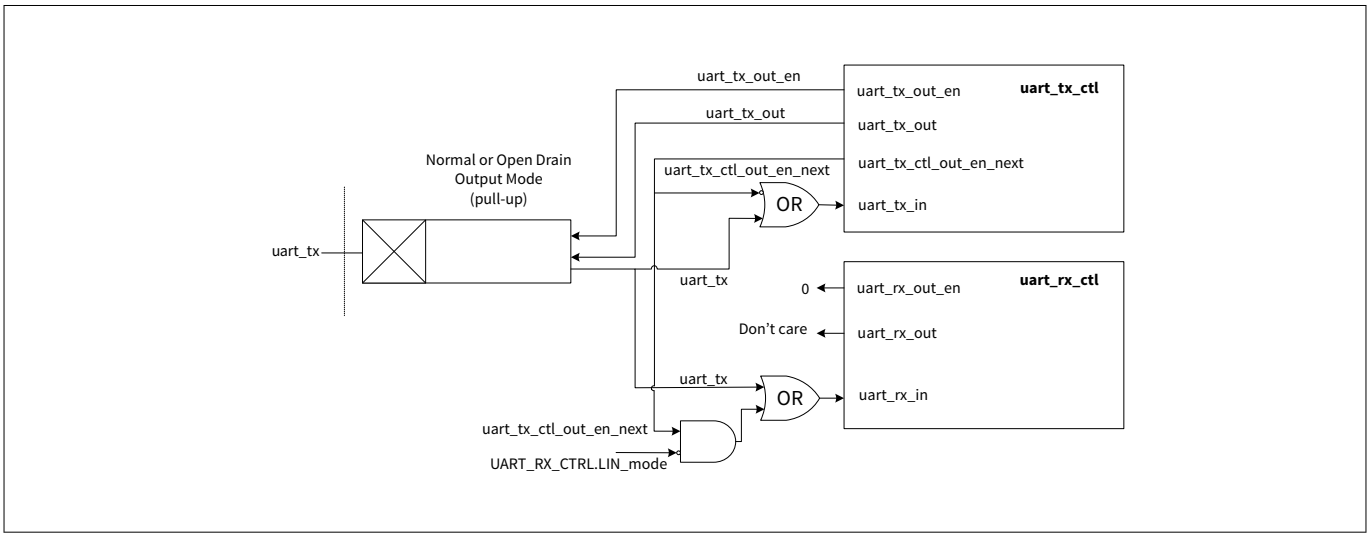


Figure 170 Half-duplex mode I/O pad connections

Table 212 Half-duplex mode I/O pad connections

I/O pads	Drive mode	On-chip I/O signals	Usage
uart_tx	Push-Pull Open-drain Input enabled	uart_tx_out_en uart_tx_out uart_tx_in	Transmit and receive a data element. While uart_tx_out_en is high (transmission), the uart_rx_in path is blocked and during receiving data the uart_tx_in path is blocked. Both are controlled by uart_tx_out_en.

26.4.6.2 SmartCard mode

Figure 171 and Table 213 list the use of the I/O pads for the SmartCard mode.

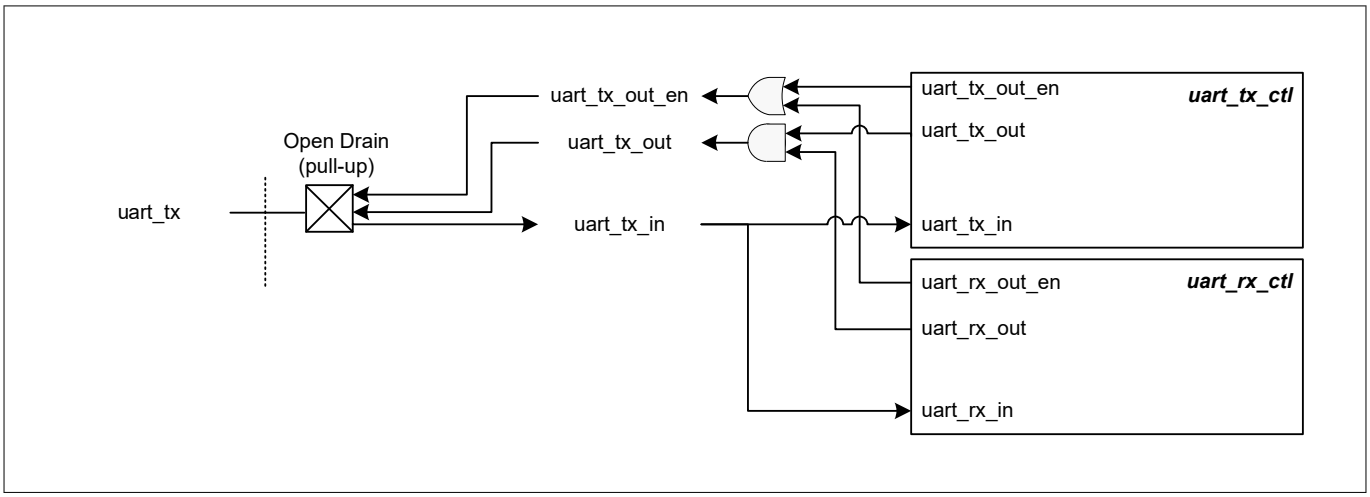


Figure 171 SmartCard mode I/O pad connections

Table 213 SmartCard mode I/O pad connections

I/O pads	Drive mode	On-chip I/O signals	Usage
uart_tx	Open drain with pull-up	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.

26.4.6.3 IrDA mode

Figure 172 and Table 214 list the use of the I/O pads for IrDA mode.

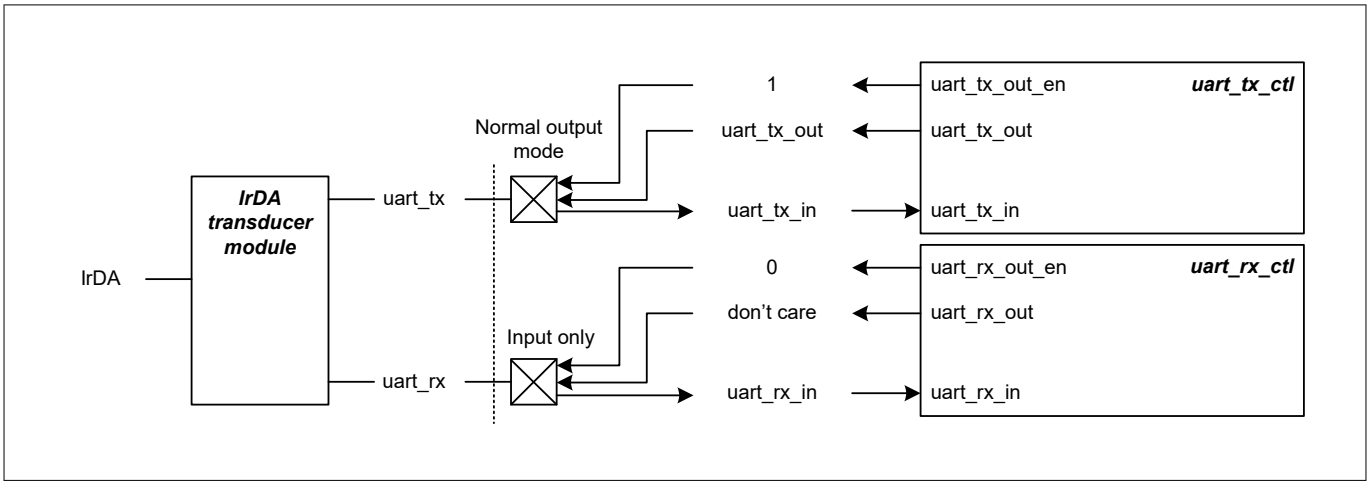


Figure 172 IrDA mode I/O pad connections

Table 214 IrDA mode I/O pad connections

I/O pads	Drive mode	On-chip I/O signals	Usage
uart_tx	Normal output mode	uart_tx_out_en uart_tx_out	Transmit a data element
uart_rx	Input only	uart_rx_in	Receive a data element

***Note**: **IrDA invalid use case scenario:** A glitch is observed on the TX pad when SCB is configured as master using IrDA mode and if the TX pad is '0', SCB is enabled and uart_tx_out_en = 1, and uart_tx_out = 0. The default value of the TX pad is '0', that means, before SCB is enabled, TX pad is '0'. On the pad there is a delay on the output signal which is set when the enable signal is set, thereby triggering a small glitch of around 400 ps. This behavior is caused by delays between pad enable and output signal.*

26.4.6.4 LIN mode

Figure 173 and Table 215 list the use of the I/O pads for LIN mode.

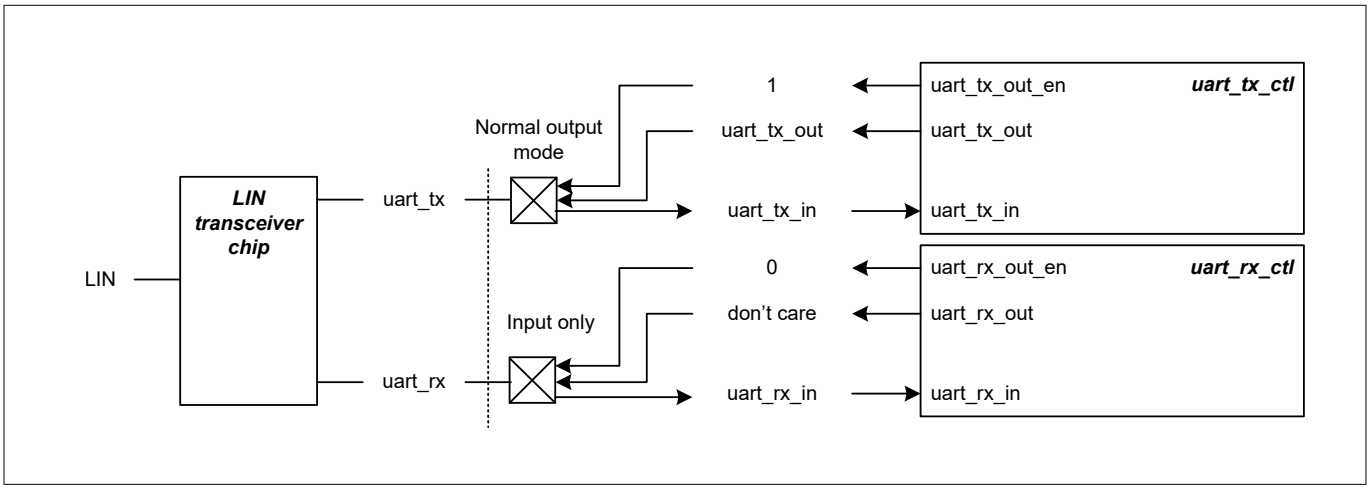


Figure 173 LIN mode I/O pad connections

Table 215 LIN mode I/O pad connections

I/O pads	Drive mode	On-chip I/O signals	Usage
uart_tx	Normal output mode	uart_tx_out_en uart_tx_out	Transmit a data element
uart_rx	Input only	uart_rx_in	Receive a data element

26.4.7 Registers summary

The UART interface is controlled using a set of 32-bit registers. For detailed information on these registers, see the PSOC™ Control C3M/P8, C3M/P7 registers reference manual.

26.5 Inter-Integrated Circuit (I2C)

This section explains the I2C implementation in the PSC3M/P8, PSC3M/P7 MCUs. For more information on the I2C protocol specification, see the I2C-bus specification available on NXP's Website. In the PSC3M/P8,

PSC3M/P7 MCUs, all SCB blocks support both I2C master and slave modes; only one SCB (SCB[0]) is available in Deep Sleep power mode and allows externally-clocked operations.

26.5.1 Features

This block supports the following features:

- Master, slave and master-slave functionalities
- Standard mode (100 kHz), fast mode (400 kHz) and fast plus mode (1 MHz)
- Standard 7-bit slave addressing
- 10-bit addressing supported through software
- Multi-master modes and bus arbitration
- Analog glitch filter
- Clock stretching
- Collision detection
- Programmable oversampling of I2C clock signal (SCL)
- FIFO, EZ, and CMD_RESP buffer modes
- Local loopback control
- General call address detection
- Wake up from sleep modes

26.5.2 General description

Figure 174 illustrates an example of an I2C communication network.

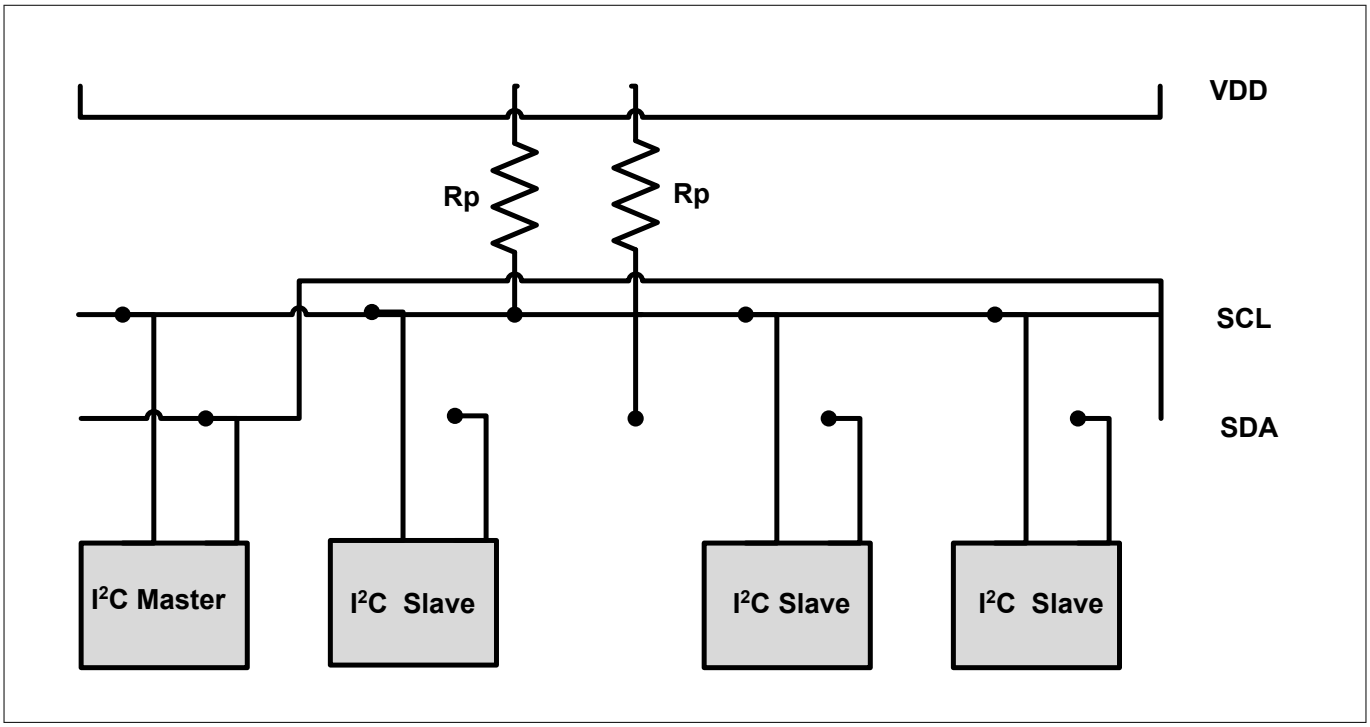


Figure 174 I2C interface block diagram

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.

26.5.3 Terms and definitions

Table explains the commonly used terms in an I2C communication network.

Table 216 Definition of I2C bus terminology

Term	Description
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

26.5.3.1 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 PSC3M/P8, PSC3M/P7 can and will stretch the clock.

26.5.3.2 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.

26.5.4 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 217 illustrates the I2C modes of operation.

Table 217 I2C modes

Mode	Description
Slave	Slave only operation (default)
Master	Master only operation
Multi-master	Supports more than one master on the bus

Table 218 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 218 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 the slave.

26.5.4.1 Write transfer

- A typical write transfer begins with the master generating a START condition on the I2C. The master then writes a 7-bit I2C 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 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 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

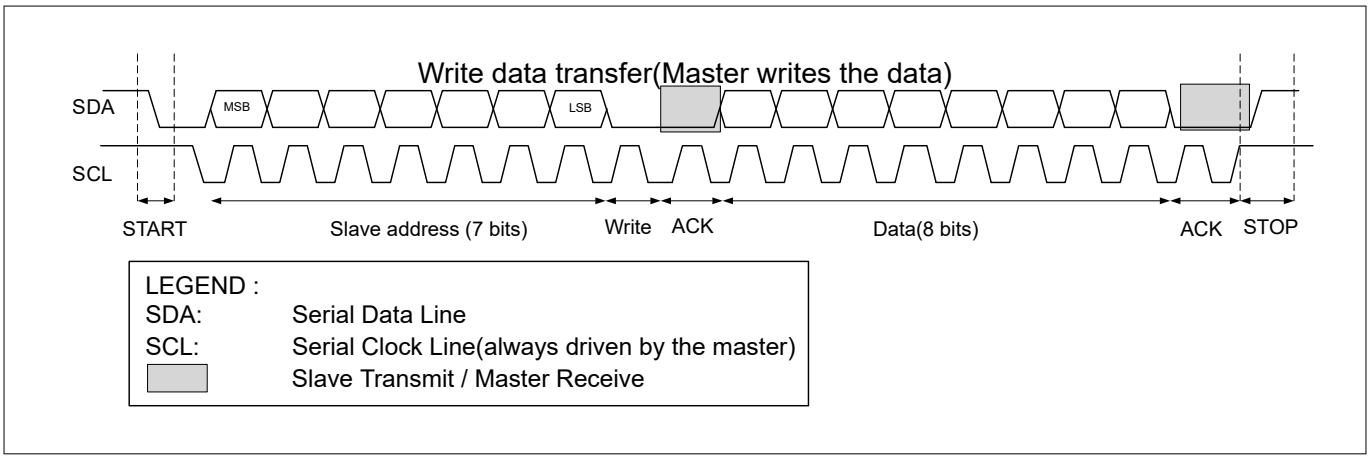


Figure 175 Master write data transfer

26.5.4.2 Read transfer

- A typical read transfer begins with the master generating a START condition on the I2C. 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. Combined data transfers consist of multiple individual transfers that are not separated by STOP events, but by repeated START events only

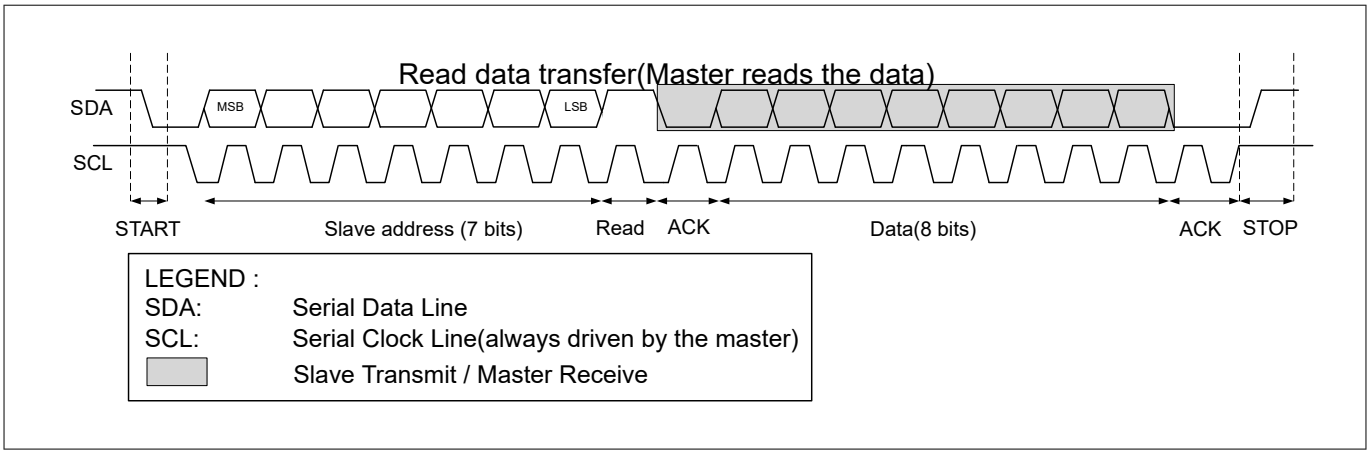


Figure 176 Master read data transfer

26.5.5 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.

26.5.5.1 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 64 elements deep with 16-bit data elements or 128 elements deep with 8-bit data elements. The width of the data elements is 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 CPU Active and CPU Sleep power modes. In the system Deep Sleep power mode, only the system Deep Sleep-capable SCB can wake up the device up from sleep through slave address match. For more information, see the "System Deep Sleep to CPU Active mode transition in FIFO buffer mode" section.

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`. When in debug mode, a read from this register behaves as a read from the `SCBx_RX_FIFO_RD_SILENT` register that is, data will not be removed from the FIFO.

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:

- For a TX FIFO, a trigger is generated when the number of entries in the transmit FIFO is less than TRIGGER_LEVEL
- For the RX FIFO, a trigger is generated when the number of entries in the FIFO is greater than the TRIGGER_LEVEL

Furthermore, several interrupt status bits are provided as well, which indicate if the FIFOs are full, empty, and so on.

System Deep Sleep to CPU Active mode transition in FIFO buffer mode

SCBx_CTRL.EC_AM_MODE = 1, SCBx_CTRL.EC_OP_MODE = 0, FIFO Mode. The following descriptions only apply to slave mode.

Master Write:

- 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 reenabled; this is when the clock stretch will be released
- 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 reenabled, do this before setting SCBx_I2C_S_CMD.S_ACK or SCBx_I2C_S_CMD.S_NACK
- 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 I2C 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:

- 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 reenabled before writing data into the TX FIFO
- 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 reenable after the PLL/FLL have stabilized

***Note**: When doing a repeated start after a write, wait until the UNDERFLOW interrupt status is asserted before setting the I2C_M_CMD.START bit and writing the new address into the TX_FIFO. Otherwise, the address in the FIFO will be sent as data and not as an address.*

26.5.5.2 EZI2C mode

The EZI2C protocol is a unique communication scheme built on top of the I2C protocol by Infineon Technologies AG. It uses a metaprotocol around the standard I2C protocol to communicate to an I2C 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 (an 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 I2C base and current addresses are provided through I2C_STATUS. At the end of a transfer, 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. EZI2C can be implemented through firmware or hardware. All SCBs can implement EZI2C through a firmware implementation in both CPU Active and CPU Sleep power modes. The Deep Sleep SCB can implement a hardware-and firmware-based EZI2C with a system Deep Sleep power 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**: A slave's base address is updated by the master and not by the CPU.*

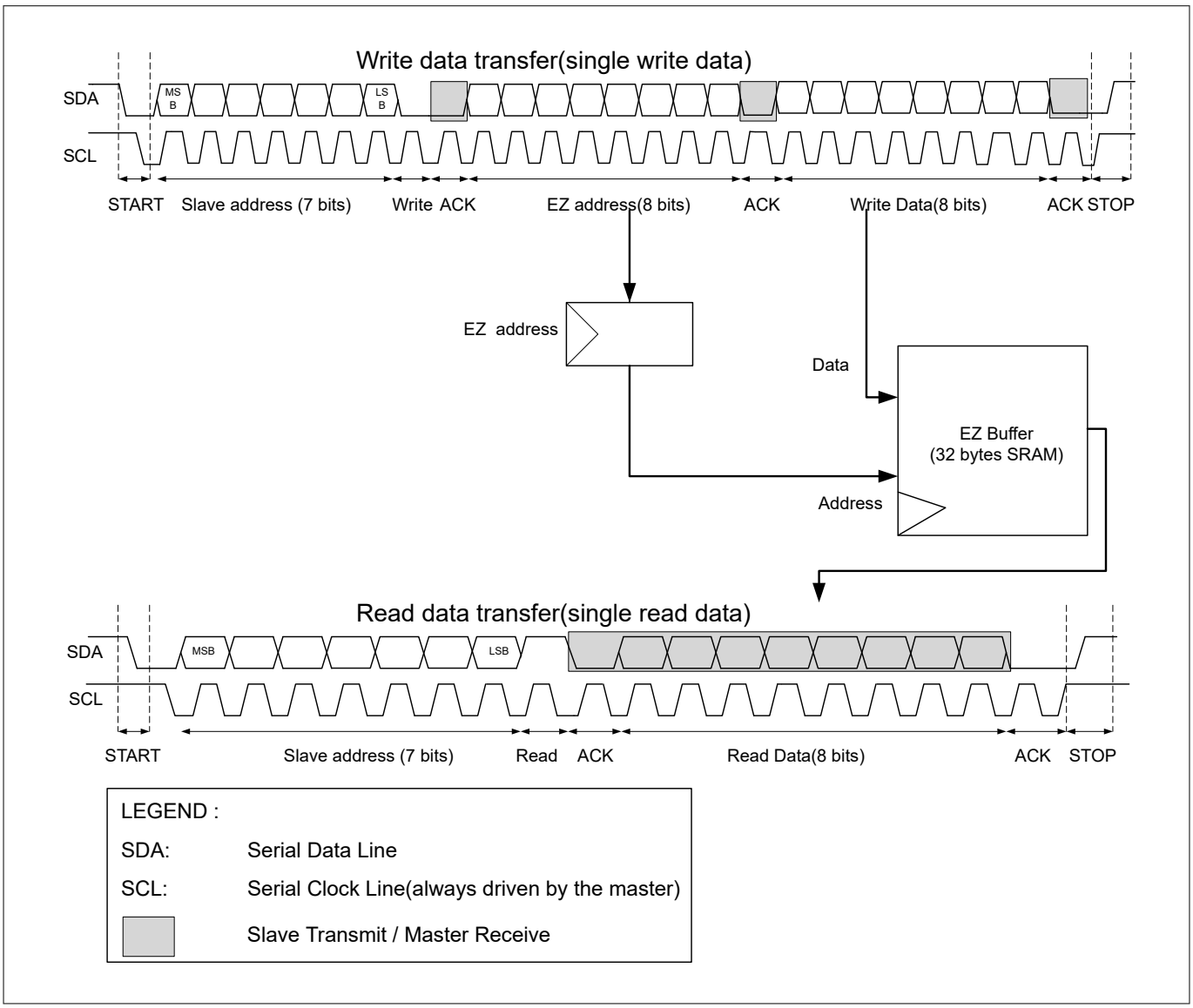


Figure 177 EZI2C Write and Read data transfer

System Deep Sleep to CPU Active mode transition in EZ buffer mode

SCBx_CTRL.EC_AM_MODE = 1, SCBx_CTRL.EC_OP_MODE = 0, EZ Mode.

- 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 reenabled this is when the clock stretch will be released

- 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 reenable after the PLL/FLL have stabilized

26.5.5.3 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 is 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 (I^2C 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. The current addresses can be viewed in SCBx_CMD_RESP_STATUS. At the end of a transfer (I^2C stop), the difference between a base and current address indicates how many read/write accesses were performed. This block provides interrupts to identify the end of a transfer, which can be found in SCBx_INTR_I2C_EC and SCBx_INTR_SPI_EC register sections.

Command-response mode operation is available in CPU Active and system 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

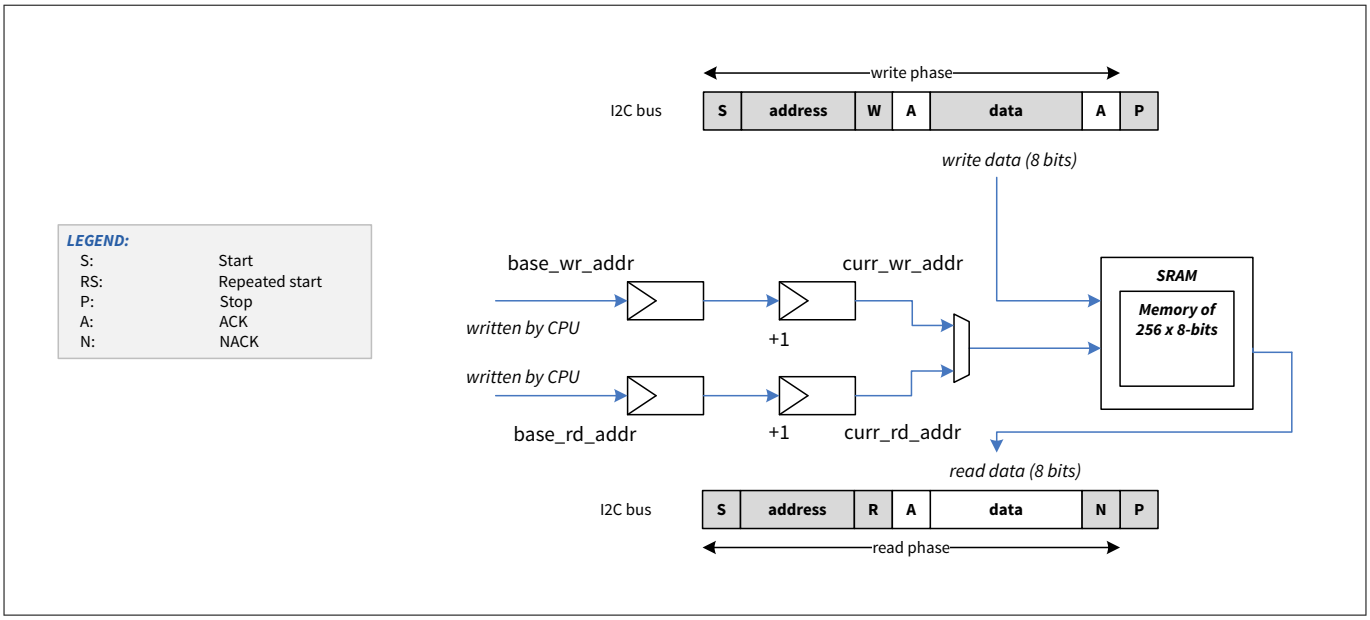


Figure 178 I2C command-response mode

***Note**: A slave's base address is updated by the CPU and not by the master.*

26.5.6 Clocking and oversampling

The SCB I2C supports both internally and externally clocked operation modes. Two bitfields (SCBx_CTRL.EC_AM_MODE and SCBx_CTRL_EC_OP_MODE) in the SCB_CTRL register determine the SCB clock mode. SCBx_CTRL.EC_AM_MODE indicates whether I2C address matching is internally (0) or externally (1) clocked. 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 internally (0) or externally (1) clocked. The externally clocked mode of operation is supported only in the I2C slave mode.

An internally-clocked operation uses the programmable clock dividers. For I2C, an integer clock divider must be used for both master and slave. For more information on system clocking, see the Clocking system section. The internally-clocked mode does not support the command-response mode.

***Note**: In the PSC3M/P8, PSC3M/P7, only one SCB supports an externally-clocked mode of operation.*

The SCBx_CTRL.EC_AM_MODE and SCBx_CTRL.EC_OP_MODE can be configured in the following ways.

- EC_AM_MODE is '0' and SCBx_CTRL.EC_OP_MODE is '0': Use this configuration when only CPU Active mode functionality is required
  - FIFO mode: Supported
  - EZ mode: Supported
  - Command-response mode: Not supported. The slave NACKs every slave address
- EC_AM_MODE is '1' and SCBx_CTRL.EC_OP_MODE is '0': Use this configuration when both CPU Active and system Deep Sleep functionalities 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 "hand over" 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 system Deep Sleep functionality interrupt brings the system into CPU Active power mode
  - FIFO mode: See FIFO mode
  - EZ mode: See EZI2C Mode
  - CMD_RESP mode: Not supported. The slave NACKs every slave address
- EC_AM_MODE is ‘1’ and SCBx_CTRL.EC_OP_MODE is ‘1’. Use this mode when both CPU Active and system Deep Sleep functionalities are required. When the slave is selected, SCBx_INTR_I2C_EC.WAKE_UP is set to ‘1’. The associated system Deep Sleep functionality interrupt brings the system into CPU Active power mode. When the slave is deselected, SCBx_INTR_I2C_EC.EZ_STOP and/or SCBx_INTR_I2C_EC.EZ_WRITE_STOP are set to ‘1’
  - FIFO mode: Not supported
  - EZ mode: Supported
  - CMD_RESP mode: Supported

Table 219 Clock configuration and mode support

Mode	SCBx_CTRL.EC_AM_MODE is '0'; SCBx_CTRL.EC_OP_MODEis '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
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 (I2C SCL). This allows for EZ and CMD_RESP mode functionality in CPU Active and system Deep Sleep power modes.

In CPU Active power mode, the memory buffer requires arbitration between an external interface logic(on I2C 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 or two serial interface clock/bit periods for the I2C. During this period, the internal logic is denied service to the memory buffer. The PSC3M/P8, PSC3M/P7 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 I2C 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 the 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, as the returned value is 0xFFFF:FFFF. This value is unique as nonfailing 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 access-es) 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.BLOCKED 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).

26.5.6.1 Glitch filtering

The PSC3M/P8, PSC3M/P7 SCB I2C 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 bit fields, only change the _SEL bit fields 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.

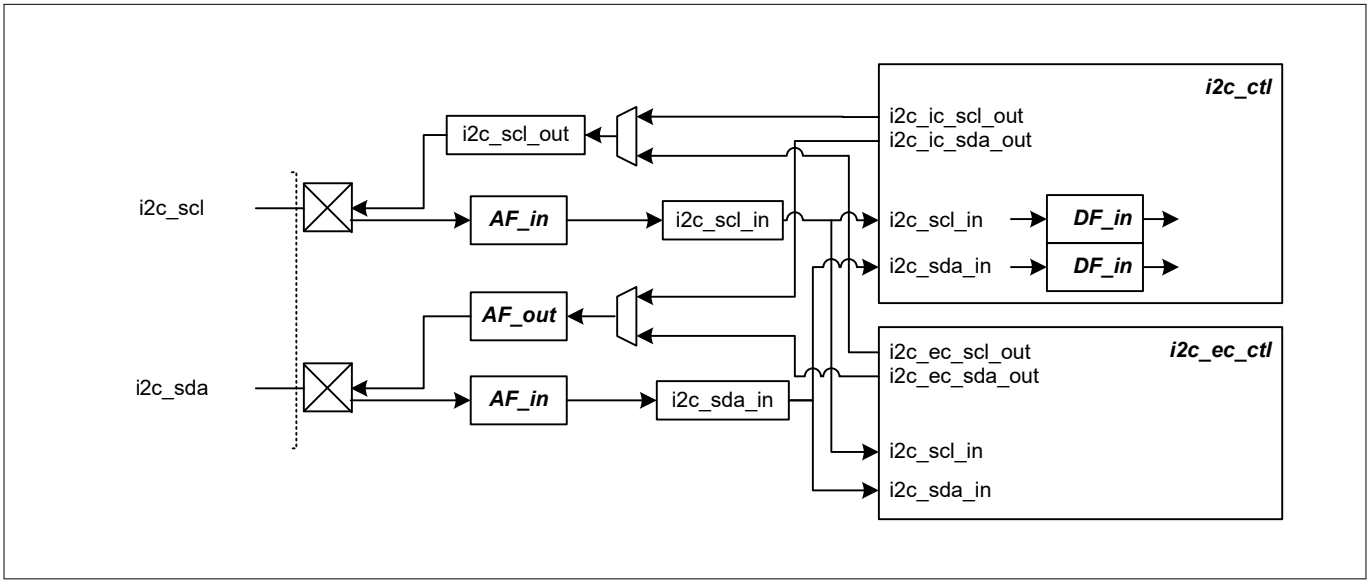


Figure 179 I2C glitch filtering connection

The following table lists the useful combinations of glitch filters.

Table 220 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	Comments
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

26.5.6.2 Oversampling and bit rate

Internally-clocked Master

The PSC3M/P8, PSC3M/P7 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 221 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
			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, 15]	[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 assumes worst-case conditions on the I2C bus. The following equations can be used to determine the settings for your own system. This involves 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.

Internally-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 assumes worst-case conditions on the I2C bus including the chip internal delay.

Table 222 SCB input clock requirements in 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

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.

26.5.7 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.

26.5.8 Enabling and initializing I2C

26.5.8.1 Configuring for I2C FIFO mode

Do the following to program the I2C interface:

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 '00' 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 I^2C mode. For a complete description of the I2C registers, see the PSOC™ Control C3M/P8, C3M/P7 registers RM

26.5.8.2 Configuring for EZ and CMD_RESP modes

To configure the I2C block for EZ and CMD_RESP modes, set the following I2C register bits:

1. For EZ mode, select the EZI2C mode by writing '1' to the SCBx_CTRL.EZ_MODE register
2. For CMD_RESP mode, select CMD_RESP mode by writing a '1' to the SCBx_CTRL.CMD_RESP register
3. Set the S_READY_ADDR_ACK (bit 12) and SCBx_I2C_CTRL.S_READY_DATA_ACK (bit 13) register

26.5.9 I/O pad connection

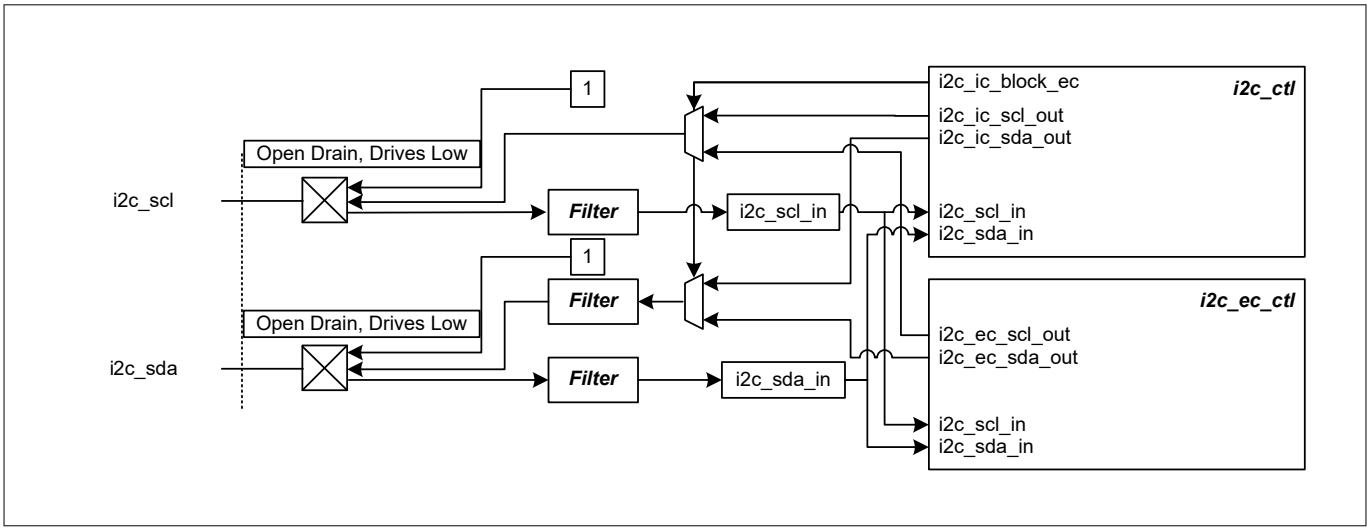


Figure 180 I2C I/O pad connections

Table 223 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_scl_out_en = 1	i2c_scl_out_en is fixed to high (enabled).
i2c_sda	Open drain drives low	i2c_sda_in	Receive data
		i2c_sda_out	Transmit data

(table continues...)

Table 223 (continued) I2C I/O pad descriptions

I/O pads	Drive mode	On-chip I/O signals	Usage
		i2c_sda_out_en = 1	i2c_sda_out_en is fixed to high (enabled).

26.5.10 Registers summary

The I2C interface is controlled by reading and writing a set of configuration, control, and status registers. For detailed information on these registers, see the PSOC™ Control C3M/P8, C3M/P7 registers reference manual.

26.6 Timer generation support

The Timeout Generation Support (TGS) functionality in SCB provides a robust mechanism to monitor and handle timeout events in SMBUS/PMBUS protocols. There are three 18-bit down-counters designed to offer flexibility and precision in protocol monitoring, while supporting various communication interfaces (I2C, SPI, UART). The following sections provide an in-depth understanding of the TGS feature, its use cases, behavior, and considerations.

26.6.1 Key features of TGS functionality

- **Event-Driven Timeout Monitoring:**
  - The TGS monitors the time between communication events (e.g., clock edges or start/stop conditions)
  - Timeout events (when a counter reaches 0x0000) trigger interrupts SCB (SPI, UART and I2C) and/or resets (only available in I2C Slave mode) to address stuck or incomplete communication states
  - Customers can configure the sensitivity and granularity of timeout behavior
- **Three 18-bit Down-Counters**:
  - Each counter operates independently
  - Gated by the tgs_clk_en input, providing fine-grained control of timeout monitoring
  - Reload, Gating, and Stop events influence the counter behavior
- **Flexibility with Protocol Monitoring**:
  - The TGS supports I2C, SPI (Motorola, TI, NS modes), and UART communication protocols
  - For I2C, the TGS ensures compliance with SMBUS/PMBUS specifications by measuring high/low periods of the SCL line (communication is stuck or incomplete)
  - For SPI, it accommodates protocol-specific behaviors like frame delineation and start/stop conditions
- **Design-Time Configurability**:
  - TGS feature available in both the **Active and Deep Sleep SCB instances**
- **Retention and Non-Retention Registers**:
  - Retention: TGS_CTL (control registers) and TGS_REL (reload values)
  - Non-Retention: TGS_CNT (counter values), reset upon power cycle or deep-sleep entry

26.6.2 Functional description

Counter Operation

- The counters begin decrementing from a reload value upon a reload event

- If the counter value reaches 0x0000:
  - An interrupt is triggered
  - The SCB can optionally reset (via tgs_rst) only in I2C slave mode
- A **Reload event** reloads the counter with its reload value. A **Stop event** stops the counter by writing zero into it and halts decrementing without triggering an interrupt
- Once the counter is enabled, the present counter value will be decremented until 0x0000. In case the initial value is already at 0x0000, nothing will happen until the first trigger event reloads the counter value. Each subsequent reload trigger event also reloads the counter

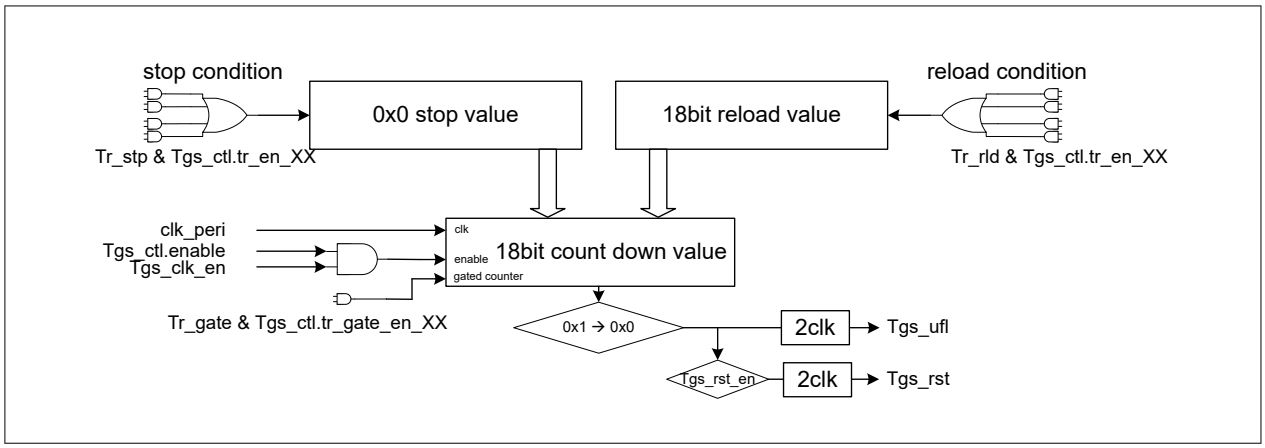


Figure 181 TGS Block diagram

Event Types

1. **Reload Event**:
   - Reloads the counter with the value stored in TGS_RELx.RELOAD
   - Typically triggered at the **start of a frame**
   - Subsequent reload events ensure the counter restarts
2. **Gating Event**:
   - Controls whether the counter decrements or pauses
   - When trigger gate (tr_tgs_gated) is high, the counter pauses. When low, decrementing resumes
3. **Stop Event**:
   - Stop event stops the counter by writing 0x0 into counter
   - Does not trigger an interrupt or reset
   - All events at 0x1 take precedence over the transition to 0x0

Interface Signals

- **Inputs**:
  - clock_tgs_en: Enables the clock for decrementing
- **Outputs**:
  - tr_tgs_reload[2:0]: Indicates that a reload event occurred for a specific counter
  - tr_tgs_stop[2:0]: Indicates that a stop event occurred
  - tr_tgs_gated[2:0]: Indicates when the counter is paused (gated)
  - **Generated Signals**:
    - tgs_underflow: Signals that a counter reached 0x0 (used for interrupts)
    - tgs_rst: Resets the SCB for 2 clock cycles to reinitialize the state machine

Priority of Counter Triggers:

1. Software Write

2. Stop event
3. Reload event
4. Gating Event
5. If the counter is not zero, it will decrement

***Note**: If there is a software write to the TGS counter, the frequency of TGS clock can be changed, since counter write has the priority. Therefore, the customer must expect that the bit-field of the register is updated. An uncertainty of -1 is expected when software is writing to the TGS counter, since frequency of TGS clock is changed*

26.6.3 Intended use case

Setup for a 25ms Timeout:

```
// Configure timeout counter
 Write(TGS_RELx.RELOAD = 0x9C40; TGS_CNTx.COUNT = 0x0000);
 Write(TGS_CTLx.ENABLE = 1b; TGS_CTLx.TR_I2C_P = 1b;
 TGS_CTLx.TR_I2C_S = 1b; TGS_CTLx.TR_FCLK = 1b);
```

- Wait for the first external clock edge.

- Start counting low periods between I2C start and stop conditions.

- On transition to `0x0000`, an interrupt triggers. software can react on the stuck-at clock

Additionally by setting TGS_CTLx.STATE_RST =1b, SCB (Only in I2C slave mode) disable for two clock cycles followed by an enable of SCB can be done, which resets the internal protocol state machine without software interaction. This sequence creates a 2-cycle window, where the module is disabled and will therefore not react on any external event. After reset the FIFO is also initialized

26.6.4 Special cases and protocol-specific behavior

1. **Special scenarios for SPI Triggers (valid for FIFO triggers only):**

   Expected behavior: The priority order of TGS triggers, from high to low: STOP -> RELOAD -> GATE -> decrementing. Due to pulse extenders were introduced, some TGS triggers can now be wider than 1 TGS clock. In case of the SPI, only FIFO (both RX and TX) triggers are affected due to pulse extension and can be extended on scenarios where TGS clock is greater than SCB clock. If a scenario where RELOAD(FIFO) and STOP(spi_clk) conditions are set at the same TGS cycle. The expectation is counter is stopped, because STOP has greater priority then RELOAD. But since RELOAD pulse is wider, counter will reload one clock cycle later. The behavior is expected as per design in Internal Clock mode

2. **Special scenarios for I2C Triggers (valid for FIFO and ACK triggers):**

   Expected behavior: The priority order of TGS triggers, from high to low: STOP -> RELOAD -> GATE -> decrementing. Due to pulse extenders were introduced, some TGS triggers can now be wider than 1 TGS clock. In case of the I2C, FIFO (both TX and RX) triggers and interface triggers (ACK on the rising and falling edge of SCL) are affected due to pulse extension and can be extended on scenarios where TGS clock is greater than SCB clock. If a scenario where RELOAD and STOP conditions are set at the same TGS cycle. The expectation is counter is stopped, because STOP has greater priority then RELOAD. But since RELOAD pulse is wider, counter will reload one clock cycle later. The behavior is expected as per design in IC mode

3. **SPI Triggers behavior for TI mode:**

   In TI mode, the spi_select signal is used to indicate the start of a data transfer. The start of data transfer is indicated by a high active pulse. So, spi_select is the indication of the start of the frame – there is no

indication of the end of the frame. In TGS_CTRL register, the counter can be configured to be gated inside or outside of the frame. And the end of frame cannot be declared based on the spi_select signal. Additionally, the polarity of the spi_select is different compared to Motorola and TI modes.

For TI mode,

- polarity=0, high pulse of the spi_select is indication of the start of the frame. Current implementation of the TGS will give TGS_RELOAD on falling edge of the spi_select, TGS_STOP on rising edge of the spi_select
- polarity=1 low (negative)pulse on spi_select is indication of the start of the frame. Current implementation of the TGS will give TGS_RELOAD on rising edge of the spi_select, TGS_STOP on falling edge of the spi_select

4. SPI Master behavior for National Semiconductors (NS) mode:

For the SPI Master, NS mode, with late sample enabled, active instance and valid only for IC mode, the falling edge of the spi_select should be the START condition (start of frame) and rising edge of the spi_select should be the STOP condition (end of frame). The customer must be aware of the fact that in case of SPI master SCB with an always running SPI clock, TGS clock(clk_tgs) should have same or higher frequency than SCB clock (clk_scb). For SCB clock (clk_scb) faster than TGS clock (clk_tgs), select line is de-asserted just before the next SPI clock rising edge (spi_clk is always running). The STOP condition gets cleared on the next rising edge of spi_clk. Since TGS clock (clk_tgs) is slow, so the counter misses the stop condition, so STOP trigger won't be detected

5. Deep-sleep to Active Transition Behavior

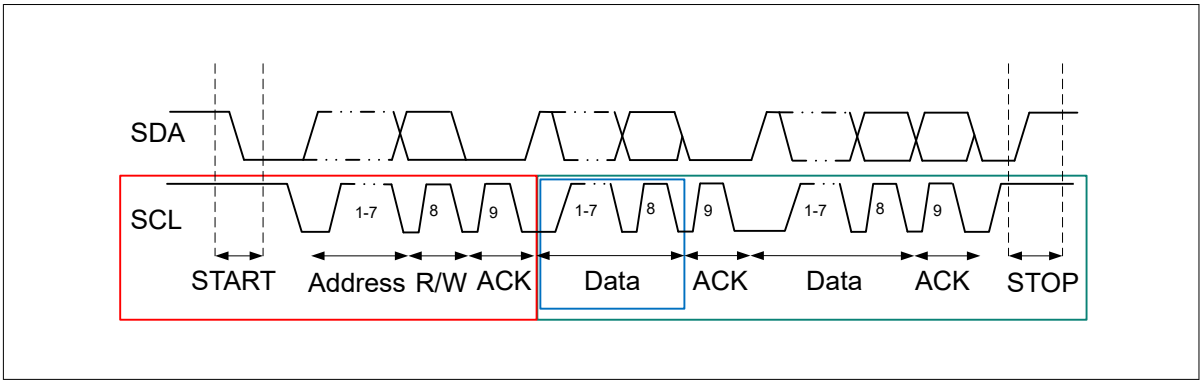


Figure 182 Deep-sleep to Active transition

In the above figure, the red box defines a deep-sleep power mode and the green box defines an active mode. In the case of transition from deep-sleep to active, address matching happens in deep-sleep, address acknowledgment can happen also in deep-sleep. SCB wakes up the system and the remaining of the transaction happens in active state. TGS functionality is only available in active state. There is a risk that TGS does not measure the stuck clock at the edge of the transition because of missing reload conditions which happened in deep-sleep state

A solution for a such scenario is to perform an automatic reload upon transition. For implementation purpose, there is an additional control bit TGS_CTLx.TR_DPSLP2ACT which enables a reload when power transition happens from deep-sleep to active

The counter is reloaded by the value in TGS_RELx.RELOAD like other reload events. Transition between states happen in microseconds range while the timeout is 25ms. So, same reload value can be used without a major effect of transition time

If the TGS_CTLx.TR_DPSLP2ACT is '0', the counter resets upon the power transition without a reload. The counter is idle until the next reload event occurs and proceeds with normal active functionality

6. Constant PMBus input voltage switching level

The I^2C specification defines the switching thresholds for logic levels as a percentage of the supply voltage (VDD). Specifically, the thresholds are:Low Level: Below 30% of VDD (0.3 × VDD).High Level:

Above 70% of VDD (0.7 × VDD).This means the switching thresholds dynamically scale with the supply voltage, making the I^2C spec inherently flexible for any VDD level. The PMBus (Power Management Bus) is built on top of the SMBus electrical layer. Therefore, it inherits the SMBus electrical characteristics, including its fixed switching thresholds. The SMBus specification introduces fixed voltage thresholds for its logic levels, regardless of the supply voltage. These thresholds are:Low Level: ≤ 0.8 V (VIL).High Level: ≥ 2.1 V (VIH).This fixed requirement is stricter and not dependent on the supply voltage, unlike I^2C. GPIO pads GPIO_STD or GPIO_OVT do not meet the fixed switching level requirements of SMBus/PMBus. only TTL or CMOS level can be selected

26.7 Multiple slave address support

Multiple Slave Address is implemented in UART and I2C.

In SCB, instead of a single address with a corresponding mask, 6 slave addresses are been defined with corresponding mask definitions. The SCB slave address is considered a match if any of the six independent addresses correspond to the address transmitted by the master on the bus.

The MMIO register RX_MATCH defines the slave address.

In UART multiprocessor mode, all 8 bits are used.

In I2C slave mode, only bits 7 down to 1 are used. This reflects the organization of the first transmitted byte in an I2C transfer: the first 7 bits represent the address of the addressed slave, and the last 1 bit is a read/write indicator ('0': write, '1': read).

Slave device address mask is a mask that specifies which of the ADDR field bits in the ADDR field take part in the matching of the slave address:

MATCH = ((ADDR & MASK) == ("slave address" & MASK)).

For MSA pairs, if MASK1=0 it means this pair is disabled and will not affect address matching.

The MMIO register RX_MATCH is extended with one additional address with the corresponding MASK definition. In the same way, two additional RX_MATCH registers are added with the same bit definition. This gives the customer the possibility to define up to 6 independent slave addresses with corresponding mask definitions.

For internal clocked FIFO I2C slaves, register CTRL.ADDR_ACCEPT is to control whether the received matched I2C slave address can be pushed into RXFIFO.

26.8 SCB interrupts

SCB supports interrupt generation on various events. The interrupts generated by the SCB block vary depending on the mode of operation.

Table 224 SCB interrupts

Interrupt	Functionality	CPU Active/System Deep Sleep	Registers
interrupt_master	master and SPI master functionality	CPU Active	SCBx_INTR_M, SCBx_INTR_M_SET, SCBx_INTR_M_MASK, SCBx_INTR_M_MASKED
interrupt_slave	I2C slave and SPI slave functionality	CPU Active	SCBx_INTR_S, SCBx_INTR_S_SET, SCBx_INTR_S_MASK, SCBx_INTR_S_MASKED
interrupt_tx	UART transmitter and TX FIFO functionality	CPU Active	SCBx_INTR_TX, SCBx_INTR_TX_SET, SCBx_INTR_TX_MASK, SCBx_INTR_TX_MASKED
interrupt_rx	UART receiver and RX FIFO functionality	CPU Active	SCBx_INTR_RX, SCBx_INTR_RX_SET, SCBx_INTR_RX_MASK, SCBx_INTR_RX_MASKED
interrupt_i2c_ec	Externally-clocked I2C slave functionality	System Deep Sleep	SCBx_INTR_I2C_EC, SCBx_INTR_I2C_EC_MASK, SCBx_INTR_I2C_EC_MASKED
interrupt_spi_ec	Externally-clocked SPI slave functionality	System Deep Sleep	SCBx_INTR_SPI_EC, SCBx_INTR_SPI_EC_MASK, SCBx_INTR_SPI_EC_MASKED
Interrupt_tgs	Timer generation support	CPU Active	SCBx_INTR_TGS, SCBx_INTR_TGS_SET, SCBx_INTR_TGS_MASK, SCBx_INTR_TGS_MASKED

26.8.1 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 the 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 PSOC™ Control C3M/P8, C3M/P7 registers 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 fewer 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)
- SPI Timer generation support (SCBx_INTR_TGS)
  - Timeout counter 0 underflow (TGS0)
  - Timeout counter 1 underflow (TGS1)
  - Timeout counter 2 underflow (TGS2)

26.8.2 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 the 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 PSOC™ Control C3M/P8, C3M/P7 registers. The UART blocks generate interrupts on the following events:

- UART TX (SCBx_INTR_TX)
  - TX FIFO has fewer 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 LIN or 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)
  - LIN baud rate detection is completed (BAUD_DETECT)
  - LIN break detection is successful (BREAK_DETECT)
- UART Timer generation support (SCBx_INTR_TGS)
  - Timeout counter 0 underflow (TGS0)
  - Timeout counter 1 underflow (TGS1)
  - Timeout counter 2 underflow (TGS2)

26.8.3 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 the 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 PSOC™ Control C3M/P8, C3M/P7 registers 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 slave received STOP at the end of multiple restarts when address matches at least once (I2C_STOP_ANY)
- I2C TX (SCBx_INTR_TX)
  - TX FIFO has fewer 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)
  - EZ memory (EZ data access) blocked (BLOCKED)
- I2C 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)
- I2C Externally-clocked (SCBx_INTR_I2C_EC)
  - Wake up request on address match (WAKE_UP)

  - I2C STOP detection at the end of each transfer (EZ_STOP)
  - I2C STOP detection at the end of a write transfer (EZ_WRITE_STOP)
  - I2C STOP detection at the end of a read transfer (EZ_READ_STOP)
  - I2C STOP detection at the end of multiple restarts when address matches at least once (EZ_STOP_ANY)
- I2C Timer generation support (SCBx_INTR_TGS)
  - Timeout counter 0 underflow (TGS0)
  - Timeout counter 1 underflow (TGS1)
  - Timeout counter 2 underflow (TGS2)

27 Programmable Power Control Accelerator (PPCA)

PPCA is a programmable subsystem with Cortex® M33 CPUs and advanced peripherals, primarily targeted towards implementing AC/DC, DC/AC, DC/DC, and Switch Mode Power Supply (SMPS) converters, such as Totem Pole Interleaved PFC, Half Bridge Interleaved LLC, Full Bridge interleaved LLC, Phase Shift Interleaved Full Bridge, Dual Active Bridge Dual Stage, and CLLC. It interfaces with WBG devices switching at high frequencies.

27.1 Features

- 2x Cortex®-M33 CPU cores, running at up to 200 MHz
- 112 KB RAM, organized into 5 memory blocks: 32 KB, 16 KB, 32 KB, 16 KB, and 16 KB
- 2x 12-bit ADCs, maximum 25 Msps, with multiplexed channels, configurable as up to 4 differential inputs (one with or without selectable gain) and/or 8 single-ended high-bandwidth inputs, or up to 7 single-ended high-bandwidth channels and 8 auxiliary low-bandwidth single ended inputs
- 2x 12-bit ADCs, maximum 25 Msps, configurable up to 2 differential input channels, one analog input channel with or without selectable gain, and/or 4 single-ended channels
- 9x fast analog comparators (<10 ns) with programmable thresholds provided by an embedded 12-bit DAC, supporting a maximum refresh rate of 50 MHz
- 2x 12-bit DAC outputs with up to 2 MHz bandwidth
- 1x IDAC current generator to sense up to 2 external resistances
- 6x ADC local filters modules connected to the ADC output
- 12x digital comparators connected to the ADC output
- 6x programmable and configurable 3-pole, 3-zeroes hardware filters
- 12x 32-bit high-resolution PWMs with <100 ps resolution and complementary outputs
- 14x 16-bit PWM with CPU clock resolution and complementary output
- Capture compare units (shared with PWMs)
- Configurable and programmable Events Processing Unit (EPU)
- Custom Logic Block Unit (CLB) implementing specific functions demanding very high computing power (digital current observer and matrix commutation)
- 2x CORDIC modules for hardware support of trigonometric and square root functions

27.2 Architecture

27.2.1 Block diagram

Figure 183 shows major subsystems and components of PPCA Subsystem.

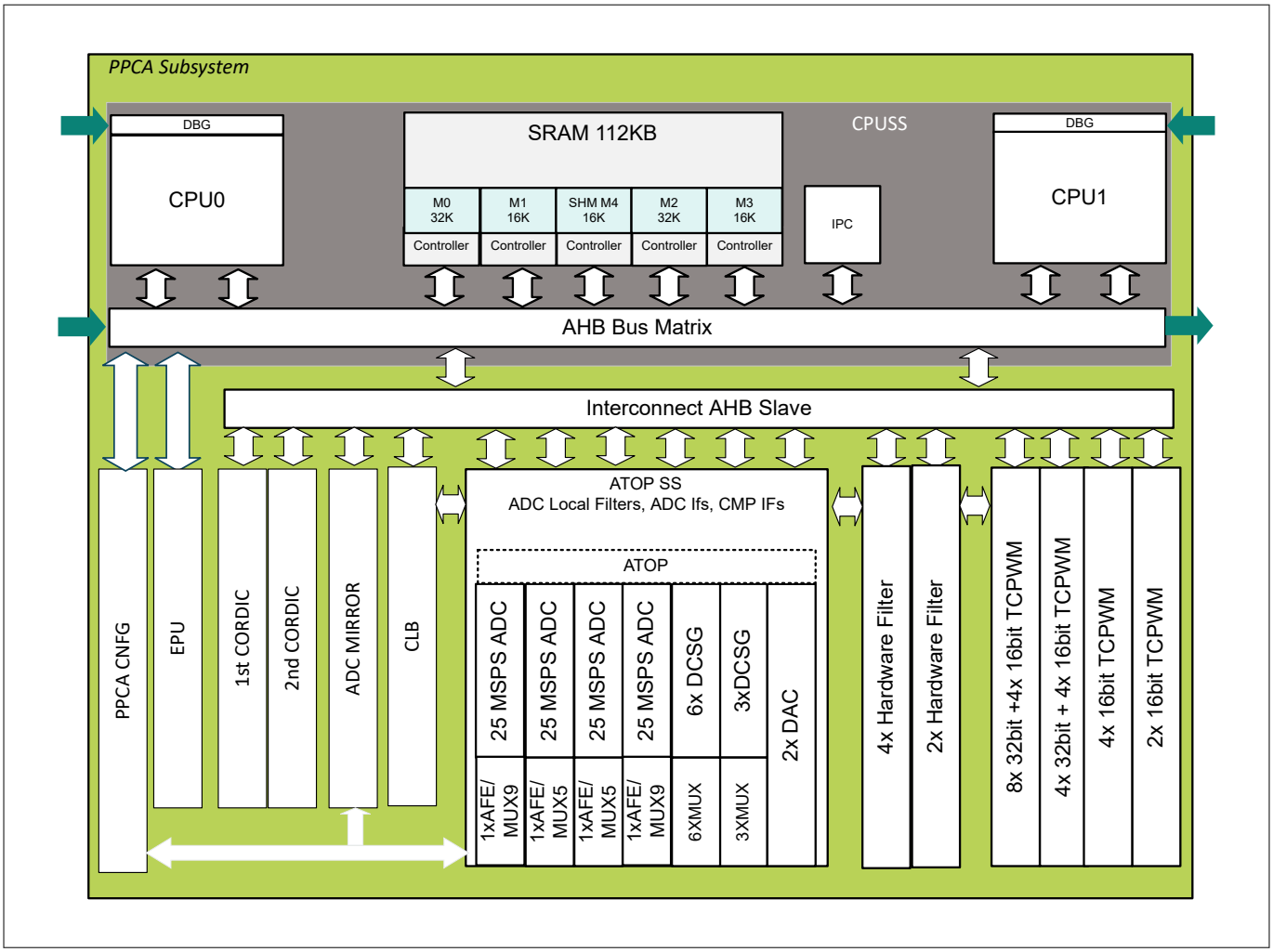


Figure 183 PPCA architecture

PPCA is partitioned in two main sections,

- PPCA CPU Subsystem
- PPCA Peripherals Subsystem

The CPU subsystem is based on an AMBA bus infrastructure, it embeds two dedicated Cortex®-M33 processors, local memory and shared memory, and the bus infrastructure to manage the access to the available embedded peripherals. The CPU cores are loosely coupled, each CPU Core operates with its own assigned resources with the only exception of the shared memory accessible by both CPUs.

The PPCA Peripherals section embeds all the peripherals aimed to support the implementation of Switch Mode Power Supply converters. All peripherals are AMBA bus compliant. The PPCA Peripherals Section also contains the bus infrastructure to interconnect the CPU SS exported peripherals AHB Bus Master interfaces with the instantiated PPCA Peripherals AHB Slave interfaces. Peripherals are assigned to one of the two CPU cores depending on the programmed configuration. The configuration is controlled through a register programmed only through the external AHB Slave input bus.

The PPCA subsystem is interfaced to the main CPU through an AHB bus as shown in the Figure 183. The CPU subsystem memories, as well as the embedded peripherals, are visible to an external AHB master agent through AHB slave interface.

27.2.2 Operating modes

PPCA is an Active mode only peripheral. Registers and memories are accessible in the Active mode only. SRAM PPU registers and memories can retain their content. Also defined retention registers retain their values in DeepSleep. Selected PPCA_CNFG and CPU/SRAMC configuration registers content will be retained in DeepSleep. Please refer register reference manual for register retention status.

Still all platform operating conditions and power modes are supported, although only in Active mode operation can be performed. Active digital power domain and PPCA input clock relation is reported in Operating modes.

Table 225 Power modes and clock frequencies

Power mode	VCCACT	Maximum PPCA_CLK	ATOP_CLK
OD	1.2 V, +5%, -10%	200 MHz	100 MHz
LP	1.1 V, +/-10%	150 MHz	75 MHz
MF	1.0, +/-10%	70 MHz	35 MHz
ULP	0.9 V, +/-10%	50 MHz	25 MHz

PPCA supports the low power modes shown in Operating modes.

Table 226 PPCA low power modes

Low Power mode	VCCACT	VCCRET	Data retention
DeepSleep	OFF	ON	Retention PPCA registers are retained
DeepSleep-RAM/ DeepSleep-OFF	OFF	OFF	PPCA RAMs are retained in DeepSleep-RAM.
Hibernate	OFF	OFF	

27.2.3 Clock

PPCA Subsystem requires one clock input (PPCA_CLK) that supports upto 200 MHz frequency. All internal peripherals clocks are generated from PPCA_CLK.

Main internal generated clocks of PPCA Subsystem are:

- CPU0_CLK – CPU0 clock signal
- CPU1_CLK – CPU1 clock signal
- ATOP_CLK – ATOP clock signal is to feed the ADCs and the DACs in the ATOP Subsystem

Clock used by other peripherals are also derived from the PPCA_CLK. ATOP_CLK is derived from the PPCA_CLK through a fixed integer divider of 2 with 50% duty.

The peripheral's provided clocks can be individually enabled or disabled, dynamically and statically. At the global reset assertion or power-up all the generated clocks are disabled. All the peripheral clocks can be disabled all together.

PPCA implements two clock domains:

- PPCA_CLK (CPU0_CLK, CPU_1CLK, PERIPHERALS_CLK, and DCSG_CLKs)
- ATOP_CLK (ADC_CLKs)

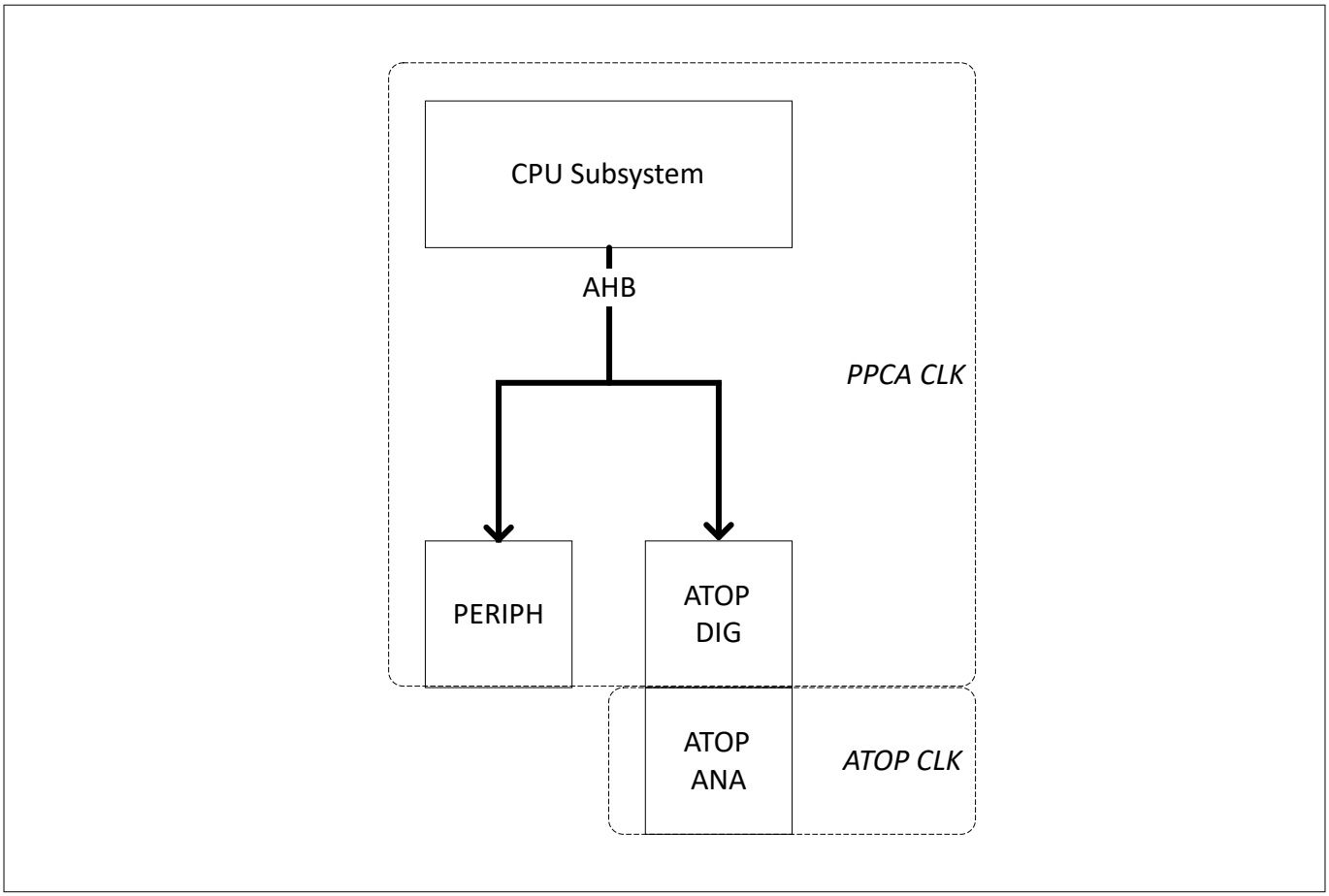


Figure 184 PPCA clock

As shown in the Figure 184, ATOP_CLK is used by the analog components in the ATOP sub system, such as ADCs and DACs all other components in the PPCA subsystem are clocked by PPCA_CLK.

27.2.4 Operation

1. Startup

At power up or global reset assertion, all the PPCA resources are initialized and kept under reset. As first step, the Main CPU should enable the PPCA operations by enabling the PPCA_EN bit in the PPCA CTL register. Then, before releasing the CPUs operation, the Main CPU should configure the PPCA resources by releasing the proper peripherals reset and peripherals clock. Configuring the peripherals allocation register initialization is not required if only CPU0 is intended to be used or if none of the CPUs are intended to be used. In this case the main CPU can directly operate with the PPCA resources

The main CPU is responsible for copying the PPCA code, from the FLASH to the PPCA CPUs code memory. During this time, it is recommended to keep the PPCA CPUs inactive by clearing CM33_CMD.ENABLED. CPUs should be enabled by asserting the bit CM33_CMD.ENABLED only after the CPUs memories have been initialized. The CPU resets can be released only after the memory initialization, and then the PPCA will be in operation

To save PPCA code memory space, it is recommended to delegate PPCA peripheral initialization to main CPU

2. Shutdown

In case a protection/alarm is triggered, the PPCA must be stopped and driven into a safe state until the protection/alarm is removed by a proper procedure. To stop (or shut down) the PPCA operation it is recommended to sequentially disable the used resources. Particularly, before disabling the PWM, the

PWM output should be driven to a safe state to avoid damaging the power stage connected to it. PWM kill feature can be used in some cases

3. **Entering DeepSleep**

   The PPCA supports the DeepSleep power mode but it is not operating in DeepSleep. When PPCA is ready to enter to deep sleep, it will assert the IRQ_RDY_FOR_DEEPSLEEP signal to inform the system that the PPCA is ready to go in deep sleep/sleep. Assertion of the IRQ_RDY_FOR_DEEPSLEEP is subsequent of the overall internal PPCA peripherals clock gating

4. **Exiting from DeepSleep**

   The PPCA exits from Deep Sleep power mode when the system restores the active power and releases the PPCA_CLK gated clock. Then for the PPCA to start running, main CPU has to copy the PPCA Core firmware to from FLASH to the RAM used by the PPCA Cores

27.3 CPU subsystem (CPUSS)

The CPU subsystem is based on the 32-bit Arm® Cortex®-M33 (CM33) CPU. It is designed for control, communication, short interrupt response time, high code density, and high 32-bit throughput while maintaining a strict cost and power consumption budget. This section provides only an overview of the Arm® Cortex® CPU in PSC3M/P8, PSC3M/P7 MCU. For details, see the Arm® documentation sets for Cortex®-M33.

27.3.1 Features

The PPCA has two Arm® Cortex®-M33 (CM33) CPU (CPU0, CPU1) having the following features:

- Maximum clock frequency of 200 MHz @ over drive (OD) mode
- Single-precision floating-point unit (FPU) and DSP extension
- Nested vectored interrupt controller (NVIC) integrated with the CPU core yielding low interrupt latency
- Fast integrated multiplier and hardware divider
- Four CPU Local Memories (32 KB, 16 KB, 32 KB, 16 KB), where the first two are strictly allocated to CPU0 (code and data) while the other two are usually allocated to CPU1 (code and data) or CPU0 (code extension and data extension) in case CPU1 is not used
- 16 KB Shared Memory accessible from CPU0 or CPU1, or used as memory data extension of CPU0 or CPU1
- Bus Matrix, 5 AHB slave input and 12 AHB master outputs

27.3.2 Architecture

CM33 is a 32-bit processor with a 32-bit data path, 32-bit registers, and a 32-bit memory interface. The CPU supports a wide variety of instructions in the Thumb instruction set. The CM33 CPU supports thread and handler operating modes (see Operating modes and privilege levels).

The CM33 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 2 to 11 cycles (depending on values)
- DSP instructions
- Complex memory load and store access
- Complex bit manipulation

The CM33 processor with FPU supports single precision arithmetic as defined by the FPv5 architecture. This is compliant with the ANSI/IEEE Std 754-2008, IEEE standard for binary floating-point arithmetic.

27.3.2.1 CPUSS

The PPCA CPUSS is a control, and communication CPU that is built on a Cortex®-M33(CM33) core(s) and runs at up to 200 MHz. The CM33 CPU is a 3-stage pipeline (3PIP) from Arm® based on the Armv8-M architecture having a low-power domain with rapid clock switching. CPUSS supports on-chip SRAM controller and memories. It provides C-AHB bus access from SRAM for code execution. In CPUSS, the active power domain becomes active or initialized first (by default, following the power-on reset (POR)). Refer the Device power modes section for more details.

27.3.2.1.1 CPU0

CPU0 is a Cortex®-M33 with the following configuration:

- Fast Integrated Multiplier
- Fast Integrated Hardware Divider
- DSP Unit
- FPU Unit
- No TZM
- No DAP
- No ETM
- Running at maximum frequency of 200 MHz

27.3.2.1.2 CPU1

CPU1 is a Cortex®-M33 with the following configuration:

- Fast Integrated Multiplier
- Fast Integrated Hardware Divider
- DSP Unit
- FPU Unit
- No TZM
- No DAP
- No ETM
- Running at same frequency of CPU0

27.3.2.2 CPU0/CPU1 memories

Five instances of the Memory Module are available inside the CPUSS subsystem. Each Memory Module hosts an AHB RAM wrapper (SRAM Controller) and a configurable RAM module. Each memory supports ECC capable to detect two errors and to correct one error. ECC detection of two errors shall trigger a Fault signal: Correctable Fault and Not Correctable Fault signals shall be propagated at the PPCA boundary. Non Correctable Fault signal shall bring the PWM or CLB output signals into the quiescent state. One error detection shall be automatically repaired.

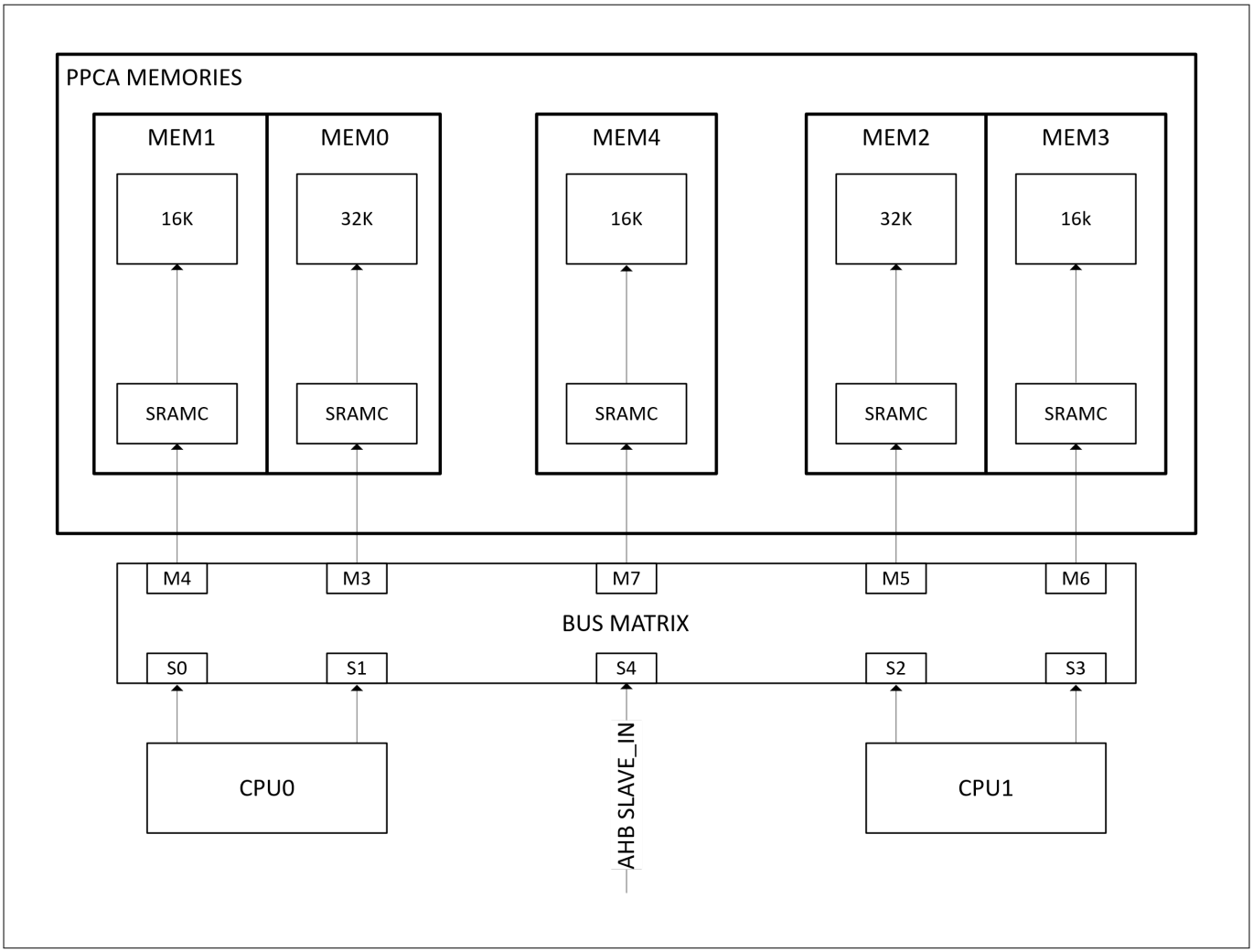


Figure 185 PPCA memories

All memories are retained in deep sleep mode, to ensure a faster wakeup.

27.3.2.3 Address and memory maps

The following figures show the Memory map visible from CPU0 and CPU1, and also from the external bus master interface.

27.3.2.3.1 Memory map visible from CPU0

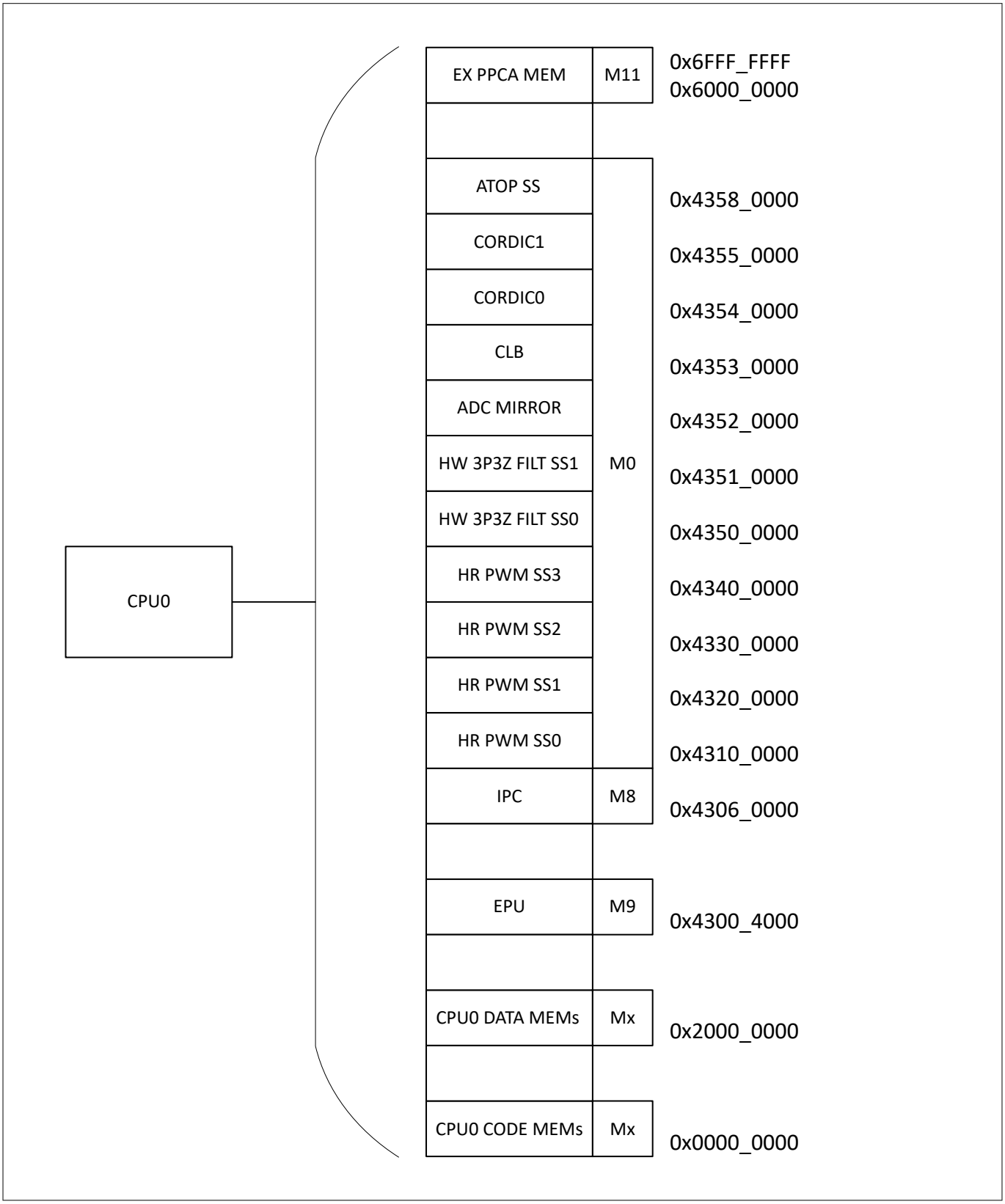


Figure 186 Memory map visible from CPU0

Register PPCA_CNFG_CTRL BIT0 = REMAP_MEM

Below is the memory map for the CPU0 DATA MEMs

CPU0-S	CPU0 DATA MEM
0x2000_0000	MEM1
0x2000_1FFF	
0x2000_2000	X
0x2000_3FFF	
0x2004_0000	MEM4
0x2004_3FFF	

Below is the memory map for the CPU0 CODE MEMs

CPU0-C	REMAP_MEM = 0	REMAP_MEM = 1
0x0000_0000	MEM0	MEM0
0x0000_7FFF		
0x0000_8000	X	MEM2
0x0000_FFFF		
0x0001_0000	X	MEM3
0x0001_1FFF		
0x0001_2000	X	MEM3[1]
0x0001_3FFF		
0x0001_4000	X	X
0x0001_7FFF		
0x0001_8000	X	X
0x0001_FFFF		
0x0002_0000	X	X
0x0002_1FFF		
0x0002_2000	X	X
0x0002_3FFF		

***Note**: 8 KB region available on select devices, check datasheet for details.*

27.3.2.3.2 Memory map visible from CPU1

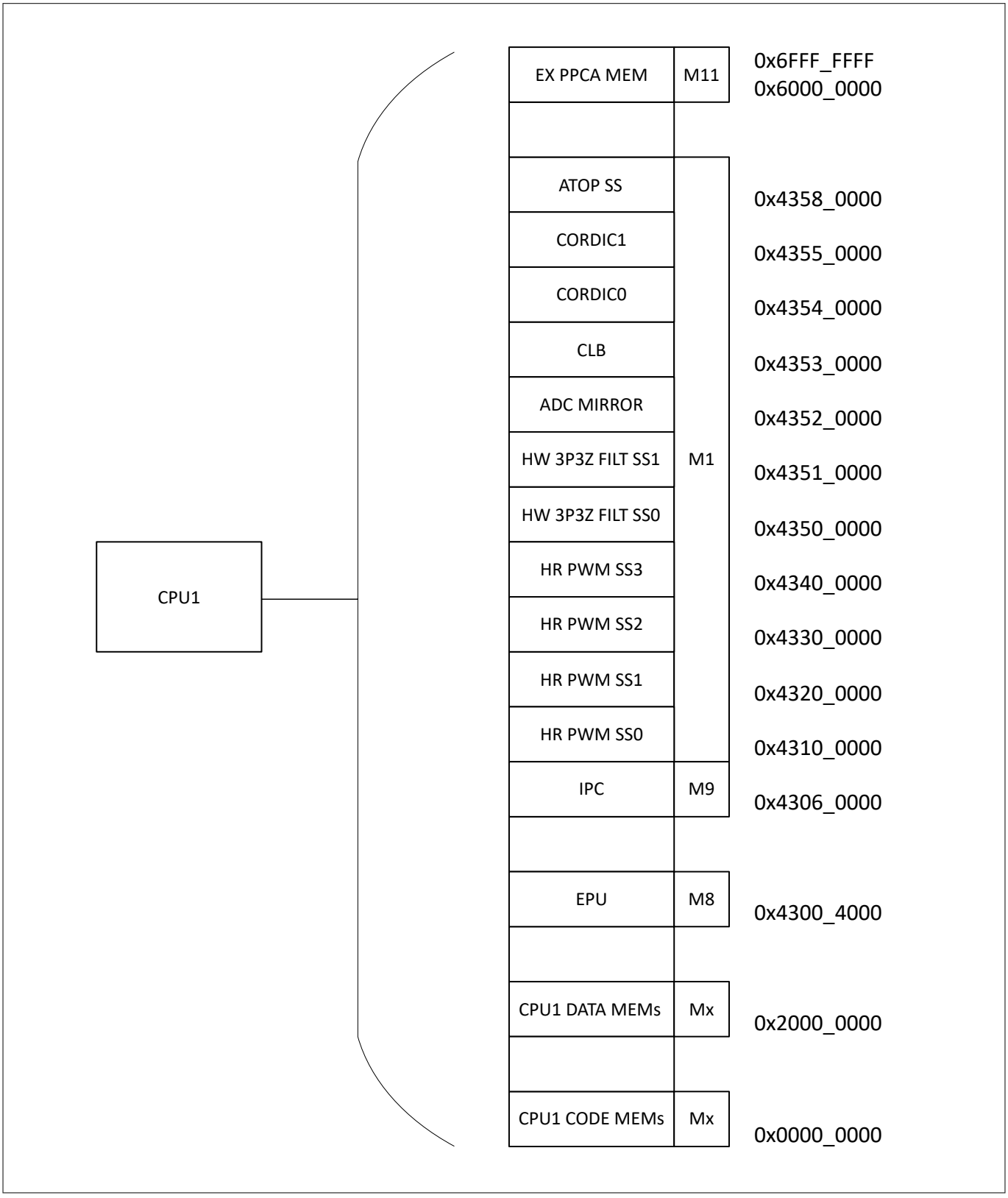


Figure 187 Memory map visible from CPU1

Register PPCA_CNFG_CTRL BIT0 = REMAP_MEM
Below is the CPU1 CODE MEMs

CPU1-C	REMAP_MEM = 0	REMAP_MEM = 1
0x0000_0000	MEM2	X
0x0000_7FFF		
0x0000_8000	X	X
0x0000_FFFF		

Below is the CPU1 DATA MEMs

CPU1-S	REMAP_MEM = 0	REMAP_MEM = 1
0x2000_0000	MEM3	X
0x2000_1FFF		
0x2000_2000	X	X
0x2000_3FFF		
0x2004_0000	MEM4	X
0x2004_3FFF		

***Note**: When REMAP_MEM is enabled, the code, data, and shared data are no longer linked to CPU1, making CPU1 unusable in this configuration.*

27.3.2.3.3 Overall Peripherals map table

Peripherals Name	Absolute address
EPU SS	0x4300_4000
IPC	0x4306_0000
HR PWM SS0	0x4310_0000
HR PWM SS1	0x4320_0000
HR PWM SS2	0x4330_0000
HR PWM SS3	0x4340_0000
HW 3P3Z FILT SS0	0x4350_0000
HW 3P3Z FILT SS1	0x4351_0000
ADC MIRROR	0x4352_0000
CLB	0x4353_0000
CORDIC0	0x4354_0000
CORDIC1	0x4355_0000
ADC_GRP0	0x4358_0000
ADC_GRP1	0x4358_2000
ADC_GRP2	0x4358_4000
ADC_GRP3	0x4358_6000
DCSG_GRP0	0x4358_8000

Peripherals Name	Absolute address
DCSG_GRP1	0x4358_A000

The PPCA CPU0, CPU1, and EXT_AHB_MASTER refer to the same absolute addresses for peripheral(s) access.

27.3.2.3.4 Addressing External PPCA address space

PPCA IP provides an AHB Master port mapped at 0x6000_0000 (only Non-Secure access), 256 MBytes (28-bits) address space, partitioned into four (4) sections, each one having a maximum of 64 Mbytes, to access external memories or peripherals.

AHB Master port is accessed only by CPU0 or CPU1.

AHB Master port will import the HMASTER ID from the parameter configured at IP integration (basically hard-coded 0x12 EXP_MS_0/PPCA_MS). The exported LSB of the HMASTER ID shall be set to “1” if the access has been issued from the debugger port of the CPUs.

The generated address shall be remapped as per the following table:

Internal base address	Remapped Base Address
0x6000_0000	0x02000000
0x6400_0000	0x24000000
0x6800_0000	0x42000000
0x6C00_0000	Not accessible

***Note**: An access to Not accessible address results in HRESP ERROR (that is, illegal access address)*

There is timeout implemented to monitor the access-elapsed time to avoid the PPCA master agent (CPUs and Debuggers) to be stalled in case the external accessed resource in not available. On the occurrence of this condition the HRESP ERROR signal shall be asserted, the access terminated, and a fault error signal reported to the PPCA boundary. A further access to a stalled interface shall be terminated immediately and the fault error signal reported.

27.3.2.3.5 Remap function

The Remap function logic enables the External AHB Master input interface to access the internal PPCA CPUs assignable peripherals at unique address, even though the PPCA peripherals are allocated to one or the other CPU address space depending on the programmed configuration defined by the CNFG register.

Basically, the Remap function logic steers an external peripheral access to the AHB matrix port corresponding to the CPU, it is assigned to, so that it is not blocked in the slave interconnect.

External Base AddressHADDR [22:0]	PPCA_CNFG_CNFG0[15:0]		Remapped Base Address HADDR_remap [23:0]	Address mapped to PPCA peripheral
	BIT_POS	BIT_VALUE 0 = CPU01= CPU1		
0x10_0000	0	0	0x10_0000	HR PWM SS0
0x20_0000	1	0	0x20_0000	HR PWM SS1
0x30_0000	2	0	0x30_0000	HR PWM SS2
0x40_0000	3	0	0x40_0000	HR PWM SS3
0x50_0000	4	0	0x50_0000	HW 3P3Z FILT SS0
0x51_0000	5	0	0x51_0000	HW 3P3Z FILT SS1

External Base AddressHADDR [22:0]	PPCA_CNFG_CNFG0[15:0]		Remapped Base Address HADDR_remap [23:0]	Address mapped to PPCA peripheral
	BIT_POS	BIT_VALUE 0 = CPU01= CPU1		
0x52_0000	6	0	0x52_0000	ADC MIRROR
0x53_0000	7	0	0x53_0000	CLB
0x54_0000	8	0	0x54_0000	CORDIC1
0x55_0000	9	0	0x55_0000	CORDIC2
0x58_0000	10	0	0x58_0000	ADC_GRP0
0x58_2000	11	0	0x58_2000	ADC_GRP1
0x58_4000	12	0	0x58_4000	ADC_GRP2
0x58_6000	13	0	0x58_6000	ADC_GRP3
0x58_8000	14	0	0x58_8000	DCSG_GRP0
0x58_A000	15	0	0x58_A000	DCSG_GRP1
0x10_0000	0	1	0x90_0000	HR PWM SS0
0x20_0000	1	1	0xA0_0000	HR PWM SS1
0x30_0000	2	1	0xB0_0000	HR PWM SS2
0x40_0000	3	1	0xC0_0000	HR PWM SS3
0x50_0000	4	1	0xD0_0000	HW 3P3Z FILT SS0
0x51_0000	5	1	0xD1_0000	HW 3P3Z FILT SS1
0x52_0000	6	1	0xD2_0000	ADC MIRROR
0x53_0000	7	1	0xD3_0000	CLB
0x54_0000	8	1	0xD4_0000	CORDIC1
0x55_0000	9	1	0xD5_0000	CORDIC2
0x58_0000	10	1	0xD8_0000	ADC_GRP0
0x58_2000	11	1	0xD8_2000	ADC_GRP1
0x58_4000	12	1	0xD8_4000	ADC_GRP2
0x58_6000	13	1	0xD8_6000	ADC_GRP3
0x58_8000	14	1	0xD8_8000	DCSG_GRP0
0x58_A000	15	1	0xD8_A000	DCSG_GRP1

The Remap function logic does not require an additional programming, the logic is implemented in hardware.

27.3.2.3.6 AHB Master ID remapped

The AHB Master ID associated to the available AHB Master agents and entering in the Bus Matrix is remapped as per the following table, to enable the EPU embedded Bus Matrix, to uniquely recognize which AHB Master is accessing the EPU resources.

AHB Master IF	New AHB Master ID [4:0]
EXTERNAL AHB MASTER IF	[0000x]

AHB Master IF	New AHB Master ID [4:0]
CPU C0 AHB MASTER IF	[0001x]
CPU S0 AHB MASTER IF	[0010x]
CPU C1 AHB MASTER IF	[0011x]
CPU S1 AHB MASTER IF	[0100x]

27.3.3 Operating modes and privilege levels

The CPU has thread and handler operating modes and can run in either thumb or debug operating states. In addition, the processor can limit or exclude access to some resources by executing code in privileged or unprivileged mode. See the Armv8-M Architecture reference manual for more information about the modes of operation and execution.

27.3.3.1 Operating modes

The processor operates in two modes:

Thread Mode:

Used to execute application software. The processor enters Thread mode on reset, or as a result of an exception return. Thread mode supports both privileged and unprivileged execution levels.

Handler Mode:

Used to handle exceptions. The processor enters Handler mode when an exception is triggered. Handler mode always runs in privileged state.

27.3.3.2 Operating states

The processor can operate in the thumb or debug state:

- The thumb state is the state of normal execution running 16-bit and 32-bit half-word aligned thumb instructions
- The debug state is the state when the processor is in halting debug

27.3.3.3 Privilege levels

- Unprivileged:

  The software has limited access to the move from the general-purpose register to the special register (MSR) and move from special register to general-purpose register (MRS) and cannot use the change processor state/disable interrupts (CPSID) and change processor state/enable interrupts (CPSIE) instructions. It cannot access the system timer, NVIC, or system control block. Restricted access to memory or peripherals
- Privileged:

  The software can use all the instructions and has access to all resources In the thread mode, the CONTROL register controls whether software execution is privileged or unprivileged. In the 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 the privileged software.

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

  In the 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.

27.3.4 Instruction set

The Cortex®-M33 (CM33) CPU supports all 16-bit Thumb instructions defined by the Armv8-M architecture. In addition, it implements the T32 instruction set based on Thumb-2 technology, ensuring high code density and reduced program memory requirements.

The CM33 CPU implements the following from Armv8-M:

- All base instructions
- All instructions in the main extension
- All instructions in the DSP extension
- All single-precision instructions and double-precision load/store instructions in the floating-point extension

For more details, refer to Section 3 – The Cortex®-M33 Instruction Set in the Arm® Cortex®-M33 Devices Generic User Guide.

27.3.5 Registers summary

The CM33 CPU has the following 32-bit core registers.

The PPCA CM33 cores, CPU0 and CPU1, have TrustZone disabled. As a result, these cores do not support secure or non-secure execution modes.

Table 227 CM33 core register set summary

Name	Type[1)]	Required privilege[2)]	Reset value	Description
R0-R12	RW	Either	UNKNOWN	R0-R12 are general-purpose registers for data operations.
MSP (R13)	RW	Either	-[3)]	The stack pointer (SP) is register R13. In Thread mode, the CONTROL register indicates the stack pointer to use, Main Stack Pointer (MSP) or Process Stack Pointer (PSP).
PSP (R13)	RW	Either	UNKNOWN	
MSPLIM	RW	Privileged	0x00000000	The stack limit registers limit the extent to which the MSP and PSP registers can descend respectively.
PSPLIM	RW	Privileged	0x00000000	MSPLIM_NS for the non-secure state MSPLIM_S for the secure state
LR (R14)	RW	Either	UNKNOWN	The Link Register (LR) is register R14. It stores the return information for subroutines, function calls, and exceptions.
PC (R15)	RW	Either	-[3)]	The Program Counter (PC) is register R15. It contains the current program address.

(table continues...)

Table 227 (continued) CM33 core register set summary

Name	Type[1]	Required privilege[2]	Reset value	Description
xPSR	RW	Either	-[4]	The Program Status Register (PSR) combines: Application Program Status Register (APSR) Interrupt Program Status Register (IPSR) Execution Program Status Register (EPSR) These registers provide different views of the PSR.
APSR	RW	Either	UNKNOWN	The APSR contains the current state of the condition flags from previous instruction executions.
IPSR	RO	Privileged	0x00000000	The IPSR contains the exception number of the current ISR.
EPSR	RO	Privileged	-[4]	The EPSR contains the thumb state bit and the execution state bits for the If-then (IT) instruction, and interruptible-continuable instruction (ICI) field for an interrupted load multiple or store multiple instruction.
PRIMASK	RW	Privileged	0x00000000	The PRIMASK register prevents activation of exceptions with configurable priority.
BASEPRI	RW	Privileged	0x00000000	The BASEPRI register defines the minimum priority for exception processing.
FAULTMASK	RW	Privileged	0x00000000	The FAULTMASK register prevents activation of all exceptions except for NON-MASKABLE INTERRUPT (NMI) and optionally secure HardFault.
CONTROL	RW	Privileged	0x00000000	The CONTROL register controls the stack used, and optionally the privilege level, when the processor is in the thread mode.

1) Describes the access type during program execution in the thread mode and the handler mode. Debug access can differ.
2) An entry of either means privileged and unprivileged software can access the register.
3) Soft reset to the value retrieved by the reset handler.
4) Bit[24] is the T-bit and is loaded from bit[0] of the reset vector. All other bits are reset to 0.

The CM33 floating-point unit (FPU) has the following registers:

- Thirty-two 32-bit single-precision registers, S0 to S31. These registers can also be addressed as sixteen 64-bit double-precision registers, D0 to D15
- The four FPU control and status registers are as follows:
  - FPCCR – Floating-point Context Control Register
  - FPCAR – Floating-point Context Address Register
  - FPSCR – Floating-point Status Control Register
  - FPDSCR – Floating-point Default Status Control Register

For more information on how these registers are used, see the Arm® Cortex®-M33 reference manual.

Use the MSR and MRS instructions to access the PSR, PRIMASK, CONTROL, FAULTMASK, and BASEPRI registers. The following tables show how the PSR bits are assigned. See the Combined Program Status Register section in the Arm® Cortex®-M33 reference manual for detailed information.

Table 228 CM33 APSR bit assignments

Bit	Name	Usage
31	N	Negative flag
30	Z	Zero flag
29	C	Carry or borrow a flag
28	V	Overflow flag
27	Q	DSP overflow and saturation flag
26:20	-	Reserved
19:16	GE[3:0]	Greater than or equal flags
15:0	-	Reserved

Table 229 CM33 IPSR bit assignments

Bit	Name	Usage
31:9	-	Reserved
8:0	Exception number	This is the number of the current exceptions: • 0 = Thread mode • 1 = Reset • 2 = NMI • 3 = HardFault • 4 = MemManage • 5 = BusFault • 6 = UsageFault • 7 = SecureFault • 8-10 = Reserved • 7-10 = Reserved • 11 = SVCall • 12 = DebugMonitor • 13 = Reserved • 14 = PendSV • 15 =SysTick • 16 = IRQ0 • ... 495 = IRQ479

Table 230 CM33 EPSR bit assignments

Bit	Name	Usage
31:27	-	Reserved, RES0

(table continues...)

Table 230 (continued) CM33 EPSR bit assignments

Bit	Name	Usage
26:25, 15:10	ICI	Interruptible-continuable instruction bits
26:25, 15:10	IT	Indicates the execution state bits of the IT instruction
26:25, 11:10, 15:12	ECI	Exception continuation flags for beat-wise vector instructions. This field encodes which beats of the in-flight instructions have completed.
24	T	Thumb state bit
23:16	-	

27.4 Bus infrastructure

The bus infrastructure inside PPCA block forms the backbone of communication between various components within the PPCA IP. It is a bus network that connects different IP blocks, such as processors, SRAM memories, peripherals, and accelerators. The bus infrastructure provides a standardized interface for data and control signals, enabling seamless interaction and data exchange among these components inside PPCA block. The bus infrastructure implements the AMBA bus infrastructure of the CPUSS subsystem using the Arm® bus matrix (Bus Matrix) and AHB slave mux.

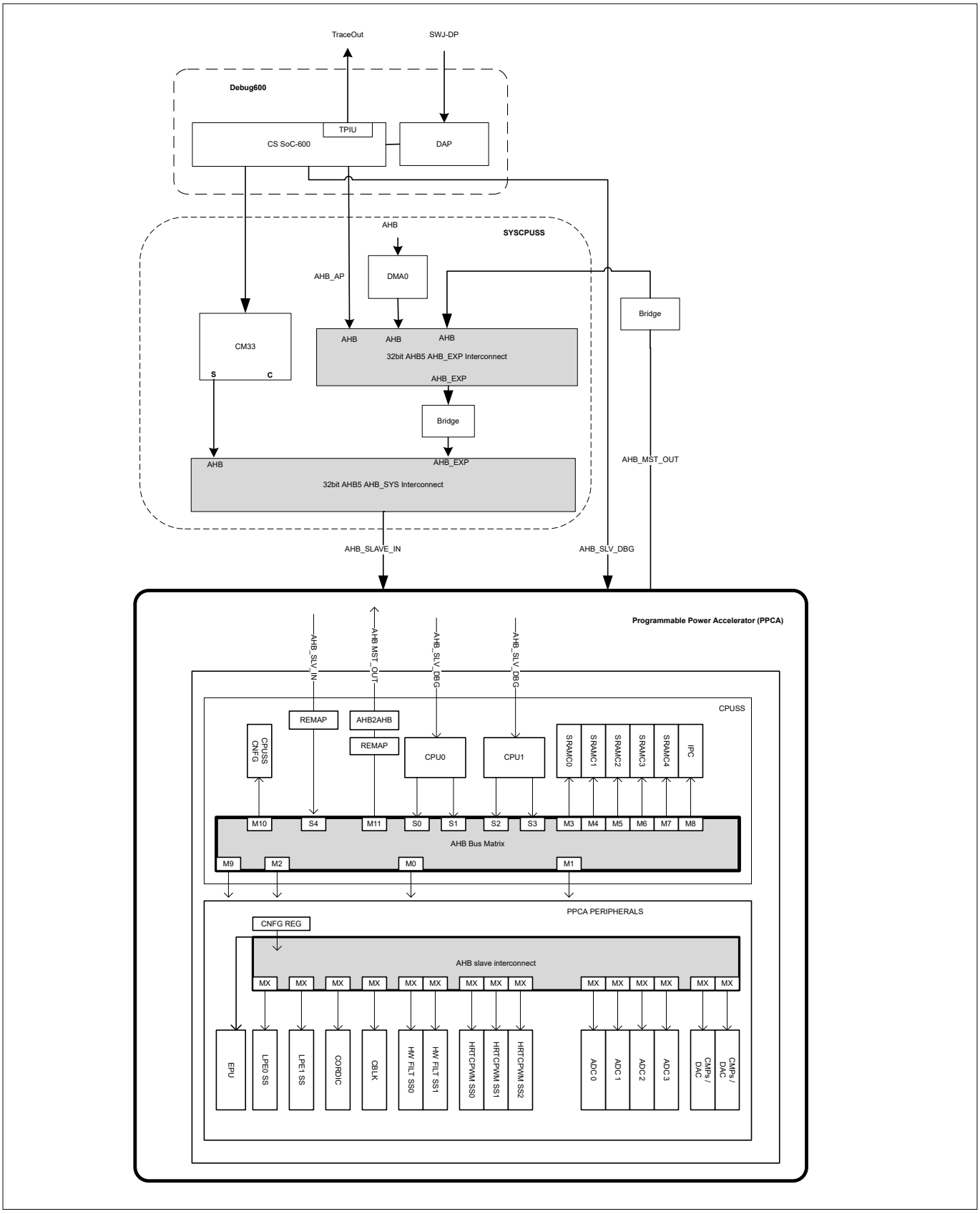


Figure 188 PPCA bus infrastructure

27.4.1 AHB bus matrix

The AHB Bus Matrix interconnects the following masters and slaves.

Masters:

- Two (2) for CPU0 (code bus and system bus) inside CPUSS
- Two (2) for CPU1 (code bus and system bus)inside CPUSS
- One (1)Bridge from External CPU subsystem

Slaves:

- Five (5)SRAM connected to local and shared memories inside CPUSS
- One (1) Bridge to the external CPU subsystem
- One (1) connected to the IPC
- One (1) connected to the PPCA CPUSS CNFG
- Two (2) ports to the secondary peripheral interconnect
- One (1) EPU
- One (1) PPCA-CNFG

The Bus Matrix supports a fixed priority arbitration, where the main CPU sub-system as more priority and the CPU1 lower priority.

27.4.1.1 AHB bus matrix interconnect

The general Interconnection between AHB Slave ports (Sx) and AHB Master ports (Mx) are provided in the following picture. For example, AHB Slave port S4 is assigned to the external AHB Slave Interface, AHB Slave S0 (code bus) and S1 (system bus) to the CPU0, AHB Slave S2 (code bus) and S3 (system bus) to the CPU1.

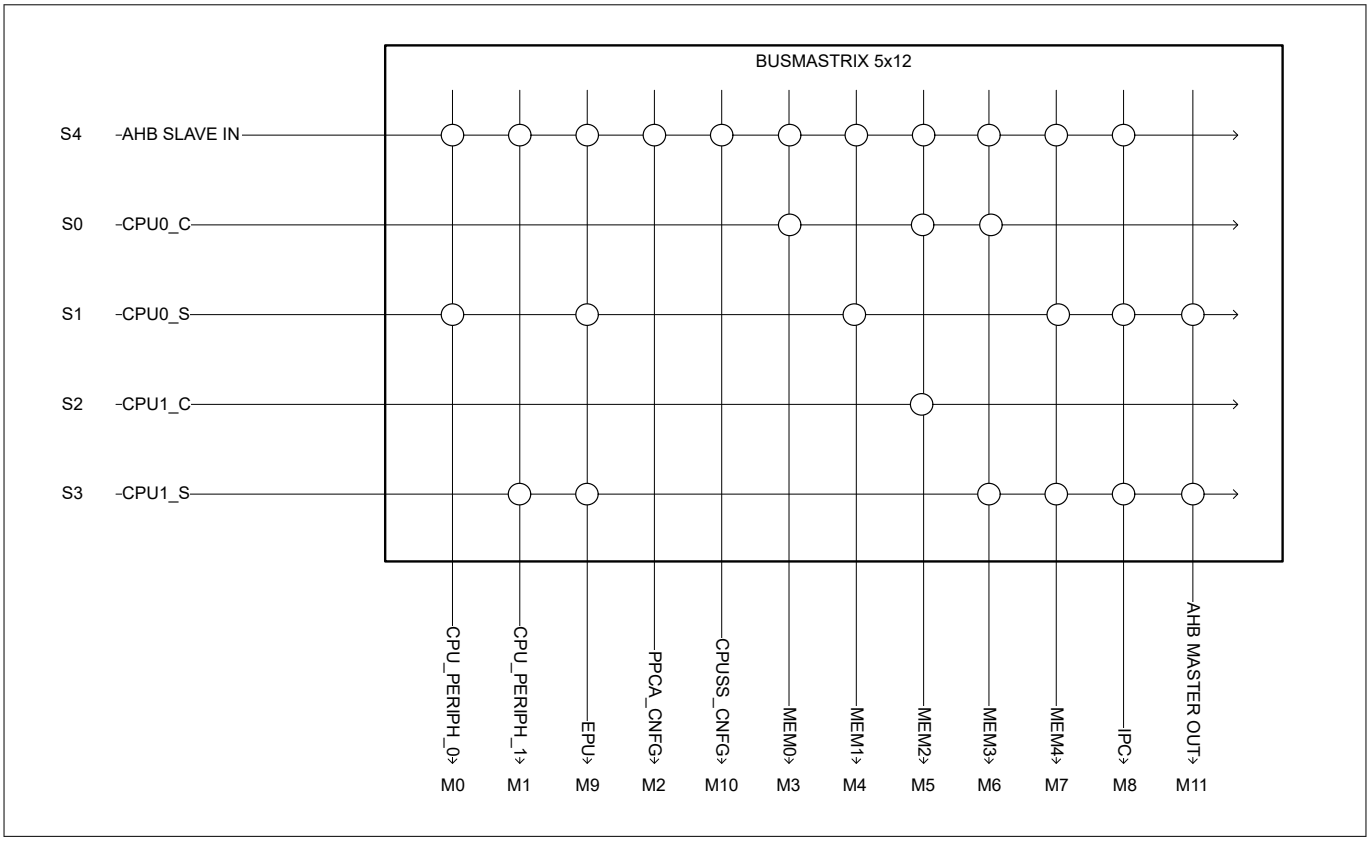


Figure 189 AHB bus matrix interconnect

27.4.2 AHB slave interconnect

The AHB Bus slave interconnect module provides an infrastructure to connect the AHB Bus Master coming from the CPUSS subsystem to the AHB Slave Peripherals assigned to a specific CPU Core without arbitration.

The AHB slave interconnect block simplified diagram can be seen below:

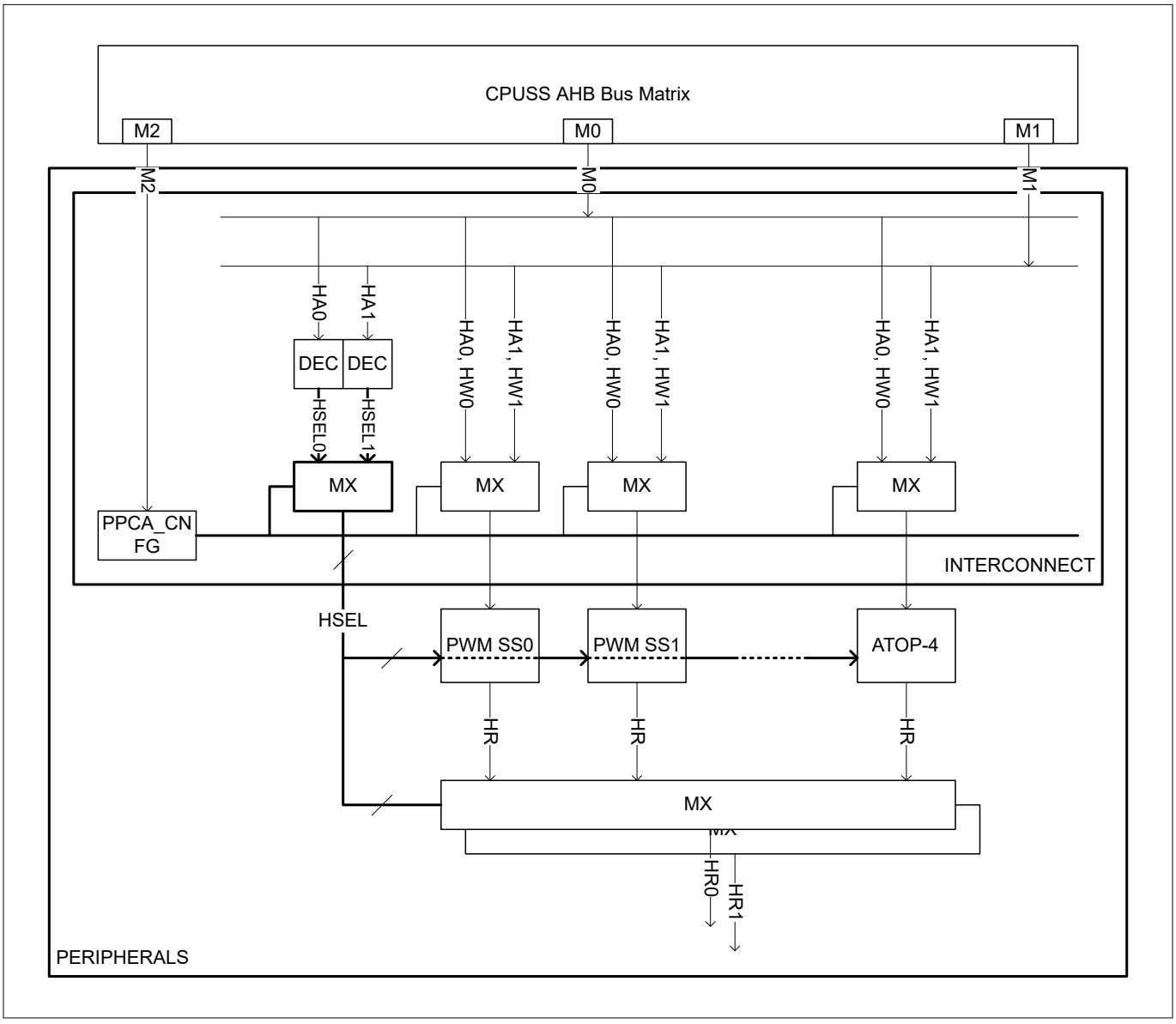


Figure 190 AHB slave interconnect

Peripherals are assigned to the M0/CPU0 master interface by default. The main CPU subsystem (M2) can reassign them to M1/CPU1 by writing to the PPCA_CNFG.CNFGx registers. An AHB Bus Master (M0 or M1) trying to access a peripheral belonging to the other AHB Bus Master (M1 or M0) receive an HRESP error (indicating an illegal access).

27.4.3 Dedicated Data Path

The dedicated data-path provides a dedicated infrastructure to connect ATOP_SS (ADCs or ADCs filtered data) to the HWFILT3P3Z instances or to CLB, and to connect the HWFILT3P3Z to the PWMs instances, enabling a fast data processing and injecting a lower phase delay.

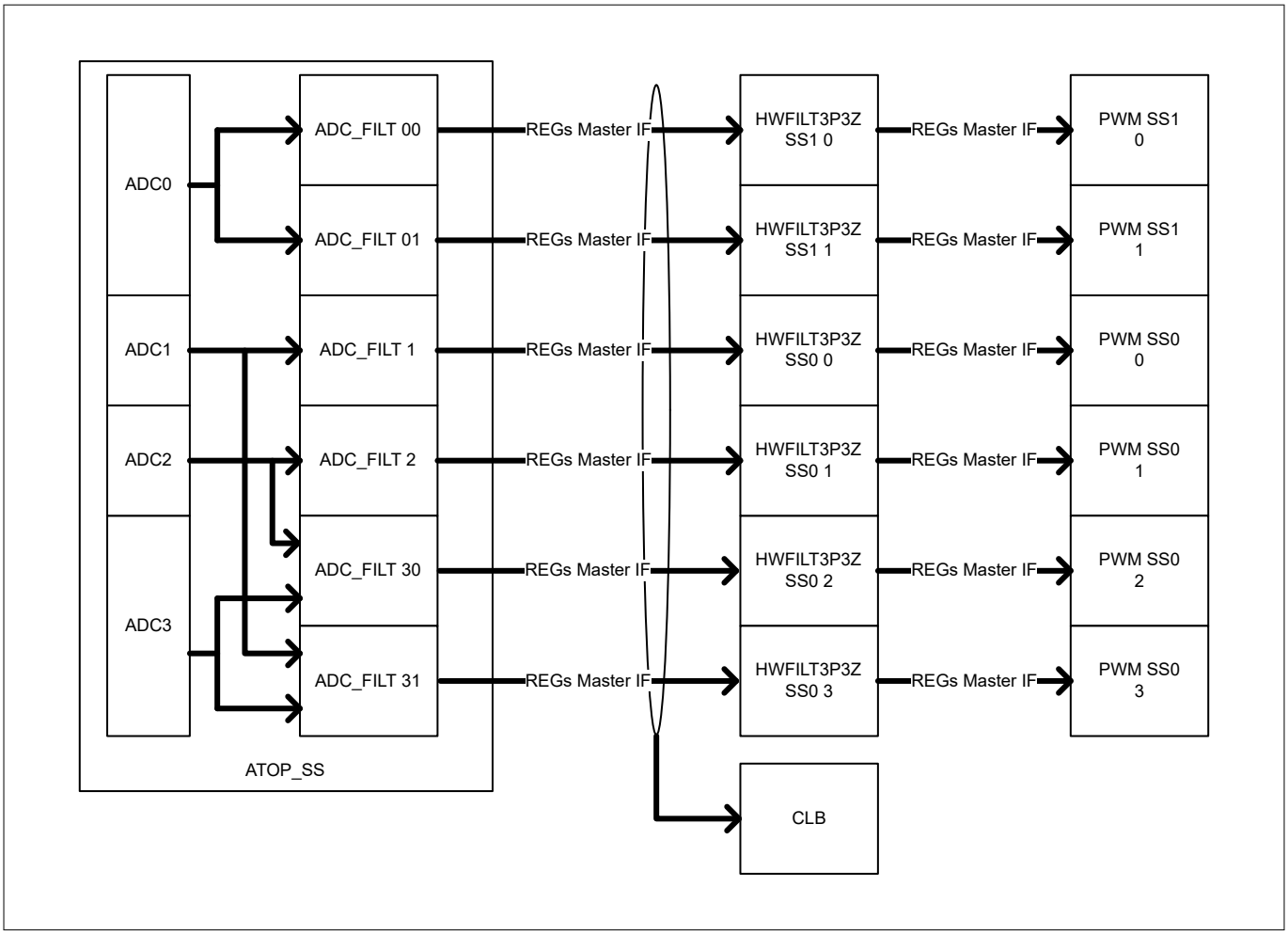


Figure 191 Dedicated Data Path

ADC_FILT source selection shall be defined by SEL_SRC_ADC_FILT configuration bit in the ATOPSS-ADC_GRP0-AFLT-AFLT_CNFG Configuration register, status of the DISPERI bits should also be considered.

Register Master/Slave interface includes the following signals:

Signal Name	Description
STB	Pulse Strobe Data bus, synchronous to the PPCA_CLK
DATA [31:0]	Data bus

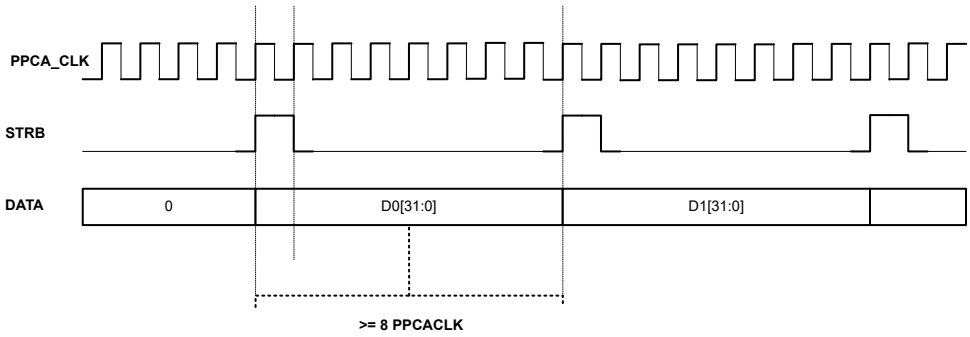


Figure 192 Register Master/Slave interface

***Note**: For more information and applications use case, refer to Getting started Programmable Power Control Accelarator (PPCA) and peripheral configurations AN, section Hardware Architecture of the ACMC Using PPCA.*

27.4.4 CPUSS buses

27.4.4.1 Cortex®-M33 C-AHB bus

This bus connects the C-AHB bus of each Cortex®-M33 core to the bus matrix through the instruction cache. This bus is used for instruction fetch and data access to the internal and external memories mapped in the code region. The target of this bus are the flash, is internal SRAM.

27.4.4.2 Cortex®-M33 S-AHB bus

This bus connects the S-AHB bus of the Cortex®-M33 core to the bus matrix. This bus is used by the core to access data located in a peripherals or SRAM area. The targets of this bus are the aliased internal SRAM, and peripherals connected.

27.5 PPCA Inter processor communication (IPC)

Inter-processor communication (IPC) provides the functionality for PPCA CPU0 and PPCA CPU1 applications to communicate and synchronize their activities. IPC hardware is implemented using two register structures.

- IPC Channel: Communication and synchronization between processors are 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

The channel and interrupt structures are independent and have no correlation to each other, as shown in Figure 193. This allows for building varying models of interfaces, as shown in the Typical usage models.

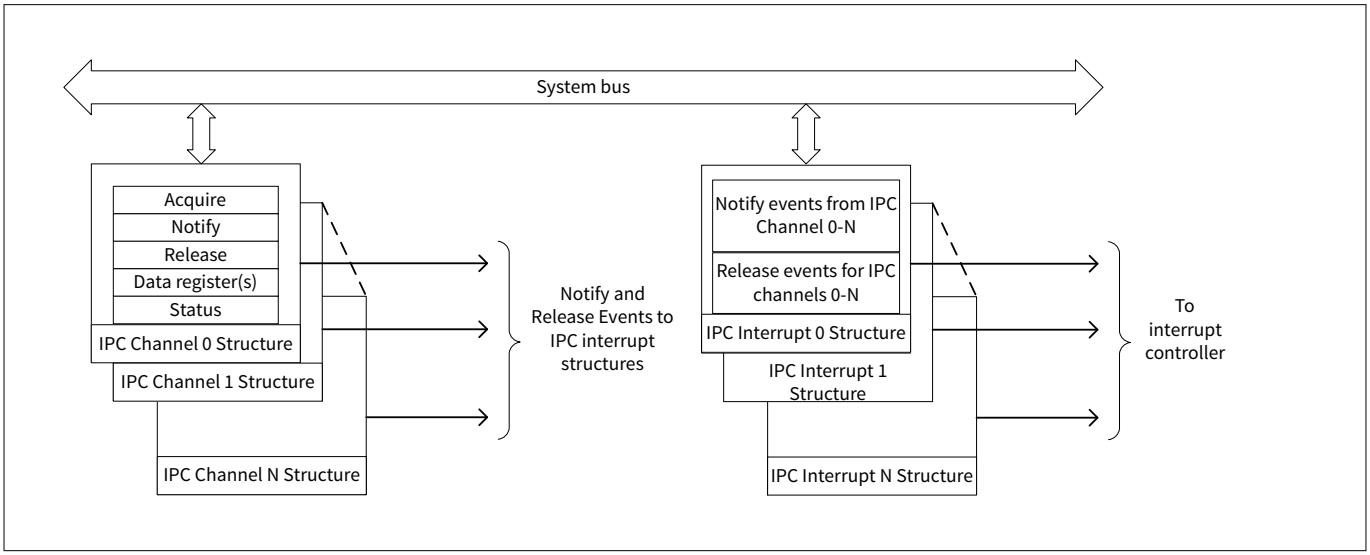


Figure 193 IPC register architecture

27.5.1 Features

The features of IPC are as follows:

- Implements locks for mutual exclusion and synchronization between PPCA CPU0 and CPU1 applications

- Allows sending messages between PPCA CPU0 and CPU1 core applications
- Supports up to 4 channels for communication
- Supports up to 4 interrupts, which can be triggered using notify or release events from the channels

27.5.2 Architecture

27.5.2.1 IPC channel

An IPC channel is implemented as six hardware registers, as shown in Figure 194. The IPC channel registers are accessible to both PPCA CPU0 and CPU1 applications in the system.

- PPCA_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 value of '1', it means that the read acquired the lock. If the SUCCESS field returns a value of '0', it means that the read operation failed to acquire the lock. The P, NS, PC, and MS fields represent the access attributes of the transaction that previously obtained the lock successfully, and the current access does not affect these fields.

  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 CPU with multiple tasks that can preempt each other. When acquired, this register is released by writing any value into the PPCA_IPC_STRUCTx_RELEASE register. If the register is already in an acquired state, another attempt to read it will not be able to acquire it
- PPCA_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 any or multiple interrupt structures
- PPCA_IPC_STRUCTx_RELEASE: This field unlocks and triggers release events for the IPC interrupt structures, but only when the lock is obtained (LOCK_STATUS.ACQUIRED is '1'). The release event generated by an IPC channel can trigger any or multiple interrupt structures. To only release the IPC channel and not generate an interrupt, you can write a zero into the IPC release register. Masking of the interrupt can also be achieved by the PPCA_IPC_INTR_STRUCTx_INTR_MASK register
- PPCA_IPC_STRUCTx_DATA0 and PPCA_IPC_STRUCTx_DATA1: These registers are designed to hold data, and they are both 32-bit in size. These registers can be considered as the shared data memory for the channel. Typically, these registers will hold messages that need to be communicated between PPCA CPU0 and CPU1 applications. If the messages are larger than the combined 64-bit size, place pointers in one or both of these registers
- PPCA_IPC_STRUCTx_LOCK_STATUS: This register provides the instantaneous lock status for the IPC channel. If the channel is acquired (LOCK_STATUS.ACQUIRED is '1'), it shows the protection context, security, and other information in the respective bit fields. However, if the lock is not acquired (LOCK_STATUS.ACQUIRED is '0'), the values in fields P, NS, PC, and MS are not valid. The lock status reading only gives an immediate status, which can change in the next cycle

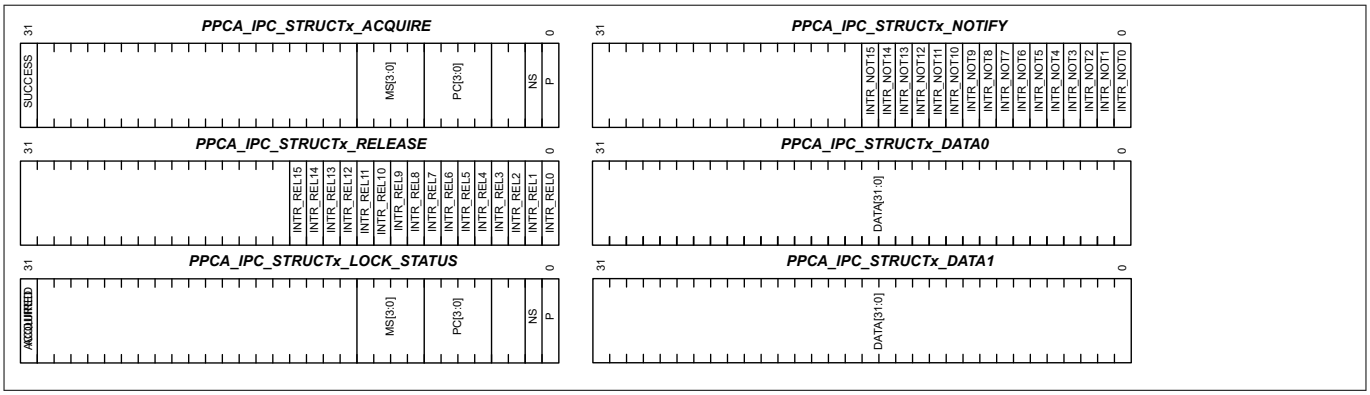


Figure 194 IPC channel structure

27.5.2.2 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. You can choose to mask any of the sources of these events using the IPC interrupt registers. Figure 195 shows the registers in an IPC interrupt structure.

- PPCA_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 register. These are the notify and release events corresponding to the 16 IPC channels. When a notify event is triggered in IPC channel 0, the corresponding Notify 0 bit is activated in the interrupt registers. A write of '1' to a bit will clear the interrupt
- PPCA_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
- PPCA_IPC_INTR_STRUCTx_INTR_SET: A write of '1' into this register sets the interrupt
- PPCA_IPC_INTR_STRUCTx_INTR_MASKED: This register provides the instantaneous value of the interrupts after they are masked. The values in this register are PPCA_IPC_INTR_STRUCTx_INTR and PPCA_IPC_INTR_STRUCTx_INTR_MASK

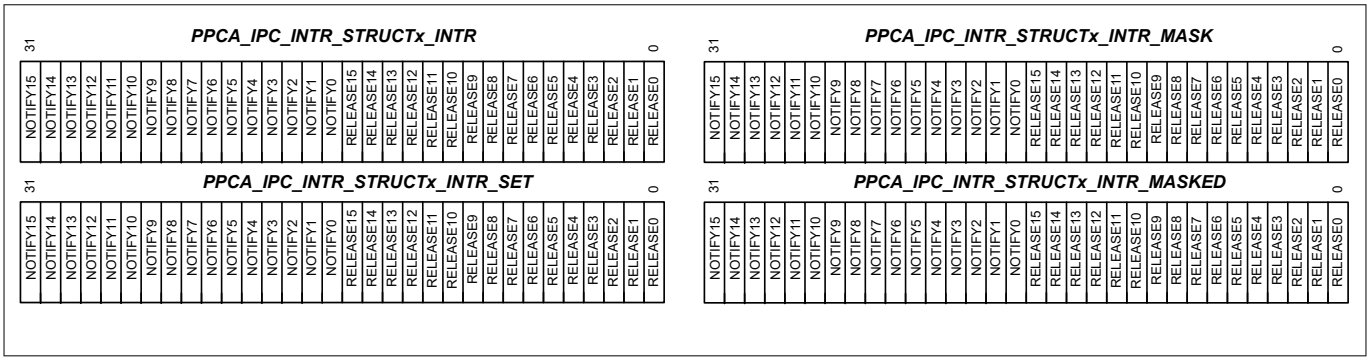


Figure 195 IPC interrupt structure

27.5.2.3 IPC channels and interrupts

The IPC block has a set of associated IPC interrupts. 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 is dependent on the device architecture.

Each IPC channel has a release and notify register, which can drive events to any of the IPC interrupts. An illustration of this relationship between the IPC channels and the IPC interrupt structure is shown in Figure 196.

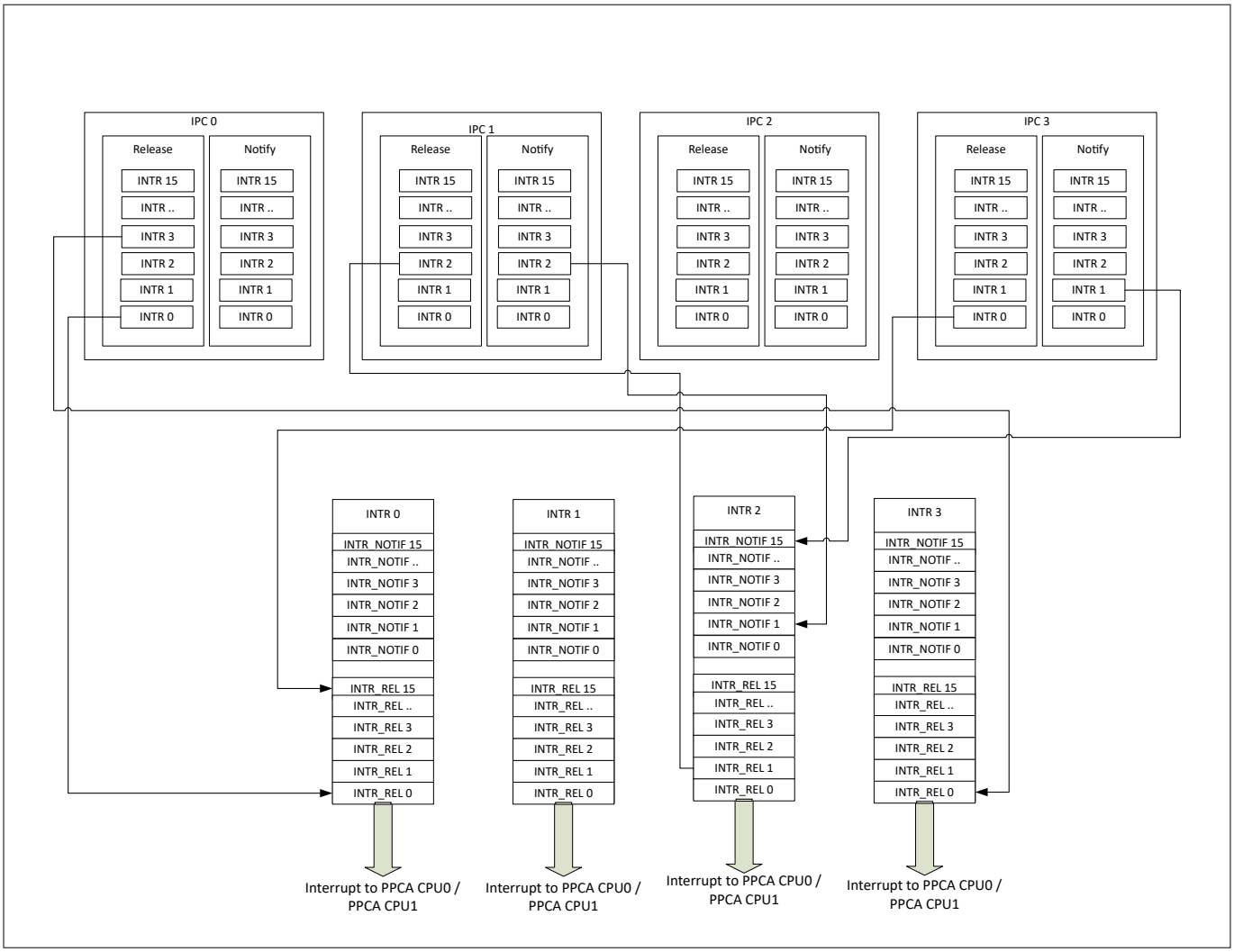


Figure 196 IPC channels and interrupts

27.5.3 Implementing locks

The IPC channels can be used to implement locks, which are typically used between PPCA CPU0 and CPU1 applications to enable some form of mutually exclusive access to a shared resource. When PPCA CPU0 and CPU1 applications share a resource, the PPCA CPU0 or CPU1 application can acquire and release the IPC channel. So PPCA CPU0 and CPU1 applications can assume an IPC channel as a lock. The semantics of this code are that access to the shared resource is gated by the PPCA CPU0 and CPU1 application ownership of the channel. So the PPCA CPU0 or CPU1 application must acquire the IPC channel before they can access the shared resource.

A failure to acquire the IPC channel by the PPCA CPU0 or CPU1 application signifies a lock on the shared resource. Note that the IPC channel does not enforce which application acquires or releases the channel. PPCA CPU0 or CPU1 applications can both acquire or release the IPC channel.

27.5.4 Message passing

IPC channels can be used to communicate messages between PPCA CPU0 and CPU1 applications. In this use case, the channel is used in conjunction with the interrupt structures. The IPC channel is used to lock access to the data registers. 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 transmitting data into the channel and the receiver reading it, the channel is locked for all other task access. 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 registers. After the reception is complete, the receiver releases the channel and can also generate a release event for the sender's IPC interrupt. Note that the action of locking the channel does not restrict access to the data registers in hardware. This is a semantic that should be enforced by software.

Figure 197 shows an example of a sender (a PPCA CPU1 application) sending data to a receiver (a PPCA CPU0 application). IPC interrupt A is configured to interrupt PPCA CPU1 application. IPC interrupt B is configured to interrupt PPCA CPU0 application.

1. The sender will attempt to acquire the IPC channel by reading the PPCA_IPC_STRUCTx_ACQUIRE register. If the channel is acquired, the sender has ownership of the channel for data transmission. This also changes the status of the channel and its corresponding PPCA_IPC_STRUCTx_LOCK_STATUS register. If the channel is not acquired, the sender should wait until the channel is free for acquisition. This can be done by polling the IPC channel's PPCA_IPC_STRUCTx_LOCK_STATUS register
2. After the IPC channel is acquired, the sender has control of the channel for communication and places the data in the PPCA_IPC_STRUCTx_DATA0 and PPCA_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 achieves this by setting the corresponding bit in the IPC channel's PPCA_IPC_STRUCTx_NOTIFY register. This event creates a notify event at IPC interrupt B (PPCA_IPC_INTR_STRUCTx_INTR). If the IPC channel's notify event is enabled by setting the mask bit (PPCA_IPC_INTR_STRUCTx_INTR_MASK [31:16]) in the IPC interrupt B, this will generate an interrupt in the receiver
4. When it receives IPC interrupt B, the receiver can poll the PPCA_IPC_INTR_STRUCTx_INTR_MASKED register to understand which IPC channel triggered the notify event. Based on this, the receiver identifies the channel to read and reads from the IPC channel's PPCA_IPC_STRUCTx_DATA0 and PPCA_IPC_STRUCTx_DATA1 registers. The receiver has now received the data sent by the sender. It needs to release the channel so that other 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 generates a release event interrupt on the sender if the corresponding channel release event is masked

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

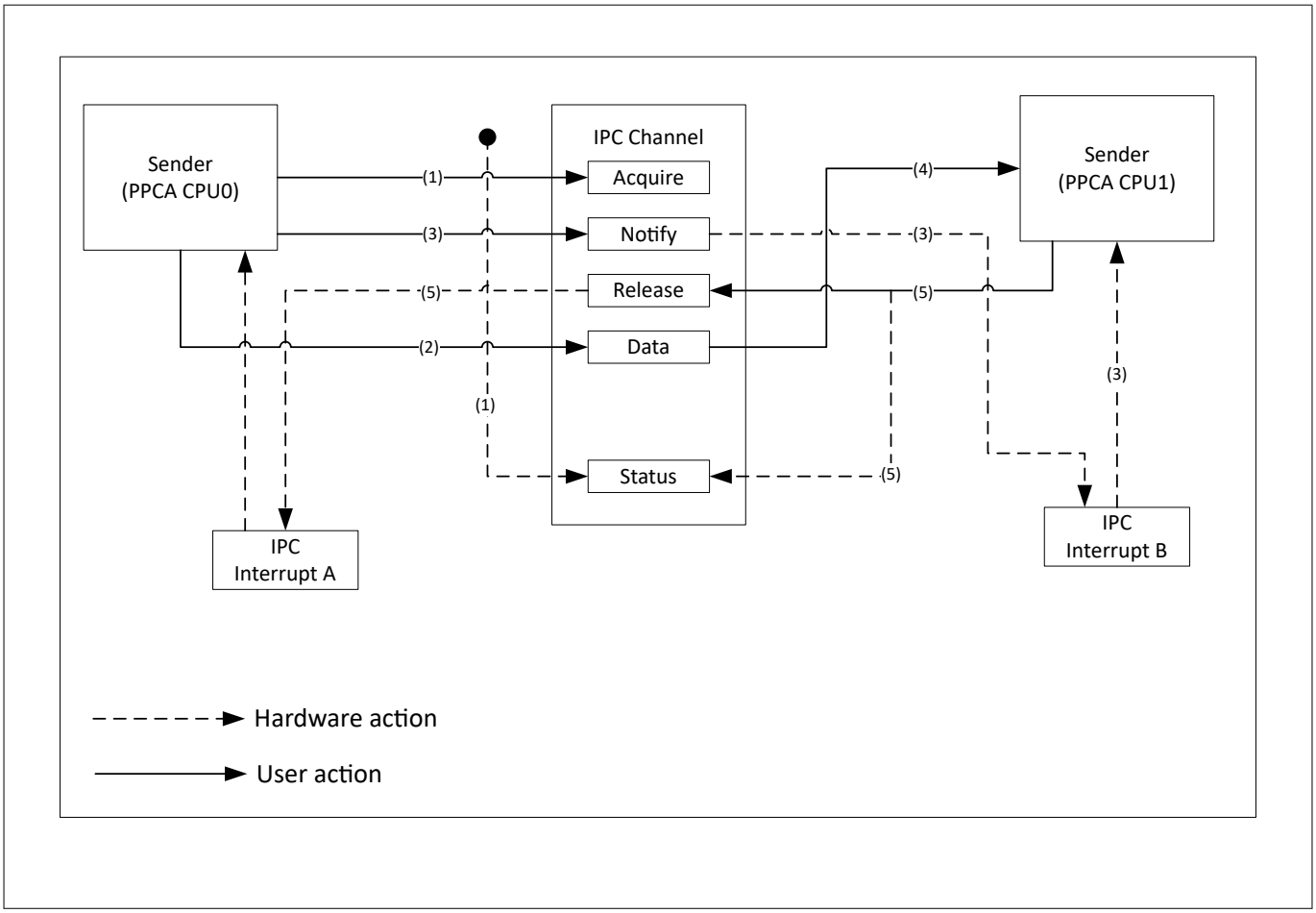


Figure 197 Sending messages using IPC

In the previous example, the size of the data being transmitted was only 64 bits. Larger messages can be sent as pointers. The sender can allocate a larger message structure in memory and pass the pointers to the data registers, see Figure 198 for usage. Note that the user code must implement the synchronization of the message read process.

- The implementation can stall the channel until the receiver has used all the data in the message packet and the message packet can be rewritten. This is worthless because it will stall other inter-process 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 the receiver clears on read complete

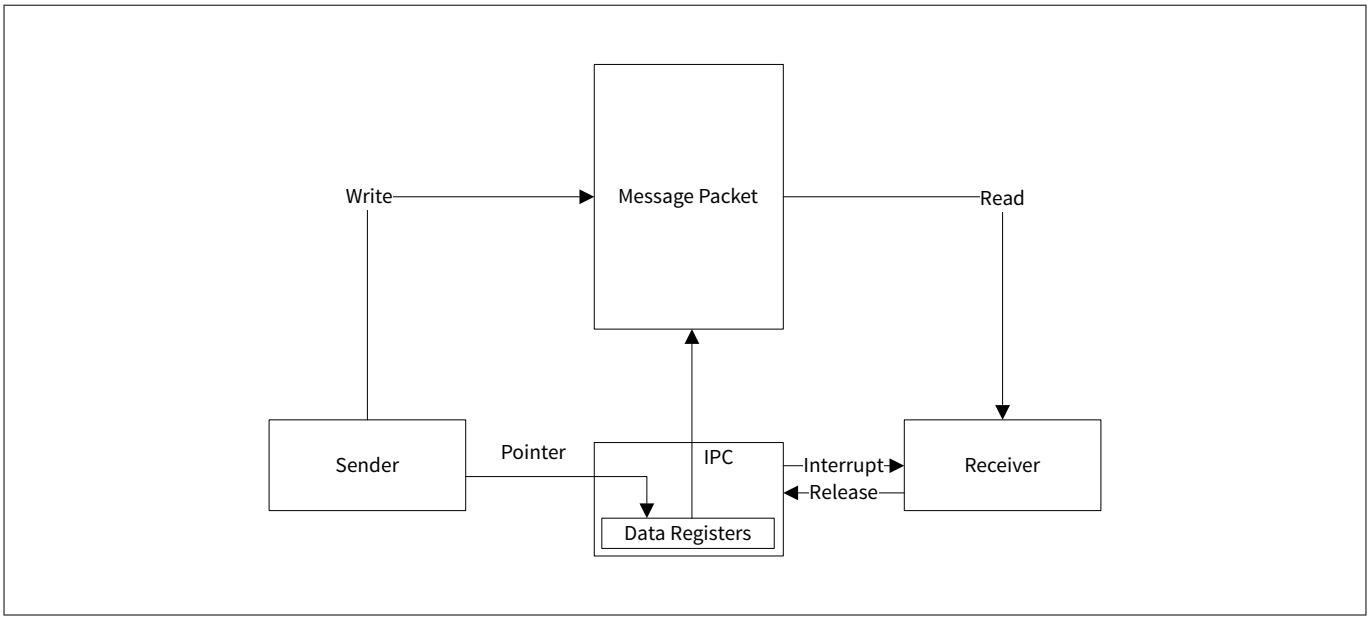


Figure 198 Communicating larger messages

27.5.5 Typical usage models

The unique channel and interrupt architecture of the PSC3M/P8, PSC3M/P7 IPC allows for a range of usage models for communication between PPCA CPU0 and CPU1 applications. Some of these are listed here as an example. Note that the communication models possible, based on the IPC architecture, are not restricted to the ones listed in this document. Also note that this document only provides a high-level usage model and does not provide details of data management in the communication. This must be determined based on the specific application use case.

27.5.5.1 Full-duplex communication

In this usage model, an IPC channel is used according to the direction of communication between PPCA CPU0 and CPU1 applications. In this example, IPC channel X is dedicated to data communication from PPCA CPU1 application to PPCA CPU0 application and IPC channel Y is for data communication from PPCA CPU0 application to PPCA CPU1 application. The IPC interrupt X will signal events on PPCA CPU0 application. Therefore, its interrupt output is connected to the PPCA CPU0 application's system interrupts. The events are triggered by writing into the IPC interrupt register structure over the system bus. Similarly, IPC interrupt Y is dedicated to PPCA CPU1 application.

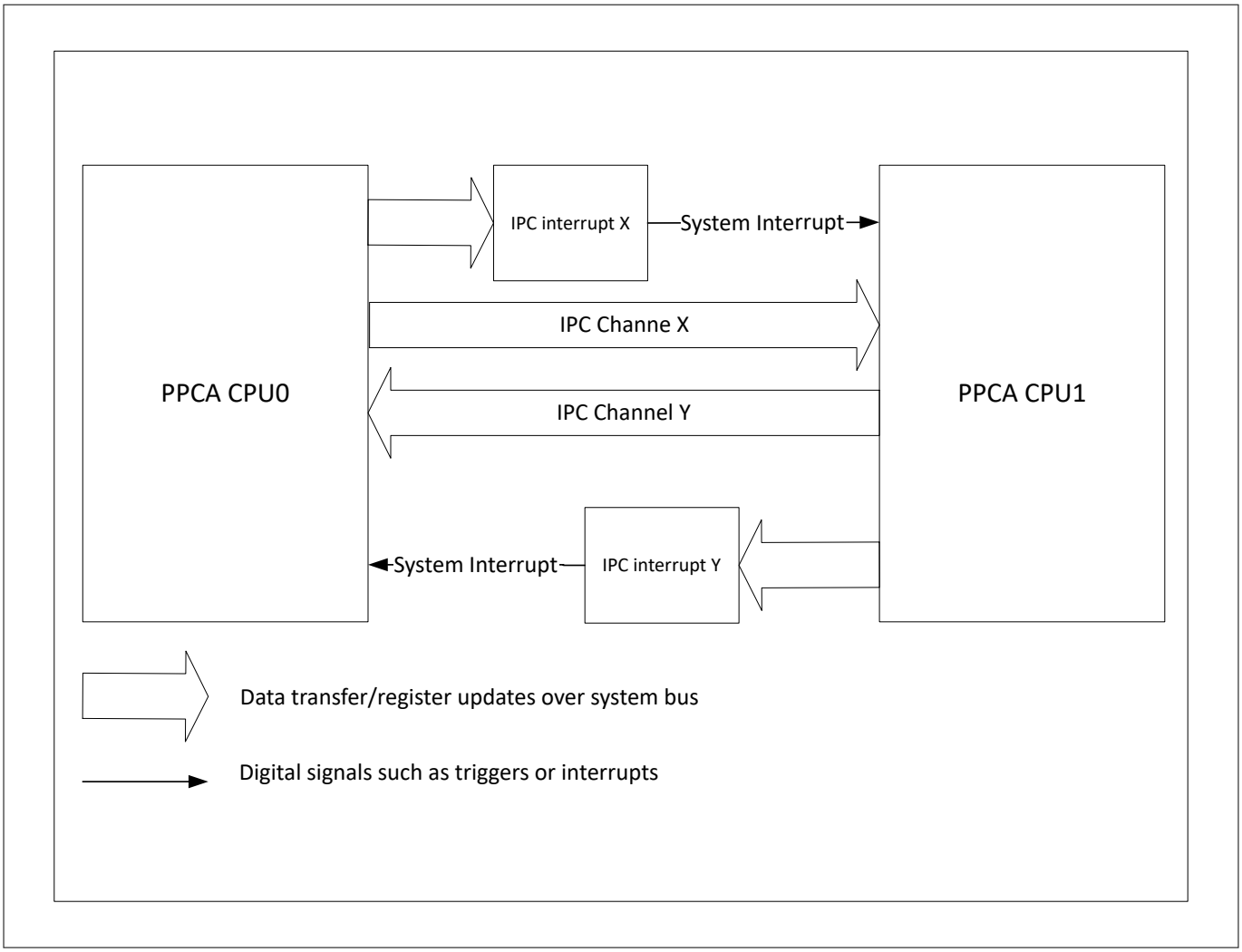


Figure 199 Full-duplex IPC between secure and non-secure application

27.5.5.2 Half-duplex with independent event handling

In this case, only one IPC channel is used to set up the transfer between the PPCA CPU0 and CPU1 applications. This means that only one side controls data transfer at a time. The channel's lock register must be used to avoid contention of the one shared IPC channel. Two independent events are supported since two IPC interrupt structures are used. This model is shown in Figure 200.

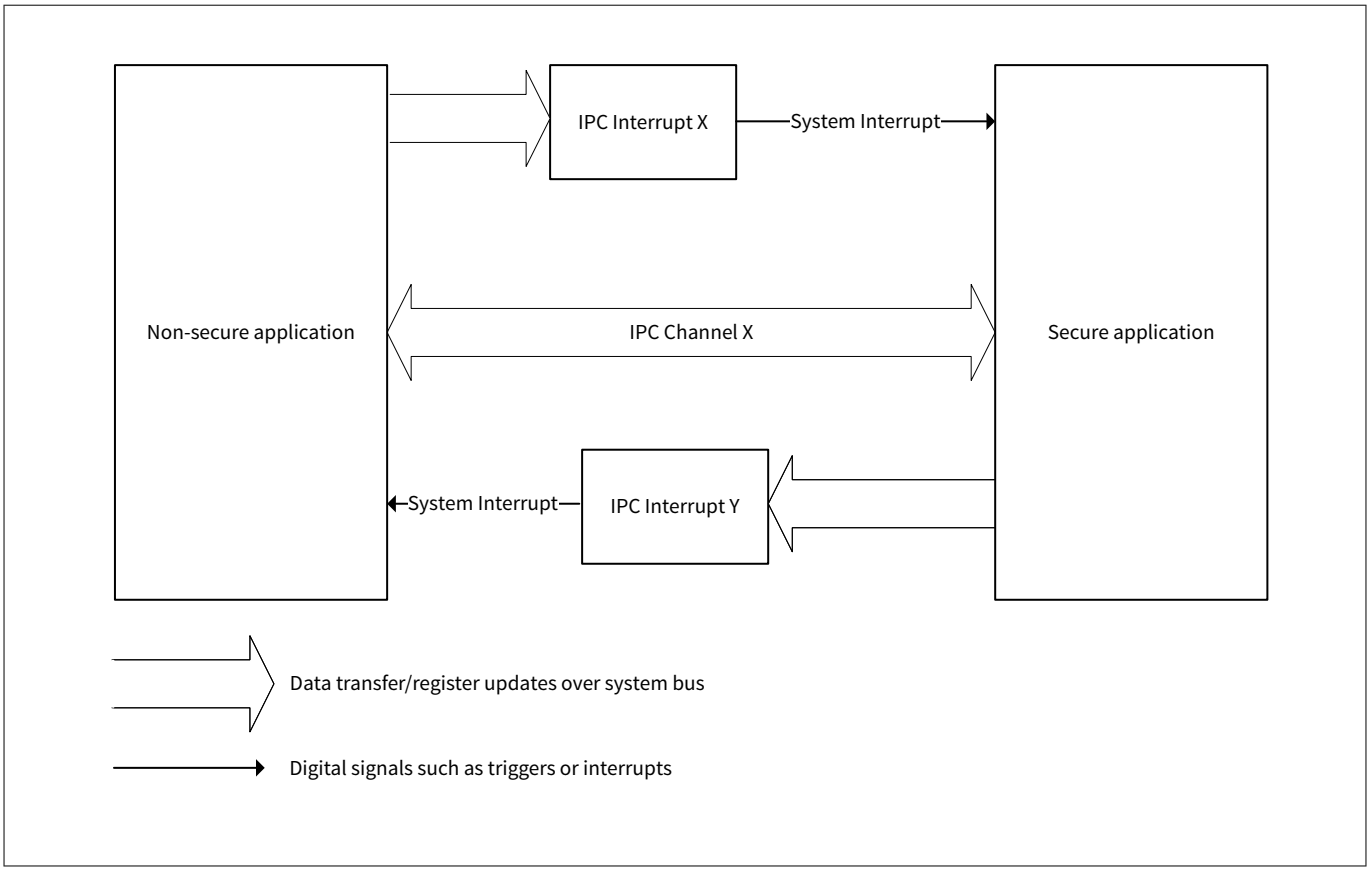


Figure 200 Half-duplex with independent event handling

27.5.5.3 Half-duplex with shared event handling

In this model, both the IPC channel and interrupt are shared between the PPCA CPU0 and CPU1 applications. Since the interrupt is also shared, access to the interrupt registers must be managed using the IPC lock of the channel. As shown in Figure 201, the IPC interrupt is set up to trigger interrupts in both PPCA CPU0 and CPU1 applications. Therefore, PPCA CPU0 or CPU1 application interrupts should have logic in their ISR to check which application is in control of the IPC and determine if the message and event were for that application.

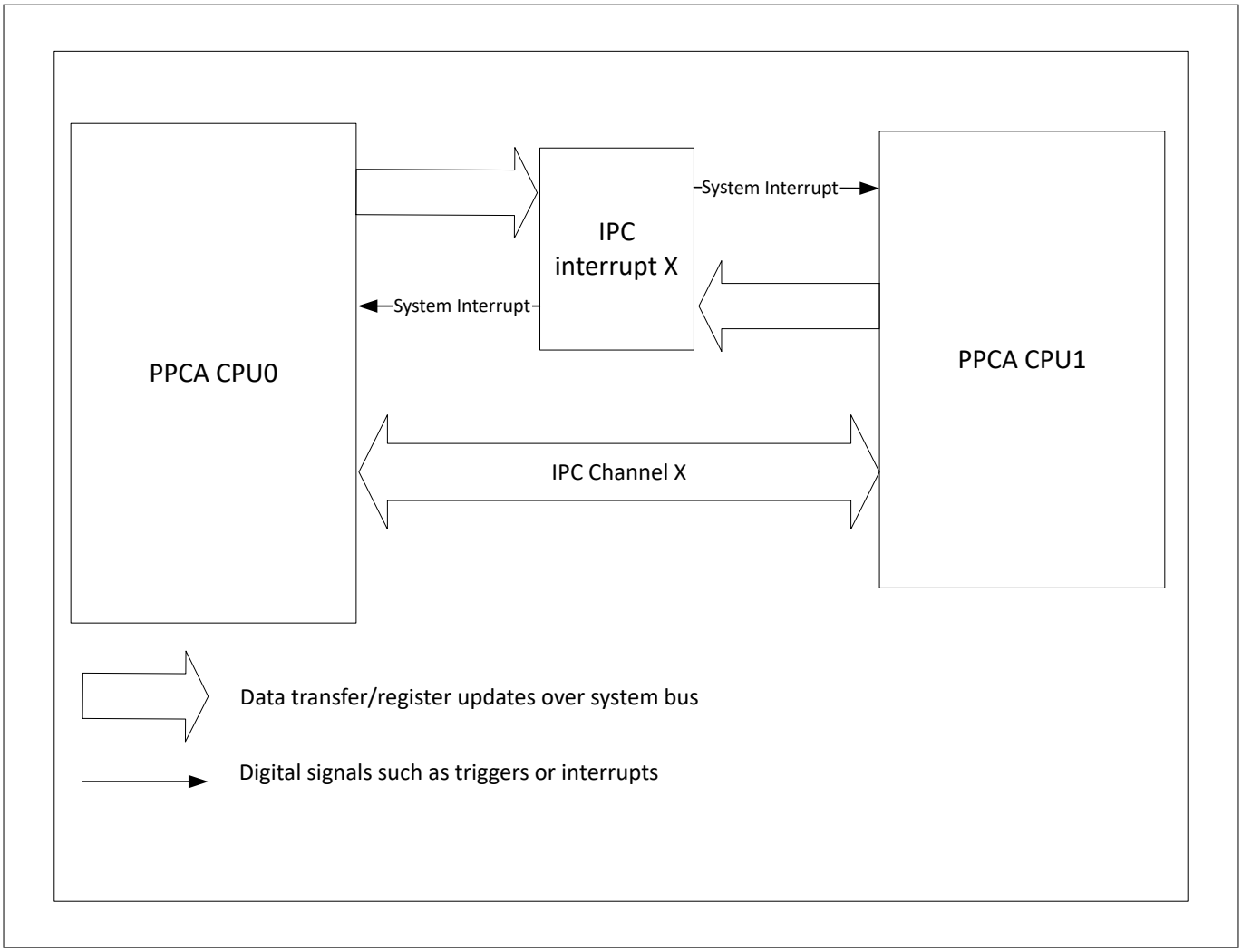


Figure 201 Half-duplex with shared event handling

27.6 SRAM controller

This section explains the PPCA SRAM controller, its features, architecture, and wait states. The SRAM controller enables the CPUs to read and write the PPCA SRAMs.

In the PPCA there are five (5) SRAMs , three 16 KB sizes and two 32KB of sizes. To know more about SRAM instances and their start addresses, see the Address and memory maps section of the PPCA CPUSS subsystem.

27.6.1 Features

The SRAM controller has the following features:

- Supports 8, 16, and 32-bit access
- Provides exclusive access support using Exclusive Access Monitor (EAM) from Arm® SIE-200
- SRAM power mode control using Arm® PPU (Power Policy Unit)
- The SRAM controller supports the Error Correction Code (ECC) for the SRAM

See the device datasheet for electrical specifications.

27.6.2 Architecture

The SRAM controller has an AHB5 interface that connects to the AHB5 infrastructure. The AHB5 transfers are the origin for all SRAM accesses. The SRAM controller differentiates between the following three types of AHB5 transfers:

- AHB5 read transfers
- 32-bit AHB5 write transfers
- 8-bit and 16-bit AHB5 write transfers (also referred to as Partial AHB5 write transfers)

27.6.2.1 AHB5 read transfers

AHB5 reads transfers, and a read transfer is translated into SRAM read access.

27.6.2.2 32-bit AHB5 write transfers

A 32-bit AHB5 write transfer is translated into SRAM write access.

27.6.2.3 Partial AHB5 write transfers

Partial writes (8-bits and 16-bits) are supported through read, modify, and write by applying the associated mask during write.

27.6.2.4 Power partition control

The RAMCx_PWR_MACROx_CTL (where x = 0 to 4, example RAMC0_PWR_MACRO0_CTL) register controls the power state of each SRAM power partition independently. Each bit of RAMCx_PWR_MACROx_CTL.OFF[31:0] represents the individual state of each SRAM power partition. Enabling the bit corresponding to the particular power partition turns OFF the partitions, and disabling the bit turns ON the respective partitions. It is mandatory to ensure STATUS.PWR_DONE becomes 1 after these RAMCx_PWR_MACROx_CTL register contents are changed to ensure the SRAM mode transition is completed successfully. In PPCA SRAMs, there is just one power partition available in each SRAM macro.

The RAMCx_PWRx_MACRO_CTL_LOCK register controls the accidental writes into the RAMCx_PWR_MACRO_CTL register by prohibiting read/write access when this (PWR_MACRO_CTL_LOCK[1:0]) field is not equal to zero.

27.6.3 Error correcting code (ECC)

The SRAM controller supports Error Correcting Code (ECC) for the SRAM.

The 7-bit SECDED parity covers the 32-bit data word as a single entity, there is no dedicated parity for the four 8-bit bytes within the data.

- 32-bit AHB5 write bus transfers require only a single SRAM write access
- 8-bit and 16-bit AHB5 write bus transfers require an additional SRAM read access that precedes the SRAM write access to retrieve the missing data bytes

27.6.3.1 ECC error injection

The Error Injection mechanism is supported by 32-bit data word write into an RAMCx_ECC_MATCH (where, x = 0 to 4) register. Reading from the address defined in ECC_MATCH.WORD_ADDR triggers correctable or uncorrectable fault indication, if the value in ECC_CTL.PARITY has 1, 2, or more errors compared to expected ECC value.

27.6.3.2 ECC error report functionality

The ECC syndrome logic reports recoverable and non-recoverable errors to the CPUSS fault reporting infrastructure. The ECC syndrome logic corrects recoverable errors, Errors up to one bit are recoverable.

27.6.3.2.1 Partial AHB5 write transfers

Error reported during partial write transfer involving SRAM read are listed as below

- **Correctable Error (1-bit error):** This corrected 32-bit data word is then merged with the write data with appropriate mask and stored in the write buffer. A future write-buffer request results in a SRAM write access with the merged write data
- **Multiple-bit error:** The error is communicated through the CPUSS fault reporting infrastructure

27.6.3.2.2 AHB5 read transfers

During the AHB read transfers below are possible scenarios.

- **Correctable Error (1-bit error):** The corrected data is returned to the AHB5 interface. If ECC auto-correct functionality is enabled, then the corrected data is also stored in write-buffer. A future write-buffer request results in a SRAM write access with the corrected data
- **Multiple-bit error:** Corrupted-data (multiple-bit error data) is returned to the AHB master on hrdata[31:0]

27.6.3.2.3 Correctable ECC error reporting

Auto-correction is enabled by default. When a 1-bit error is detected during an SRAM read, the hardware automatically corrects the error before passing the data to the AHB. The corrected data is also written back to the SRAM to maintain data integrity. Optionally if Correctable ECC fault is enabled in the Fault sub-system, a PPCA SRAMx_C ECC fault (correctable) will be communicated to the main CPUSS via Fault reporting infrastructure.

27.6.3.2.4 Non-correctable ECC error reporting

When a non-recoverable error—defined as an error affecting more than one bit—is detected during an AHB5 bus transfer, the resulting data word is incorrect. However, no AHB5 bus error is generated. Instead, the error is reported exclusively through the fault monitoring subsystem. Specifically, it is flagged as a PPCA SRAMx_NC ECC fault (non-correctable), where x ranges from 0 to 4. These faults are communicated to the main CPUSS via the Fault Reporting Infrastructure.

27.6.4 Wait states

Under standard operating conditions, no wait states are introduced, as only AHB5 requests are issued. This behaviour remains consistent even when ECC is enabled.

However, a single wait state is inserted under the following conditions:

- Read-after-Write Dependency: A read operation immediately follows a write operation.
- ECC Correction Overhead: An ECC correction was triggered during the preceding memory access.

27.7 Interrupts

27.7.1 Features

The PPCA CPU0/CPU1 supports the following interrupt features:

- Supports 53 interrupts
  - Four interrupts can be mapped to the CPU's non-maskable interrupt (NMI)
- Nested vectored interrupt controller (NVIC) integrated with CPU core, yielding low interrupt latency
- Configurable priority levels for each interrupt, eight levels on the Cortex®-M33
- Level-triggered interrupt signals

27.7.2 Architecture

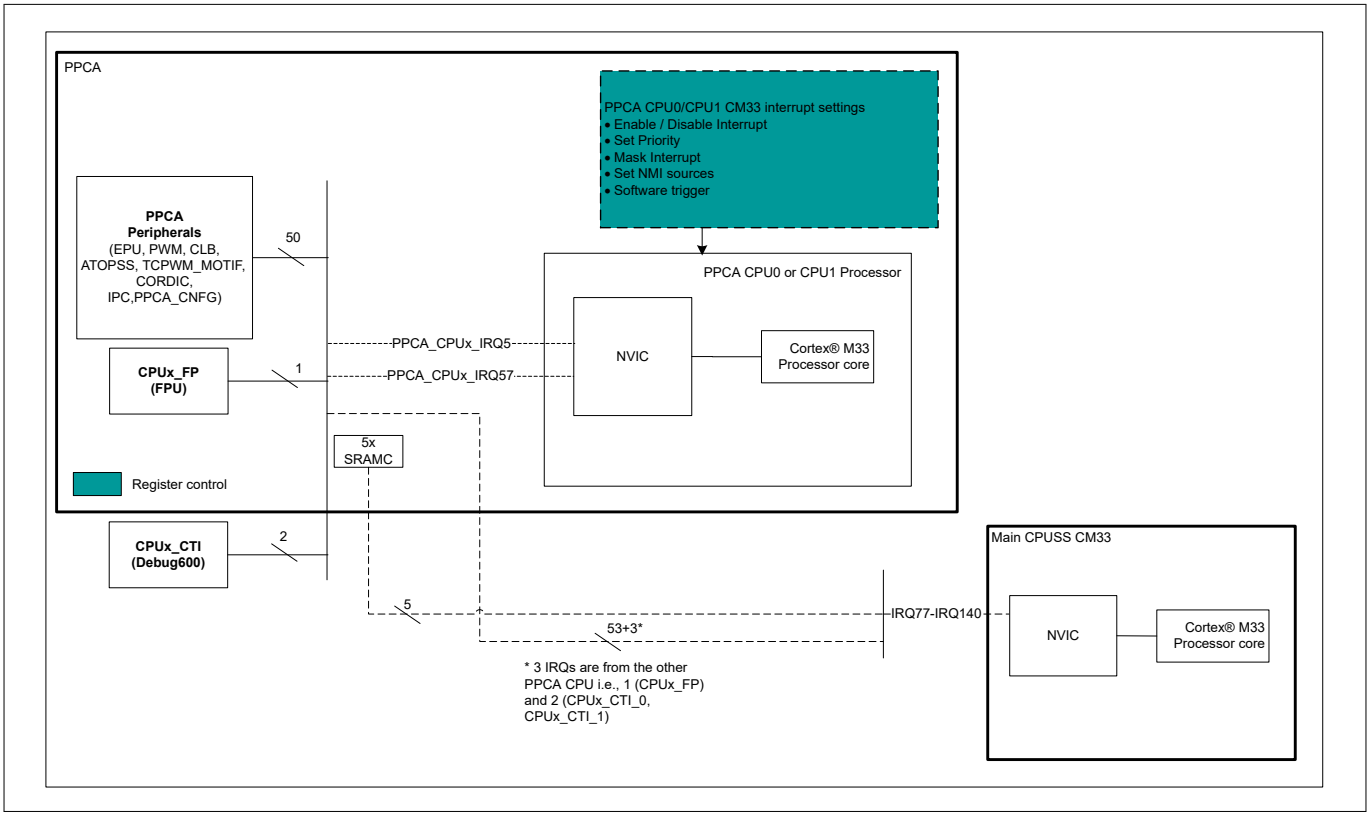


Figure 202 PPCA CPU0 / CPU1 interrupt architecture block diagram

Figure 202 shows the PPCA CPU0 / CPU1 interrupt architecture. The Cortex®-M33 has system interrupt sources directly connected to the IRQn of the NVIC. The NVIC handles enabling/disabling individual interrupt IRQs, priority resolution, and communication with the CPU core. The other exceptions, such as NMI and hard faults, are not shown in Figure 202 because they are part of CPU core-generated events, unlike interrupts, which are generated by peripherals external to the CPU within the PPCA sub-system.

27.7.3 Interrupts and exceptions operation

27.7.3.1 Interrupt/exception handling

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

- The assumption is that all the interrupt and exception signals are initially low (idle or inactive state) and the processor executes the main code. Any interrupt signal that is held asserted 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
- On detecting the signal from the NVIC, the CPU stores its current context by pushing the contents of the CPU registers onto the stack
- The CPU also receives the exception number of the triggered interrupt from the NVIC. All interrupts and exceptions have a unique exception number. By using this exception number, the CPU fetches the address of the specific exception handler from the vector table. Each interrupt can be configured to target the secure or non-secure vector table
- The CPU then branches to this address and executes the exception handler that follows
- After completion of the exception handler, the CPU registers are restored to their original state using stack pop operations, and the CPU resumes the main code execution

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. The Armv8-M CPU also supports tail-chaining. This means that on completion of an exception handler, if the next pending exception has a higher priority than all stacked exceptions, the stack pop operation is skipped and control is transferred to the new exception handler. Exceptions are handled in the same way that interrupts are handled. Each exception event has a unique exception number, which is used by the CPU to execute the appropriate exception handler.

27.7.3.2 Level interrupts

PPCA CPU 0 / CPU1 only supports level signals on the interrupt lines (IRQn).

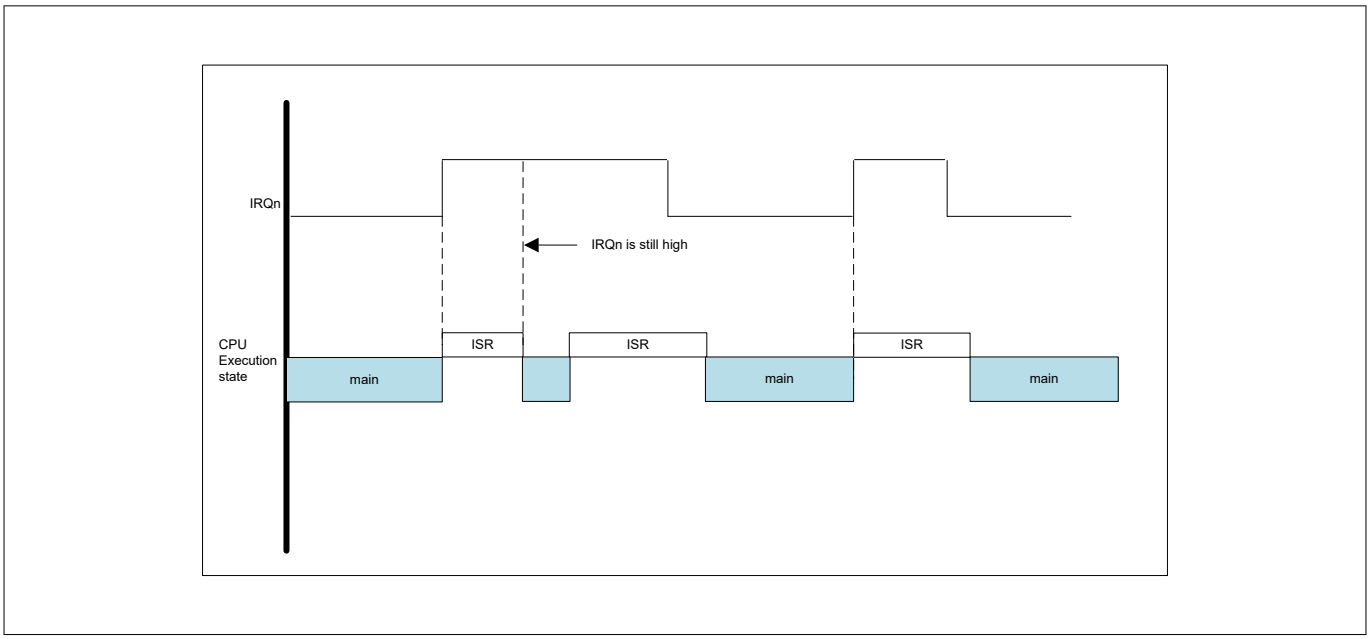


Figure 203 Level interrupts

***Note**: Pulse interrupts are not supported by this device.*

Figure 203 shows how a level interrupt is handled. Assuming the interrupt signal is initially inactive (logic low), the following sequence of events explains the handling of the level:

- For level interrupts, the interrupt signal remains asserted until the NVIC registers the interrupt request. The interrupt is now in the pending state, which means the CPU has not serviced the interrupt requests
- 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 remains high after completing the ISR, it will be pending, and the ISR will be executed again. Figure 203 shows this for level-triggered interrupts, where the ISR is executed as long as the interrupt signal is high

27.7.3.3 Exception vector table

PPCA CPU 0 / CPU1 has one exception vector table that store the entry point addresses for all exception handlers. The CPU fetches the appropriate address based on the exception number.

Table 231 Cortex®-M33 exception vector

Exception number	Secure exceptions	Exception priority	Vector address
-	Initial stack pointer value	-	Start_Address = CM33_NS_VECTOR_TABLE_BASE
1	Reset	-4	Start_Address + 0x0004
2	non-maskable interrupt (NMI_NS)	-2	Start_Address + 0x0008
3	HardFault	(-1 or -3)	Start_Address + 0x000C
4	Memory Management	Configurable (0-7)	Start_Address + 0x0010
5	Bus Fault	Configurable (0-7)	Start_Address + 0x0014
6	Usage Fault	Configurable (0-7)	Start_Address + 0x0018
7	Secure Fault	Configurable (0-7)	Start_Address + 0x001C
8-10	Reserved	-	...
11	Supervisory Call (SVCall_NS)	Configurable (0-7)	Start_Address + 0x002C
12	Debug Monitor	Configurable (0-7)	Start_Address + 0x0030
13	Reserved	-	...
14	Pending Supervisory (PendSV_NS)	Configurable (0-7)	Start_Address + 0x0038
15	Sys Tick Timer (SysTick_NS)	Configurable (0-7)	Start_Address + 0x003C
16	IRQ(0)	Configurable (0-7)	Start_Address + 0x0040
...	...	...	...
63	IRQ(57)	Configurable (0-7)	Start_Address + 0x0228

***Note**: Start address = CM33_NS_VECTOR_TABLE_BASE on reset and can be later modified by firmware by updating the CM33's Armv8-M VTOR_NS register.*

The first word (4 bytes) in each vector table 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. The vector table offset register (VTOR) present as part of the Armv8-M System Control Block registers (SCB) configures the vector table offset from the base address (0x0000). The VTOR value determines the location of the vector table, which can be placed in volatile and non-volatile memory. Note that the VTOR registers can be updated only in privilege CPU mode. The advantage of moving the vector table to volatile memory, such as SRAM, is that the exception handler addresses can be dynamically changed by modifying the vector table contents. However, a non-volatile memory vector table must be modified by a non-volatile memory write. Note that the exception table must be 1024 byte-aligned for the Cortex®-M33. The exception sources (exception numbers 1 to 15) are explained in the Exception sources section. The exceptions marked as reserved are not used, although they have addresses reserved for them in the vector table. The interrupt sources (exception numbers 16 and higher) are explained in the Interrupt sources section.

27.7.4 Exception sources

This section explains the different exception sources listed in Table 231 (exception numbers 1 to 15).

27.7.4.1 Reset exception

A device reset is treated as an exception. The reset exception is always enabled with a fixed priority of –4, the highest priority exception. A device reset can occur for multiple reasons, such as power-on-reset (POR), an external reset signal on the XRES pin, or a watchdog reset.

27.7.4.2 Non-maskable interrupt exception

Non-maskable interrupt (NMI) is always enabled with a fixed priority of –2.

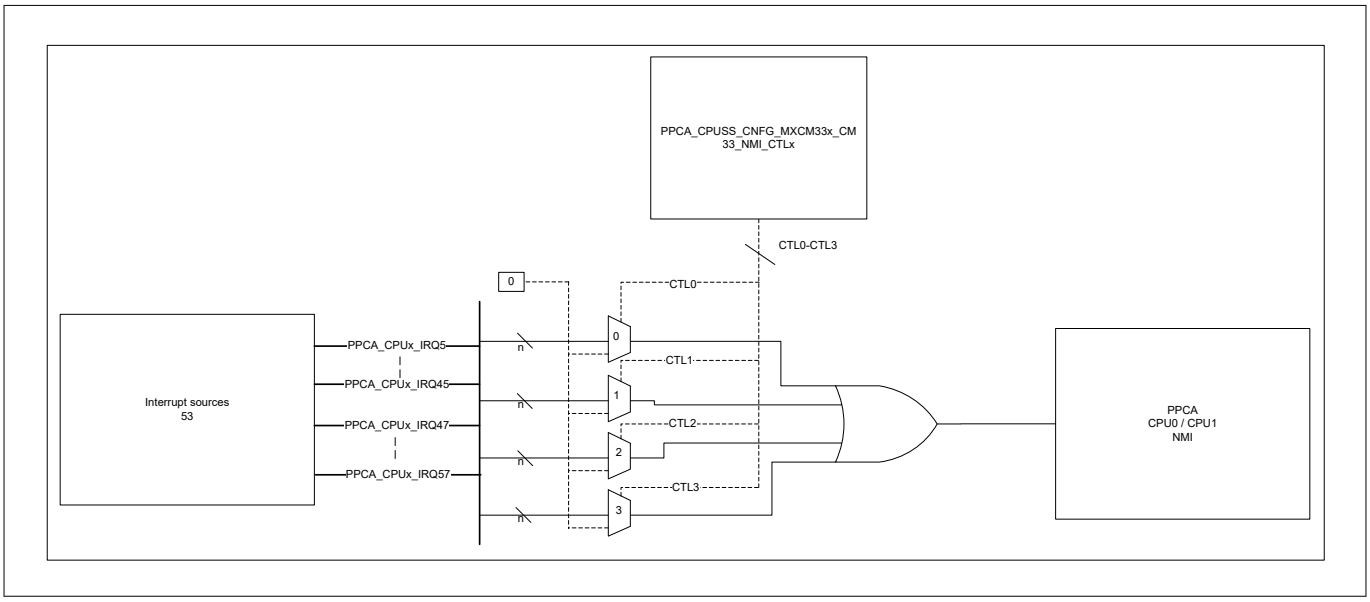


Figure 204 NMI trigger

The Cortex®-M33 provides an option to trigger an NMI exception from up to four of the interrupts the CPU has access to (IRQ0—IRQ63). The NMI exception triggered by the interrupt executes the NMI handler pointed to by the active exception vector table. The PPCA_CPUSS_CNFG_MXCM33x_CM33_NMI_CTLx registers select the interrupt source that triggers the NMI from hardware. The registers have a default value of 1023; that is, if the register is set to 1023, then that particular register does not map any interrupt source to the NMI. There are four

such registers, and each can map one interrupt vector to the NMI. NMI is triggered when any of the four interrupts are triggered; that is, the interrupts are logically ORed.

27.7.4.3 HardFault exception

PPCA CPU0 / CPU1 supports the HardFault exception. HardFault is an always-enabled exception that occurs because of an error during normal or exception processing. HardFault has a priority of at least –1, meaning it has a higher priority than any exception with configurable priority. A HardFault exception is a catch-all exception for different types of fault conditions, which include executing an undefined instruction and accessing an invalid memory address. 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 can recover from the fault situation.

27.7.4.4 MemManage fault exception

A MemManage 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 access to Execute Never (XN) memory regions. The priority of the exception is configurable from 0 (highest) to 7 (lowest).

The exception is banked between security states.

27.7.4.5 BusFault exception

A BusFault is an exception that occurs because of a memory-related fault for an instruction or data memory transaction. This may be from an error detected on a bus in the memory system. The priority of the exception is configurable from 0 (highest) to 7 (lowest).

27.7.4.6 UsageFault exception

A UsageFault 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 half-word memory access
- Division by zero

The priority of the exception is configurable from 0 (highest) to 7 (lowest).

27.7.4.7 Supervisor Call (SVCall) exception

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 an SVCall that requires privileged access to the system. The priority of the exception is configurable from 0 (highest) to 7 (lowest).

27.7.4.8 DebugMonitor exception

Armv8-M CPUs support the DebugMonitor exception. DebugMonitor is only available if halting debug is disabled and debug monitoring is enabled. This can be done by setting the C_DEBUGEN bit to disabled in the Armv8-M Debug Halting Control and Status Register (DHCSR) and the MON_EN bit to enabled in the Armv8-M Debug Exception and Monitor Control Register (DEMCR). The DebugMonitor exception is handled by dedicated debug monitoring software running on the same core. A debug event causes a DebugMonitor exception only when the priority of the DebugMonitor exception is greater than the current exception priority. The priority of the exception is configurable from 0 (highest) to 7 (lowest).

27.7.4.9 PendSupervisory (PendSV) exception

Armv8-M CPUs support the 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 priority of the exception is configurable from 0 (highest) to 7 (lowest).

27.7.5 Interrupt sources

The PPCA CPU 0 / CPU1 supports 53 interrupts within the PPCA peripherals. The source of each interrupt is listed in Table 232. These system interrupts are mapped directly to the Cortex®-M33 NVIC.

The interrupts include interrupts from the PPCA peripherals such as EPU, TCPWM, CLB, ATOPSS, TCPWM_MOTIF, CORDIC, and IPC. 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 level interrupts. The appropriate interrupt registers should be cleared in the ISR to deassert the interrupt. Usually, a write '1' is required to clear the registers. If the interrupt register is not cleared in the ISR, the interrupt remains asserted, and the ISR is executed continuously.

Table 232 List of PPCA CPU 0 interrupt sources

Cortex®-M33 interrupt	Cortex®-M33 exception number	Interrupt source
NMI	-2	Logical OR of four of the following 64 interrupt sources
IRQ0	16	Not used
IRQ1	17	Not used
IRQ2	18	Not used
IRQ3	19	Not used
IRQ4	20	Not used
IRQ5	21	CPU0_0
IRQ6	22	CPU0_1
IRQ7	23	CPU0_2
IRQ8	24	CPU0_3
IRQ9	25	CPU1_0
IRQ10	26	CPU1_1
IRQ11	27	CPU1_2
IRQ12	28	CPU1_3

(table continues...)

Table 232 (continued) List of PPCA CPU 0 interrupt sources

Cortex®-M33 interrupt	Cortex®-M33 exception number	Interrupt source
IRQ13	29	HRTCPWM0 GRP 0 CNT 0
IRQ14	30	HRTCPWM0 GRP 0 CNT 1
IRQ15	31	HRTCPWM0 GRP 0 CNT 2
IRQ16	32	HRTCPWM0 GRP 0 CNT 3
IRQ17	33	HRTCPWM0 GRP 1 CNT 0
IRQ18	34	HRTCPWM0 GRP 1 CNT 1
IRQ19	35	HRTCPWM0 GRP 1 CNT 2
IRQ20	36	HRTCPWM0 GRP 1 CNT 3
IRQ21	37	HRTCPWM0 GRP 2 CNT 0
IRQ22	30	HRTCPWM0 GRP 2 CNT 1
IRQ23	31	HRTCPWM0 GRP 2 CNT 2
IRQ24	32	HRTCPWM0 GRP 2 CNT 3
IRQ25	33	HRTCPWM1 GRP 0 CNT 0
IRQ26	34	HRTCPWM1 GRP 0 CNT 1
IRQ27	35	HRTCPWM1 GRP 1 CNT 0
IRQ28	36	HRTCPWM1 GRP 1 CNT 1
IRQ29	37	HRTCPWM1 GRP 2 CNT 0
IRQ30	38	HRTCPWM1 GRP 2 CNT 1
IRQ31	39	HRTCPWM1 GRP 2 CNT 2
IRQ32	40	HRTCPWM1 GRP 2 CNT 3
IRQ33	41	HRTCPWM2 GRP 0 CNT 0
IRQ34	42	HRTCPWM2 GRP 0 CNT 1
IRQ35	43	HRTCPWM2 GRP 0 CNT 2
IRQ36	44	HRTCPWM2 GRP 0 CNT 3
IRQ37	45	HRTCPWM3 GRP 0 CNT 0
IRQ38	46	HRTCPWM3 GRP 0 CNT 1
IRQ39	47	CLB
IRQ40	48	ADC GROUP 0
IRQ41	49	ADC GROUP 1
IRQ42	50	ADC GROUP 2
IRQ43	51	ADC GROUP 3
IRQ44	52	CORDIC 0
IRQ45	53	CORDIC 1

(table continues...)

Table 232 (continued) List of PPCA CPU 0 interrupt sources

Cortex®-M33 interrupt	Cortex®-M33 exception number	Interrupt source
IRQ46	54	Not used
IRQ47	55	IPC 0
IRQ48	56	IPC 1
IRQ49	57	IPC 2
IRQ50	58	IPC 3
IRQ51	59	HRTCPWM0 MOTIF 0 IRQN
IRQ52	60	HRTCPWM0 MOTIF 1 IRQN
IRQ53	61	HRTCPWM1 MOTIF 0 IRQN
IRQ54	62	HRTCPWM1 MOTIF 1 IRQN
IRQ55	63	CPU0_CTI[0]
IRQ56	64	CPU0_CTI[1]
IRQ57	65	CPU0_FP

Table 233 List of PPCA CPU 1 interrupt sources

Cortex®-M33 interrupt	Cortex®-M33 exception number	Interrupt source
NMI	-2	Logical OR of four of the following 64 interrupt sources
IRQ0	16	Not used
IRQ1	17	Not used
IRQ2	18	Not used
IRQ3	19	Not used
IRQ4	20	Not used
IRQ5	21	CPU0_0
IRQ6	22	CPU0_1
IRQ7	23	CPU0_2
IRQ8	24	CPU0_3
IRQ9	25	CPU1_0
IRQ10	26	CPU1_1
IRQ11	27	CPU1_2
IRQ12	28	CPU1_3
IRQ13	29	HRTCPWM0 GRP 0 CNT 0
IRQ14	30	HRTCPWM0 GRP 0 CNT 1
IRQ15	31	HRTCPWM0 GRP 0 CNT 2
IRQ16	32	HRTCPWM0 GRP 0 CNT 3

(table continues...)

Table 233 (continued) List of PPCA CPU 1 interrupt sources

Cortex®-M33 interrupt	Cortex®-M33 exception number	Interrupt source
IRQ17	33	HRTCPWM0 GRP 1 CNT 0
IRQ18	34	HRTCPWM0 GRP 1 CNT 1
IRQ19	35	HRTCPWM0 GRP 1 CNT 2
IRQ20	36	HRTCPWM0 GRP 1 CNT 3
IRQ21	37	HRTCPWM0 GRP 2 CNT 0
IRQ22	30	HRTCPWM0 GRP 2 CNT 1
IRQ23	31	HRTCPWM0 GRP 2 CNT 2
IRQ24	32	HRTCPWM0 GRP 2 CNT 3
IRQ25	33	HRTCPWM1 GRP 0 CNT 0
IRQ26	34	HRTCPWM1 GRP 0 CNT 1
IRQ27	35	HRTCPWM1 GRP 1 CNT 0
IRQ28	36	HRTCPWM1 GRP 1 CNT 1
IRQ29	37	HRTCPWM1 GRP 2 CNT 0
IRQ30	38	HRTCPWM1 GRP 2 CNT 1
IRQ31	39	HRTCPWM1 GRP 2 CNT 2
IRQ32	40	HRTCPWM1 GRP 2 CNT 3
IRQ33	41	HRTCPWM2 GRP 0 CNT 0
IRQ34	42	HRTCPWM2 GRP 0 CNT 1
IRQ35	43	HRTCPWM2 GRP 0 CNT 2
IRQ36	44	HRTCPWM2 GRP 0 CNT 3
IRQ37	45	HRTCPWM3 GRP 0 CNT 0
IRQ38	46	HRTCPWM3 GRP 0 CNT 1
IRQ39	47	CLB
IRQ40	48	ADC GROUP 0
IRQ41	49	ADC GROUP 1
IRQ42	50	ADC GROUP 2
IRQ43	51	ADC GROUP 3
IRQ44	52	CORDIC 0
IRQ45	53	CORDIC 1
IRQ46	54	Not used
IRQ47	55	IPC 0
IRQ48	56	IPC 1
IRQ49	57	IPC 2

(table continues...)

Table 233 (continued) List of PPCA CPU 1 interrupt sources

Cortex®-M33 interrupt	Cortex®-M33 exception number	Interrupt source
IRQ50	58	IPC 3
IRQ51	59	HRTCPWM0 MOTIF 0 IRQN
IRQ52	60	HRTCPWM0 MOTIF 1 IRQN
IRQ53	61	HRTCPWM1 MOTIF 0 IRQN
IRQ54	62	HRTCPWM1 MOTIF 1 IRQN
IRQ55	63	CPU1_CTI[0]
IRQ56	64	CPU1_CTI[1]
IRQ57	65	CPU1_FP

Table 234 PPCA System Interrupts mapped to Main CPUSS

Interrupts	From PPCA peripherals number	PPCA IRQ mapped to Main CPUSS CM33	PPCA CPU0 IRQ Table 232	PPCA CPU1 IRQ Table 233
IRQ_PPCA_PPU	SRAMC	PPCA_IRQ[4:0]	Not available	Not available
IRQ_PPCA_CPU0[3:0]	EPU	PPCA_IRQ[8:5]	IRQ[8:5]	IRQ[8:5]
IRQ_PPCA_CPU1[3:0]	EPU	PPCA_IRQ[12:9]	IRQ[12:9]	IRQ[12:9]
IRQ_HRTCPWM [5:0]	HRTCPWM	PPCA_IRQ[38:13]	IRQ[38:13]	IRQ[38:13]
IRQ_CLB	CLB	PPCA_IRQ[39]	IRQ[39]	IRQ[39]
IRQ_PPCA_ADC_GROUP[3:0]	ATOPSS	PPCA_IRQ[43:40]	IRQ[43:40]	IRQ[43:40]
IRQ_PPCA_CORDIC[1:0]	CORDIC1 & CORDIC2	PPCA_IRQ[45:44]	IRQ[45:44]	IRQ[45:44]
IRQ_RDY_FOR_DEEPSLEEP	PPCA_CNFG	PPCA_IRQ[46]	Not available	Not available
IRQ_PPCA_IPC	IPC	PPCA_IRQ[50:47]	IRQ[50:47]	IRQ[50:47]
IRQ_PPCA_MOTIF[3:0]	HRTCPWM SS0 & SS1	PPCA_IRQ[54:51]	IRQ[54:51]	IRQ[54:51]
IRQ_PPCA_CPU0_CTI[1:0]	CPUSS	PPCA_IRQ[56:55]	IRQ[56:55]	Not applicable
IRQ_PPCA_CPU1_CTI[1:0]	CPUSS	PPCA_IRQ[58:57]	Not applicable	IRQ[56:55]
IRQ_PPCA_CPU0_FP	CPUSS	PPCA_IRQ[59]	IRQ[57]	Not applicable
IRQ_PPCA_CPU1_FP	CPUSS	PPCA_IRQ[60]	Not applicable	IRQ[57]

27.7.6 Interrupt/exception target state

27.7.6.1 Interrupt target state

On the PPCA CPU0/CPU1, each interrupt can be configured to non-secure state. As a result, an interrupt can be configured to service the Interrupt Service Routine (ISR) in the non-secure vector table. In the Armv8-M architecture, there are 16 Interrupt Target non-secure registers (ITNSn), which are used to configure the specific vector table that an interrupt uses. Each ITNSn register has 32 bits. The interrupt number is represented as 32n + m, where m is the bit number in the register and n is the register instance. Set bit m to 1 configure the interrupt target state.

27.7.7 Interrupt/exception priority

Exception priority is useful for exception arbitration when there are multiple exceptions that need to be serviced by the CPU. All exceptions other than Reset, NMI, and HardFault can be assigned a configurable priority level. The Reset and NMI have a fixed priority of –4 and –2, respectively. HardFaults on the CPU can have priority of –1 and –3. For more information on HardFault configurations with TrustZone, see the HardFault exception section. 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 7.

The Armv8-M CPU supports nested exceptions, in which a higher-priority exception can obstruct (interrupt) the currently active exception handler. This pre-emption does not happen if the incoming exception priority is the same as or lower than the active exception. The CPU resumes execution of the lower-priority exception handler after servicing the higher-priority exception. With eight priority levels supported, this allows up to eight nested exceptions. When the CPU receives two or more exception requests of the same priority, the lowest exception number is serviced first.

The priority of the exception numbers 4 to 15 can be configured by writing to the three System Handler Priority Registers (SHPR1, SHPR2, SHPR3). Each SHPRn stores the priority of four exceptions, PRI_N. Each priority number corresponds to an exception number. SHPR1 stores the priority for exceptions PRI_4 to PRI_7. SHPR2 stores the priority for exceptions PRI_8 to PRI_11. SHPR3 stores the priority for exceptions PRI_12 to PRI_15. The priority of the interrupts can be configured by writing to the respective Interrupt Priority registers (NVIC_IPRn). This is a group of 124 32-bit registers, with each register storing the priority values of four interrupts, as specified in Table 235. The interrupt number corresponding to each field is represented as 4n + m, where m is the field number in the register and n is the register instance.

Table 235 Interrupt priority register bit definitions for NVIC_IPRn

Bits	Name	Description
7:5	PRI_N0	Priority of interrupt number 4N + 0
15:13	PRI_N1	Priority of interrupt number 4N + 1
23:21	PRI_N2	Priority of interrupt number 4N + 2
31:29	PRI_N3	Priority of interrupt number 4N + 3

27.7.8 Enabling and disabling interrupts

The NVICs of the Armv8-M CPUs provide registers to individually enable and disable the interrupts in software. If an interrupt is not enabled, the NVIC does not process the interrupt requests on that interrupt line. The Armv8-M Interrupt Set Enable Register (ISERn) and the Armv8-M Interrupt Clear-Enable Register (ICERn) are used to enable and disable the interrupts, respectively. There are 16 instances of the 32-bit ISER/ICER registers in the Armv8-M architecture. The interrupt number corresponding to each bit is represented as 32n+m, where m is the bit number in the register and n is the register instance. These registers can also be read in software to get the enable status of the interrupts. Enabling and disabling interrupts/exceptions shows the register access

properties for these two registers. Note that writing zero to these registers has no effect. For CPU, ISER and ICER are not banked between security states.

Table 236 Interrupt enable/disable registers

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

The ISER and ICER registers are applicable only for interrupts. These registers cannot be used to enable or disable the exception numbers 1 to 15.

The Armv8-M SHCSR register can be configured to enable and disable UsageFault, BusFault, and MemFault. The SHCSR register has the USGFAULTENA, BUSFAULTENA, and MEMFAULTENA bits, where setting them enables the exception while clearing it disables the exception.

SysTick can be configured with the ENABLE bit in the Armv8-M SysTick Control and Status Register (SYST_CSR).

DebugMonitor can be enabled with the MON_EN bit in the Armv8-M Debug Exception and Monitor Control Register (DEMCR). Note that the C_DEBUGEN bit needs to be disabled in the Armv8-M Debug Halting Control and Status Register (DHCSR) for the DebugMonitor to function.

Note that HardFault, NMI, SVCall, and PendSV are always enabled and therefore do not have an exception enable bit.

Interrupts and exceptions can also be disabled using the Armv8-M special purpose mask register (PRIMASK, FAULTMASK, and BASEPRI). The PRIMASK register 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 232. The FAULTMASK register can be used to disable all faults and interrupts except for the NMI exception. The BASEPRI register can be used to disable interrupts based on priority level. Only interrupts with a priority value lower (higher priority) than what is in the BASEPRI register can trigger. When PRIMASK, FAULTMASK, or BASEPRI are set, none of the masked exceptions can be serviced by the CPU. The masked exceptions remain in a pending state until the mask is cleared.

27.7.9 Interrupt/exception states

Each exception can be in one of the following states:

Table 237 Interrupt/exception states

Exception state	Meaning
Inactive	The exception is not active or pending. Either the exception is disabled or the enabled exception has not been triggered.

(table continues...)

Table 237 (continued) Interrupt/exception states

Exception state	Meaning
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 request has been serviced by the CPU but whose exception handler execution is not yet complete. A high-priority exception can interrupt the execution of a 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 Armv8-M Interrupt Control State Register (ICSR) contains status bits describing the various exception states.

- The VECTACTIVE bits ([8:0]) in the ICSR store the exception number for the current executing exception. This value is zero if the CPU does not execute any exception handler (the CPU is in thread mode). Note that the value in the VECTACTIVE bitfields 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 VECTPENDING bits ([20:12]) in the ICSR store the exception number of the highest priority pending exception. This value is zero if there are no pending exceptions

The ISRPENDING bit (bit 22) in the ICSR indicates if a NVIC-generated interrupt is in a pending state

27.7.9.1 Pending interrupts/exception

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 all interrupt lines by providing separate register bits for setting and clearing the pending states of the interrupts. The Interrupt Set-Pending register (ISPRn) and the Interrupt Clear-Pending register (ICPRn) are used to set and clear the pending status of the interrupt lines. There are 16 instances of the 32-bit ISPR/ICPR registers, where each bit corresponds to an interrupt. The interrupt number corresponding to each bit is represented as 32n + m, where m is the bit number in the register and n is the register instance. Table 237 shows the register access properties for these two registers. Note that writing zero to these registers has no effect.

Table 238 Interrupt set pending/clear pending registers

Register	Operation	Bit value	Comment
Interrupt Set-Pending Register (ISPRn)	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 (ICPRn)	Write	1	To clear a pending interrupt

(table continues...)

Table 238 (continued) Interrupt set pending/clear pending registers

Register	Operation	Bit value	Comment
		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 does not move to the pending state until it is enabled by writing to the ISER register. Note that the ISPR and ICPR registers are used only for peripheral interrupts. These registers cannot be used for pending exception numbers 1 to 15.

The pending bits for exceptions 1 to 15 can be found in the Armv8-M SHCSR, ICSR, and DEMCR registers. The SHCSR has the HARDFAULTPENDED, SVCALLPENDED, BUSFAULTPENDED, MEMFAULTPENDED, and USGFAULTPENDED bits, which are the pending bits for the exceptions with the same name. There are no clear pending registers for these exceptions. Writing and clearing these bits sets and clears the pending state of the exception.

The ICSR has the PENDNMISET, PENDSVSET, and PENDSTSET bits, which set the NMI, PendSV, and SysTick exceptions to the pending state when writing one. The ICSR also has the PENDNMICLR, PENDSVCLR, and PENDSTCLR bits, which clear the NMI, PendSV, and SysTick exceptions when writing one.

The DEMCR register has the MON_PEND bits, which are used to set or clear the pending state of the DebugMonitor.

27.7.10 Stack usage for interrupts/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. Armv8-M CPUs without TrustZone have two stack pointers: the Main Stack Pointer (MSP) and the Process Stack Pointer (PSP). The Armv8-M CPUs with TrustZone enabled have four stack pointers to separate between secure and non-secure states: MSP_NS, MSP_S, PSP_NS, and PSP_S. 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 downward 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) section for details.

27.7.11 Interrupt/exception - initialization/configuration

This section covers the different steps involved in initializing and configuring exceptions in the PPCA CPU0/ CPU1.

1. Configuring the Exception Vector Table Location: The first step in using exceptions is to configure the vector table location as required in volatile memory. This configuration is done as described in the Exception vector table section.
2. Configuring individual exceptions: The next step is to configure individual exceptions required in an application, as explained in the 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 PPCA peripherals section for more information about the interrupt configuration supported by them
   b. For the PPCA CPU0/CPU1, configure the NVIC to use the vector table when an interrupt source is triggered, as described in the interrupt target state
   c. Define the exception handler function and write the address of the function to the exception vector table. Table 232 gives the exception vector table format; write the exception handler address to the appropriate exception number entry in the table
   d. Set up the exception priority, as explained in the Interrupt/exception priority section
   e. Enable the exception, as explained in Enabling and disabling interrupts/exceptions

28 PPCA configuration (PPCA CNFG)

The PPCA CNFG (CoNFiGuration block) module embeds the PPCA Control register (CTRL), the Peripherals Allocation register (CNFG), and the CPUs Clock and Reset registers (CPU_CTRL).

Additionally, the CNFG module contains the mirrored ADC Results Data registers. This module allows for the selection of inputs to be presented to the EPU (Event Processing Unit) and determines which outputs should be propagated to the PPCA (Programmable Peripheral Control Array) boundary.

The PPCA CNFG is an AHB slave peripheral, and can only be accessed from the main CPU, as the main CPU is the only one entitled to modify the configuration.

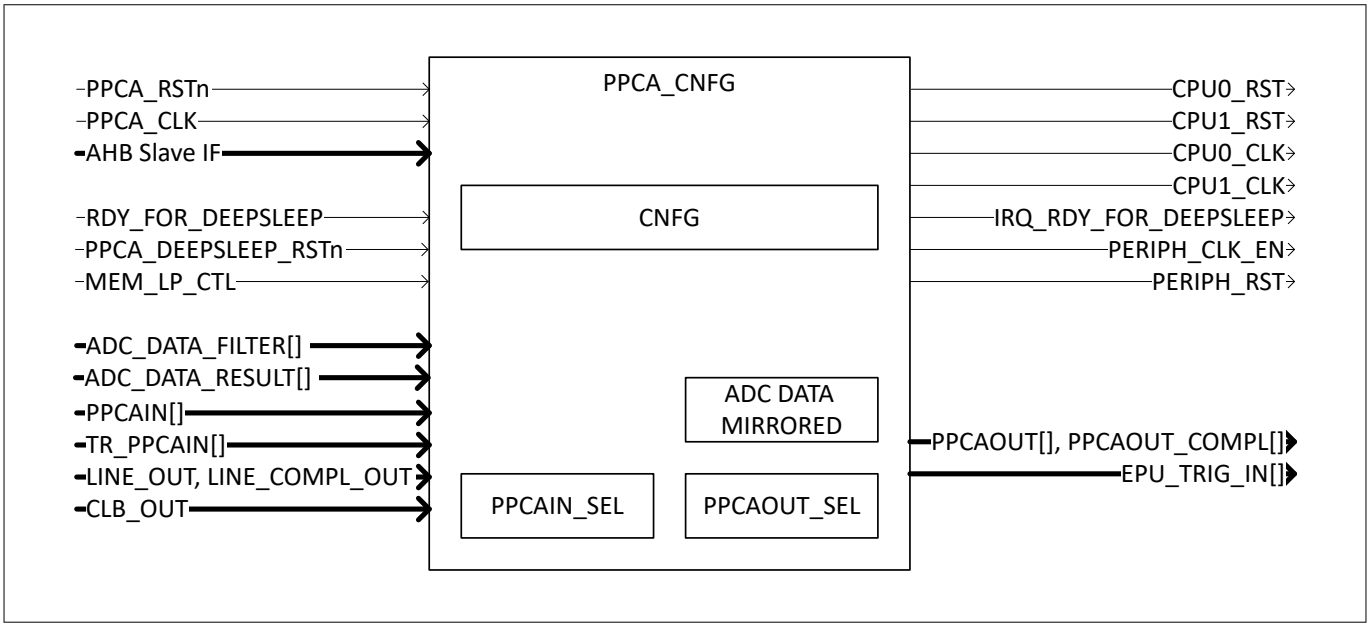


Figure 205 PPCA CNFG

28.1 CNFG

The CNFG section includes the PPCA Peripherals Allocation registers, the CPUs Clock and Reset Control registers, and the main PPCA Control Register.

The functionality of the PPCA has to be enabled by setting the proper bit in the Control Register, when the PPCA is disabled, the PPCA functions are in reset, and the clock is gated.

The PPCA Control Register also manages the Power State of the PPCA, by enabling the possibility to gate all the PPCA peripherals clocks, in case the RDY_FOR_DEEPSLEEP signal is asserted. The signal is also propagated to the output through the IRQ bus.

28.2 PPCA peripheral interconnect

The peripheral interconnect block is multiplexing the two input master driven by the PPCA CPUSS bus matrix to all the peripheral AHB interfaces. It also grants or denies access from each of the master interfaces to the peripherals according to the programmed configuration in the PPCA_CNFG block. Each peripheral can only be assigned to one master at a time.

In case a master is accessing a denied peripheral, an AHB error response is given to indicate an illegal access.

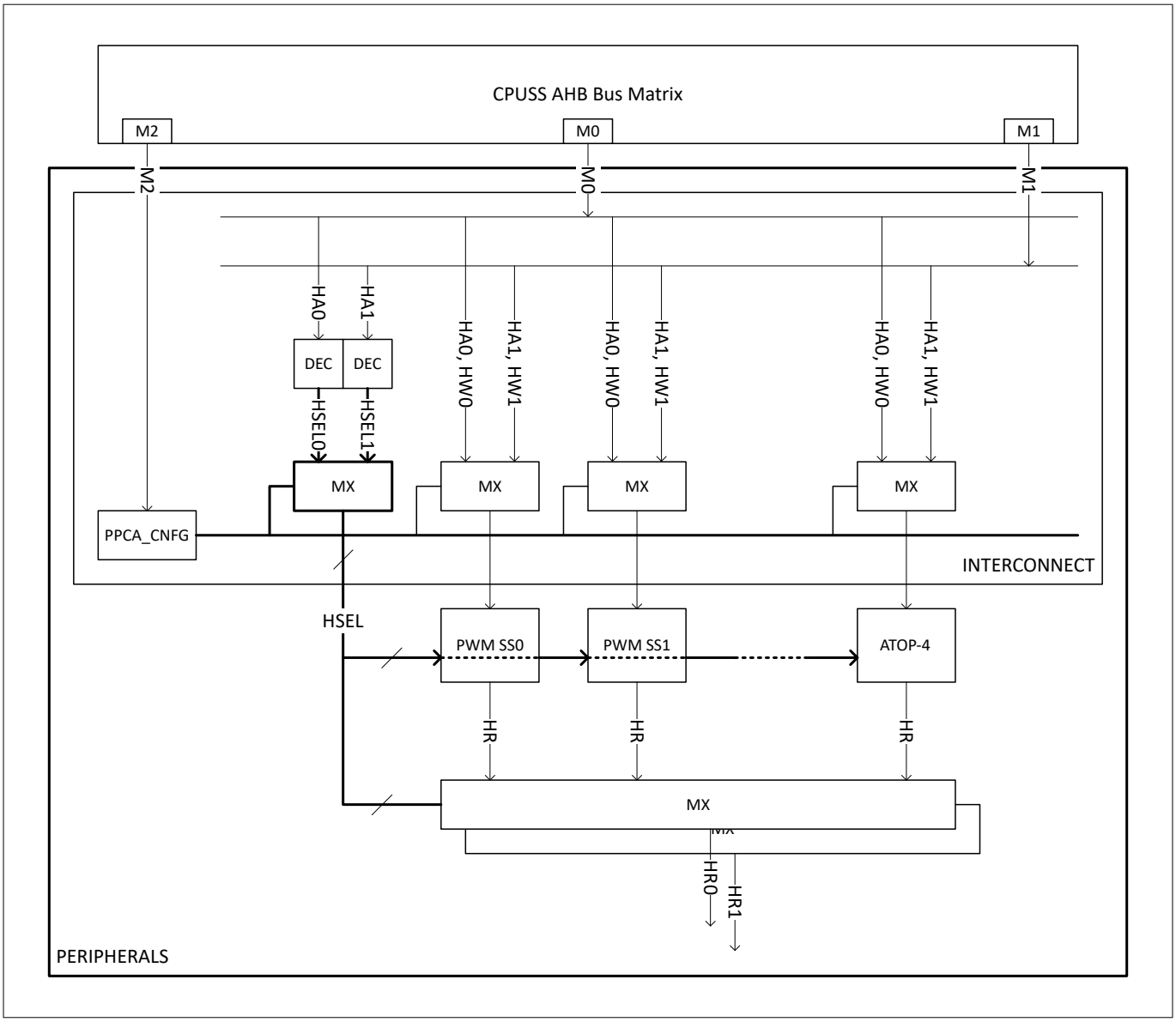


Figure 206 PPCA peripherals interconnect

28.3 ADC data mirror

The ADC MIRROR module implements a multiplexer, enabling the main CPU to read the ADC results data registers.

This register set provides the readability of data from ATOP ADC to be accessed by the external main CPU, without interference to the internal CPU data paths, removing the need for arbitration.

28.4 PPCA input selector module (PPCAIN_SEL)

The PPCA Input Selector module (PPCAIN_SEL) instantiates a group of multiplexers to select up to 16 PPCAIN input signals and up to 16 TR_PPCAIN input signals, available at the PPCA boundary, to be propagated to the EPU module.

The multiplexers are controlled by a dedicated Control register and each output is synchronized or pipelined in the PPCA_CLK domain.

Table 239 PPCAIN source

Source	To the boundary of PPCA
P8.0	PPCAIN[0]
P8.1	PPCAIN[1]
P8.2	PPCAIN[2]
P8.3	PPCAIN[3]
P8.4	PPCAIN[4]
P8.5	PPCAIN[5]
P9.0	PPCAIN[6]
P9.1	PPCAIN[7]
P9.2	PPCAIN[8]
P9.3	PPCAIN[9]
P1.4	PPCAIN[10]
P1.5	PPCAIN[11]
P1.6	PPCAIN[12]
P1.7	PPCAIN[13]
P5.2	PPCAIN[14]
P5.3	PPCAIN[15]
P6.0	PPCAIN[16]
P6.1	PPCAIN[17]
P6.2	PPCAIN[18]
P6.3	PPCAIN[19]
P4.3	PPCAIN[20]
P4.2	PPCAIN[21]
P4.1	PPCAIN[22]
P4.0	PPCAIN[23]
P4.4	PPCAIN[24]
P4.5	PPCAIN[25]
P4.6	PPCAIN[26]
P4.7	PPCAIN[27]
P5.0	PPCAIN[28]
P5.1	PPCAIN[29]
P3.0	PPCAIN[31]
P3.1	PPCAIN[30]
P7.0	PPCAIN[32]

(table continues...)

Table 239 (continued) PPCAIN source

Source	To the boundary of PPCA
P7.1	PPCAIN[33]
P7.2	PPCAIN[34]
P7.3	PPCAIN[35]
P7.4	PPCAIN[36]
P7.5	PPCAIN[37]
P7.6	PPCAIN[38]
P7.7	PPCAIN[39]
P3.2	PPCAIN[40]
P3.3	PPCAIN[41]
P3.4	PPCAIN[42]
P3.5	PPCAIN[43]
P3.6	PPCAIN[44]
P3.7	PPCAIN[45]
P6.4	PPCAIN[46]
P6.5	PPCAIN[47]
P2.0	PPCAIN[48]
P2.1	PPCAIN[49]
P2.2	PPCAIN[50]
P2.3	PPCAIN[51]
P2.4	PPCAIN[52]
P2.5	PPCAIN[53]
P0.0	PPCAIN[54]
P0.1	PPCAIN[55]
P1.0	PPCAIN[56]
P1.1	PPCAIN[57]
P1.2	PPCAIN[58]
P1.3	PPCAIN[59]
NA	PPCAIN[60:63]

TR_PPCAIN is output of the Trigger Mux

Table 240 TR_PPCAIN signal routing

From the boundary of PPCA	MUX_TR_PPCAIN #	To EPU
TR_PPCAIN[63:0]	15, . . . , 0	TR_PPCAIN_SEL[MUX_TR_PPCAIN]

Table 241 PPCAIN signal routing

From the boundary of PPCA	MUX_PPCAIN #	To ATOP/EPU
PPCAIN [63:0]	5, …, 0	SDIN [MUX_PPCAIN]/PPCAIN_SEL [MUX_PPCAIN]
PPCAIN [63:0]	15, … , 6	PPCAIN_SEL [MUX_PPCAIN]

The selected PPCAIOs are also monitored on a dedicated read only register.

28.5 PPCA output selector module (PPCAOUT_SEL)

The PPCA Output Selector module (PPCAOUT_SEL) has a group of multiplexers to select which signal between PWM outputs (LINE_OUT and LINE_COMPL_OUT) and CLB outputs (CLB_LINE_OUT, CLB_LINE_COMPL_OUT) that needs to be propagated to the PPCA boundary signals (PPCAOUT).

The multiplexers are controlled by a dedicated Control register bit (PPCAOUT_SEL_SRC) or by a dedicated input signal (HW_SEL_SRC), coming from EPU.

The PPCA Output Selector module (PPCAOUT_SEL) also has a group of multiplexers to select, which signal must be associated and propagated to the TR_PPCAOUT_SEL [51:0] bus.

Those multiplexers are controlled by the same signals controlling the multiplexer related to the PPCAOUT signal.

Table 242 PPCAOUT, TR_PPCAOUT_SEL signal routing (part 1)

Input from Peripherals	Signals	PPCAOUT_SEL_SRC or HW_SEL_SRC	Output to Boundary of PPCA (PPCAOUT, TR_PPCAOUT_SEL)
TCPWM SS0	LINE_OUT[i], i = 0, 1, 2, 3, 4, 5, 6, 7	0	PPCAOUT[k], k = 0, 2, 4, 6, 8, 10, 12, 14
	LINE_COMPL_OUT[i], i = 0, 1, 2, 3, 4, 5, 6, 7	0	PPCAOUT[k], k = 1, 3, 5, 7, 9, 11, 13, 15
CLB	CLB_LINE_OUT[i], i = 0, 1, 2, 3, 4, 5, 6, 7	1	PPCAOUT[k], k = 0, 2, 4, 6, 8, 10, 12, 14
	CLB_LINE_COMPL_OUT[i], i = 0, 1, 2, 3, 4, 5, 6, 7	1	PPCAOUT[k], k = 1, 3, 5, 7, 9, 11, 13, 15

Table 243 PPCAOUT, TR_PPCAOUT_SEL signal routing (part 2)

Input from Peripherals	Signals	Output to Boundary of PPCA (PPCAOUT, TR_PPCAOUT_SEL)
TCPWM SS1	LINE_OUT[i], i = 0, 1, 2, 3	PPCAOUT[k], k = 16, 18, 20, 22

(table continues...)

Table 243 (continued) PPCAOUT, TR_PPCAOUT_SEL signal routing (part 2)

Input from Peripherals	Signals	Output to Boundary of PPCA (PPCAOUT, TR_PPCAOUT_SEL)
TCPWM SS1	LINE_COMPL_OUT[i], i = 0, 1, 2, 3	PPCAOUT[k], k = 17, 19, 21, 23
TCPWM SS0	LINE_OUT[i], i = 8, 9, 10, 11	PPCAOUT[k], k = 24, 26, 28, 30
TCPWM SS0	LINE_COMPL_OUT[i], i = 8, 9, 10, 11	PPCAOUT[k], k = 25, 27, 29, 31
TCPWM SS1	LINE_OUT[i], i = 4, 5, 6, 7	PPCAOUT[k], k = 32, 34, 36, 38
TCPWM SS1	LINE_COMPL_OUT[i], i = 4, 5, 6, 7	PPCAOUT[k], k = 33, 35, 37, 39
TCPWM SS2	LINE_OUT[i], i = 0, 1, 2, 3	PPCAOUT[k], k = 40, 42, 44, 46
TCPWM SS2	LINE_COMPL_OUT[i], i = 0, 1, 2, 3	PPCAOUT[k], k = 41, 43, 45, 47
TCPWM SS3	LINE_OUT[i], i = 0, 1	PPCAOUT[k], k = 48, 50
TCPWM SS3	LINE_COMPL_OUT[i], i = 0, 1	PPCAOUT[k], k = 49, 51

***Note**: Exported PPCAOUT signals not explicitly assigned are assumed to be grounded*

There is a PPCA enable output bit to enable PPCA outputs. In case an error event is generated in PPCA (ECC non-correctable fault, AHB error) or the PPCA is ready to DEEPSLEEP, this enable needs to be overwritten to disconnect the PPCA from the outputs.

28.6 Fault handler

The FAULT_HANDLER latches the FAULT events generated inside the CPUSS and propagated at the boundary of PPCA (that is, the Non-Correctable-FAULT, the BUS ERROR, and the HARD FAULT) to permanently assert the KILL signal input, to the TCPWM or CLB modules.

The latched fault signal (FAULT_LATCHED) can be released by the main CPU, by writing into the FAULT_STATUS bit of the PPCA_CNFG CTRL register, or by resetting the PPCA block.

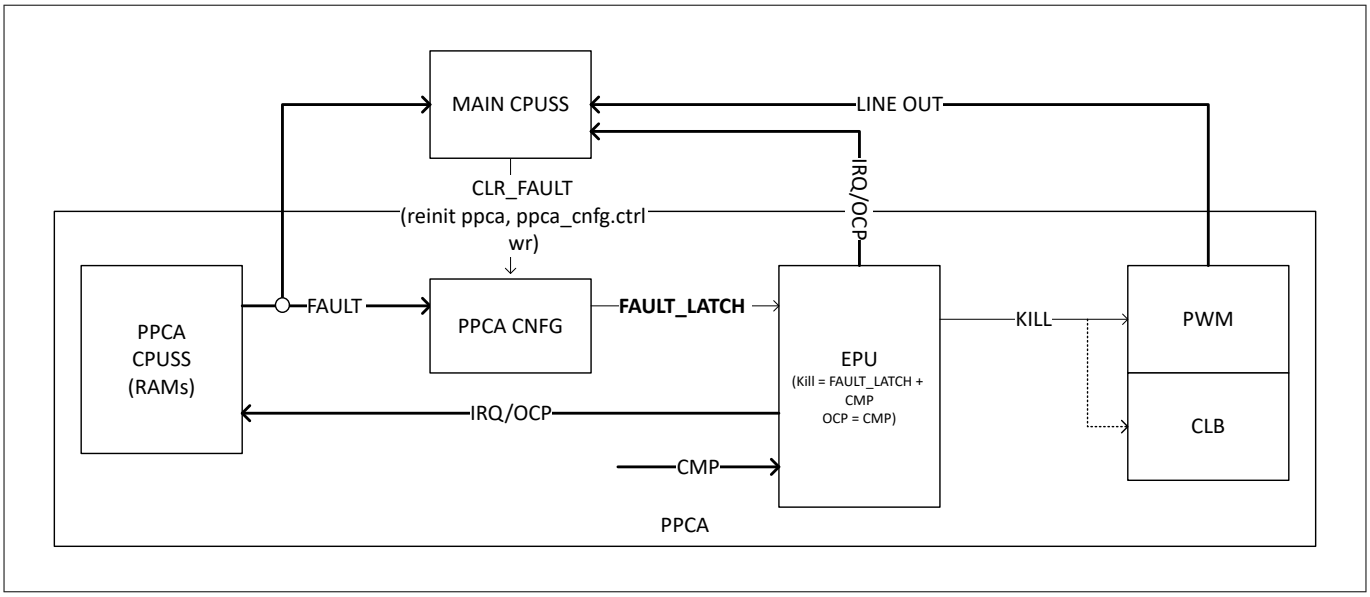


Figure 207 Fault handler

28.7 S2IRQ (Signal to interrupt)

The S2IRQ module has a logic to convert an input signal into a CPU0/CPU1 interrupt event. The rising edge of the signal is converted into interrupt events and properly managed. These interrupts are also mapped to the PPCA interrupt bus.

Table 244 Signal to interrupt mapping table

S2IRQ Block Register Index	Signal Source	Mapped Interrupt	PPCA IRQ
[0]	PPCA Internal	IRQ_RDY_FOR_DEEPSLEEP	PPCA_IRQ[46]
[1]	from CPUSS: CPU0_CTI[0]	IRQ_PPCA_CPU0_CTI[0]	PPCA_IRQ[55]
[2]	from CPUSS: CPU0_CTI[1]	IRQ_PPCA_CPU0_CTI[1]	PPCA_IRQ[56]
[3]	from CPUSS: CPU1_CTI[0]	IRQ_PPCA_CPU1_CTI[0]	PPCA_IRQ[57]
[4]	from CPUSS: CPU1_CTI[1]	IRQ_PPCA_CPU1_CTI[1]	PPCA_IRQ[58]
[5]	from CPUSS: CPU0_FP	IRQ_PPCA_CPU0_FP	PPCA_IRQ[59]
[6]	from CPUSS: CPU1_FP	IRQ_PPCA_CPU1_FP	PPCA_IRQ[60]

29 PPCA Event Processing Unit (EPU)

The PPCA includes an event processing unit (EPU), which provides a flexible event-management system capable of generating triggers based on a linear combination of processed input events. The EPU receives input events from peripherals within the PPCA, from the TriggerMux outside the PPCA, and from PORT/PADs (PPCAIN). It generates output triggers that are routed to peripherals inside the PPCA and to the TriggerMux.

The EPU can also generate interrupt events for the PPCA CPUs by processing and combining EPU input events or processed triggers. Additionally, it can process and route input triggers as events to peripherals outside the PPCA, such as the DW and SCB. This functionality ensures efficient and effective event processing across the entire system.

29.1 Features

- Processes input events, optionally within a qualified window, as the qualification window can be enabled or disabled
- Supports connection of triggers or signals from a main CPU to PPCA subsystem peripherals
- Supports up to 192 event inputs and up to 128 trigger outputs
- Includes a total of 48 processing units (PU) supporting polarity inversion, edge extraction, masking, delaying, debouncing, blanking, and bypass
- Enables the combination of multiple events and advanced event combinations
- Generates complex hardware triggers without CPU intervention
- Allows trigger outputs to be generated from either the output of a PU process, pure combinatorial logic, or a combination of both

29.2 Architecture

The event processing unit subsystem module embeds the:

- EPU module
- EPU IRQ Generator
- AHB-to-AHB bus bridge

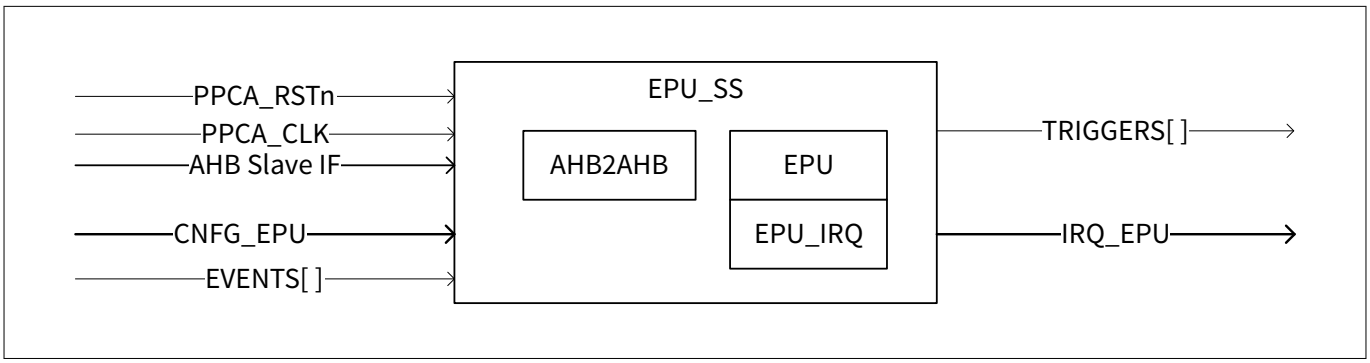


Figure 208 EPU subsystem

29.3 EPU

The EPU instantiates thirty-two (32) Processing Unit Type_1 (PU_T1) modules and sixteen (16) Processing Unit Type_2 (PU_T2) modules. These modules operate between a selected input, which is connected to an event generated by PPCA resources (CPUs and peripherals), and an output, which is connected to a trigger for PPCA resources (CPUs and peripherals).

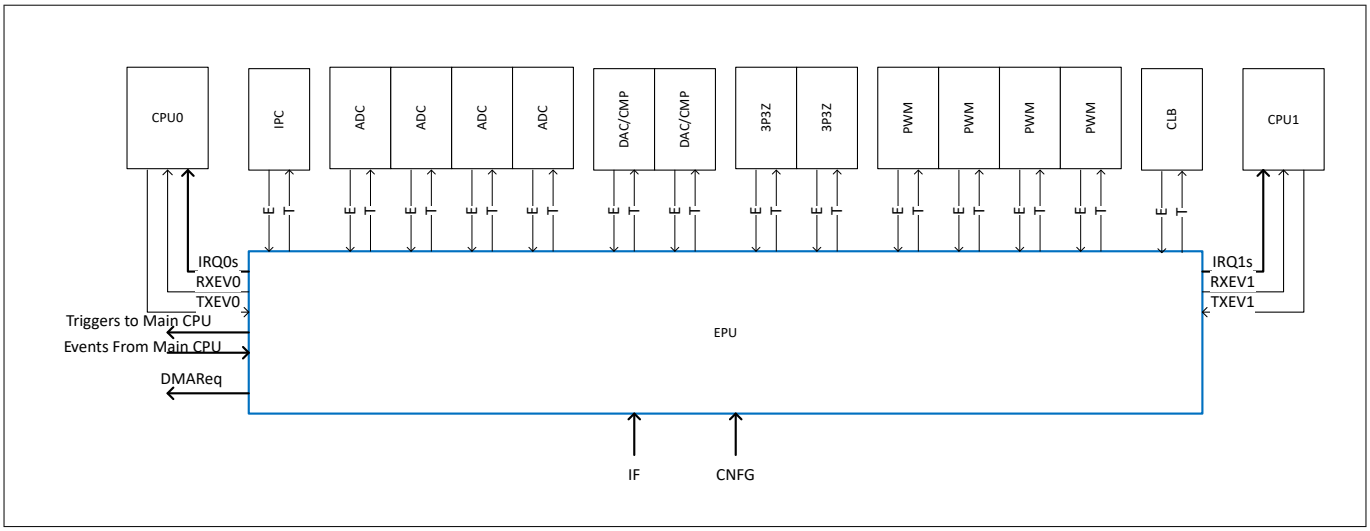


Figure 209 EPU connections

The purpose of the EPU module is to create a hardware sequence of processed, timed triggers to enable the operation of PPCA subsystem modules. This functionality helps offload the instantiated processors and accelerates operations. Additionally, the EPU is designed to send interrupts to CPU0 or CPU1 as well as at the boundary of the PPCA. At the boundary of the PPCA, it facilitates the sharing of triggers between the SoC and the PPCA.

Figure 209 shows the EPU architecture.

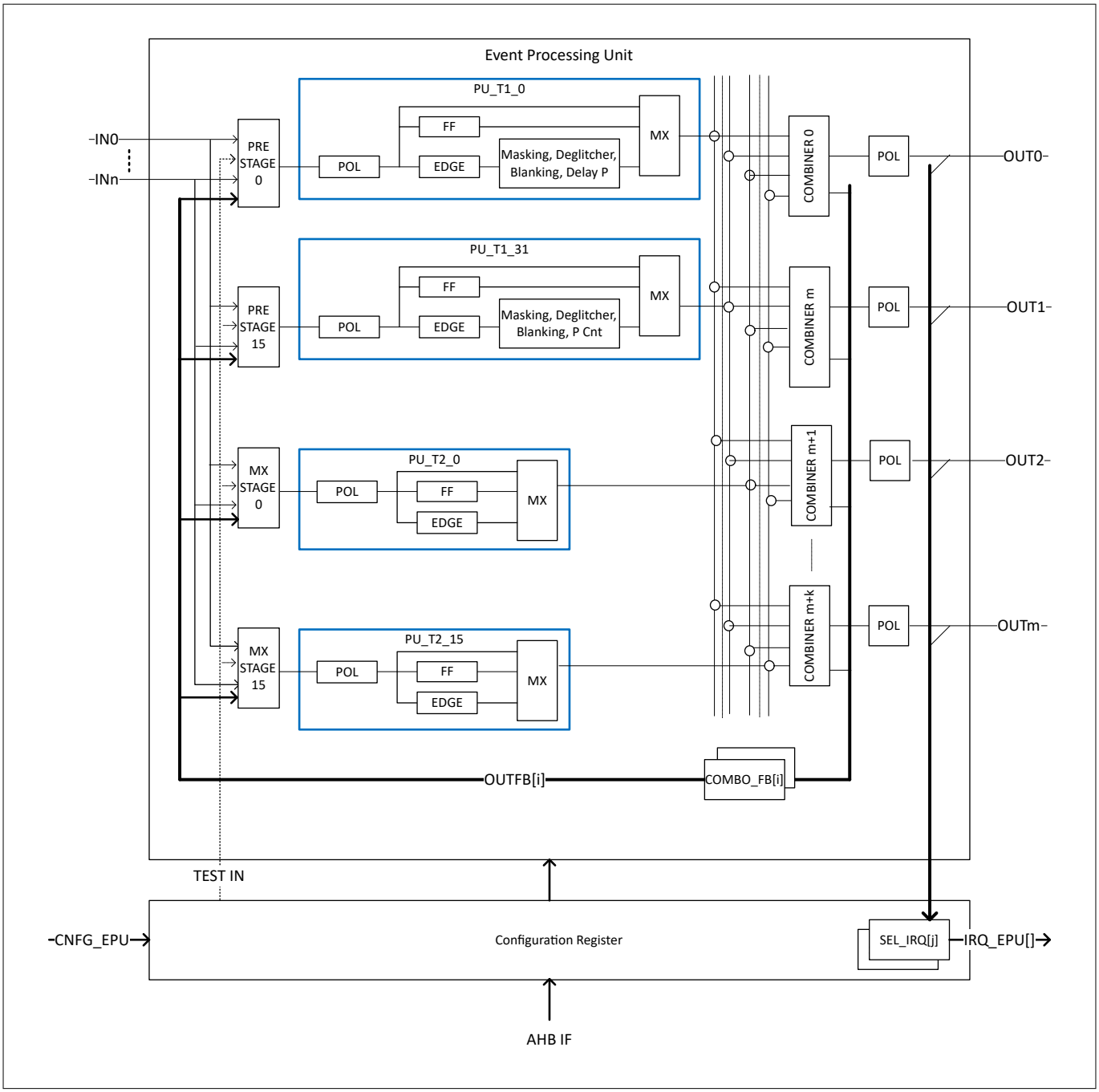


Figure 210 EPU architecture

29.4 Register blocks

There are two register blocks:

- The Configuration register block is used to configure the input source of the processing units, the function to be executed by the processing units, and the combiner associated with each defined trigger
- The Control register is used to control the EPU

Three types of errors can be reported during an AHB access:

1. **Unaligned address**: The address must be 32-bit aligned. If the address is not aligned, the access is dropped, and an error is generated on the AHB interface

2. **Invalid master access**:
   - Each PPCA CPU core is allowed to access only the processing unit resources specified for it in the provided configuration
   - The external master always has access to all resources
   - Resource allocation is defined below:
     - `hmaster_bus = 1'h0`: Access is permitted to CPU0 and main CPU
     - `hmaster_bus = 1'h1`: Access is permitted to CPU1 and main CPU
   - If a CPU attempts to access EPU registers and its access is not permitted by the EPU_CNFG register, the access is dropped, and an error is generated on the AHB interface
3. **Registers access when EPU is disabled**: When the EPU is disabled, only the CTRL register can be accessed. Any attempt to access other registers results in HRESP=ERROR

***Notes**:*

1. *For unaligned address errors, the error is generated directly by the AHB interface, and the response lasts two clock cycles*
2. *For invalid master access and disabled EPU errors, the errors are generated by the EPU logic, and the response includes an additional clock cycle*
3. *Any attempt to access PU or COMBO registers by a master other than CPU0, CPU1, or the main CPU results in an HERROR response*

29.5 Processing Unit_Type 1 (PU_T1)

- Asynchronous bypass
- Synchronous bypass with pipelined flip-flop insertion
- Masking filter supporting up to 128 clock cycles
- Blanking filter supporting up to 128 clock cycles
- Debouncing filter through integration, supporting up to 128 clock cycles
- Pulse delay supporting up to 128 clock cycles
- Input polarity inversion
- Input edge detection (low-to-high transition)

29.5.1 PU_T1 block

The Processing Unit_Type 1 (PU_T1) block consists of two parts:

- Prestage
- Filtering functions

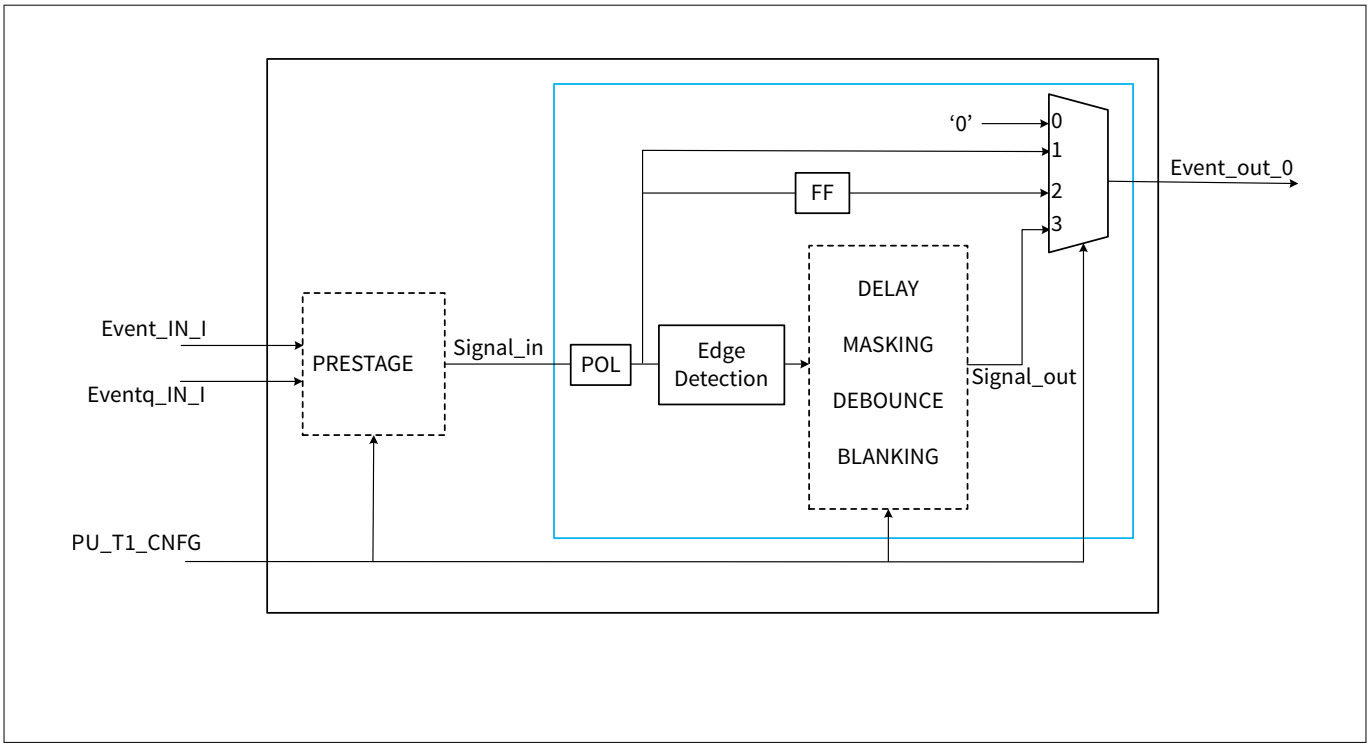


Figure 211 Processing Unit_Type 1 (PU_T1)

29.5.2 PU_T1

Prestage

The purpose of the Prestage logic is to process the input signal within a specified window, if enabled, and to change the polarity of the input signal. Additionally, a multiplexer (MUX) is used to force the input to the filtering logic to a value specified by the field `PU_T1_CNFG.CONSTANT_IN` when the field `PU_T1_CNFG.SEL_CONSTANT_IN` is set.

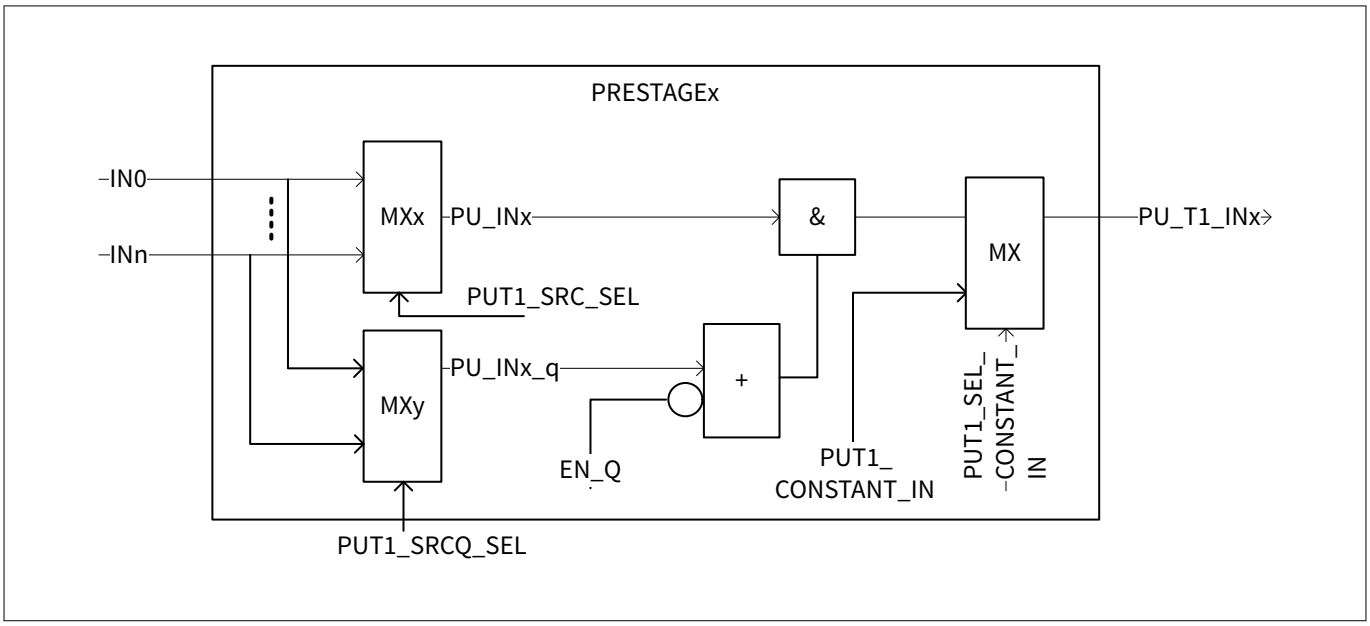


Figure 212 Prestage block diagram

Configuration details of Prestage:

1. **Selecting the input event**: Use the register `PU_T1_CNFG.PUT1_SRC_SEL` to select the input signal or event from the peripheral. This determines which signal the logic will process
2. **Configuring the qualification window**:
   - Use the field `PU_T1_CNFG.PUT1_SRCQ_SEL` to select the qualifier event
   - To enable the qualification window, set the field `PU_T1_CNFG.PUT1_ENQ` to 1. This ensures that the Prestage logic processes the input signal only during the qualification event
3. **Bypassing the qualification window**: If qualification is not required, bypass this stage by setting the field `PU_T1_CNFG.PUT1_ENQ` (or `PU_T1_CNFG.PUT1_ENQ_SEL`) to 0. This enables the input signal to proceed directly to the next stage of processing without checking for the qualification event
4. **Using the MUX with forced constant input**:
   - A MUX is included to override the input signal to the filtering logic with a constant value
   - The forced constant value is determined by the field `PU_T1_CNFG.CONSTANT_IN`
   - The forced constant input is applied when the field `PU_T1_CNFG.SEL_CONSTANT_IN` is set. This feature is useful for testing, debugging, or forcing specific conditions during operation

Polarity inversion block

The polarity inversion block modifies the signal output from the Prestage logic. Its main purpose is to invert the polarity of the signal (i.e., logical high becomes low, and logical low becomes high) based on a configuration setting:

- If polarity inversion is enabled, the signal is inverted before being passed to the next stage
- If polarity inversion is disabled, the signal is forwarded unchanged to the next stage

The PU_T1 block includes a filtering stage that processes the input signal based on the selected configuration. However, in specific scenarios, users may choose to bypass the filtering logic to handle the signal directly:

- When the field `PU_T1_CNFG.PUT1_EN` is configured for synchronous or asynchronous bypass, the input signal skips the filtering logic (see Synchronous bypass)
- In this case, the signal is passed directly from the polarity inversion block to its output (`event_out`)
- The `event_out` signal acts as an input to a combiner block

Edge detection

The edge detection feature behaves differently based on the value of the configuration field `PU_T1_CNFG.PUT1_IN_SEL`:

- When `PU_T1_CNFG.PUT1_IN_SEL = 1`:
  - The pulse detection result is used as the trigger for the filtering operation
  - Since it cannot be guaranteed whether the pulse results from an external signal change or a setup change, it is preferred to mask this pulse in the filter logic during the first clock cycle after enabling the operation
- When `PU_T1_CNFG.PUT1_IN_SEL = 0`:
  - The operation becomes level-based
  - Masking or blanking can be applied since the current signal level reflects the value of the input line at the moment the filtering operation is enabled

29.5.3 Filtering functions

- Applies configured filtering rules to the input signal based on the selected detection mode
  - Pulse delay
  - Masking filter

- Blanking filter
- Debouncing filter

- Forwards the filtered signal or event to the next stage, such as a combiner block

29.5.3.1 Pulse delay

The PU_T1 module executes the pulse delay function when the field `PU_T1_CNFG.OPCODE` is set to `0x0`.

- This function delays the rising edge of the input signal `signal_in` by a number of clock cycles specified in the field `PU_T1_CNFG.DATA + 1`
- The delay can be configured for up to 128 clock cycles

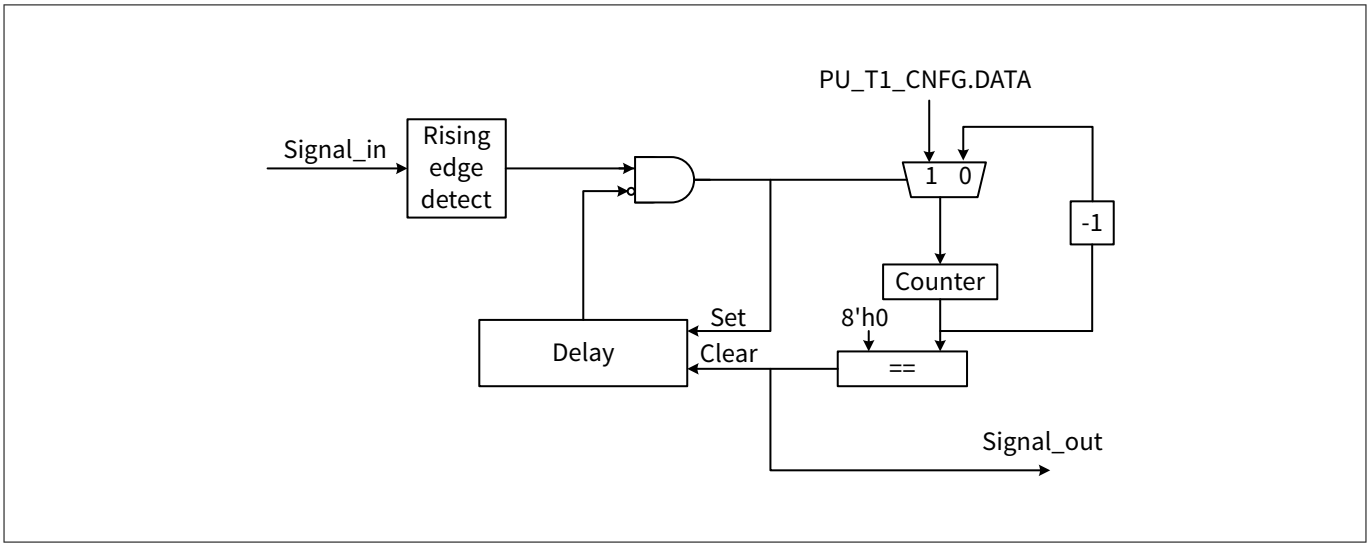


Figure 213 PU_T1 pulse delay

- When a rising edge is detected, a delay window is initiated
- Any additional rising edge that occurs during the delay window is ignored
- Once the delay window is complete, a single pulse is generated on the output signal (`signal_out`)

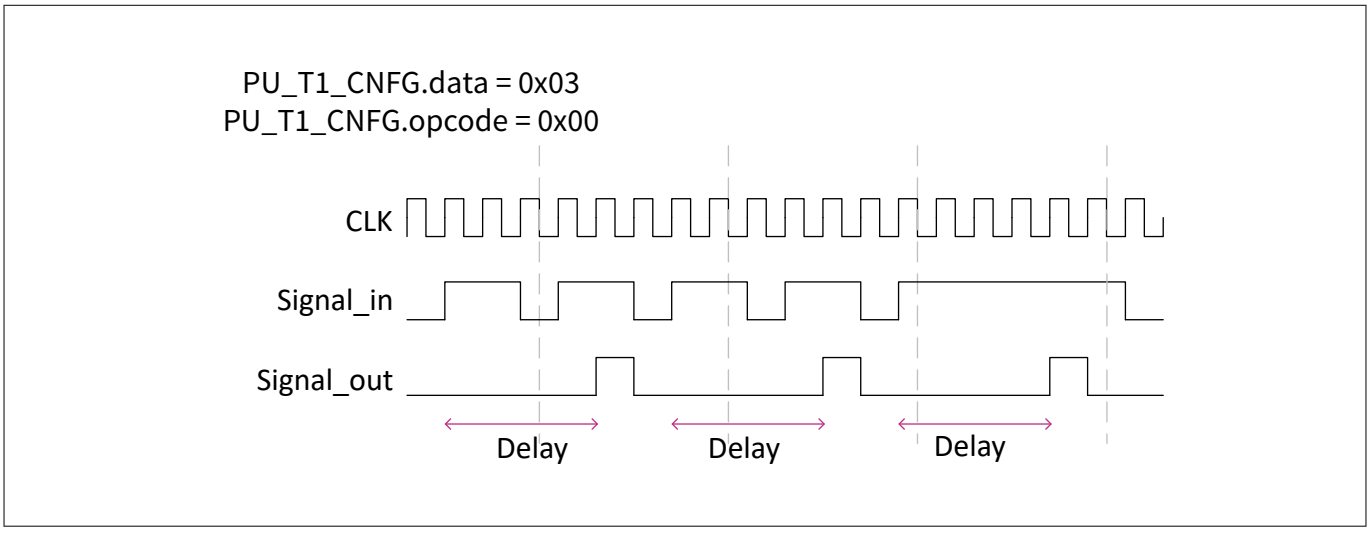


Figure 214 Pulse delay waveform

Configuration of pulse delay:

- **Pulse delay enable condition**:
  - The Pulse delay function is activated when the field `PU_T1_CNFG.OPCODE` is set to `0x0`
  - This mode delays the propagation of the rising edge of the input signal (`signal_in`) by a configurable number of clock cycles
- **Delay window configuration**:
  - The delay duration is determined by the value in the field `PU_T1_CNFG.DATA + 1` clock cycle
  - The delay can be configured for up to 128 clock cycles, allowing for a wide range of delay durations and precise timing adjustments
- **Trigger for pulse delay**:
  - Upon detecting a rising edge on `signal_in`, the delay window starts
  - Any additional rising edge of the input signal occurring during the delay window is ignored (dropped). This ensures that only the initial rising edge within the delay period is processed, preventing multiple triggers
- **Output pulse behavior**:
  - After the delay window completes, a single pulse is generated on the output signal (`signal_out`)
  - This output pulse corresponds to the delayed version of the first rising edge detected in `signal_in`

***Note**: In this configuration, the value of the field `PU_T1_CNFG.PUT1_IN_SEL` is disregarded, except in synchronous bypass mode (see Synchronous bypass).*

29.5.3.2 Masking filter

The PU_T1 block operates in Masking mode when the field `PU_T1_CNFG.OPCODE` is set to `0x1`.

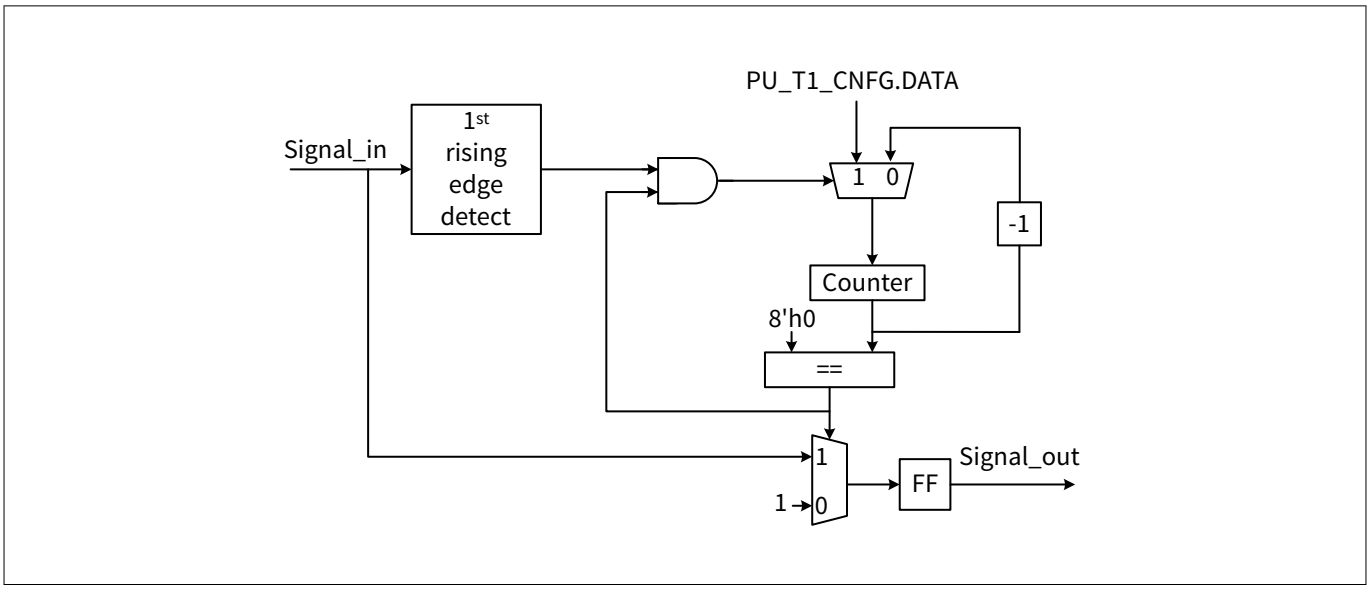


Figure 215 PU_T1 masking filter

The masking filter is used to stretch the input signal for a duration specified by the value in the field `PU_T1_CNFG.DATA + 1` (up to 128 clock cycles).

- When a rising edge of the input signal is detected, the masking window begins
- The input signal is stretched for the duration of the specified mask window
- At the end of the masking window, the output signal takes the current value of the input signal

The waveform shown in Figure 216 the behavior when `PU_T1_CNFG.DATA` is set to `0x07`.

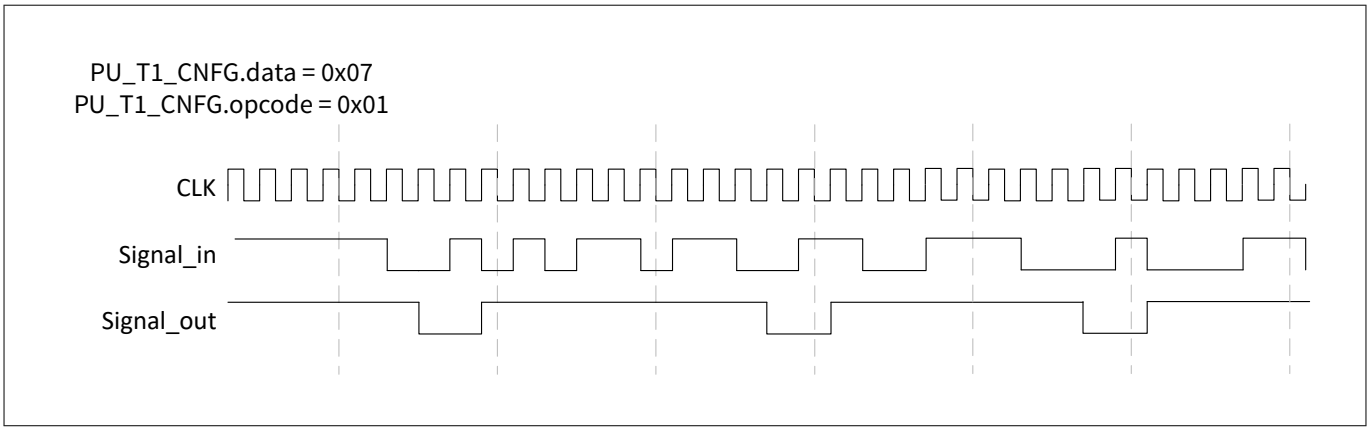


Figure 216 Masking filter waveform

Configuration of masking filter

- **Masking mode enable condition**: The PU_T1 block operates in Masking mode when the field `PU_T1_CNFG.OPCODE` is set to `0x1`. This configuration activates the masking filter logic
- **Duration of the masking window**:
  - The length of the masking window is determined by the value in the field `PU_T1_CNFG.DATA`
  - The duration is calculated as:

```
Masking Window = PU_T1_CNFG.DATA + 1 clock cycles
```

  - The masking window can range from 1 to 128 clock cycles because `PU_T1_CNFG.DATA` is an 8-bit field with a maximum value of `0x7F`
- **Trigger for masking window**:
  - The masking window is triggered by detecting a rising edge on the input signal
  - Once triggered, the masking filter logic stretches the signal so that the output remains high for the full duration of the masking window, even if the input signal transitions to low during this time
- **End of masking window**:
  - At the end of the masking window, the output signal reflects the value of the input signal at that moment
  - If the input signal is still high, the output remains high. If the input signal is low, the output transitions to low

29.5.3.3 Blanking filter

The blanking filter feature is enabled when the field `PU_T1_CNFG.OPCODE` is set to `0x2`.

- **Purpose**: The blanking filter is designed to eliminate glitches that may occur following a pulse, within the defined blanking window

- **Blanking window duration**:
  - The duration of the blanking window is determined by the value in the field `PU_T1_CNFG.DATA`
  - The length of the window is calculated as:

```
Blanking Window = PU_T1_CNFG.DATA + 1 clock cycles
```

- **Output behavior**:
  - During the blanking window, the output signal is insensitive to changes in the input signal
  - The output begins responding to the input signal only after the blanking window has expired

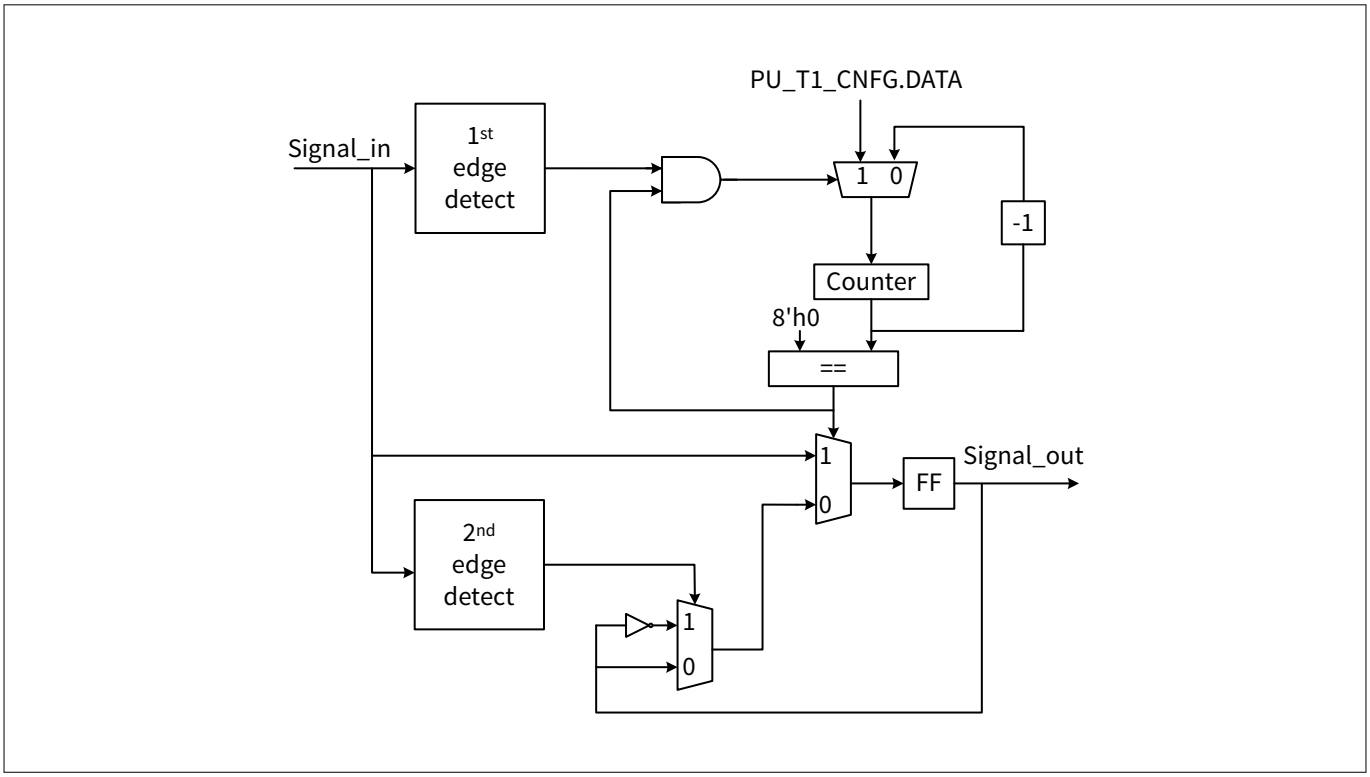


Figure 217 PU_T1 blanking filter

- The counter is initialized with the value specified in the field `PU_T1_CNFG.DATA` and counts down to `0x0`
- Once the counter reaches `0x0`, it remains at `0x0` until a new rising edge is detected on the input signal
- Upon detecting the first rising edge, the signal `signal_in` is transferred to the flip-flop `signal_out`
- When the second rising edge is detected, the flip-flop `signal_out` changes its state and remains in the updated state until the blanking window expires

The waveform as shown in Figure 218 shows the blanking filter behavior with `PU_T1_CNFG.DATA` set to `5` and `PU_T1_CNFG.PUT1_IN_SEL` set to `0`

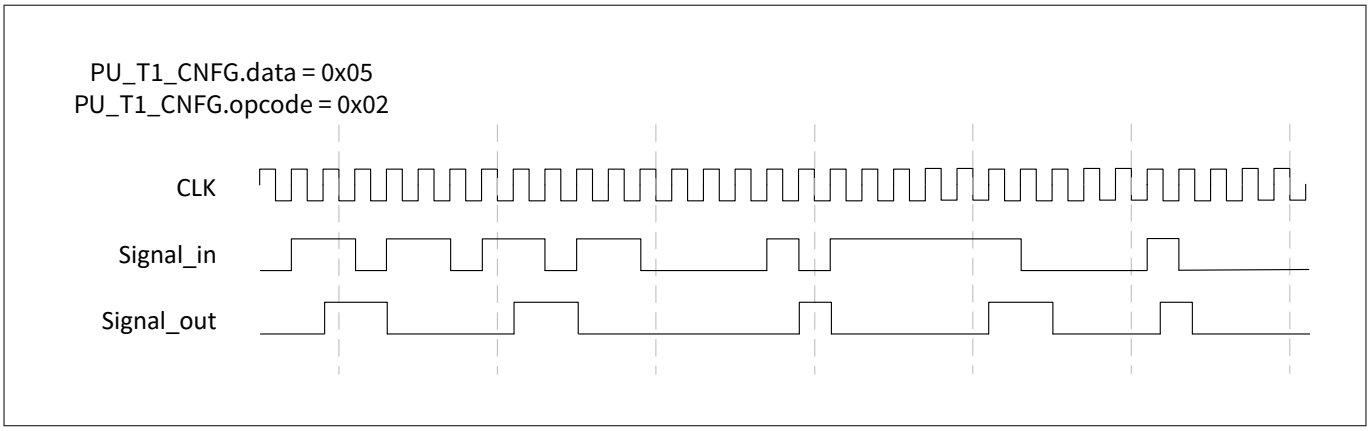


Figure 218 PU_T1 blanking filter waveform

Configuration of blanking filter:

- **Blanking filter enable condition**:
  - The blanking filter is enabled when the register field `PU_T1_CNFG.OPCODE` is set to `0x2`
  - In this mode, the system filters out glitches that may occur immediately after detecting a pulse on the input signal
- **Duration of the blanking window**:
  - The width (duration) of the blanking window is determined by the value in `PU_T1_CNFG.DATA`, incremented by 1 (`PU_T1_CNFG.DATA + 1`)
  - The blanking window duration is expressed in clock cycles:

```
Blanking Window = PU_T1_CNFG.DATA + 1 clock cycles
```

  - During the blanking window, any changes in the input signal (`signal_in`) are ignored, ensuring the output signal (`signal_out`) remains stable
- **Trigger for blanking window**:
  - When the blanking filter is triggered, the following occurs:
    1. A counter is initialized with the value from `PU_T1_CNFG.DATA` and begins counting down to `0x0`
    2. While the counter is active (non-zero), the system filters glitches or noise in the input signal
    3. Once the counter reaches `0x0`, the system allows changes in the input signal to affect the output signal
  - The system behavior is divided into two edge-triggered events:
    - **First edge detected**: The input signal (`signal_in`) is captured and loaded into the output flip-flop (`signal_out`)
    - **Second edge detected**: The flip-flop toggles its state and remains in the updated state until the blanking window expires
- **Signal processing behavior**:
  - After the blanking window expires, the system resumes sensitivity to changes in the input signal (`signal_in`)
  - Any subsequent changes in `signal_in` propagate to the output signal (`signal_out`)

29.5.3.4 Debouncing filter

The debouncing filter feature is enabled when the field PU_T1_CNFG.OPCODE is set to 0x3. This functionality eliminates pulses (positive or negative) that are shorter than or equal to the number of clock cycles specified in the field PU_T1_CNFG.DATA.

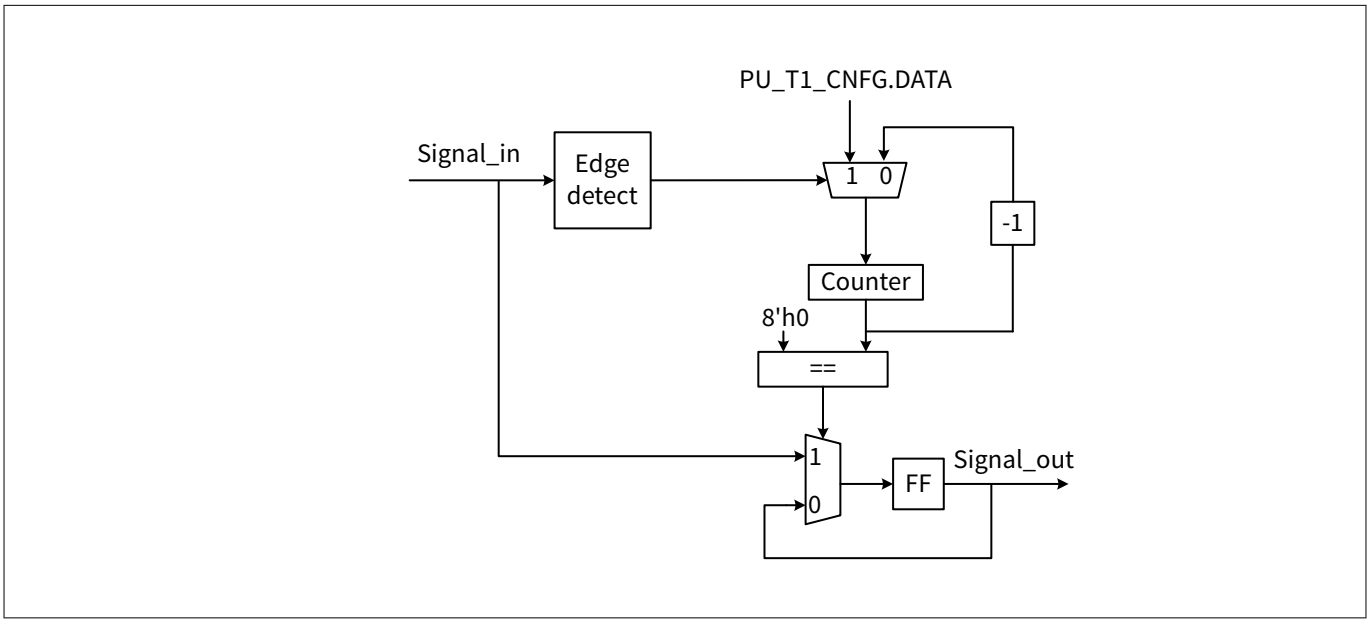


Figure 219 PU_T1 debouncing filter

The counter is initialized with the value specified in the field PU_T1_CNFG.DATA and then counts down to 0x0. Once the counter reaches 0x0, it remains at 0x0 until a new edge is detected.

- **Case 1: PU_T1_CNFG.DATA = 0x02**
  - **Debounce window duration**: 2 clock cycles
  - **Waveform behavior**:
    - Short pulses (positive or negative) lasting 2 clock cycles or fewer are filtered out and treated as noise or invalid transitions
    - Longer pulses (those exceeding 2 clock cycles) are allowed to propagate to the output signal (signal_out)

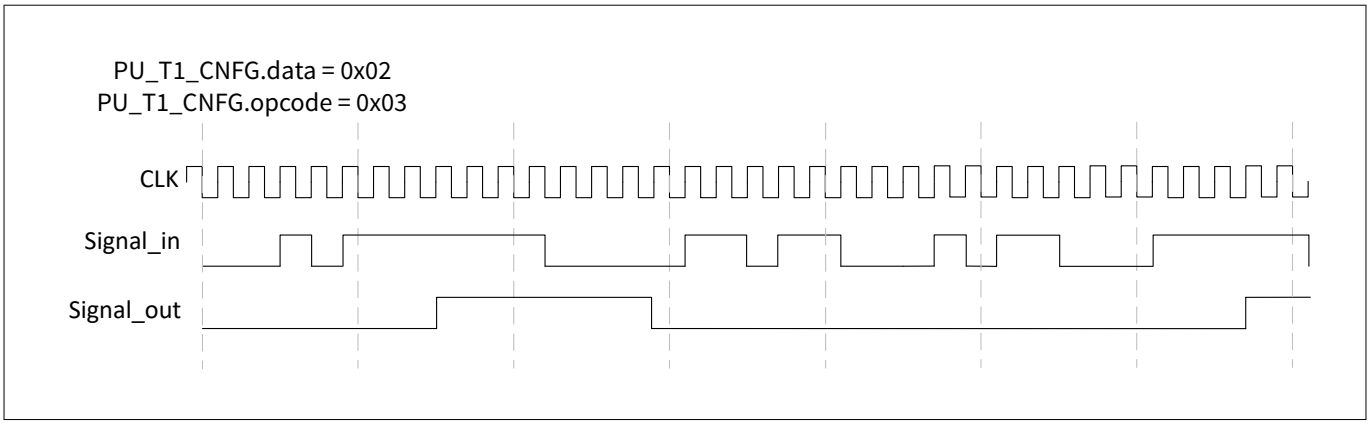


Figure 220 PU_T1 debouncing filter waveform 1

- **Case 2: PU_T1_CNFG.DATA = 0x01**
  - **Debounce window duration**: 1 clock cycle
  - **Waveform behavior**:
    - Any pulse (positive or negative) lasting 1 clock cycle or fewer is filtered out and treated as invalid
    - Pulses lasting more than 1 clock cycle are propagated to the output signal (`signal_out`)

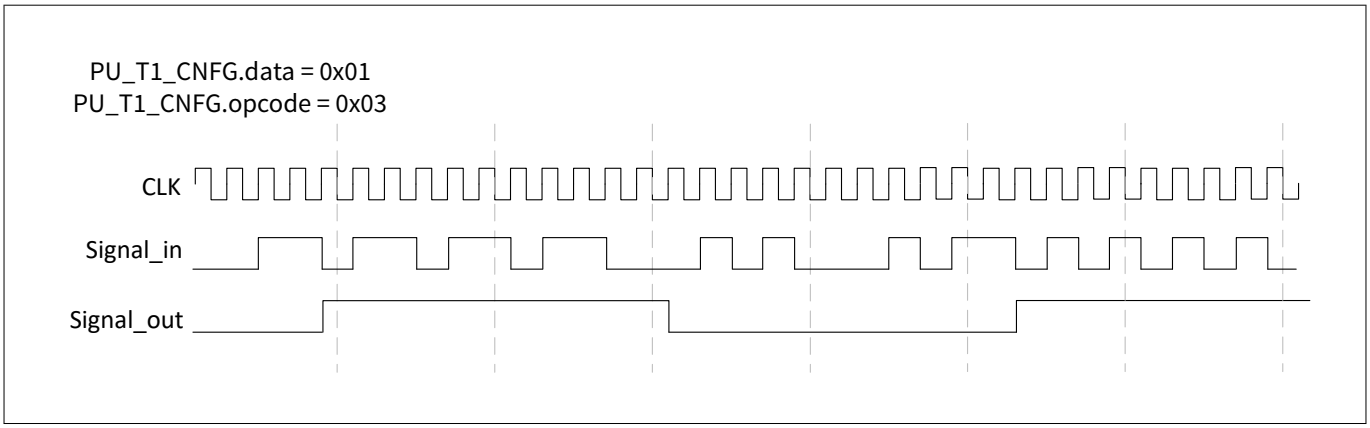


Figure 221 PU_T1 debouncing filter waveform 2

Configuration of debouncing filter

- **Debouncing filter enable condition**:
  - The PU_T1 block operates in Debouncing mode when the field `PU_T1_CNFG.OPCODE` is set to `0x3`
  - This configuration activates the debouncing filter logic
- **Duration of the debouncing window**:
  - The debouncing filter uses a counter initialized with the value in the field `PU_T1_CNFG.DATA`
  - The duration of the debouncing window is calculated as:

```
Debouncing Window = PU_T1_CNFG.DATA clock cycles
```

- **Trigger for debouncing window**:
  - When an edge (positive or negative) is detected on the input signal (`signal_in`), the counter begins counting down to `0x0`
  - During the countdown, any additional edges (positive or negative) are ignored, effectively filtering out short pulses

- **Post-countdown behavior**:
  - Once the counter reaches `0x0`, the system becomes sensitive to new edges on the input signal
  - Only edges that persist beyond the countdown period are allowed to propagate to the output signal (`signal_out`)
- **Filtering of short pulses**:
  - Pulses (both positive and negative) that are shorter than or equal to the configured countdown duration are filtered out and have no effect on `signal_out`
  - This ensures that only stable signals pass through to the output, effectively eliminating glitches and noise
- **Behavior after a valid pulse**: Once a valid edge passes through the debouncing filter (that is, it persists longer than the configured countdown duration), the system resets the counter and waits for the next edge

29.5.3.5 Synchronous bypass

When the field `PU_T1_CNFG.DATA` is set to `0x00`, the input signal `signal_in` is propagated to the output after one clock cycle, regardless of the value set in the field `PU_T1_CNFG.OPCODE`.

The waveform as shown in Figure 222 demonstrates the behavior when `PU_T1_CNFG.PUT1_IN_SEL` is configured to `1`.

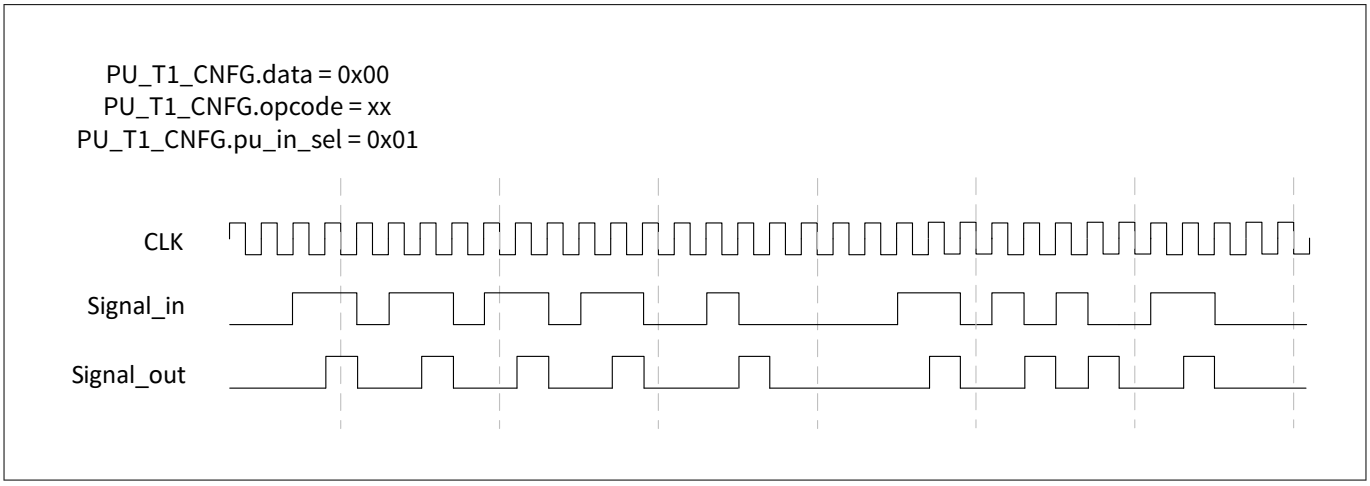


Figure 222 Synchronous bypass waveform 1

The waveform as shown in Figure 223 demonstrates the behavior when the field `PU_T1_CNFG.PUT1_IN_SEL` is configured to `0`.

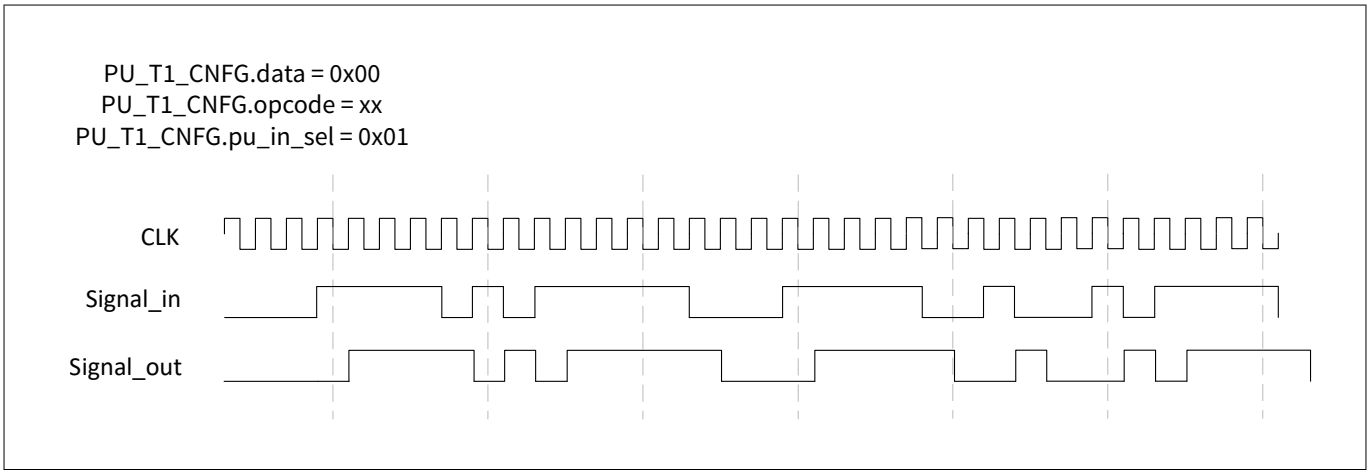


Figure 223 Synchronous bypass waveform 2

29.6 Processing Unit_Type 2 (PU_T2)

Features include:

- Asynchronous bypass
- Synchronous bypass with pipelined flip-flop insertion
- Polarity inversion
- Input edge detection (LH)

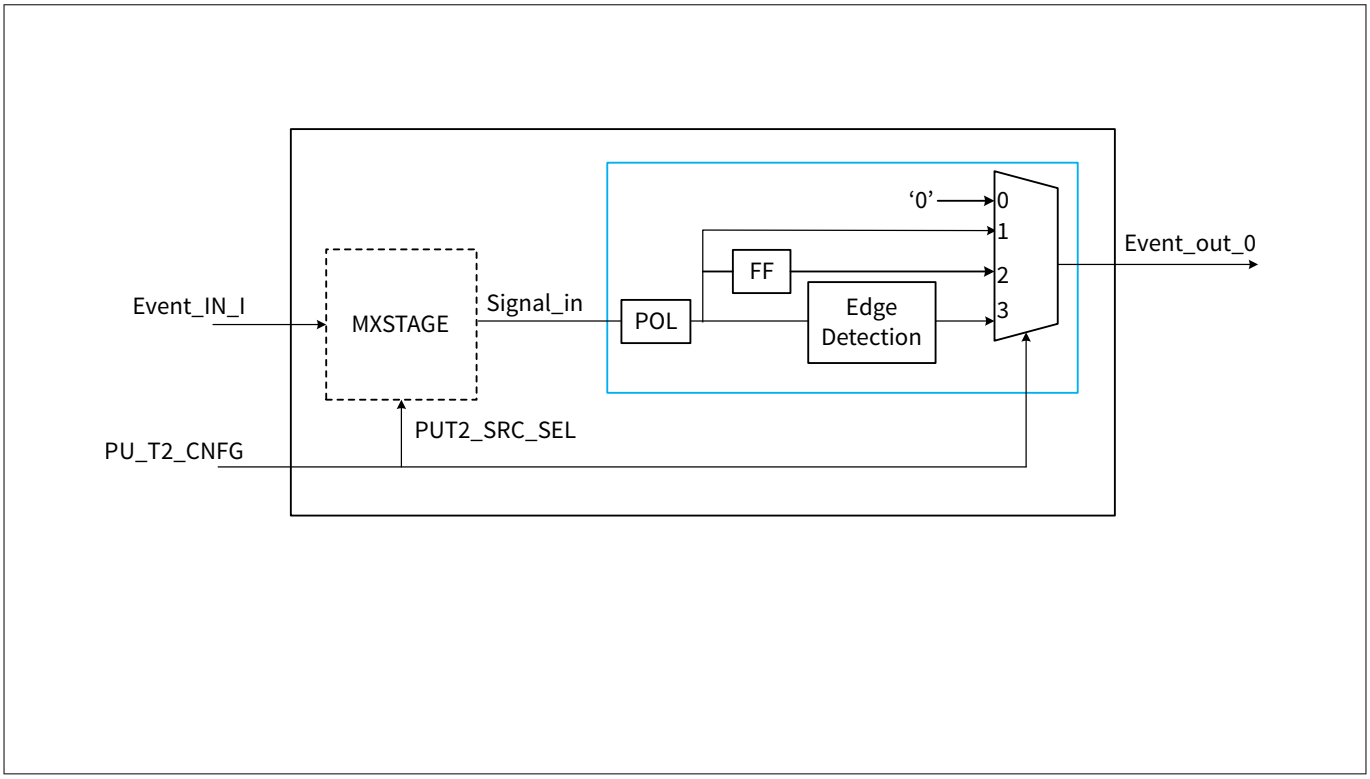


Figure 224 Processing Unit_Type 2 (PU_T2)

The PU_T2 module shares similarities with the PU_T1 module but has fewer capabilities and reduced flexibility. Specifically, PU_T2 does not support qualification windows for the input signal and does not include the filtering options available in PU_T1.

In synchronous mode, either the input signal or the detected edge is propagated to the output, depending on the value of the field `PU_T2_CNFG.PUT2_IN_SEL`. Additionally, PU_T2 includes a test feature to support debugging and validation.

29.6.1 PU_T2 block

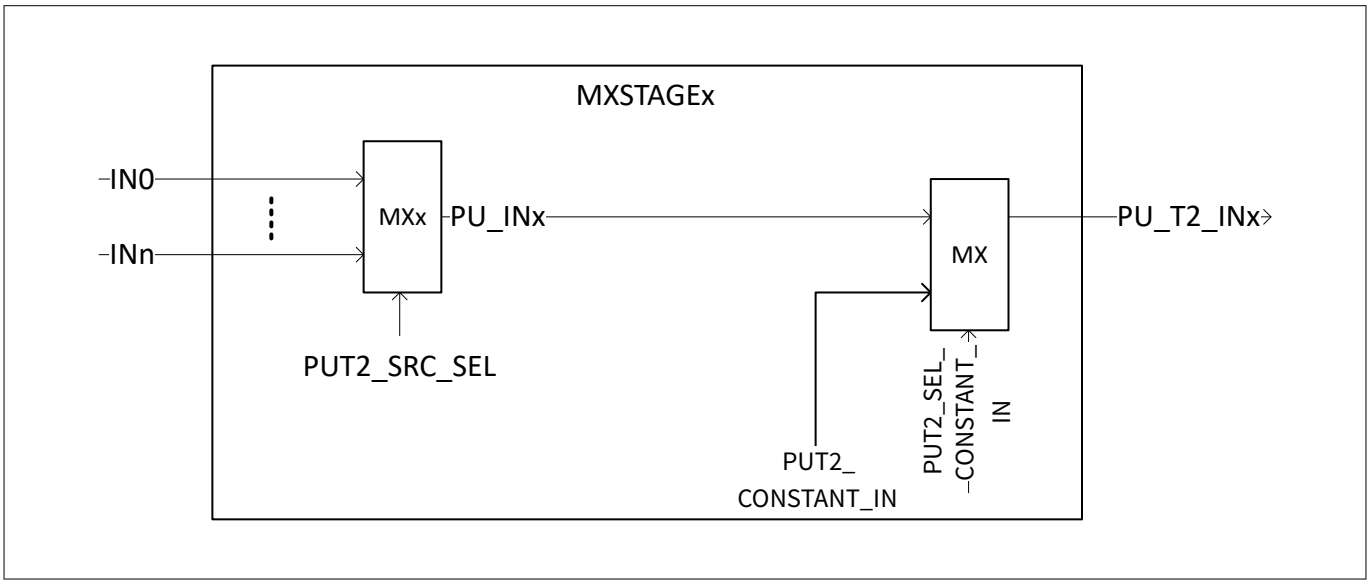


Figure 225 MXSTAGEx

The MXSTAGE allows users to select the input source for the processing units (PUs), configure the function to be executed in the PU, and force a known logic output.

Configuration details of MXSTAGE

1. **Selecting the input event**:
   - Use the PU_T2_CNFG.PUT2_SRC_SEL register to choose the input signal or event from the peripheral
   - This determines which signal the logic will process
2. **Using the MUX with forced constant input**:
   - A MUX is available to override the input signal to the next PU stage with a constant value
   - The forced constant value is specified in the field PU_T2_CNFG.CONSTANT_IN
   - This constant input is applied when the field PU_T2_CNFG.SEL_CONSTANT_IN is set
   - This feature is useful for testing, debugging, or forcing specific conditions during operation

Polarity inversion block

The polarity inversion block modifies the signal output from the MXSTAGE logic. Its primary purpose is to invert the polarity of the signal (that is, logical high becomes low, and logical low becomes high) based on a configuration setting:

- If polarity inversion is enabled, the signal is inverted before being forwarded to the next stage
- If polarity inversion is disabled, the signal remains unchanged and proceeds to the next stage

Bypass

The PU_T2 block includes an edge detection block. However, in certain scenarios, users may choose to bypass the edge detection logic to handle the signal directly.

- When the field PU_T2_CNFG.PUT2_EN is configured for synchronous or asynchronous bypass:
  - The input signal skips the edge detection logic
  - The signal passes directly from the polarity inversion block to its output (event_out)
  - The event_out signal is then used as an input to the combiner block

Edge detection

The behavior of the edge detection block depends on the value of the field PU_T2_CNFG.PUT2_IN_SEL:

- If PU_T2_CNFG.PUT2_IN_SEL is set to 1: The pulse detection result is forwarded to the final multiplexer
- If PU_T2_CNFG.PUT2_IN_SEL is set to 0:
  - The operation becomes level-based
  - The current value of the input line at that moment is passed to the final multiplexer

29.7 Combiner

The COMBO section generates a linear combination of two outputs from Processing Unit_T1 modules, selected via a configuration bit. Additionally, it combines one of 16 outputs from Processing Unit_T1 and one of 16 outputs from Processing Unit_T2. This combination is qualified using one output from four Processing Unit_T1 modules or four Processing Unit_T2 modules.

Each Combiner has a dedicated connection to a specific peripheral input. See EPU trigger outputs for the connection list.

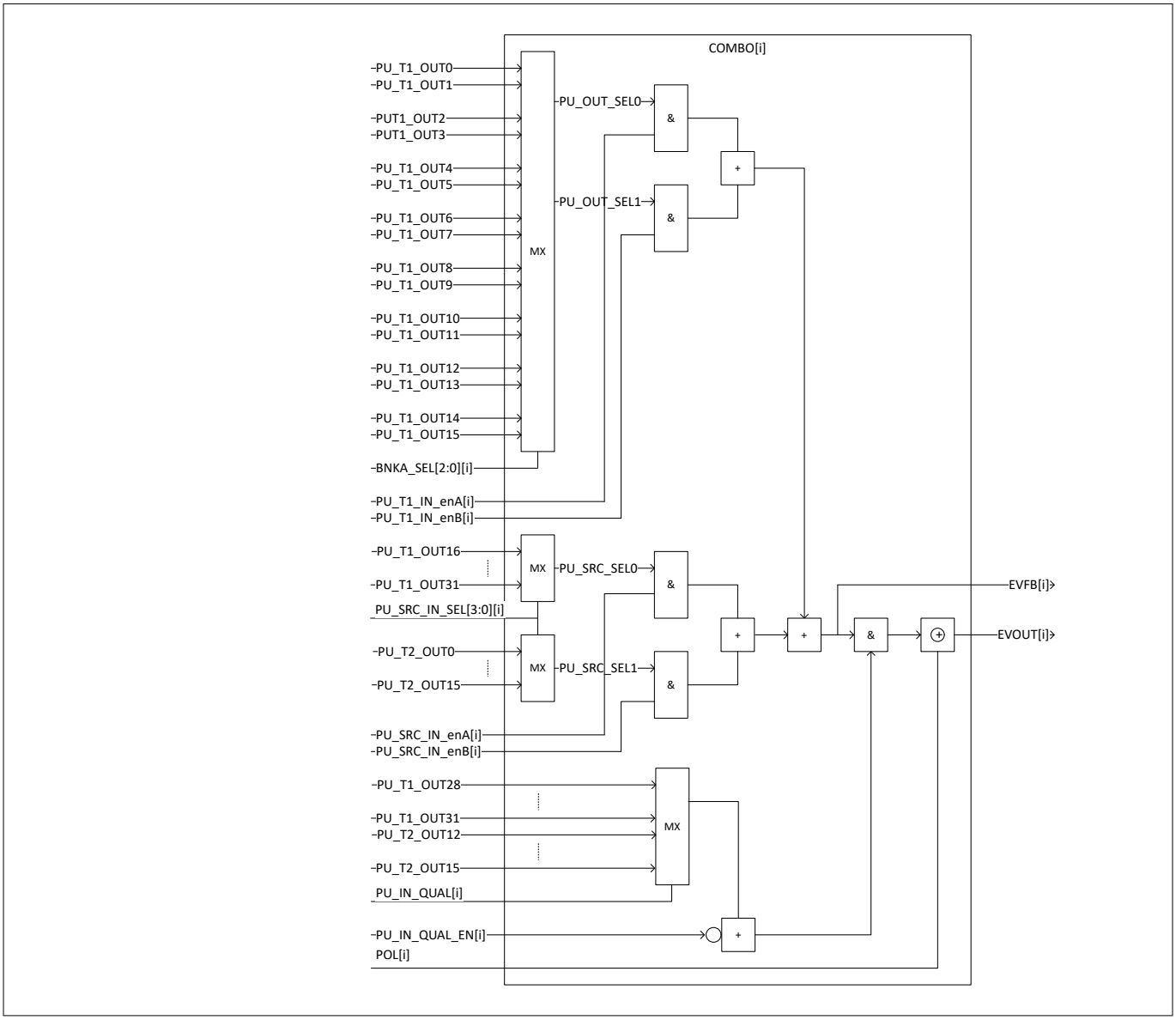


Figure 226 Combiner block diagram

$$evout(n) = (((PU_T1_OUT)_{2i}(n) * a_0 + PU_T1_OUT_{(2i+1)}(n) * a_1) + (PU_T1_OUT_{(16+j)}(n) * a_2 + PU_T2_OUT_j(n) * a_3)) * Q(n)) \hat{} POL(n)$$

$$Q(n) = \left(\left(PU_T1_OUT_{(28+k)}(n) \ * \ a_4 \ \right), \ 0 \le k \le 3\right.$$
$$Q(n) = \left(\left(PU_T2_OUT_{(12+k-4)}(n) \ * \ a_4 \ \right), \ 4 \le k \le 7\right.$$
$$evfb(n) = \left(\left(PU_T1_OUT_{2i}(n) \ * \ a_0 + \ PU_T1_OUT_{(2i+1)} \ (n) \ * \ a_1\right) + \left(PU_T1_OUT_{(16+j)}(n) \ * \ a_2 \ + \ PU_T2_OUT_j(n) \ * \ a_3\right)\right)$$

Where,

$$i \ = \ 0..7 \ = \ COMBO_CNFG[n].BNKA_SEL$$
$$j \ = 0..15 \ = \ COMBO_CNFG[n].PU_SRC_IN_SEL$$
$$a_0 = \ COMBO_CNFG[n].PU_T1_IN_enA$$
$$a_1 = \ COMBO_CNFG[n].PU_T1_IN_enB$$
$$a_2 = \ COMBO_CNFG[n].PU_SRC_IN_enA$$
$$a_3 = \ COMBO_CNFG[n].PU_SRC_IN_enA$$
$$a_4 = \ COMBO_CNFG[n].PU_IN_QUAL_EN$$
$$n \ = \ 0..NUM_OF_EVOUT$$

- **Processing units**:
  - PU_T1 provides 32 outputs, divided into two banks
  - PU_T2 provides 16 outputs
  - These units produce signals that serve as inputs for computing the final outputs
- **Configuration parameters**:
  - The behavior of the COMBO section is controlled using the configuration bit field `COMBO_CNFG[n]`, which includes the following parameters:
    - **BNKA_SEL (i)**: Selects one of 8 pairs of outputs from PU_T1
    - **PU_SRC_IN_SEL (j)**: Selects one of 16 outputs from PU_T1 or PU_T2
    - **Coefficients (a0, a1, a2, a3, a4)**: Control the contribution of different signals during calculation
- **Output equations**:
  - `evout(n)`: The main output of the COMBO section is a combination of:
    - Two selected outputs from PU_T1, weighted by coefficients `a0` and `a1`
    - One output from PU_T1 and one from PU_T2, weighted by coefficients `a2` and `a3`
    - A qualifier signal `Q(n)`, which determines whether the output is active
    - A polarity inversion function `POL(n)` applied to the combined result, if configured
  - Qualifier signal `Q(n)`:
    - Derived from one of four outputs, depending on the value of `k` (0 through 7):
      - **For 0 ≤ k ≤ 3**: Signal comes from PU_T1_{28+k} (indices 28 to 31)
      - **For 4 ≤ k ≤ 7**: Signal comes from PU_T2_{12+k-4} (indices 16 to 19)
  - Feedback signal `evfb(n)`:
    - Bypasses both the qualifier `Q(n)` and the polarity inversion function `POL(n)`
    - Sent back to the input based on the configured selection
- **Linear combination**:
  - The COMBO section essentially performs a weighted summation of multiple outputs from PU_T1 and PU_T2
  - Both the weights (coefficients `a0` to `a4`) and the signals themselves are dynamically configurable
- **Feedback to other blocks**:
  - The results of the linear combination are passed to the `COMBO_FB` block, where they are:
    - Synchronized for consistent timing
    - Sent back to earlier pipeline stages (for example, PRESTAGE and MXSTAGE) for further processing

29.8 Combiner feedback

The COMBO section provides access to the internal linear combination. These signals are sent to the COMBO_FB block, where one signal is selected, synchronized, and sent back to the input signals of the PRESTAGE and MXSTAGE blocks.

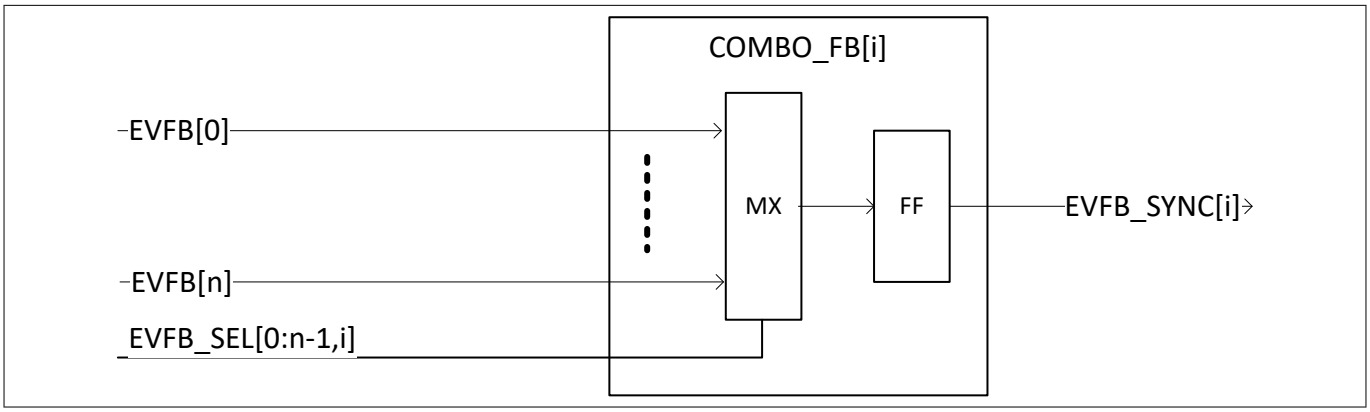


Figure 227 Combiner feedback

29.9 EPU interrupt generator

The EPU interrupt generator module (EPU_IRQ) uses a group of multiplexers to select up to 8 COMBO output signals (EVOUT) for generating interrupt events to the PPCA CPUs. A dedicated control register controls the multiplexers. The rising edge of the selected multiplexer output signal is converted into interrupt events.

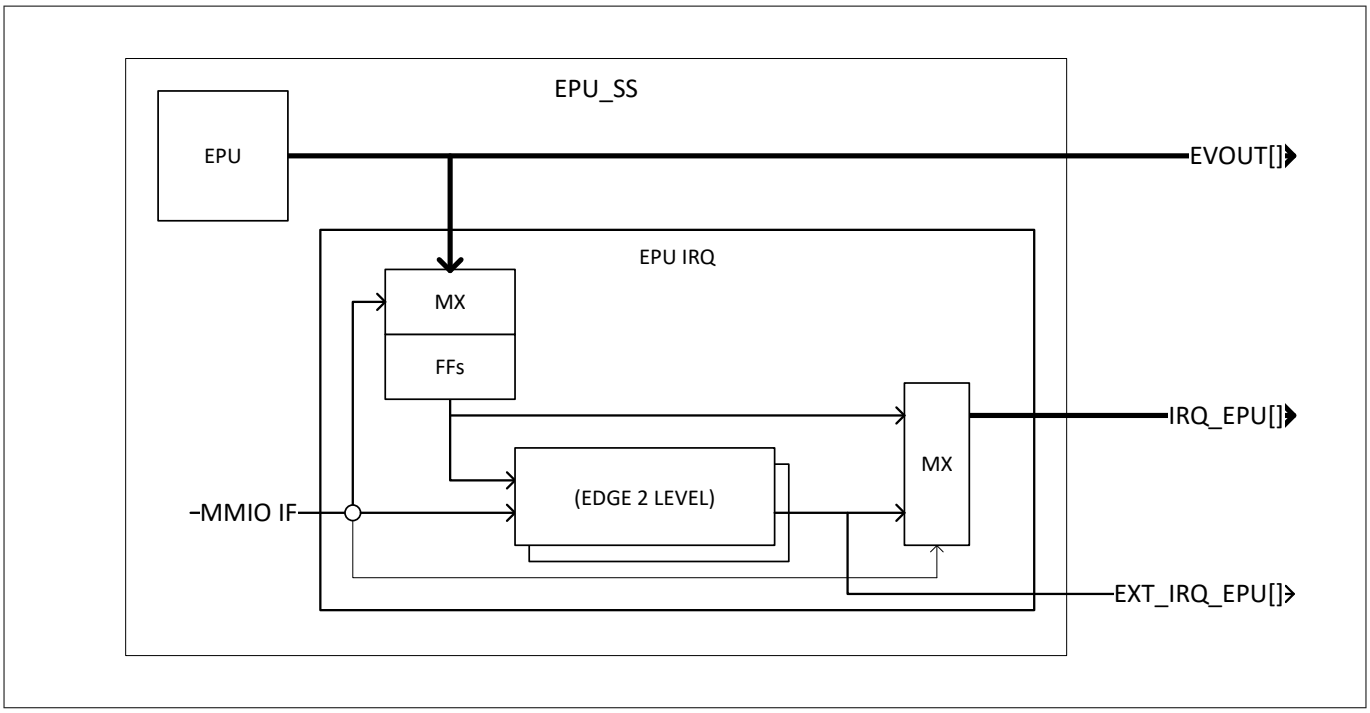


Figure 228 EPU interrupt generator

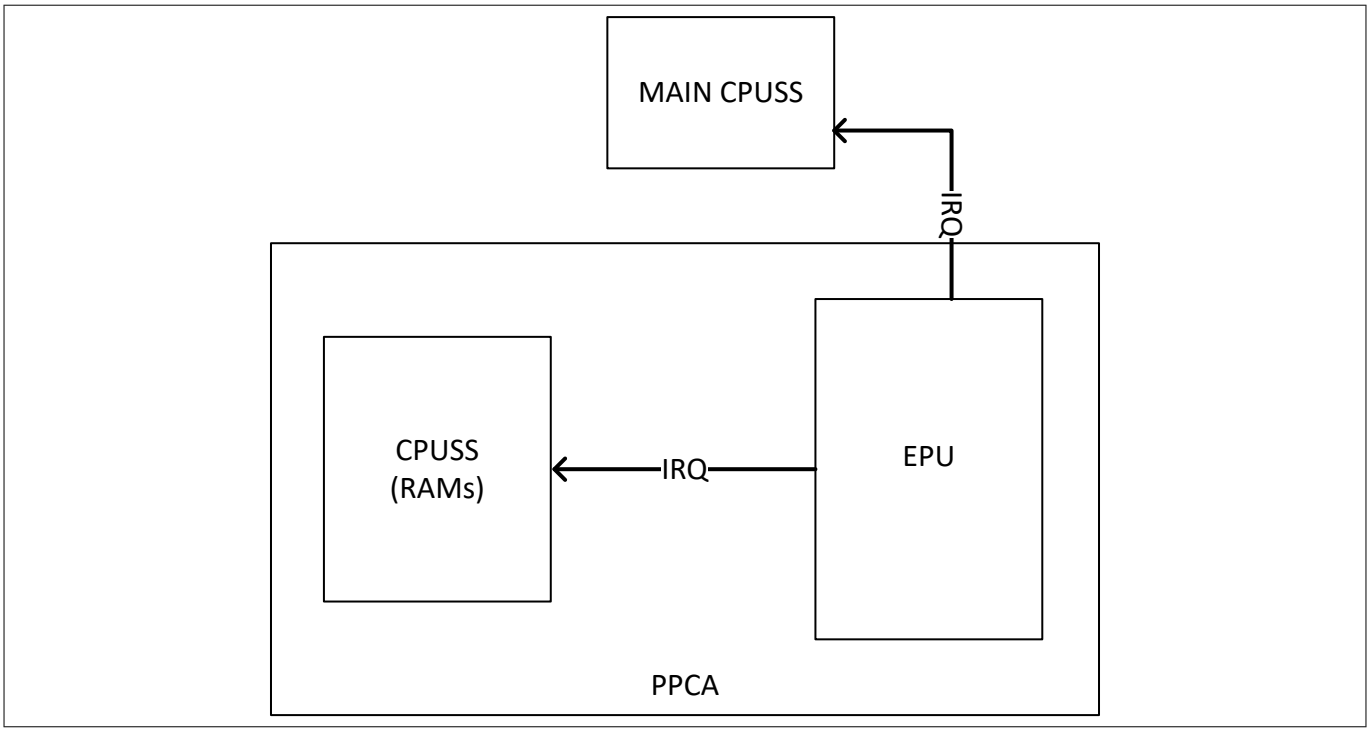


Figure 229 EPU IRQ connection

One flip-flop (FF) is assigned to each multiplexer output, with all interrupt events visible to the Main CPU and PPCA CPU cores, and an option to bypass the interrupt generation logic for internal PPCA CPUs.

Table 245 PPCA interrupts

Interrupts	From peripherals	PPCA IRQ
IRQ_PPCA_CPU0[3:0]	EPU	PPCA_IRQ [8:5]
IRQ_PPCA_CPU1[3:0]	EPU	PPCA_IRQ [12:9]

29.10 EPU trigger inputs

Table 246 Event input table (Total = 256; 8 MSB internally used)

From peripherals	Signals	EPU input	EPU input bitwise
TCPWM_SS0	TR_OUT_0 [11:0]	EVIN 0	TCPWM0_TR_OUT_0_0
		EVIN 1	TCPWM0_TR_OUT_0_1
		EVIN 2	TCPWM0_TR_OUT_0_2
		EVIN 3	TCPWM0_TR_OUT_0_3
		EVIN 4	TCPWM0_TR_OUT_0_4
		EVIN 5	TCPWM0_TR_OUT_0_5
		EVIN 6	TCPWM0_TR_OUT_0_6
		EVIN 7	TCPWM0_TR_OUT_0_7
		EVIN 8	TCPWM0_TR_OUT_0_8
		EVIN 9	TCPWM0_TR_OUT_0_9

(table continues...)

Table 246 (continued) Event input table (Total = 256; 8 MSB internally used)

From peripherals	Signals	EPU input	EPU input bitwise
		EVIN 10	TCPWM0_TR_OUT_0_10
		EVIN 11	TCPWM0_TR_OUT_0_11
TCPWM_SS0	TR_OUT_1 [11:0]	EVIN 12	TCPWM0_TR_OUT_1_0
		EVIN 13	TCPWM0_TR_OUT_1_1
		EVIN 14	TCPWM0_TR_OUT_1_2
		EVIN 15	TCPWM0_TR_OUT_1_3
		EVIN 16	TCPWM0_TR_OUT_1_4
		EVIN 17	TCPWM0_TR_OUT_1_5
		EVIN 18	TCPWM0_TR_OUT_1_6
		EVIN 19	TCPWM0_TR_OUT_1_7
		EVIN 20	TCPWM0_TR_OUT_1_8
		EVIN 21	TCPWM0_TR_OUT_1_9
		EVIN 22	TCPWM0_TR_OUT_1_10
		EVIN 23	TCPWM0_TR_OUT_1_11
TCPWM_MOTIF_SS0	MO_TR_OUT [7:0]	EVIN 24	TCPWM0_MO_TR_OUT_0
		EVIN 25	TCPWM0_MO_TR_OUT_1
		EVIN 26	TCPWM0_MO_TR_OUT_2
		EVIN 27	TCPWM0_MO_TR_OUT_3
		EVIN 28	TCPWM0_MO_TR_OUT_4
		EVIN 29	TCPWM0_MO_TR_OUT_5
		EVIN 30	TCPWM0_MO_TR_OUT_6
		EVIN 31	TCPWM0_MO_TR_OUT_7
TCPWM1_CNT	TR_OUT_0 [7:0]	EVIN 32	TCPWM1_TR_OUT_0_0
		EVIN 33	TCPWM1_TR_OUT_0_1
		EVIN 34	TCPWM1_TR_OUT_0_2
		EVIN 35	TCPWM1_TR_OUT_0_3
		EVIN 36	TCPWM1_TR_OUT_0_4
		EVIN 37	TCPWM1_TR_OUT_0_5
		EVIN 38	TCPWM1_TR_OUT_0_6
		EVIN 39	TCPWM1_TR_OUT_0_7
TCPWM1_CNT	TR_OUT_1 [7:0]	EVIN 40	TCPWM1_TR_OUT_1_0
		EVIN 41	TCPWM1_TR_OUT_1_1
		EVIN 42	TCPWM1_TR_OUT_1_2

(table continues...)

Table 246 (continued) Event input table (Total = 256; 8 MSB internally used)

From peripherals	Signals	EPU input	EPU input bitwise
		EVIN 43	TCPWM1_TR_OUT_1_3
		EVIN 44	TCPWM1_TR_OUT_1_4
		EVIN 45	TCPWM1_TR_OUT_1_5
		EVIN 46	TCPWM1_TR_OUT_1_6
		EVIN 47	TCPWM1_TR_OUT_1_7
TCPWM1_MOTIF	MO_TR_OUT [7:0]	EVIN 48	TCPWM1_MO_TR_OUT_0
		EVIN 49	TCPWM1_MO_TR_OUT_1
		EVIN 50	TCPWM1_MO_TR_OUT_2
		EVIN 51	TCPWM1_MO_TR_OUT_3
		EVIN 52	TCPWM1_MO_TR_OUT_4
		EVIN 53	TCPWM1_MO_TR_OUT_5
		EVIN 54	TCPWM1_MO_TR_OUT_6
		EVIN 55	TCPWM1_MO_TR_OUT_7
TCPWM2_CNT	TR_OUT_0 [3:0]	EVIN 56	TCPWM2_TR_OUT_0_0
		EVIN 57	TCPWM2_TR_OUT_0_1
		EVIN 58	TCPWM2_TR_OUT_0_2
		EVIN 59	TCPWM2_TR_OUT_0_3
TCPWM2_CNT	TR_OUT_1 [3:0]	EVIN 60	TCPWM2_TR_OUT_1_0
		EVIN 61	TCPWM2_TR_OUT_1_1
		EVIN 62	TCPWM2_TR_OUT_1_2
		EVIN 63	TCPWM2_TR_OUT_1_3
TCPWM3_CNT	TR_OUT_0 [1:0]	EVIN 64	TCPWM3_TR_OUT_0_0
		EVIN 65	TCPWM3_TR_OUT_0_1
TCPWM3_CNT	TR_OUT_1 [1:0]	EVIN 66	TCPWM3_TR_OUT_1_0
		EVIN 67	TCPWM3_TR_OUT_1_1
HWFILT3P3Z_SS0	EOP [3:0]	EVIN 68	HWFILT3P3Z0_EOP_0
		EVIN 69	HWFILT3P3Z0_EOP_1
		EVIN 70	HWFILT3P3Z0_EOP_2
		EVIN 71	HWFILT3P3Z0_EOP_3
HWFILT3P3Z_SS1	EOP [1:0]	EVIN 72	HWFILT3P3Z1_EOP_0
		EVIN 73	HWFILT3P3Z1_EOP_1
CLB	LINE_OUT [7:0]	EVIN 74	CLB_LINE_OUT_0
		EVIN 75	CLB_LINE_OUT_1

(table continues...)

Table 246 (continued) Event input table (Total = 256; 8 MSB internally used)

From peripherals	Signals	EPU input	EPU input bitwise
		EVIN 76	CLB_LINE_OUT_2
		EVIN 77	CLB_LINE_OUT_3
		EVIN 78	CLB_LINE_OUT_4
		EVIN 79	CLB_LINE_OUT_5
		EVIN 80	CLB_LINE_OUT_6
		EVIN 81	CLB_LINE_OUT_7
CLB	LINE_COMPL_OUT [7:0]	EVIN 82	CLB_LINE_COMPL_OUT_0
		EVIN 83	CLB_LINE_COMPL_OUT_1
		EVIN 84	CLB_LINE_COMPL_OUT_2
		EVIN 85	CLB_LINE_COMPL_OUT_3
		EVIN 86	CLB_LINE_COMPL_OUT_4
		EVIN 87	CLB_LINE_COMPL_OUT_5
		EVIN 88	CLB_LINE_COMPL_OUT_6
		EVIN 89	CLB_LINE_COMPL_OUT_7
ADC	ADC_EOC [23:0]	EVIN 90	ADC_EOC_0
		EVIN 91	ADC_EOC_1
		EVIN 92	ADC_EOC_2
		EVIN 93	ADC_EOC_3
		EVIN 94	ADC_EOC_4
		EVIN 95	ADC_EOC_5
		EVIN 96	ADC_EOC_6
		EVIN 97	ADC_EOC_7
		EVIN 98	ADC_EOC_8
		EVIN 99	ADC_EOC_9
		EVIN 100	ADC_EOC_10
		EVIN 101	ADC_EOC_11
		EVIN 102	ADC_EOC_12
		EVIN 103	ADC_EOC_13
		EVIN 104	ADC_EOC_14
		EVIN 105	ADC_EOC_15
		EVIN 106	ADC_EOC_16
		EVIN 107	ADC_EOC_17
		EVIN 108	ADC_EOC_18

(table continues...)

Table 246 (continued) Event input table (Total = 256; 8 MSB internally used)

From peripherals	Signals	EPU input	EPU input bitwise
		EVIN 109	ADC_EOC_19
		EVIN 110	ADC_EOC_20
		EVIN 111	ADC_EOC_21
		EVIN 112	ADC_EOC_22
		EVIN 113	ADC_EOC_23
DCMP	DCMP_OUT [11:0]	EVIN 114	DCMP_OUT_0
		EVIN 115	DCMP_OUT_1
		EVIN 116	DCMP_OUT_2
		EVIN 117	DCMP_OUT_3
		EVIN 118	DCMP_OUT_4
		EVIN 119	DCMP_OUT_5
		EVIN 120	DCMP_OUT_6
		EVIN 121	DCMP_OUT_7
		EVIN 122	DCMP_OUT_8
		EVIN 123	DCMP_OUT_9
		EVIN 124	DCMP_OUT_10
		EVIN 125	DCMP_OUT_11
DCSG	DCSG_CMP_OUT [8:0]	EVIN 126	DCSG_CMP_OUT_0
		EVIN 127	DCSG_CMP_OUT_1
		EVIN 128	DCSG_CMP_OUT_2
		EVIN 129	DCSG_CMP_OUT_3
		EVIN 130	DCSG_CMP_OUT_4
		EVIN 131	DCSG_CMP_OUT_5
		EVIN 132	DCSG_CMP_OUT_6
		EVIN 133	DCSG_CMP_OUT_7
		EVIN 134	DCSG_CMP_OUT_8
CPU0	TXEV0	EVIN 135	CPU0_TXEV0
CPU1	TXEV1	EVIN 136	CPU1_TXEV1
CPUSS	FAULT	EVIN 137	FAULT
CLB	CLB_OUT [5:0]	EVIN 138	CLB_OUT_0
		EVIN 139	CLB_OUT_1
		EVIN 140	CLB_OUT_2
		EVIN 141	CLB_OUT_3

(table continues...)

Table 246 (continued) Event input table (Total = 256; 8 MSB internally used)

From peripherals	Signals	EPU input	EPU input bitwise
		EVIN 142	CLB_OUT_4
		EVIN 143	CLB_OUT_5
CNFG	NA	EVIN 144	NA
		EVIN 145	NA
CNFG	PPCAIN_SEL [15:0]	EVIN 146	PPCAIN_SEL_0
		EVIN 147	PPCAIN_SEL_1
		EVIN 148	PPCAIN_SEL_2
		EVIN 149	PPCAIN_SEL_3
		EVIN 150	PPCAIN_SEL_4
		EVIN 151	PPCAIN_SEL_5
		EVIN 152	PPCAIN_SEL_6
		EVIN 153	PPCAIN_SEL_7
		EVIN 154	PPCAIN_SEL_8
		EVIN 155	PPCAIN_SEL_9
		EVIN 156	PPCAIN_SEL_10
		EVIN 157	PPCAIN_SEL_11
		EVIN 158	PPCAIN_SEL_12
		EVIN 159	PPCAIN_SEL_13
		EVIN 160	PPCAIN_SEL_14
		EVIN 161	PPCAIN_SEL_15
CNFG	TR_PPCAIN_SEL [15:0]	EVIN 162	TR_PPCAIN_SEL_0
		EVIN 163	TR_PPCAIN_SEL_1
		EVIN 164	TR_PPCAIN_SEL_2
		EVIN 165	TR_PPCAIN_SEL_3
		EVIN 166	TR_PPCAIN_SEL_4
		EVIN 167	TR_PPCAIN_SEL_5
		EVIN 168	TR_PPCAIN_SEL_6
		EVIN 169	TR_PPCAIN_SEL_7
		EVIN 170	TR_PPCAIN_SEL_8
		EVIN 171	TR_PPCAIN_SEL_9
		EVIN 172	TR_PPCAIN_SEL_10
		EVIN 173	TR_PPCAIN_SEL_11
		EVIN 174	TR_PPCAIN_SEL_12

(table continues...)

Table 246 (continued) Event input table (Total = 256; 8 MSB internally used)

From peripherals	Signals	EPU input	EPU input bitwise
		EVIN 175	TR_PPCAIN_SEL_13
		EVIN 176	TR_PPCAIN_SEL_14
		EVIN 177	TR_PPCAIN_SEL_15
SD_CMP	SD_CMP [5:0]	EVIN 178	SD_CMP_0
		EVIN 179	SD_CMP_1
		EVIN 180	SD_CMP_2
		EVIN 181	SD_CMP_3
		EVIN 182	SD_CMP_4
		EVIN 183	SD_CMP_5
NA	NA	EVIN [247:184]	NA
COMBO_FB	EVFB_SYNC [7:0]	EVIN 248	EVFB_SYNC_0
		EVIN 249	EVFB_SYNC_1
		EVIN 250	EVFB_SYNC_2
		EVIN 251	EVFB_SYNC_3
		EVIN 252	EVFB_SYNC_4
		EVIN 253	EVFB_SYNC_5
		EVIN 254	EVFB_SYNC_6
		EVIN 256	EVFB_SYNC_7

29.11 EPU trigger outputs

Table 247 lists resources that can be assigned to either CPU0 or CPU1, as well as resources directly controlled by the external bus master (main CPU).

Table 247 Trigger outputs (total = 123)

To peripherals	Allocatable resources	Signals	EPU output	EPU output bitwise
TCPWM SS0	Y	TR_ONE_CNT_IN [11:0]	EVOUT 0	TCPWM0_TR_ONE_CNT_IN_0
			EVOUT 1	TCPWM0_TR_ONE_CNT_IN_1
			EVOUT 2	TCPWM0_TR_ONE_CNT_IN_2
			EVOUT 3	TCPWM0_TR_ONE_CNT_IN_3
			EVOUT 4	TCPWM0_TR_ONE_CNT_IN_4

(table continues...)

Table 247 (continued) Trigger outputs (total = 123)

To peripherals	Allocatable resources	Signals	EPU output	EPU output bitwise
			EVOUT 5	TCPWM0_TR_ONE_CNT_IN_5
			EVOUT 6	TCPWM0_TR_ONE_CNT_IN_6
			EVOUT 7	TCPWM0_TR_ONE_CNT_IN_7
			EVOUT 8	TCPWM0_TR_ONE_CNT_IN_8
			EVOUT 9	TCPWM0_TR_ONE_CNT_IN_9
			EVOUT 10	TCPWM0_TR_ONE_CNT_IN_10
			EVOUT 11	TCPWM0_TR_ONE_CNT_IN_11
TCPWM SS0	Y	TR_ALL_CNT_IN [5:0]	EVOUT 12	TCPWM0_TR_ALL_CNT_IN_0
			EVOUT 13	TCPWM0_TR_ALL_CNT_IN_1
			EVOUT 14	TCPWM0_TR_ALL_CNT_IN_2
			EVOUT 15	TCPWM0_TR_ALL_CNT_IN_3
			EVOUT 16	TCPWM0_TR_ALL_CNT_IN_4
			EVOUT 17	TCPWM0_TR_ALL_CNT_IN_5
TCPWM-MOTIF SS0	Y	MO_TR_IN [7:0]	EVOUT 18	TCPWM0_MO_TR_IN_0
			EVOUT 19	TCPWM0_MO_TR_IN_1
			EVOUT 20	TCPWM0_MO_TR_IN_2
			EVOUT 21	TCPWM0_MO_TR_IN_3
			EVOUT 22	TCPWM0_MO_TR_IN_4
			EVOUT 23	TCPWM0_MO_TR_IN_5

(table continues...)

Table 247 (continued) Trigger outputs (total = 123)

To peripherals	Allocatable resources	Signals	EPU output	EPU output bitwise
			EVOUT 24	TCPWM0_MO_TR_IN_6
			EVOUT 25	TCPWM0_MO_TR_IN_7
TCPWM SS1	Y	TR_ONE_CNT_IN [7:0],	EVOUT 26	TCPWM1_TR_ONE_CNT_IN_0
			EVOUT 27	TCPWM1_TR_ONE_CNT_IN_1
			EVOUT 28	TCPWM1_TR_ONE_CNT_IN_2
			EVOUT 29	TCPWM1_TR_ONE_CNT_IN_3
			EVOUT 30	TCPWM1_TR_ONE_CNT_IN_4
			EVOUT 31	TCPWM1_TR_ONE_CNT_IN_5
			EVOUT 32	TCPWM1_TR_ONE_CNT_IN_6
			EVOUT 33	TCPWM1_TR_ONE_CNT_IN_7
TCPWM SS1	Y	TR_ALL_CNT_IN [5:0]	EVOUT 34	TCPWM1_TR_ALL_CNT_IN_0
			EVOUT 35	TCPWM1_TR_ALL_CNT_IN_1
			EVOUT 36	TCPWM1_TR_ALL_CNT_IN_2
			EVOUT 37	TCPWM1_TR_ALL_CNT_IN_3
			EVOUT 38	TCPWM1_TR_ALL_CNT_IN_4
			EVOUT 39	TCPWM1_TR_ALL_CNT_IN_5
TCPWM-MOTIF SS1	Y	MO_TR_IN [7:0]	EVOUT 40	TCPWM1_MO_TR_IN_0
			EVOUT 41	TCPWM1_MO_TR_IN_1
			EVOUT 42	TCPWM1_MO_TR_IN_2

(table continues...)

Table 247 (continued) Trigger outputs (total = 123)

To peripherals	Allocatable resources	Signals	EPU output	EPU output bitwise
			EVOUT 43	TCPWM1_MO_TR_IN_3
			EVOUT 44	TCPWM1_MO_TR_IN_4
			EVOUT 45	TCPWM1_MO_TR_IN_5
			EVOUT 46	TCPWM1_MO_TR_IN_6
			EVOUT 47	TCPWM1_MO_TR_IN_7
TCPWM SS2	Y	TR_ONE_CNT_IN [3:0]	EVOUT 48	TCPWM2_TR_ONE_CNT_IN_0
			EVOUT 49	TCPWM2_TR_ONE_CNT_IN_1
			EVOUT 50	TCPWM2_TR_ONE_CNT_IN_2
			EVOUT 51	TCPWM2_TR_ONE_CNT_IN_3
TCPWM SS2	Y	TR_ALL_CNT_IN	EVOUT 52	TCPWM2_TR_ALL_CNT_IN_0
TCPWM SS3	Y	TR_ONE_CNT_IN [1:0]	EVOUT 53	TCPWM3_TR_ONE_CNT_IN_0
			EVOUT 54	TCPWM3_TR_ONE_CNT_IN_1
TCPWM SS3	Y	TR_ALL_CNT_IN	EVOUT 55	TCPWM3_TR_ALL_CNT_IN_0
CLB	Y	TRIG _IN [3:0]	EVOUT 56	CLB_TRIG_IN_0
			EVOUT 57	CLB_TRIG_IN_1
			EVOUT 58	CLB_TRIG_IN_2
			EVOUT 59	CLB_TRIG_IN_3
ADC_GRP0	Y	ADC_SOC [7:0],	EVOUT 60	ADC_SOC_0
			EVOUT 61	ADC_SOC_1
			EVOUT 62	ADC_SOC_2
			EVOUT 63	ADC_SOC_3
			EVOUT 64	ADC_SOC_4
			EVOUT 65	ADC_SOC_5
			EVOUT 66	ADC_SOC_6

(table continues...)

Table 247 (continued) Trigger outputs (total = 123)

To peripherals	Allocatable resources	Signals	EPU output	EPU output bitwise
			EVOUT 67	ADC_SOC_7
ADC_GRP0	Y	SAMPLE_W [1:0],	EVOUT 68	SAMPL_W_0
			EVOUT 69	SAMPL_W_1
ADC_GRP0	Y	MIN_MAX [1:0]	EVOUT 70	MIN_MAX_0
			EVOUT 71	MIN_MAX_1
ADC_GRP1	Y	ADC_SOC [11:8],	EVOUT 72	ADC_SOC_8
			EVOUT 73	ADC_SOC_9
			EVOUT 74	ADC_SOC_10
			EVOUT 75	ADC_SOC_11
ADC_GRP1	Y	SAMPLE_W [2],	EVOUT 76	SAMPL_W_2
ADC_GRP1	Y	MIN_MAX [2]	EVOUT 77	MIN_MAX_2
ADC_GRP2	Y	ADC_SOC [15:12],	EVOUT 78	ADC_SOC_12
			EVOUT 79	ADC_SOC_13
			EVOUT 80	ADC_SOC_14
			EVOUT 81	ADC_SOC_15
ADC_GRP2	Y	SAMPLE_W [3],	EVOUT 82	SAMPL_W_3
ADC_GRP2	Y	MIN_MAX [3]	EVOUT 83	MIN_MAX_3
ADC_GRP3	Y	ADC_SOC [23:16],	EVOUT 84	ADC_SOC_16
			EVOUT 85	ADC_SOC_17
			EVOUT 86	ADC_SOC_18
			EVOUT 87	ADC_SOC_19
			EVOUT 88	ADC_SOC_20
			EVOUT 89	ADC_SOC_21
			EVOUT 90	ADC_SOC_22
			EVOUT 91	ADC_SOC_23
ADC_GRP3	Y	SAMPLE_W [5:4],	EVOUT 92	SAMPL_W_4
			EVOUT 93	SAMPL_W_5
ADC_GRP3	Y	MIN_MAX [5:4]	EVOUT 94	MIN_MAX_4
			EVOUT 95	MIN_MAX_5
DCSG_GRP0	Y	DCSG_CMP_TRIG [1:0],	EVOUT 96	DCSG_CMP_TRIG_0
			EVOUT 97	DCSG_CMP_TRIG_1
DCSG_GRP1	Y	DCSG_CMP_TRIG [2],	EVOUT 98	DCSG_CMP_TRIG_2

(table continues...)

Table 247 (continued) Trigger outputs (total = 123)

To peripherals	Allocatable resources	Signals	EPU output	EPU output bitwise
PPCA_CNFG	N	RDY_FOR_DEEPSLEEP	EVOUT 99	RDY_FOR_DEEPSLEEP
External DMA	N	DREQ [3:0]	EVOUT 100	PPCA_DREQ_0
			EVOUT 101	PPCA_DREQ_1
			EVOUT 102	PPCA_DREQ_2
			EVOUT 103	PPCA_DREQ_3
CPU0	Y	IRQ [3:0], RXEV0	EVOUT 104	CPU0_IRQ_0
			EVOUT 105	CPU0_IRQ_1
			EVOUT 106	CPU0_IRQ_2
			EVOUT 107	CPU0_IRQ_3
			EVOUT 108	CPU0_RXEV_0
CPU1	Y	IRQ [3:0], RXEV1	EVOUT 109	CPU1_IRQ_0
			EVOUT 110	CPU1_IRQ_1
			EVOUT 111	CPU1_IRQ_2
			EVOUT 112	CPU1_IRQ_3
			EVOUT 113	CPU1_RXEV_1
EXTERNAL OUTPUTs/ ATOP DCMP	N	TR_EPU [0]/ DCMP_TRIG [0], DCMP_TRIG [1]	EVOUT 114	TR_EPU_0_DCMP_0_1
EXTERNAL OUTPUTs/ ATOP DCMP	N	TR_EPU [1]/ DCMP_TRIG [2], DCMP_TRIG [3]	EVOUT 115	TR_EPU_1_DCMP_2_3
EXTERNAL OUTPUTs/ ATOP DCMP	N	TR_EPU [2]/ DCMP_TRIG [4], DCMP_TRIG [5]	EVOUT 116	TR_EPU_2_DCMP_4_5
EXTERNAL OUTPUTs/ ATOP DCMP	N	TR_EPU [3]/ DCMP_TRIG [6], DCMP_TRIG [7]	EVOUT 117	TR_EPU_3_DCMP_6_7
EXTERNAL OUTPUTs/ ATOP DCMP	N	TR_EPU [4]/ DCMP_TRIG [8], DCMP_TRIG [9]	EVOUT 118	TR_EPU_4_DCMP_8_9
EXTERNAL OUTPUTs/ ATOP DCMP	N	TR_EPU [5]/ DCMP_TRIG [10], DCMP_TRIG [11]	EVOUT 119	TR_EPU_5_DCMP_10_11
EXTERNAL OUTPUTs/CLB	N	TR_EPU [7:6] (or TRIG_IN [5:4])	EVOUT 120	TR_EPU_6_TRIG_IN_4

(table continues...)

Table 247 (continued) Trigger outputs (total = 123)

To peripherals	Allocatable resources	Signals	EPU output	EPU output bitwise
			EVOUT 121	TR_EPU_7_TRIG_IN_5
PPCA_CNFG	N	HW_SEL_SRC	EVOUT 122	HW_SEL_SRC

29.12 Use case

This use case demonstrates the configuration of one PPCA counter to trigger three PPCA TCPWMs. Two of the TCPWMs are preceded by additional PPCA counters to introduce a phase delay. This setup is particularly useful for motor control, synchronized PWM generation, or applications requiring precisely timed signals with adjustable phase relationships.

- **Primary PPCA counter**:
  - Acts as the master trigger source
  - Configured to generate a pulse or trigger event
  - Sends trigger signals to other components (TCPWMs or intermediate counters)
- **Intermediate PPCA counters (Phase delay)**: Two additional counters introduce phase delays for two of the TCPWMs. These counters:
  - Count a preconfigured duration
  - Delay the start of the associated TCPWMs before triggering them
- **TCPWM instances**: Each TCPWM (PWM generator) produces a PWM output. Trigger sources include:
  - **Master PPCA counter**: Directly triggers one TCPWM
  - **Intermediate PPCA counters**: Indirectly trigger two TCPWMs with phase delay
  - The TCPWMs triggered by intermediate counters remain synchronized to the master counter but with the injected phase delay

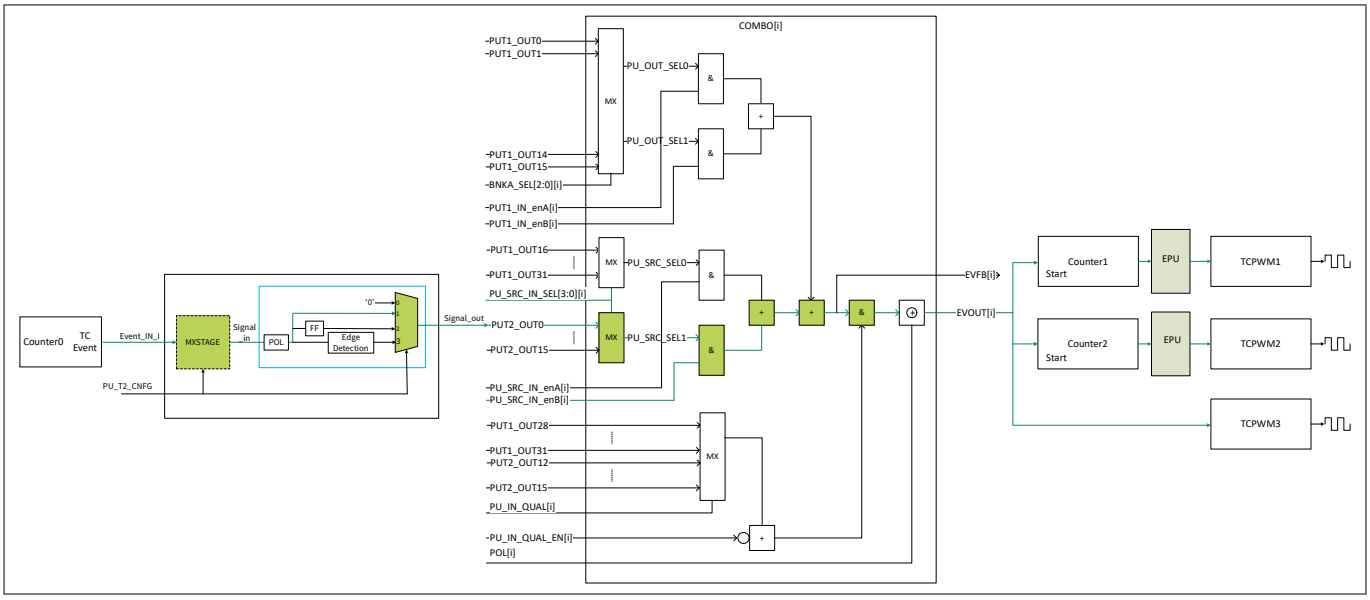


Figure 230 Synchronize TCPWM using EPU

To implement this use case, follow the steps outlined below:

1. **Enable EPU**: Use the control register `PPCA_EPU_CTRL.EPU_EN = 1` to enable the EPU

2. **Configure PU_T2 (Processing Unit_Type 2)**:
   a. Input source selection: Set `PU_T2_CNFG.PUT2_SRC_SEL = TCPWM0_TR_OUT_0_0` to select the desired input event source
   b. Enable PU_T2 in Asynchronous Bypass mode:
      1. Set `PU_T2_CNFG.PUT2_EN = 1` to enable asynchronous bypass mode
      2. In this mode, PU_T2 functions as a simple pass-through, without introducing any additional logic or processing overhead between the input and the output
3. **Configure the Combiner**:
   a. Select PU_T2 output as input to COMBO:
      1. Use the field `COMBO_CNFG.PU_SRC_IN` to select the output of PU_T2 as the input to the COMBO module
      2. Set `COMBO_CNFG.PU_SRC_IN = CY_COMBO_PUT1_OUT17_PUT2_OUT1`
   b. Enable the selected input signal for COMBO:
      1. Use the field `COMBO_CNFG.PU_SRC_IN_EN` to enable the selected input signal (PUT2_OUT1) for processing in the COMBO module
      2. Set the value `CY_COMBO_ENABLE_INPUT_SIGNAL_D` to enable input signal "D"
   c. Route COMBO output to TCPWM/counter start events:
      1. Use the output signal from the COMBO module as the start event for delay Counter1, delay Counter2, and TCPWM3
      2. The start event in TCPWM blocks is used to initiate or trigger their operations

Once the COMBO module's output signal is routed as the start event, the delay counters trigger the start events for TCPWM1 and TCPWM2 after their respective delay periods using the EPU. These counters follow the same configuration steps outlined above, except for input source selection and COMBO output connection.

29.13 Register summary

Table 248 Register summary

Register	Description
PPCA_EPU_CTRL	EPU Control Register
PPCA_EPU_IRQ0_INTR_SRCSEL	Interrupt Source Selection Register
PPCA_EPU_IRQ0_INTR	Interrupt Request Register
PPCA_EPU_IRQ0_INTR_SET	Interrupt Set Request Register
PPCA_EPU_IRQ0_INTR_MASK	Interrupt Mask Register
PPCA_EPU_IRQ0_INTR_MASKED	Interrupt Masked Register
PPCA_EPU_PU_T1_CNFG	PU_Type 1 Configuration Register
PPCA_EPU_PU_T2_CNFG	PU_Type 2 Configuration Register
PPCA_EPU_COMBO_CNFG	COMBO Configuration Register
PPCA_EPU_COMBO_FB_CNFG	COMBO FEEDBACK Configuration Register

30 PPCA Analog Top subsystem (ATOP subsystem)

The PPCA Analog Top subsystem (ATOP SS) embeds the PPCA analog components, those are the analog to digital converters, differential comparator slope generators, digital to analog converters, analog multiplexers, temperature sensor, voltage reference, current reference, etc. and the digital logic to interface and manage the analog components.

Figure 231 shows the ATOP subsystem used in the PPCA module.

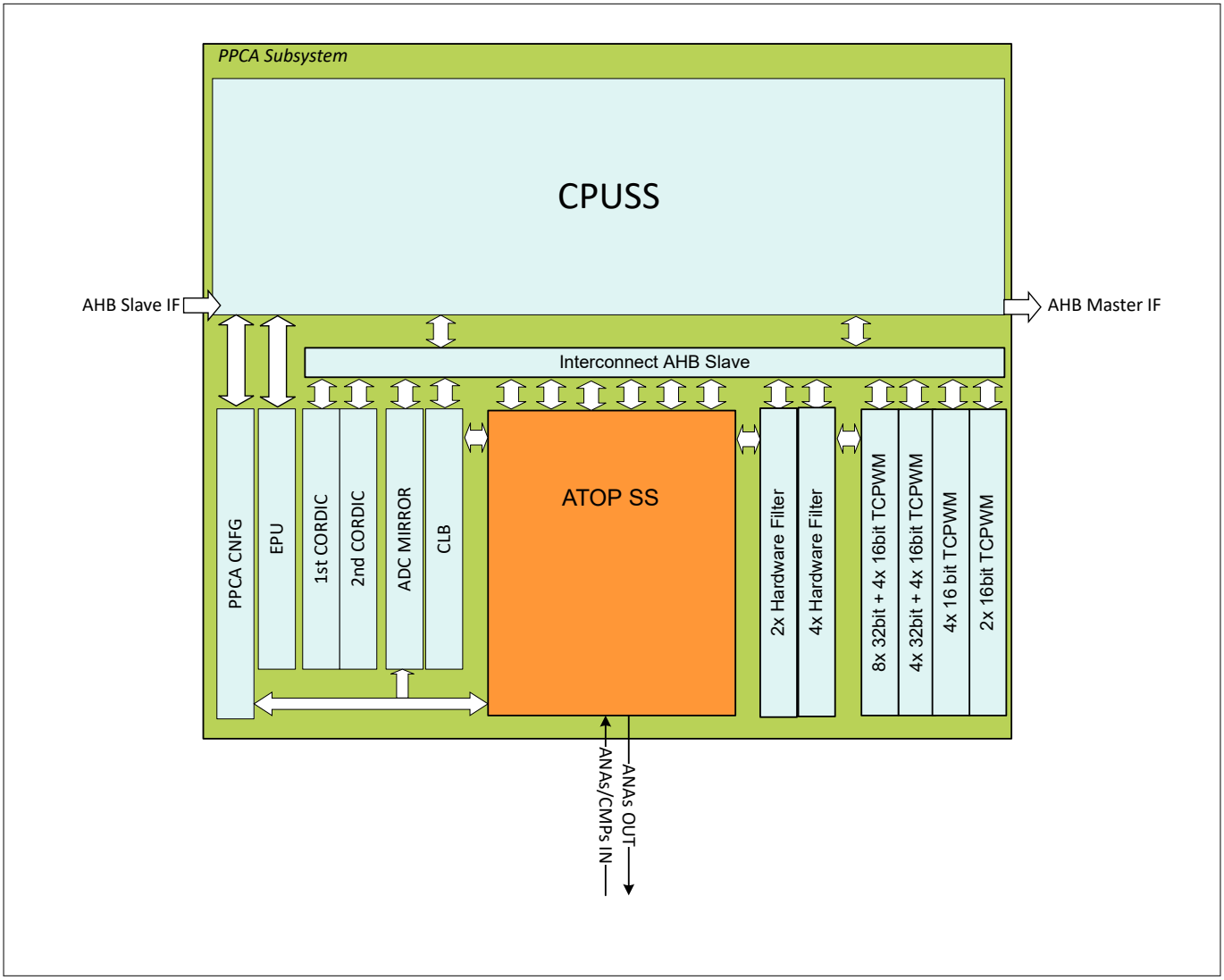


Figure 231 ATOP subsystem in PPCA subsystem

Analog Top subsystem consists of,

- Four ADC groups (ADC_GRP_0, ADC_GRP_1, ADC_GRP_2, and ADC_GRP_3)
- Two DCSG groups (DCSG_GRP_0, and DCSG_GRP_1)

It has six AHB slave interfaces enabling CPUs in the PPCA subsystem to access individually these ATOP subsystem resources.

30.1 Features

- ADC group 0 - ADC module with 8 passive samplers (8 single ended or 4 differential inputs - 1 diff input with AFE)

- ADC group 1 - ADC module with 4 passive samplers (4 single ended or 2 differential inputs - 1 diff input with AFE)
- ADC group 2 - ADC module with 4 passive samplers (4 single ended or 2 differential inputs - 1 diff input with AFE)
- ADC group 3 - ADC module with 8 passive samplers (8 single ended or 4 differential inputs - 1 diff input with AFE)
- DCSG Group 0 - 6x DCSG, 2x Slope Generator, 1x R2RDAC
- DCSG Group 1 - 3x DCSG, 1x Slope Generator, 1x R2RDAC
- TSNS (Temperature Sensor) – Connected to ADC 0 through AUX channel
- 6x ADC filters (ADC Group 0: 2, ADC Group 1: 1, ADC Group 2: 1 and ADC Group 3: 2)
- 12x digital comparator DCMP (ADC Group 0: 4, ADC Group 1: 2, ADC Group 2: 2 and ADC Group 3: 4)
- Voltage and current references
- IDAC Current source for PMBUS resistance and analog pad open circuit detection
- General Purpose Input (GPI)
- Dedicated AHB interface for accessing ADC and DCSG groups

Table 249 shows the groups of ATOP subsystem resources. These resources can be assigned to the required PPCA CPU as we want. This connection is selected by the PPCA_CNFG_CNFG0 register.

Table 249 ATOP AHB slave interfaces

ADC/DCSG Group	Resources
ADC_GRP_0	ADC_0 ADC_FILT_00 ADC_FILT_01 DCMP_00 DCMP_01 DCMP_02 DCMP_03 AREF IDAC GPI
ADC_GRP_1	ADC_1 ADC_FILT_10 DCMP_10 DCMP_11
ADC_GRP_2	ADC_2 ADC_FILT_20 DCMP_20 DCMP_21

(table continues...)

Table 249 (continued) ATOP AHB slave interfaces

ADC/DCSG Group	Resources
ADC_GRP_3	ADC_3 ADC_FILT_30 ADC_FILT_31 DCMP_30 DCMP_31 DCMP_32 DCMP_33
DCSG_GRP_0	DCSG0 DCSG1 DCSG2 DCSG3 DCSG4 DCSG5 R2R DAC 0
DCSG_GRP_1	DCSG6 DCSG7 DCSG8 R2R DAC 1

30.2 Architecture

The Figure 232 is the high-level block diagram of the PPCA ATOP SS and the resources connected to the AHB slave interfaces.

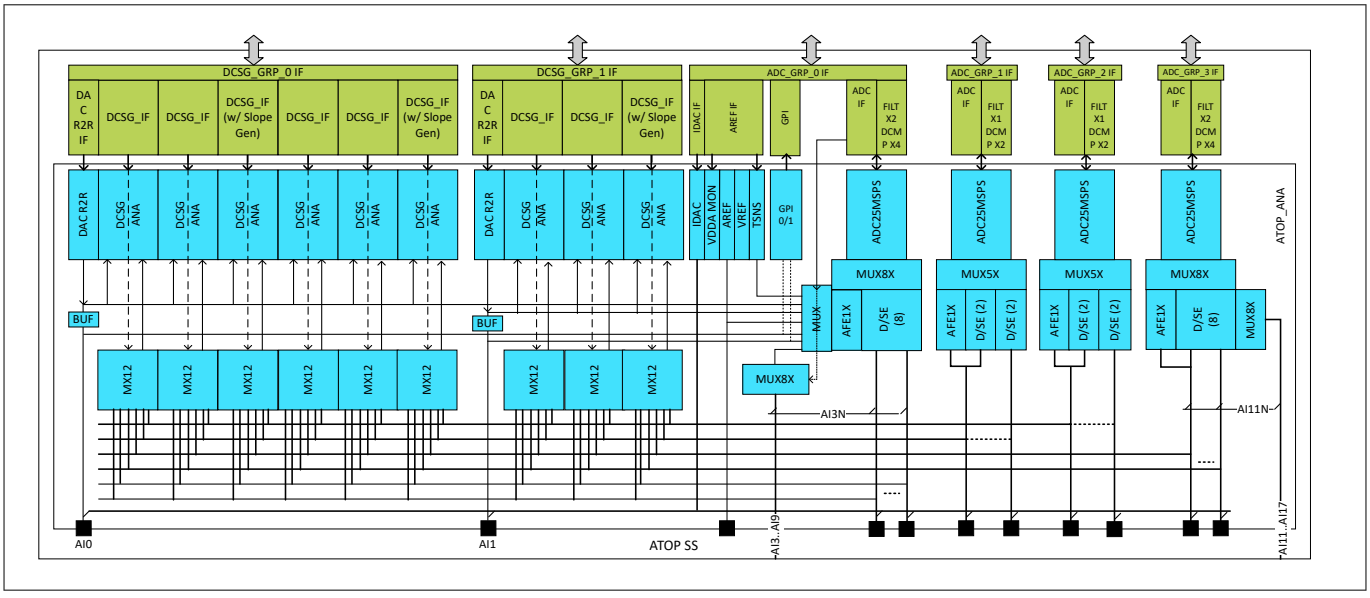


Figure 232 ATOP subsystem high level block diagram

30.2.1 IO connections

PPCA ATOP SS resources and the analog input and output connections to the resources are shown in the Figure 233. It shows the possible connections in the ATOP SS. The multiplexers associated with the resources provide additional flexibility in the connections. See Table 250 for the details of all possible combinations. Refer to the device specific datasheet to see the availability of pins in a selected MCU variant.

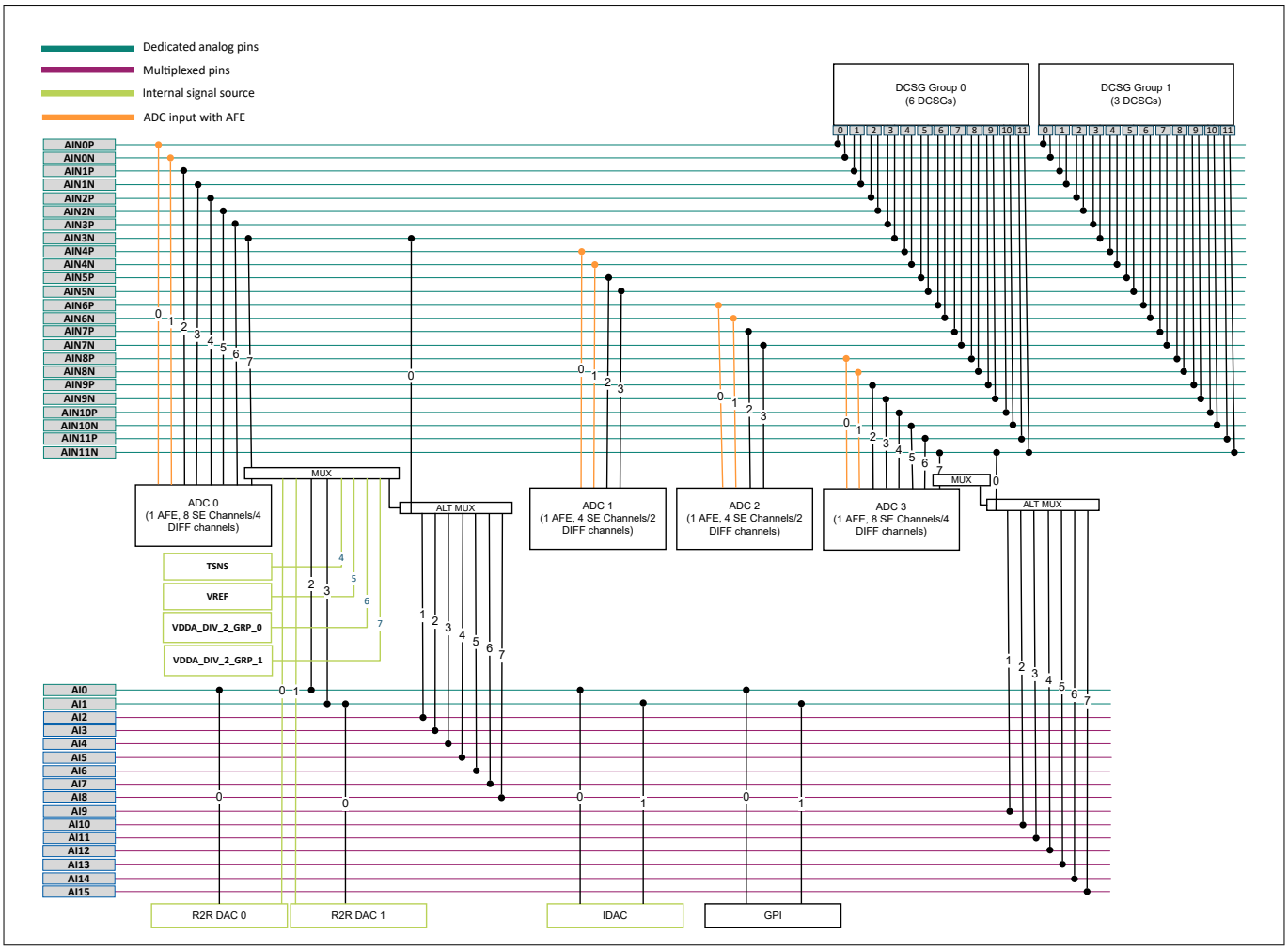


Figure 233 Analog IO connection in ATOP subsystem

The Table 250 summarizes the input pins that can be connected to the PPCA ATOP subsystem and the connections to the resources available.

Table 250 ATOP IO connections

No	Input	In/Out in ADC	In/Out in DCSG
1	AIN0P	ADC 0 DFF Ch 0 Positive input with AFE ADC 0 DFF Ch 0 Positive input ADC 0 SE Ch 0 input	DCSG 0 to 8 Positive input

(table continues...)

Table 250 (continued) ATOP IO connections

No	Input	In/Out in ADC	In/Out in DCSG
2	AIN0N	ADC 0 DFF Ch 0 Negative input with AFE ADC 0 DFF Ch 0 Negative input ADC 0 SE Ch 1 Ve input	DCSG 0 to 8 Negative input
3	AIN1P	ADC 0 DFF Ch 1 Positive input ADC 0 SE Ch 2 input	DCSG 0 to 8 Positive input
4	AIN1N	ADC 0 DFF Ch 1 Negative input ADC 0 SE Ch 3 input	DCSG 0 to 8 Negative input
5	AIN2P	ADC 0 DFF Ch 2 Positive input ADC 0 SE Ch 4 input	DCSG 0 to 8 Positive input
6	AIN2N	ADC 0 DFF Ch 2 Negative input ADC 0 SE Ch 5 input	DCSG 0 to 8 Negative input
7	AIN3P	ADC 0 DFF Ch 3 Positive input ADC 0 SE Ch 6 input	DCSG 0 to 8 Positive input
8	AIN3N	ADC 0 DFF Ch 3 Negative input ADC 0 SE Ch 7 input ADC 0 ALT MUX in 0	DCSG 0 to 8 Negative input
9	AIN4P	ADC 1 DFF Ch 0 Positive input with AFE ADC 1 DFF Ch 0 Positive input ADC 1 SE Ch 0 input	DCSG 0 to 8 Positive input
10	AIN4N	ADC 1 DFF Ch 0 Negative input with AFE ADC 1 DFF Ch 0 Negative input ADC 1 SE Ch 1 input	DCSG 0 to 8 Negative input
11	AIN5P	ADC 1 DFF Ch 1 Positive input ADC 1 SE Ch 2 input	DCSG 0 to 8 Positive input
12	AIN5N	ADC 1 DFF Ch 1 Negative input ADC 1 SE Ch 3 input	DCSG 0 to 8 Negative input

(table continues...)

Table 250 (continued) ATOP IO connections

No	Input	In/Out in ADC	In/Out in DCSG
13	AIN6P	ADC 2 DFF Ch 0 Positive input with AFE ADC 2 DFF Ch 0 Positive input ADC 2 SE Ch 0 Negative input	DCSG 0 to 8 Positive input
14	AIN6N	ADC 2 DFF Ch 0 Negative input with AFE ADC 2 DFF Ch 0 Negative input ADC 2 SE Ch 1 input	DCSG 0 to 8 Negative input
15	AIN7P	ADC 2 DFF Ch 1 Positive input ADC 2 SE Ch 2 input	DCSG 0 to 8 Positive input
16	AIN7N	ADC 2 DFF Ch 1 Positive input ADC 2 SE Ch 3 input	DCSG 0 to 8 Negative input
17	AIN8P	ADC 3 DFF Ch 0 Positive input with AFE ADC 3 DFF Ch 0 Positive input ADC 3 SE Ch 0 input	DCSG 0 to 8 Positive input
18	AIN8N	ADC 3 DFF Ch 0 Negative input with AFE ADC 3 DFF Ch 0 Negative input ADC 3 SE Ch 1 input	DCSG 0 to 8 Negative input
19	AIN9P	ADC 3 DFF Ch 1 Positive input ADC 3 SE Ch 2 input	DCSG 0 to 8 Positive input
20	AIN9N	ADC 3 DFF Ch 1 Negative input ADC 3 SE Ch 3 Negative input	DCSG 0 to 8 Negative input
21	AIN10P	ADC 3 DFF Ch 2 Positive input ADC 3 SE Ch 4 input	DCSG 0 to 8 Positive input
22	AIN10N	ADC 3 DFF Ch 2 Negative input ADC 3 SE Ch 5 input	DCSG 0 to 8 Negative input

(table continues...)

Table 250 (continued) ATOP IO connections

No	Input	In/Out in ADC	In/Out in DCSG
23	AIN11P	ADC 3 DFF Ch 3 Positive input ADC 3 SE Ch 6 input	DCSG 0 to 8 Positive input
24	AIN11N	ADC 3 DFF Ch 3 Negative input ADC 3 SE Ch 7 input ADC 3 ALT MUX in 0	DCSG 0 to 8 Negative input
25	AI0	IDAC 0 Out/GPI 0 input/ ADC 0 MUX input 2	R2R DAC 0 Out
26	AI1	IDAC 0 Out/ADC 0 MUX input 3	R2R DAC 1 Out
27	AI2	ADC 0 ALT MUX input 1	
28	AI3	ADC 0 ALT MUX input 2	
29	AI4	ADC 0 ALT MUX input 3	
30	AI5	ADC 0 ALT MUX input 4	
31	AI6	ADC 0 ALT MUX input 5	
32	AI7	ADC 0 ALT MUX input 6	
33	AI8	ADC 0 ALT MUX input 7	
34	AI9	ADC 3 ALT MUX input 1	
35	AI10	ADC 3 ALT MUX input 2	
36	AI11	ADC 3 ALT MUX input 3	
37	AI12	ADC 3 ALT MUX input 4	
38	AI13	ADC 3 ALT MUX input 5	
39	AI14	ADC 3 ALT MUX input 6	
40	AI15	ADC 3 ALT MUX input 7	

In addition to the IO pins shown in the Table 250, 6 internal signal sources are connected to the ADC 0 MUX. Those are shown in the Table 251

Table 251 ATOP ADC0 MUX connections

Signal Source	ADC 0 AUX Input index	Remark
DAC_R2R_0_O	0	Output of R2R DAC 0
DAC_R2R_1_O	1	Output of R2R DAC 1
TSNS	4	Output of Temperature sensor
VREF	5	Reference voltage
VDDA_DIV_2_GRP_0	6	Group 0 VDDA/2 reference
VDDA_DIV_2_GRP_1	7	Group 1 VDDA/2 reference

30.2.2 Clock

There are two clock sources provided to ATOP subsystem namely PPCA_CLK and ATOP_CLK. These two clocks are in phase.

- PPCA_CLK – PPCA clock signal, maximum frequency is 200 MHz. ATOPSS will provide 200 MHz clock to DCSGs
- ATOP_CLK – ATOP clock signal fed to the ADCs, maximum frequency is 100 MHz. This clock is derived from PPCA_CLK through an integer divider, which has a fixed divider ratio of 2

Table 225 tabulated the frequency combination scenario that will be observed in ATOP subsystem.

30.2.3 BUS interface

ATOP SS has an AHB Bus interface to its resources, single register read/write access and only word transfer size are supported through this interface.

A set of ATOP SS specific MMIO registers are available to the CPUs to access the ATOP resources:

- ADC_GRP registers are used to control ADC, AFE, Filter, digital comparator, AREF, IDAC, GPI configuration and functional operation
- DCSG_GRP registers are used to control DCSG and DAC_R2R configuration and functional operation

30.3 Analog to Digital Converter (ADC)

As mentioned in the section PPCA Analog Top subsystem (ATOP subsystem), ATOP subsystem contains 4 ADC Groups.

Each ADC Group (ADC_GRP) is partitioned into two sections:

- ADC Group Interface (ADC_GRP IF)
- ADC Group Analog (ADC_GRP ANA)

The ADC_GRP_IF contains:

- an AHB Slave Interface(AHB_IF)
- an ADC Interface(ADC_IF) module for the ADC
- DCMP modules
- ADC_FILT modules
- an adapter of different clock domains (PPCA_CLK and ATOP_CLK)

Two ADC converted/filtered data bus are also exported at the ADC_GRP module interface and connected to the hardware 3P3Z Filters. All ADC Result Data registers can be accessed from the main CPU.

The ADC_GRP ANA contains the analog section (ADC_CORE) of the embedded mixed-signal/analog components.

Figure 234 shows the ADC_GRP block diagram.

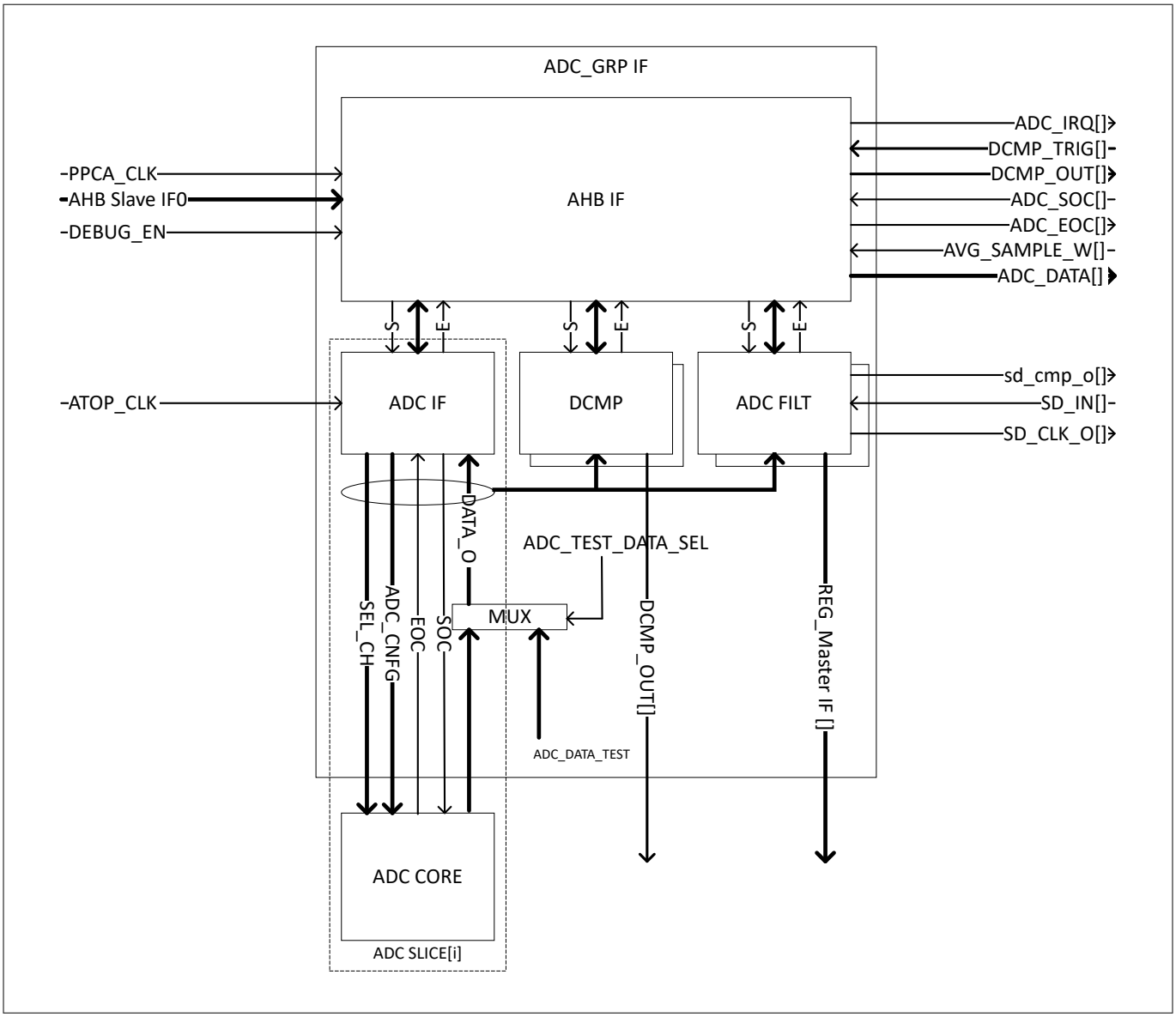


Figure 234 ADC_GRP block diagram

30.3.1 ADC Group 0 (ADC_0)

30.3.1.1 Architecture

ADC_0 supports 4 multiplexed differential analog input channels or 8 multiplexed single ended input channel channels, where one of the single ended channels(channel 7) is multiplexed with a multiplexer with 6 internal sources and two external inputs. One input in it is connected to a second multiplexer that supports 8 auxiliary low bandwidth single ended channels (7 of them are mapped to external inputs). Figure 235 shows the block diagram for ADC_0.

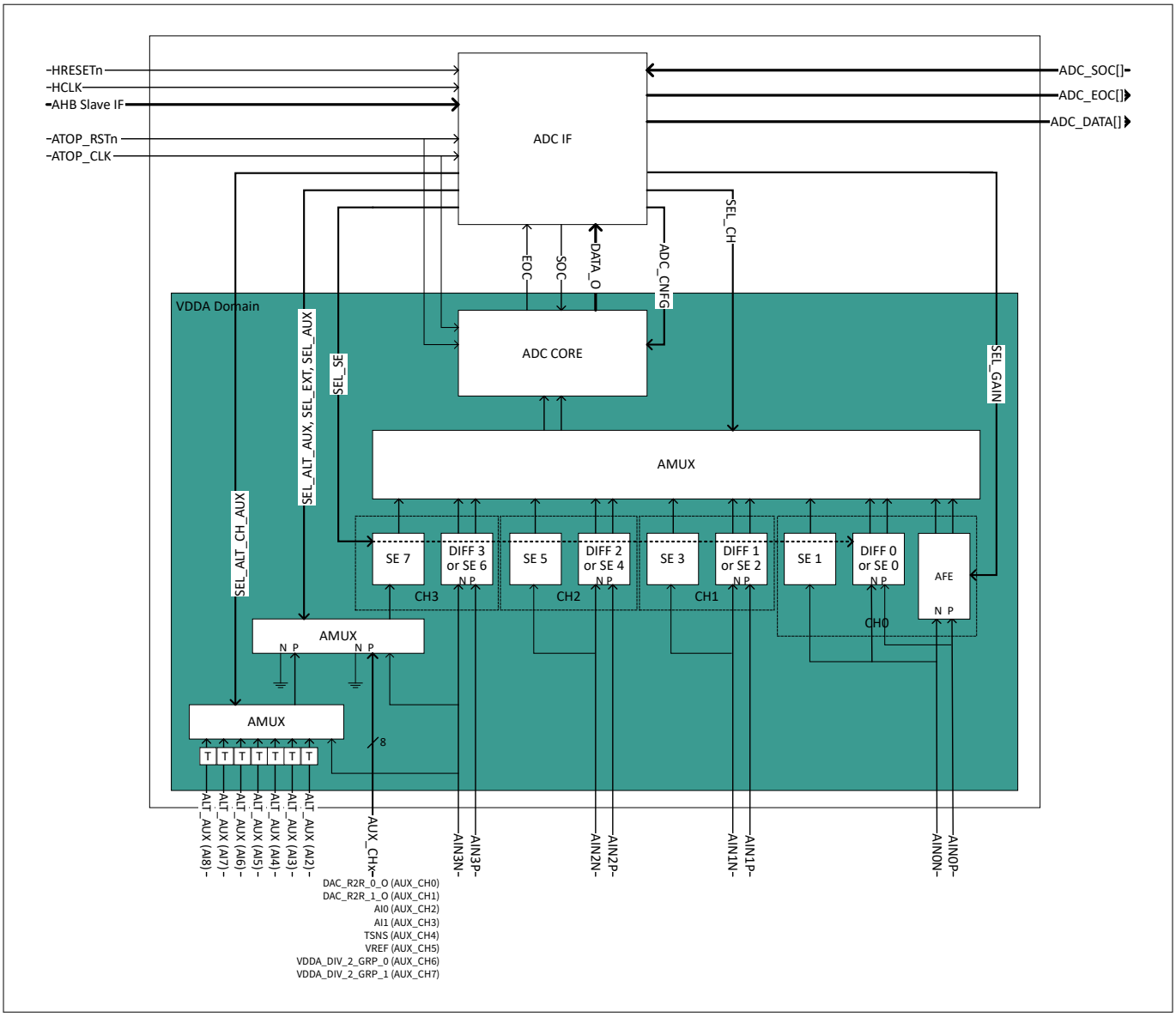


Figure 235 ADC 0 block diagram

30.3.1.2 Features

- SAR architecture
- 4 differential (one input is connected to an AFE) or 8 single ended, multiplexed input channels
- Up to 25 MSPS conversion rate (overall)
- 12-bit (> 10-bit ENOB)
- -0.4V/3.3V input voltage range
- Selectable input gain (x8.25, x16.5, x33, x66) only for the differential input connected to the AFE, with 10 MHz input signal Bandwidth on x8.25 and x16.5, and with 100KHz input signal BW on x33 and x66
- Selectable internal or external 1.2V ADC voltage reference
- Simultaneous sampling or arbitrary sampling sequence or continuous programmable sampling sequence
- Analog multiplexers on one single ended channel, to allow the measurements of internal nodes likes selected DACs, temperature sensor, input voltage supply, band gap, and analog signals AI0, AI1 and 7 analog input signals (AI2-AI9)

- Start of conversion (SOC) is done by a dedicated signal or by writing into the ADC control register
- End of Sampling (EOS) is aligned to the provided SOC or by the control ADC conversion FSM
- AHB slave interface for configuration, control, and data channel read
- Register master interface to directly export two converted data channels

The ADC_0 comprises two sections:

- **ADC 0 Core:**
  The ADC_CORE section implements the ADC conversion functions (digital and analog):
  - Sampler
  - Converter (ADC_CORE)
  - Channels selector (MUX)
  - Gain Amplifier (AFE)
  - Differential/Single Ended signal formatter
- **ADC 0 Interface:**
  The ADC_IF module implements the logic to manage the ADC data conversion, that includes the generation of the SOC signal as per programmed sampling configuration. In particular, the ADC_IF implements:
  - ADC configuration registers (ADC_CNFG),
  - ADC control register (ADC_CTL)
  - ADC status register (ADC_STAT)
  - ADC channels data registers (ADC_DATA [7:0])
  - Auxiliary ADC channels data registers (ADC_AUX_DATA [7:0]),
  - Alternate auxiliary ADC channels data registers (ADC_ALT_AUX_DATA [7:0]),
  - Generation of the SOC and the SEL_CH for the programmed sampling configuration
  - ADC gain and compensation logic
  - Debug state handler

30.3.2 ADC Group 1 (ADC_1)

30.3.2.1 Architecture

ADC_1 implements an analog to digital converter configured to support 2 multiplexed differential analog input channels and 4 multiplexed single ended input channels.

Figure 236 shows the block diagram for ADC_1.

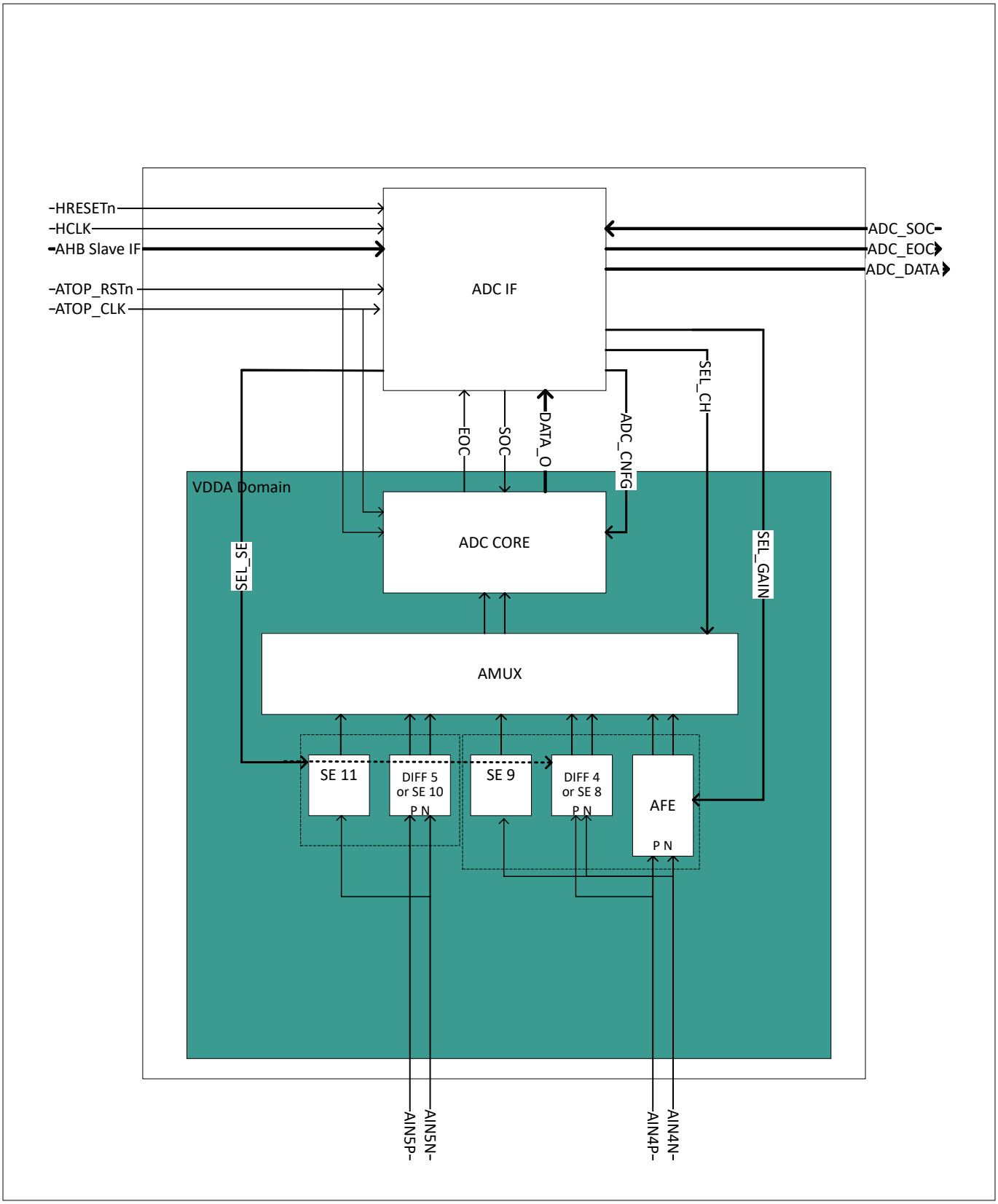


Figure 236 ADC 1 block diagram

30.3.2.2 Features

ADC_1 is similar to ADC_0, only the differences are the following:

- ADC_1 has only 2 differential (one input is connected to an AFE) or 4 single ended, multiplexed input channels
- Auxiliary and alternate auxiliary channels are not present in ADC_1

30.3.3 ADC Group 2 (ADC_2)

30.3.3.1 Architecture

ADC 2 supports 2 multiplexed differential analog input channels and 4 multiplexed single ended input channels.

Figure 237 shows the block diagram for ADC 2.

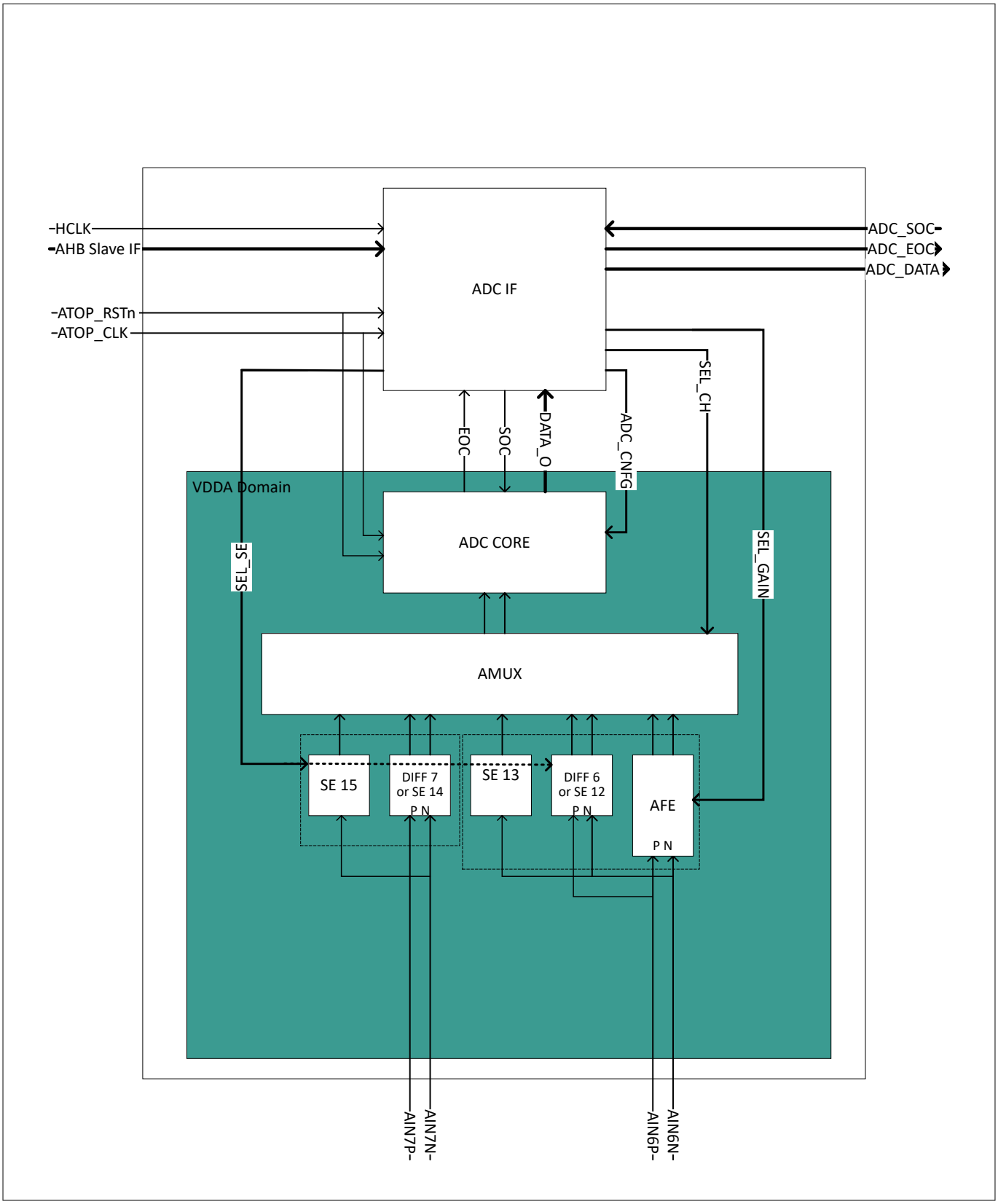


Figure 237 ADC 2 block diagram

30.3.3.2 Features

ADC 1 and ADC 2 are having the same features.

30.3.4 ADC Group 3 (ADC_3)

30.3.4.1 Architecture

ADC_3 supports 4 multiplexed differential analog input channels and 8 multiplexed single ended input channels, where one of the single ended channels can be multiplexed with 8 multiplexed auxiliary low bandwidth single ended channels (7 of them are mapped to analog inputs).

Figure 238 shows the block diagram of ADC_3.

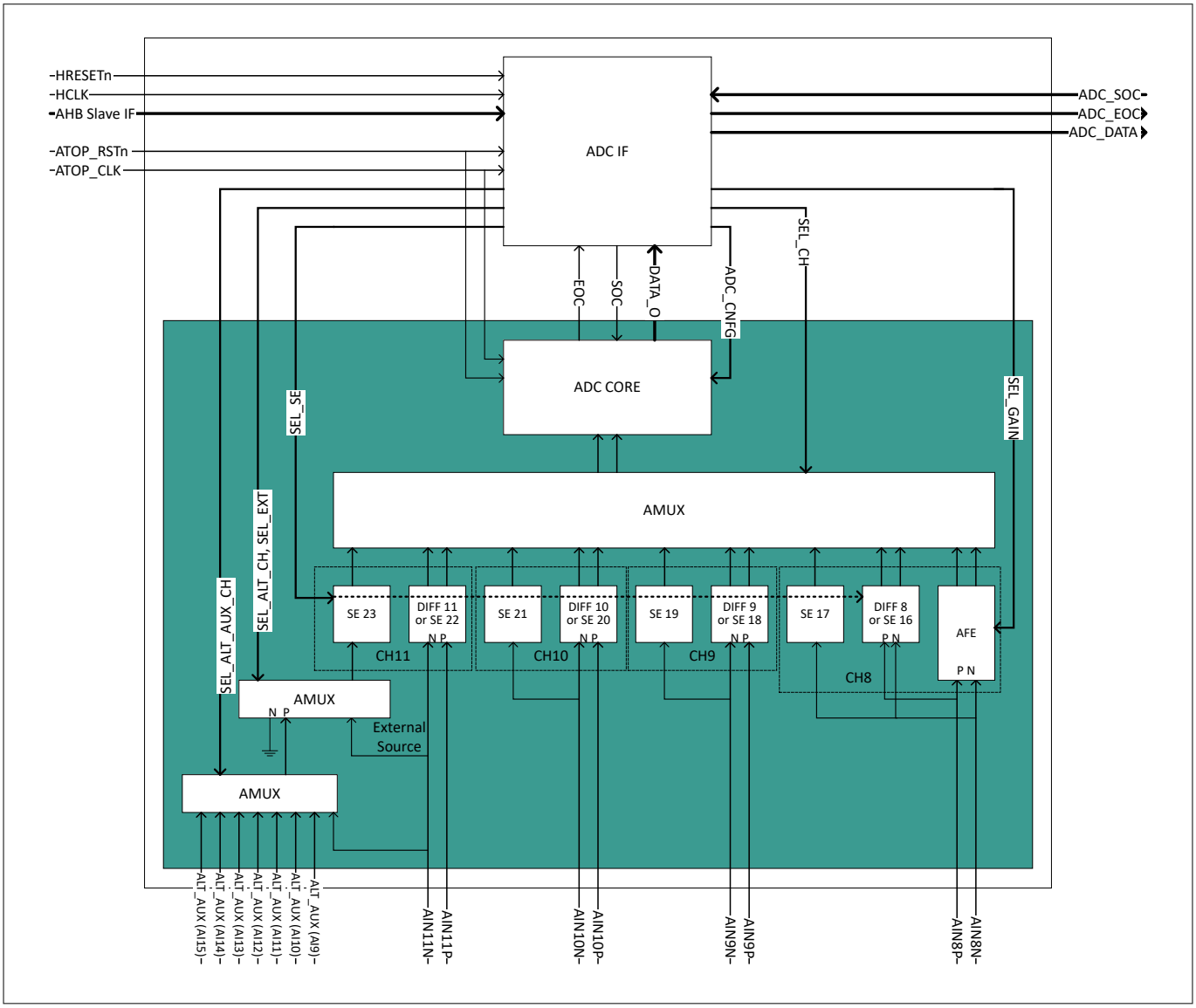


Figure 238 ADC 3 Block diagram

30.3.4.2 Features

ADC_3 is similar to ADC_0. The only differences are the following:

- Analog multiplexer is available on one single ended channel (AIN11N), to allow the measurements of 7 analog input signals (AI9-AI15) these are alternate auxiliary channels
- There are no auxiliary channels in ADC_3

30.3.5 ADC core

The ATOP ADC is an asynchronous 12b SAR ADC which can operate up to 25Msps and is able to convert multiple input samplers in a true differential and single ended fashion. The ADC has SE-DIFF passive samplers and 1 AFE (Analog Front End) sampler which allows for the active amplification of its input signal.

The transfer characteristics of the ADC in the Differential, and single ended configuration are shown in the Figure 239.

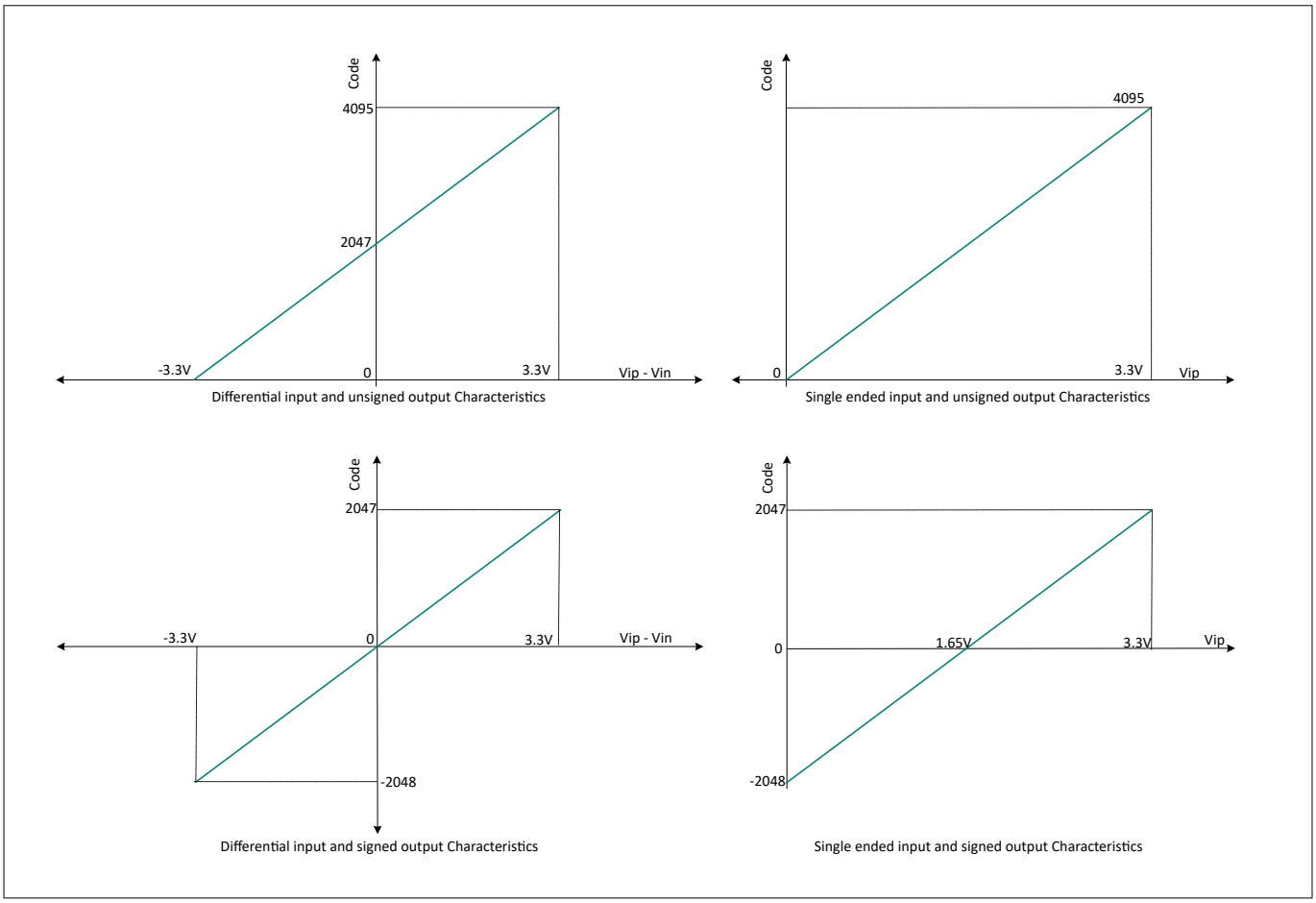


Figure 239 ADC output characteristics

The digital portion of the ADC contains the FSMs for control of the startup, timing, offset, gain calibrations, conversion processes and interface with the analog portion. This portion also manages the handling of the conversion result by applying the offset/gain correction and moving the data out of the ADC core.

The analog portion contains the SAR logic, AMUX, upto 8X Passive samplers, 1X AFE samplers and other analog components associated with these.

30.3.5.1 AFE

In all 4 ADC Groups, the first two channels are connected through the AFE. Usage of AFE is optional. When used, it is used in the differential configuration and should be statically programmed. Gain x8.25, x16.5, x33, x66 can be applied to the AFE, with 10 MHz input signal bandwidth on x8.25 and x16.5 gain, and with 100KHz input signal bandwidth on x33 and x66 gain. Single ended input and unsigned output are not supported when AFE is used.

Output characteristics of the input with AFE configuration is shown in the Figure 240.

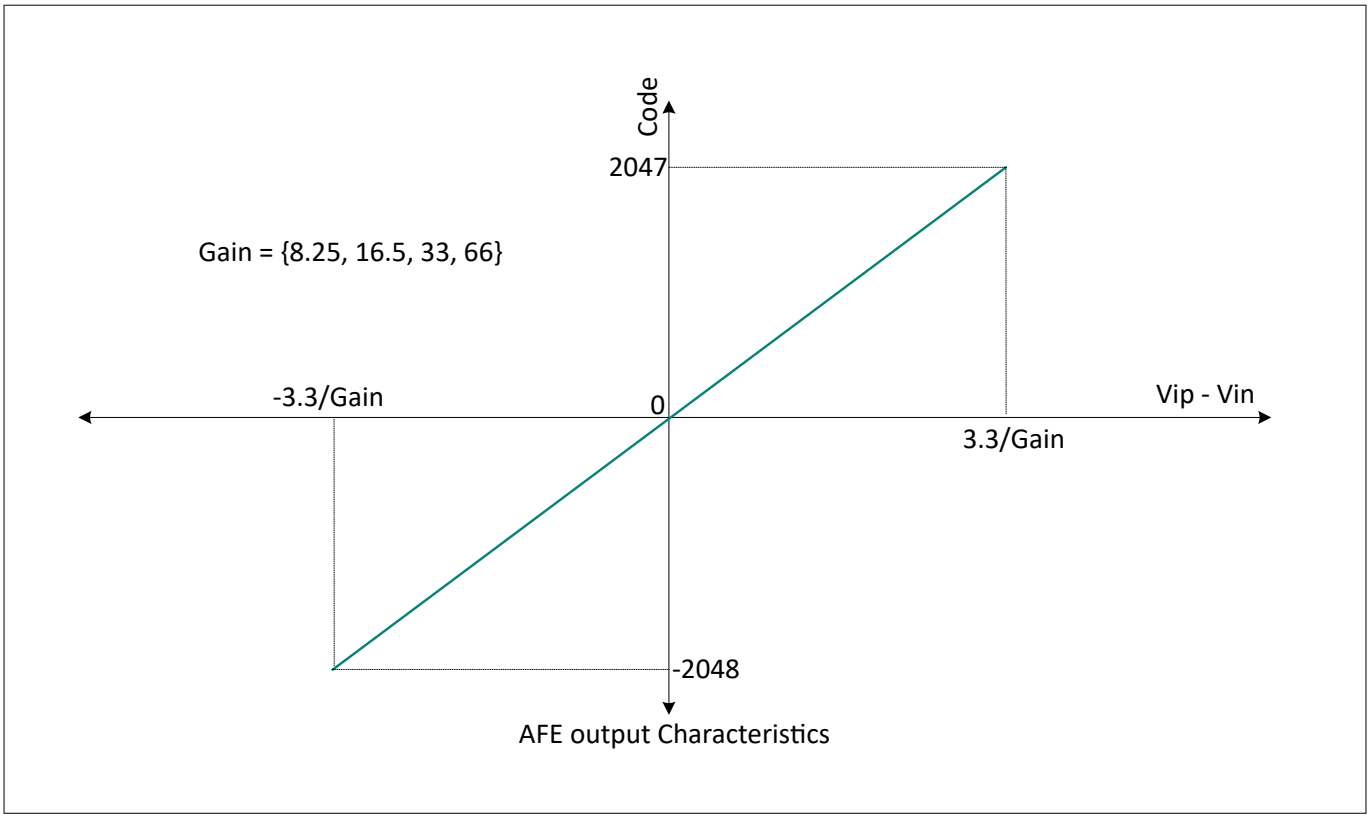


Figure 240 ADC AFE output characteristics

As shown in the Figure 240, based on the AFE configuration, the input voltage represented by the output digital number will vary. To achieve the gain factor 'A' shown in the figure, apply the configuration in the configuration register AFE_CNFG as per the Table 252. When AFE GAIN Settings is 'AFE_GAIN_SET_0', AFE will be OFF.

Table 252 AFE configuration table

AFE GAIN Settings	AFE Clock Ratio Setting	AFE Power Setting	Gain Factor (A)	Sample Rate Msps	Maximum signal frequency
AFE_GAIN_SET_1	ATOP_CLK/4	AFE_PWR_SET_0	8.25	25 Msps	10 Mhz
AFE_GAIN_SET_1	ATOP_CLK/8	AFE_PWR_SET_1	8.25	12.5 Msps	5 Mhz
AFE_GAIN_SET_1	ATOP_CLK/16	AFE_PWR_SET_2	8.25	6.25 Msps	2.5 Mhz
AFE_GAIN_SET_2	ATOP_CLK/4	AFE_PWR_SET_0	16.5	25 Msps	10 Mhz
AFE_GAIN_SET_2	ATOP_CLK/8	AFE_PWR_SET_1	16.5	12.5 Msps	5 Mhz
AFE_GAIN_SET_2	ATOP_CLK/16	AFE_PWR_SET_2	16.5	6.25 Msps	2.5 Mhz
AFE_GAIN_SET_3	ATOP_CLK/32	AFE_PWR_SET_3	33	3.125 Msps	0.1 MHz
AFE_GAIN_SET_4	ATOP_CLK/32	AFE_PWR_SET_3	66	3.125 Msps	0.1 MHz

***Note**:*

- *In single ended mode, the input between 0 and 3.3 V can be provided to the inputs. If the supply is smaller than 3.3 V, it may not be possible to reach the high codes outside of the range [VSSA-0.25 V to VDDA+0.25 V]. In differential mode, positive input minus negative input can be in the [-3.3 V to 3.3 V] range. the common mode of the input can vary in the supply range, here the voltage at the pins is limited by the range [VSSA-0.25 V, VDDA+0.25 V]. If AFE is enabled, positive input minus negative input can be in the [-3.3V/A,3.3V/A] range where A can be 8.25,16.5,33 or 66. Here the voltage at the pins is limited by the design to be inside the [VSSA-0.25 V, 1 V] range to ensure proper operation*
- *The analog inputs for the ADC are in the 3.3 V domain, however the core of the ADC works with a 1.2V reference voltage. This is to enable the relatively high sample rate of 25 Msps and provide an absolute reference to the ADC which is not related to supply. Due to this, the ADC passive samplers apply a 2.75 attenuation to the input signal. The AFE sampler do not attenuate as in this channel we are trying to amplify the input*

30.3.6 Start of conversion (SOC) and end of conversion (EOC) generation

Start of conversion(SOC) and end of conversion(EOC) are signals between ADC core and FSM. The generation of the start of conversion (SOC) is handled by an FSM which is triggered by the TRIGGER signal which is coming from firmware, other peripheral or from the FSM itself. The FSM supports manual, auto and hardware triggering modes, for single channel, and a programmable group of channels conversion. The source of TRIGGER signal to trigger the SOC will depend on the trigger modes.

For manual mode, the TRIGGER signal can come from two sources, namely ADC_SOC signal (external signal/ other peripherals) or ADC_TRIGGER.ADC_TRIGGER bit. In the manual trigger mode, each trigger occurrence will start the programmed conversion sequence and then it waits for the next trigger.

For auto mode, the first trigger signal comes from the ADC_TRIGGER.ADC_TRIGGER bit. In the auto trigger mode, the first trigger occurrence will start the auto trigger sequence, where the elapsed time between the conversion is defined by ADC_CNV_CNFG.AUTO_TRIGGER_SAMPLE_TCLK. To stop the auto trigger mode, the user needs to set to manual trigger mode during the operation. Once FSM detects that the user has set to manual trigger mode, it will complete the existing group conversion before stopping the auto trigger mode. To reconfigure any group conversion, the user is advised to stop the existing group conversion operation before updating the conversion setup.

For arbitrary mode, the TRIGGER signal comes from ADC_SOC signal or ADC_TRIGGER.ADC_TRIGGER bit. There are one TRIGGER signal or one ADC_TRIGGER.ADC_TRIGGER bit for each sampler. In the arbitrary mode, each sample and hold (S/H) is triggered at the occurrence of the dedicated TRIGGER signal, and then the ADC conversion follows. If a trigger occurs during an ongoing ADC conversion, the trigger request will be queued in a FIFO (depth 4 for ADC 0 and 3, 2 for ADC 2 and 3) and will be served when the previous ADC conversion is done. Refer to Arbitrary trigger control scheme for more details.

***Note**: The user must define unique channels in ADC_CH_CNFG0/ADC_CH_CNFG1 for conversion.*

Figure 241 shows the FSM control block diagram.

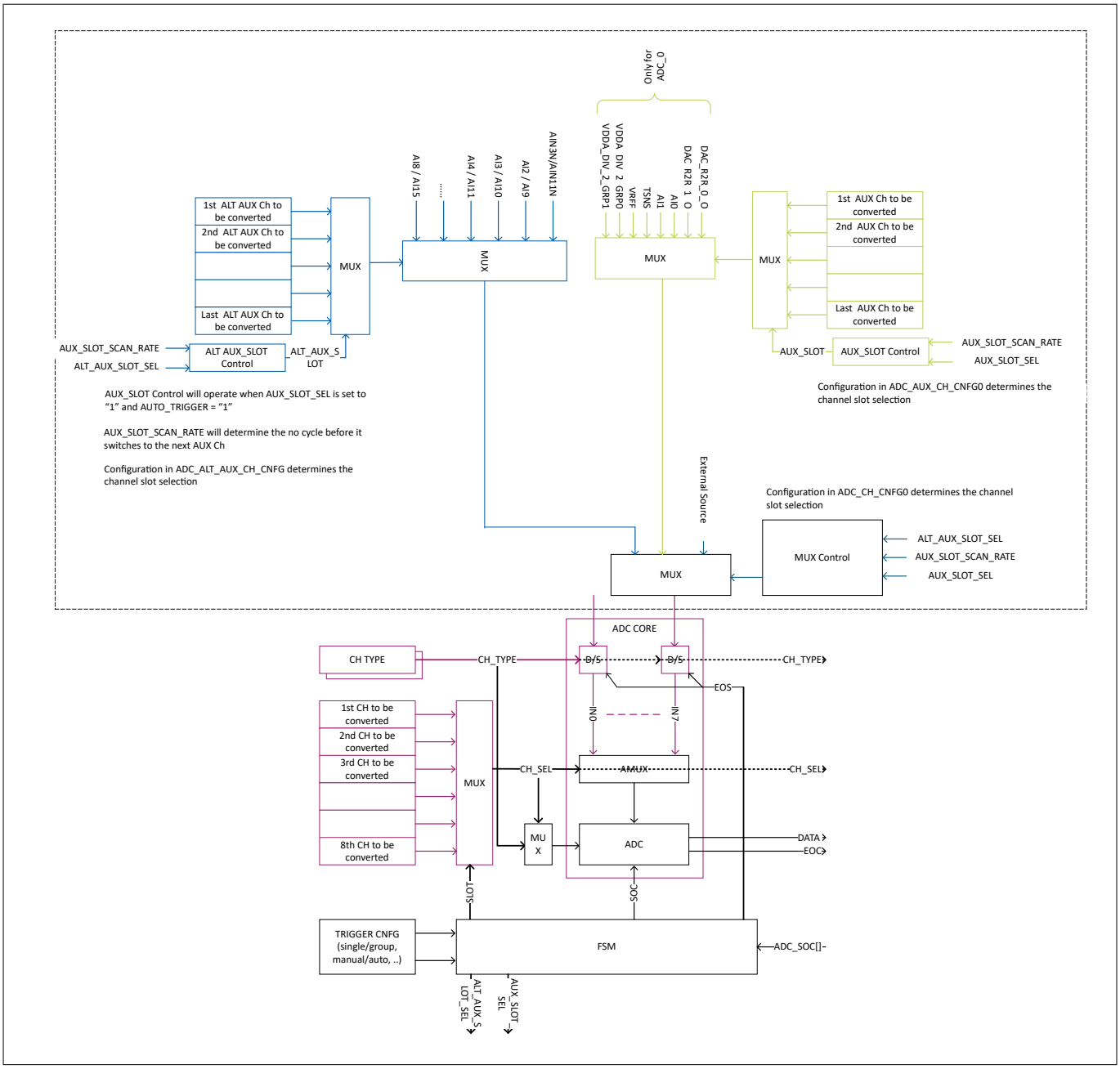


Figure 241 ADC FSM control

30.3.6.1 Slot control for AUX and ALT_AUX channels

AUX and ALT_AUX channels are sharing the same sampler, that is they are multiplexed inputs to the ADC. AUX channels are available for ADC 0 only and ALT_AUX channels are available for ADC0 and ADC3.

When ADC_CTL.AUX_SLOT_SEL or ADC_CTL.ALT_SLOT_SEL is set to '1' and ADC_CNV_CNFG.AUTO_TRIGGER = '1', there is a nested slot control for the AUX/ALT_AUX channel. The AUX/ALT_AUX channel is routed to the last channel, channel 7 of the ADC.

When the last channel is programmed in ADC_CH_CNFGx register and ADC_CTL.AUX_SLOT_SEL is set to 1, then the nested slot control will route the corresponding AUX/ALT_AUX channel through last channel 7. Which AUX channel to be routed is defined through ADC_AUX_CH_CNFG0 and which ALT_AUX channel to be route is defined through ADC_ALT_AUX_CH_CNFG0.

If ADC_CTL.AUX_SLOT_SEL/ADC_CTL.ALT_AUX_SLOT_SET is set to ‘1’ and ADC_CNV_CNFG.TRIGGER_MODE = ‘0’ (Manual Trigger Mode), ADC_AUX_ALT_AUX_CNFG register will be used to select the AUX/ALT_AUX source for the ADC conversion. ADC_AUX_ALT_AUX_CNFG register will be used to specify which analog signal will be converted by the ADC.

The mapping of the ADC_AUX_DATA/ADC_ALT_DATA register is depending on the ADC_AUX_SEL and ALT_AUX_SEL_SRC bitfield.

***Note**: When ADC_CTL.AUX_SLOT_SEL and ADC_CTL.ALT_AUX_SLOT_SEL are set to '1' and ADC_CNV_CNFG.TRIGGER_MODE is set to '01' (auto trigger mode), the FSM will clear the AUX_CHx_TBC, followed by ALT_AUX_CHx_TBC. The clearing of these registers is determined by the values of ADC_CNV_CNFG.NUM_OF_AUX_SLOT and ADC_CNV_CNFG.NUM_OF_ALT_AUX_SLOT. If EXT_SOURCE_SEL is set to '1', then EXT_SOURCE_SEL will also be a part of the AUX and ALT_AUX control.*

Table 253 shows the mapping of the AUX channel for the slot control of AUX channel and corresponding data register where the result will be stored.

Table 253 Slot control and AUX channel mapping

AUX _CHx_TBC	Connected signal	Output data register	Remark
0	DAC_R2R_0_O	ADC_AUX_DATA[0]	For ADC0 Configuration
1	DAC_R2R_1_O	ADC_AUX_DATA[1]	For ADC0 Configuration
2	AI0	ADC_AUX_DATA[2]	For ADC0 Configuration
3	AI1	ADC_AUX_DATA[3]	For ADC0 Configuration
4	TSNS	ADC_AUX_DATA[4]	For ADC0 Configuration
5	VREF	ADC_AUX_DATA[5]	For ADC0 Configuration
6	VDDA_DIV_2_GRP_0	ADC_AUX_DATA[6]	For ADC0 Configuration
7	VDDA_DIV_2_GRP_1	ADC_AUX_DATA[7]	For ADC0 Configuration

For example, if AUX_CH1_TBC is set to 2, AI0 will be route during slot control operation.

Table 254 shows the mapping of the ALT_AUX channel for the slot control of ALT_AUX channel and corresponding data register where the result will be stored.

Table 254 Slot control and ALT_AUX channel mapping

ALT_AUX _CHx_TBC	Connected signal	Output data register	Remark
0	AIN3N/AIN11N	ADC_ALT _AUX_DATA[0]	For ADC0/ADC3 Configuration
1	AI2/AI9	ADC_ALT _AUX_DATA[1]	For ADC0/ADC3 Configuration
2	AI3/AI10	ADC_ALT _AUX_DATA[2]	For ADC0/ADC3 Configuration
3	AI4/AI11	ADC_ALT _AUX_DATA[3]	For ADC0/ADC3 Configuration
4	AI5/AI12	ADC_ALT _AUX_DATA[4]	For ADC0/ADC3 Configuration

(table continues...)

Table 254 (continued) Slot control and ALT_AUX channel mapping

ALT_AUX _CHx_TBC	Connected signal	Output data register	Remark
5	AI6/AI13	ADC_ALT _AUX_DATA[5]	For ADC0/ADC3 Configuration
6	AI7/AI14	ADC_ALT _AUX_DATA[6]	For ADC0/ADC3 Configuration
7	AI8/AI15	ADC_ALT _AUX_DATA[7]	For ADC0/ADC3 Configuration

AUX slot scan rate

The ADC_CNV_CNFG.AUX_SLOT_SCAN_RATE defined the number of times the AUX channel will be converted during the slot operation.

1. **Example 1 - Conversion of Channel 0, AI0 and VDDA_DIV2_GRP_1**

   Configuration:

   ADC_CTL.AUX_SLOT_SEL = 0x1

   ADC_CTL.ALT_AUX_SLOT_SEL = 0x0

   ADC_CNV_CNFG.GROUP_CH = 0x1

   ADC_CNV_CNFG.TRIGGER_MODE = 0x1

   ADC_CNV_CNFG.NUM_OF_SLOT = 0x01

   ADC_CNV_CNFG.NUM_OF_AUX_SLOT = 0x02

   ADC_CNV_CNFG.NUM_OF_ALT_AUX_SLOT = 0xXX (Don’t care since ADC_CTL.ALT_AUX_SLOT_SEL is set to 0x0)

   ADC_CNV_CNFG.EXT_SOURCE_SEL = 0x0

   ADC_CH_CNFG0.CH1_TBC = 0x0

   ADC_AUX_CH_CNFG0.AUX_CH1_TBC = 0x2

   ADC_AUX_CH_CNFG0.AUX_CH2_TBC = 0x7

   ADC_CNFG.CHANNEL_TYPE = 0x00

   The following table illustrate the behavior for different AUX_SLOT_SCAN_RATE

   Table 255 AUX_SLOT_SCAN_RATE behavior for example 1

Round	Conversion	Channel (AUX_SLOT_SCAN_RATE = 1)	Channel (AUX_SLOT_SCAN_RATE = 0)
1	1	Channel 0	Channel 0
	2	Map to AI0	Map to AI0
2	1	Channel 0	Channel 0
	2	Map to AI0	Map to VDDA_DIV_2_GRP_1
3	1	Channel 0	Channel 0
	2	Map to VDDA_DIV_2_GRP_1	Map to AI0
4	1	Channel 0	Channel 0
	2	Map to VDDA_DIV_2_GRP_1	Map to VDDA_DIV_2_GRP_1

(table continues...)

Table 255 (continued) AUX_SLOT_SCAN_RATE behavior for example 1

Round	Conversion	Channel (AUX_SLOT_SCAN_RATE = 1)	Channel (AUX_SLOT_SCAN_RATE = 0)
5	1	Channel 0	Channel 0
	2	Map to AI0	Map to AI0
6	1	Channel 0	Channel 0
	2	Map to AI0	Map to VDDA_DIV_2_GRP_1

From the table above, when AUX_SLOT_SCAN_RATE is set to '1', one can observe in the round 2, the slot control for AUX will maintain at AUX_CH1_TBC, it will only switch to the AUX_CH2_TBC in round 3 of the conversion. When AUX_SLOT_SCAN_RATE is set to '1', the slot control for AUX will switch to next AUX_CH2_TBC in round 2. Please take note that AUX_SLOT_SCAN_RATE support up to 32 which means that it will repeat the same AUX channels for 32 times before it switches to the next AUX channel

2. Example 2 - Conversion of Channel 0, Channel 7, AI0 and VDDA_DIV2_GRP_1

Configuration:

ADC_CTL.AUX_SLOT_SEL = 0x1

ADC_CTL.ALT_AUX_SLOT_SEL = 0x0

ADC_CNV_CNFG.GROUP_CH = 0x1

ADC_CNV_CNFG.TRIGGER_MODE = 0x1

ADC_CNV_CNFG.NUM_OF_SLOT = 0x02

ADC_CNV_CNFG.NUM_OF_AUX_SLOT = 0x02

ADC_CNV_CNFG.NUM_OF_ALT_AUX_SLOT = 0xXX (Don't care since ADC_CTL.ALT_AUX_SLOT_SEL is set to 0x0)

ADC_CNV_CNFG.EXT_SOURCE_SEL = 0x1

ADC_CH_CNFG0.CH1_TBC = 0x0

ADC_CH_CNFG0.CH2_TBC = 0x7

ADC_AUX_CH_CNFG0.AUX_CH1_TBC = 0x2

ADC_AUX_CH_CNFG0.AUX_CH2_TBC = 0x7

ADC_CNFG.CHANNEL_TYPE = 0x00

The following table illustrate the behavior for different AUX_SLOT_SCAN_RATE

Table 256 AUX_SLOT_SCAN_RATE behavior for example 2

Round	Conversion	Channel (AUX_SLOT_SCAN_RATE = 1)	Channel (AUX_SLOT_SCAN_RATE = 0)
1	1	Channel 0	Channel 0
	2	Channel 7	Channel 7
2	1	Channel 0	Channel 0
	2	Channel 7	Map to AI0
3	1	Channel 0	Channel 0
	2	Map to AI0	Map to VDDA_DIV_2_GRP_1
4	1	Channel 0	Channel 0
	2	Map to AI0	Channel 7

(table continues...)

Table 256 (continued) AUX_SLOT_SCAN_RATE behavior for example 2

Round	Conversion	Channel (AUX_SLOT_SCAN_RATE = 1)	Channel (AUX_SLOT_SCAN_RATE = 0)
5	1	Channel 0	Channel 0
	2	Map to VDDA_DIV_2_GRP_1	Map to AI0
6	1	Channel 0	Channel 0
	2	Map to VDDA_DIV_2_GRP_1	Map to VDDA_DIV_2_GRP_1

3. Example 3 - Conversion of Channel 0, AI0, AI7/A14 and enable AUX_SLOT_SCAN_RATE operation for AUX channel and ALT_AUX channel

Configuration:

ADC_CTL.AUX_SLOT_SEL = 0x1
ADC_CTL.ALT_AUX_SLOT_SEL = 0x1
ADC_CNV_CNFG.GROUP_CH = 0x1
ADC_CNV_CNFG.TRIGGER_MODE = 0x1
ADC_CNV_CNFG.NUM_OF_SLOT = 0x01
ADC_CNV_CNFG.NUM_OF_AUX_SLOT = 0x01
ADC_CNV_CNFG.NUM_OF_ALT_AUX_SLOT = 0x01
ADC_CNV_CNFG.EXT_SOURCE_SEL = 0x0
ADC_CH_CNFG0.CH1_TBC = 0x0
ADC_AUX_CH_CNFG0.AUX_CH1_TBC = 0x2
ADC_AUX_CH_CNFG0.ALT_AUX_CH1_TBC = 0x5
ADC_CNFG.CHANNEL_TYPE = 0x00

The following table illustrate the behavior for different AUX_SLOT_SCAN_RATE

Table 257 AUX_SLOT_SCAN_RATE behavior for example 3

Round	Conversion	Channel (AUX_SLOT_SCAN_RATE = 1)	Channel (AUX_SLOT_SCAN_RATE = 0)
1	1	Channel 0	Channel 0
	2	Map to AI0	Map to AI0
2	1	Channel 0	Channel 0
	2	Map to AI0	Map to AI7/AI14
3	1	Channel 0	Channel 0
	2	Map to AI7/AI14	Map to AI0
4	1	Channel 0	Channel 0
	2	Map to AI7/AI14	Map to AI7/AI14
5	1	Channel 0	Channel 0
	2	Map to AI0	Map to AI0
6	1	Channel 0	Channel 0
	2	Map to AI0	Map to AI7/AI14

***Note**: When VREF monitoring is enabled as an AUX channel (CY_AUX_CHANNEL_VREF), sampling VREF introduces kickback noise that disturbs the VREF node voltage. Since VREF is used as the ADC reference, this can degrade measurement accuracy across all ADC channels, not just the AUX channels. To mitigate this issue, set the auto-trigger sample clock to its maximum value (63) to provide sufficient settling time after the kickback noise occurs.*

30.3.6.2 Control scheme

1. ADC voltage sense control – 1 channel configuration

Figure 242 shows the FSM control scheme for ADC voltage sense control for 1 channel configuration. To setup this configuration, user needs to set the following bit:

ADC_CNV_CNFG.NUM_OF_SLOT : 0xX

ADC_CNV_CNFG.SRC_EOS : 0 or 1

(If “0”, use ADC_TRIGGER.ADC_TRIGGER to trigger, If “1”, trigger through ADC_SOC(external) signal)

ADC_CH_CFG0 : 0xXXXX (Configure to which Analog Input pad)

***Note**: In the Figure 242, black means internal signals, green means external signals and blue means mmio registers. CH_SEL<3:0> indicated which channel is being converted by ADC.*

For single channel use case, it will always be slot 0. The information on which channel is being selected will be programed in the ADC_CH_CNFG0 bits. ADC_CNV_CNFG.AUTO_TRIGGER_SAMPLE_TCLK determine the period of the EOS. The minimum is 4 clock cycle which consists of 1 clock cycle for sampling and 3 clock cycle for ADC conversion. If user needs to add additional duration for sampling phase, user can program ADC_CNV_CNFG.AUTO_TRIGGER_SAMPLE_TCLK, this will add additional duration to the sampling phase.

ADC data will be updated to ADC_DATA after 6 clock cycles after SOC issued by subsystem. Subsystem will toggle ADC_EOC_OUT signal based on the update of ADC_DATA.ADC_DATA_0.

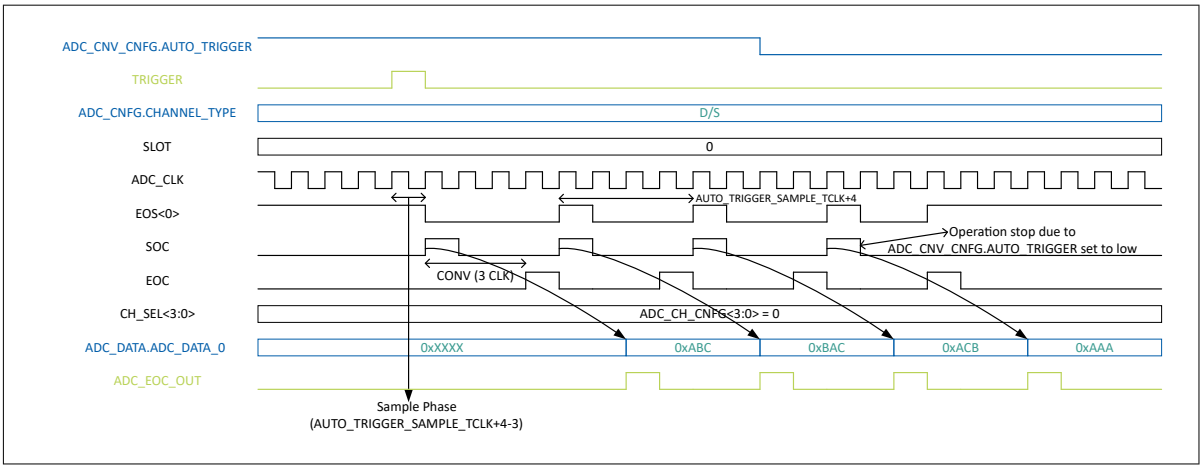


Figure 242 ADC voltage sense control for 1 channel configuration

2. ADC simultaneous voltage sensing control

Figure 243 shows the FSM control scheme for ADC simultaneous voltage sensing control. To setup this configuration, user needs to set the following bit:

ADC_CNV_CNFG.NUM_OF_SLOT : 8

ADC_CNV_CNFG.SRC_EOS : 0 or 1

(If “0”, use ADC_TRIGGER.ADC_TRIGGER to trigger, If “1”, trigger through ADC_SOC signal)

ADC_CH_CFG<15:0> : 0xXXXX (Configure to which Analog Input)

ADC_SAMPLING_MASK.ADC_CH_SAMLING_MASK: 0xXXXX

***Note**: In the Figure 243, black means internal signals, green means external signals and blue means the mmio registers. CH_SEL<3:0> indicated which channel is being converted by ADC.*

For simultaneous channel use case, it will depend on which channel is being programmed in the ADC_CH_CNFG1/ADC_CH_CNFG2 register. The FSM will update CH_SEL<3:0> bit depending on which slot is converted by ADC. The corresponding channel information will be provided.

When ADC_CNV_CNFG.AUTO_TRIGGER is set from high to low, the FSM will not stop the ADC conversion immediately. It will need to complete the whole group of ADC conversion before it will stop the ADC conversion operation. The FSM needs to add one unit clock cycle before sending SOC after EOC for simultaneous sensing operation.

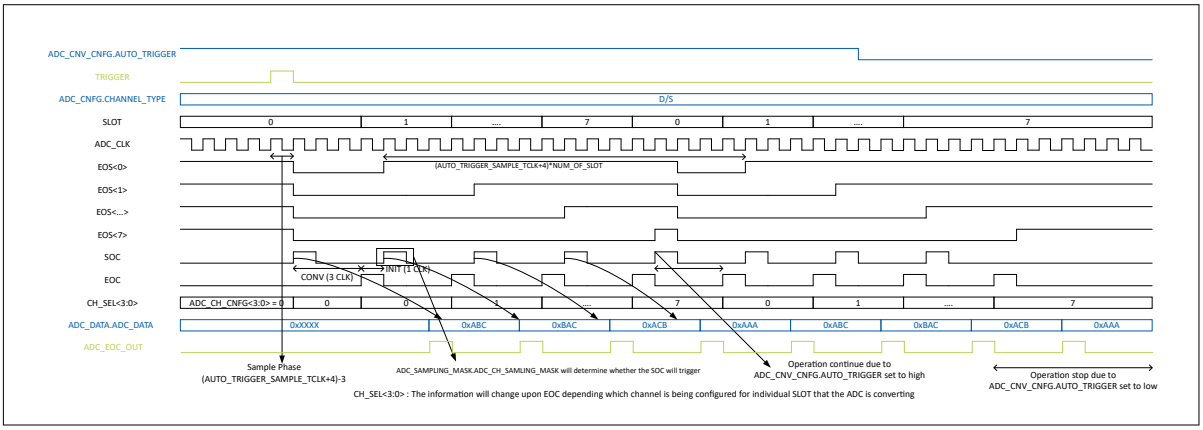


Figure 243 ADC simultaneous voltage sense control

3. ADC0/ADC1/ADC2/ADC3 control with AFE disabled – 1 channel configuration

Figure 244 shows the FSM control scheme for ADC current sensing control without AFE. To setup this configuration, user needs to set the following bit:

ADC_CNV_CNFG.NUM_OF_SLOT : 0xX

ADC_CNV_CNFG.SRC_EOS : 0 or 1

(If “0”, use ADC_TRIGGER.ADC_TRIGGER to trigger, If “1”, trigger through ADC_SOC signal)

ADC_CH_CFG0 : 0xXXXX (Configure to which analog input)

***Note**: In the Figure 244, black means internal signals, green means external signals and blue means mmio registers.*

For single channel use case, it will always be slot 0. The information on which channel is being selected will be programed in the ADC_CH_CNFG0 bits.

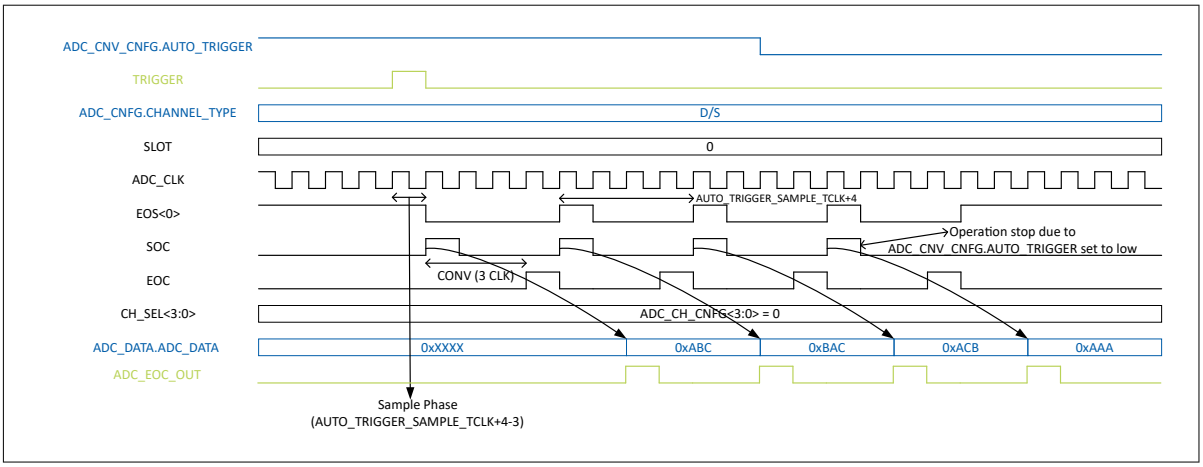


Figure 244 ADC current sensing without AFE for 1 channel configuration

4. ADC sensing control with AFE enabled – 1 channel configuration

Figure 245 shows the FSM control scheme for ADC current sensing control with AFE. To setup this configuration, user needs to set the following bit:

ADC_CNV_CNFG.NUM_OF_SLOT : 0xX

ADC_CNV_CNFG.SRC_EOS : 0 or 1

(If “0”, use ADC_TRIGGER.ADC_TRIGGER to trigger, If “1”, trigger through ADC_SOC signal)

ADC_CH_CFG0 : 0xXXXX (Configure to which Analog Input)

***Note**: In the Figure 245, black means internal signals, green means external signals and blue means mmio registers.*

For single channel use case with AFE enabled, AFE enabled channel will always be assigned to slot 0. AFE_CNFG.AFE_CLK_RATIO defines the period of EOS.

For AFE enabled channel, the sample signal falling edge shall be aligned with the falling edge of the AFE clock and the AFE sample time shall be at least a quarter of the AFE clock period.

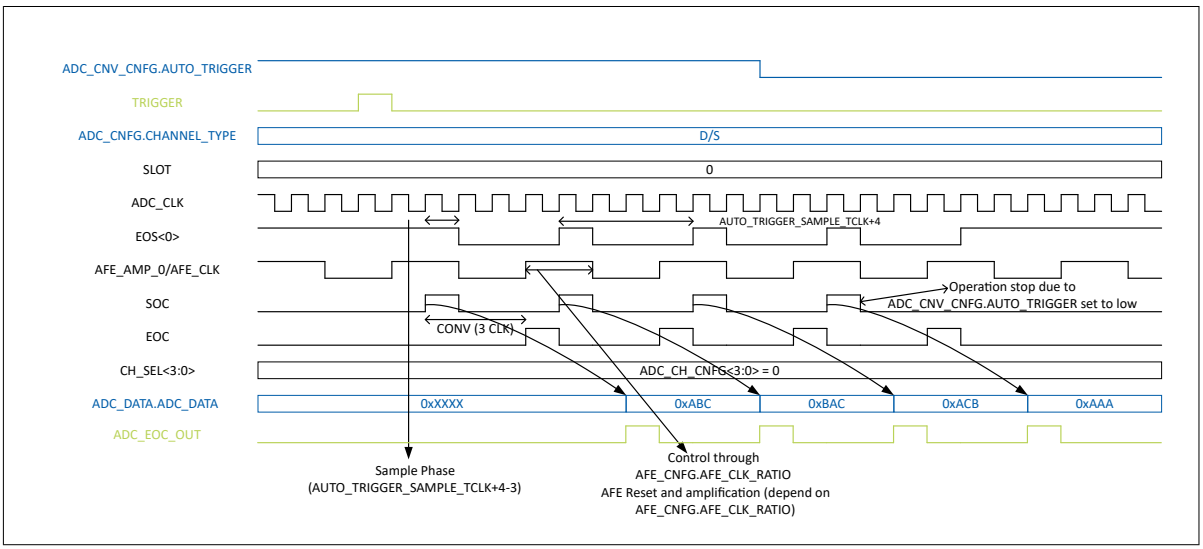


Figure 245 Sensing control (1 Ch with AFE enabled)

The Figure 246 illustrates the setup for 25MSps and 12.5MSps case for ADC current sensing with AFE for 1 channel configuration. It can be observed that the time duration of AFE_AMP_0 will be adjusted based the sampling rate and will be the determining factor the FSM to compute the EOS and SOC

***Note**: In the Figure 246, black means internal signals, green means external signals and blue means mmio registers.*

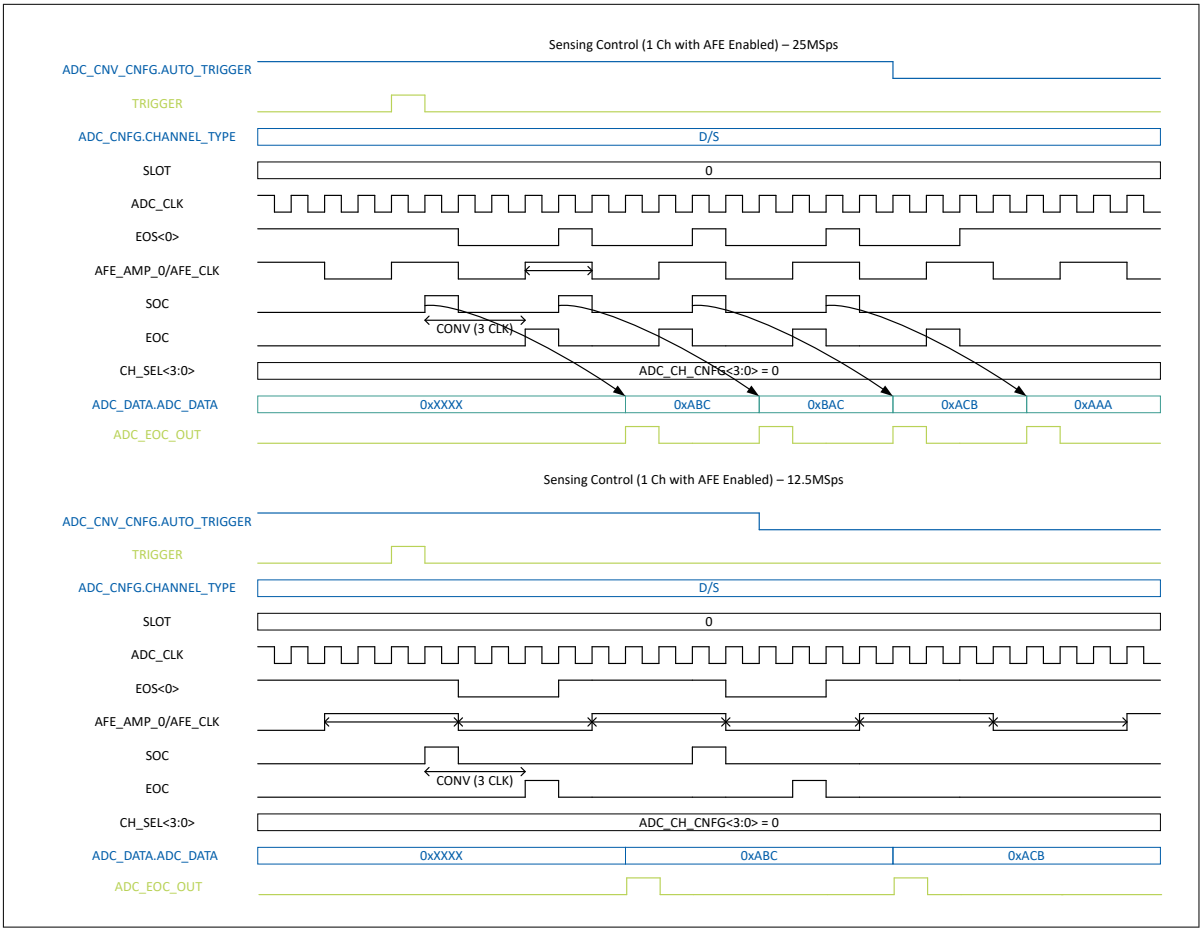


Figure 246 Sensing control (1 Ch with AFE enabled) - different configuration

5. ADC simultaneous switching between channel with AFE and without AFE

Figure 247 illustrates the FSM control scheme for ADC simultaneous switching between channel with AFE and without AFE.

To setup this configuration, user needs to set the following bit:

ADC_CNV_CNFG.NUM_OF_SLOT : 2

ADC_CNV_CNFG.SRC_EOS : 0 or 1

(If “0”, use ADC_TRIGGER.ADC_TRIGGER to trigger, If “1”, trigger through ADC_SOC signal)

ADC_CH_CFG0 : 0xXXXX (Configure to which Analog Input)

***Note**: In the Figure 247, black means internal signals, green means external signals and blue means mmio registers.*

For simultaneous channel use case with AFE, AFE enabled channel will always be assigned to slot 0. The information on which sequential channel is being selected will be programed in the ADC_CH_CNFG0 bits. The FSM will update CH_SEL<3:0> bit depending on which slot is converted by ADC. The corresponding channel information will be provided. Note that for AFE enabled channel, the sample signal falling edge will be aligned with the falling edge of the AFE clock and the AFE sample time will be at least a quarter of the AFE clock period.

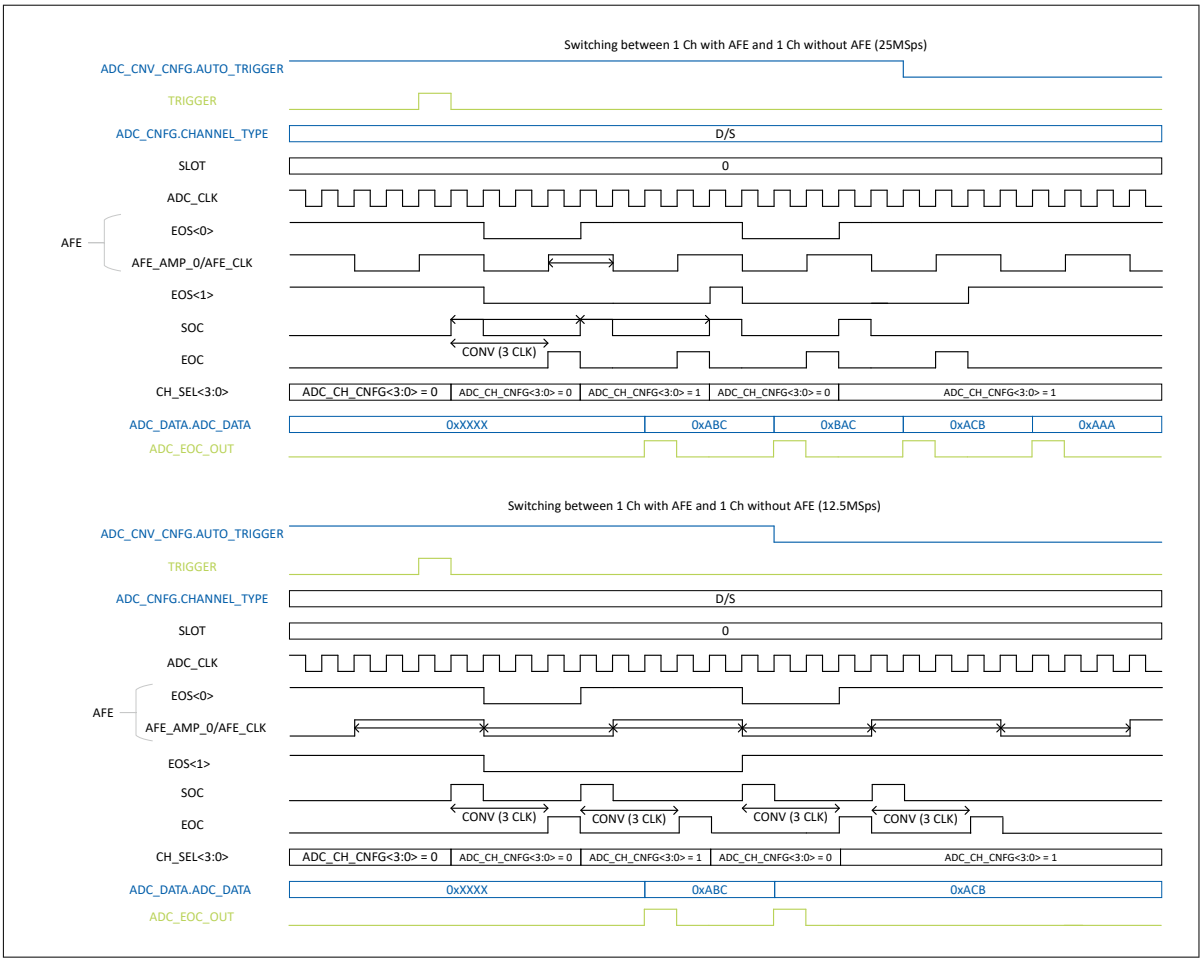


Figure 247 ADC simultaneous switching between channel with AFE and without AFE configuration

30.3.6.3 Arbitrary trigger control scheme

This section describes the behavior of the arbitrary trigger control scheme for the ADC.

1. ADC start of conversion input (ADC_SOC_IN) signal mapping

The arbitrary trigger control is depend on the ADC_SOC_IN[x] signal. This is the signal coming from other peripherals for triggering the conversion of the individual ADC channels.

Table 258 shows the mapping of ADC_SOC_IN[x] to the channel's selector of the ADC_CORE with 8 passive channels.

Table 258 ADC_SOC_IN mapping to channel of the ADC core

ADC_SOC_IN[x]	Channels selector	Remarks
0	AFE channel	When AFE is enabled
0	0	When AFE is disabled
1	1	
2	2	
3	3	
4	4	

(table continues...)

Table 258 (continued) ADC_SOC_IN mapping to channel of the ADC core

ADC_SOC_IN[x]	Channels selector	Remarks
5	5	
6	6	
7	7	

***Note**: For AFE enabled channel, the sample signal falling edge is aligned with the falling edge of the AFE clock and the AFE sample time should be at least a quarter of the AFE clock period.*

2. Arbitrary sampling control scheme

The Figure 248 shows the arbitrary trigger mode behavior based on a single trigger source. Trigger source can be from ADC_SOC_IN or ADC_TRIGGER.ADC_TRIGGER bit.

***Note**: In the Figure 248, black means internal signals, green means external signals and blue means mmio registers.*

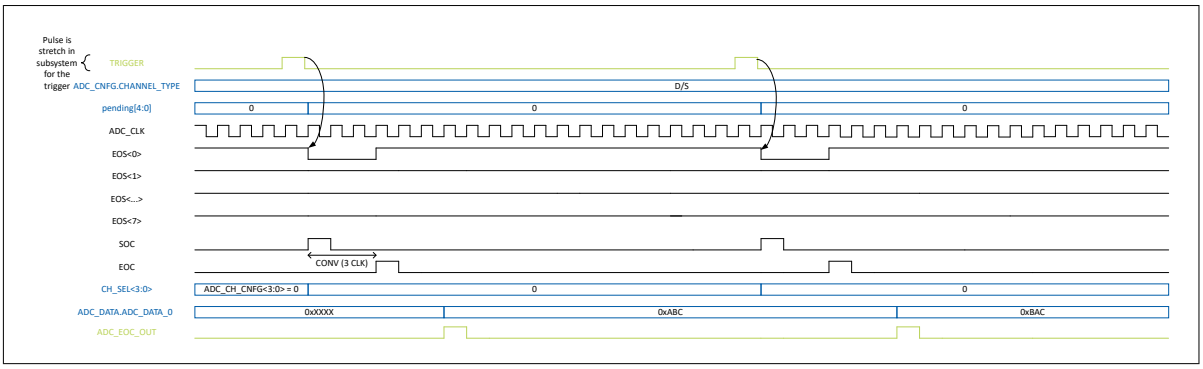


Figure 248 ADC_SOC_IN arbitrary sampling control based on single trigger source

The Figure 249 shows the hardware trigger mode behavior based on a three trigger sources. In addition, the behavior where two triggers source coincide, the FSM will convert based on the priority. Here also Black means internal signals, green means external signals and blue means mmio registers. The SOC may not be issued immediately since FSM may need time to determine the priority decision and handling of the FIFO. Rule of thumb is that lower channel number has higher priority than higher channel number (ADC_SOC_IN[0] > ADC_SOC_IN[1]). If respective ADC_SOC_IN[x] triggers when EOS<x> is low, the FSM will ignore the trigger. It will not be buffered into the FIFO queue

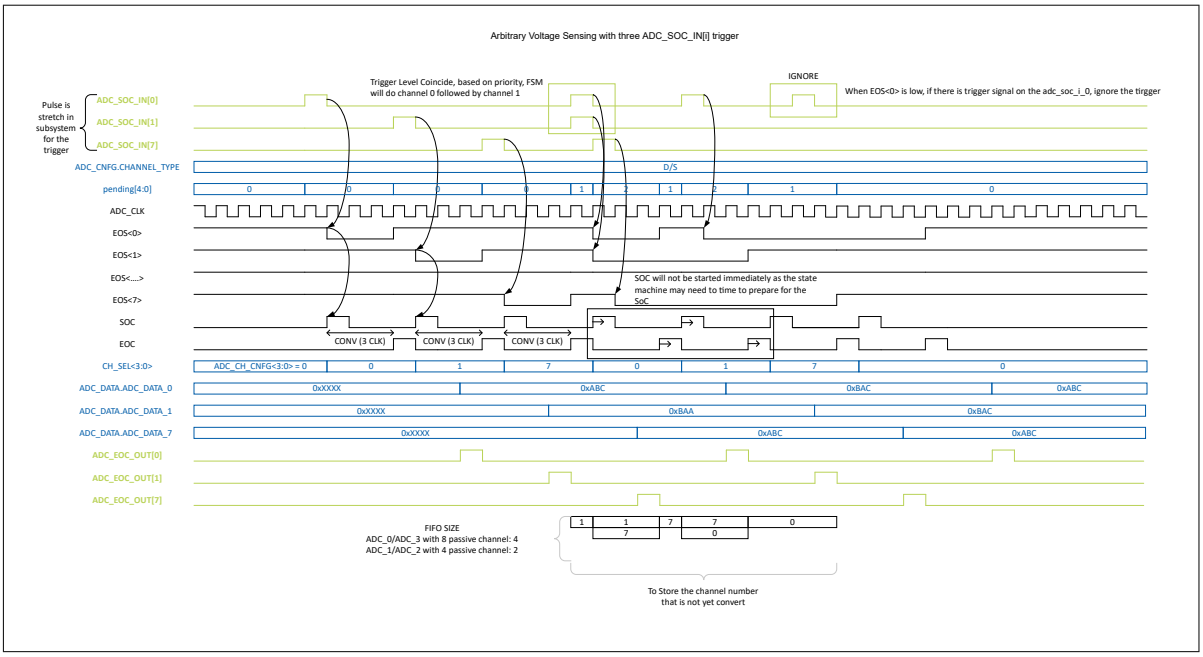


Figure 249 ADC_SOC_IN arbitrary sampling control based on three trigger source

30.3.6.4 Group channel control scheme

There are group and single channel handing being defined in the ADC_CNV_CNFG.GROUP_CH. This section describes the behavior of the group channel under auto trigger or manual trigger mode under different user setup.

The following section shows 3 examples of the behavior in group channel operation under auto trigger and manual trigger mode.

1. Group channel example 1

User wants to do conversion for channel 0, AI0, and VDDA_DIV2_GRP_1 under group channel in auto trigger mode and manual trigger mode. The following will be set:

ADC_CTL.AUX_SLOT_SEL = 0x1

ADC_CTL.ALT_AUX_SLOT_SEL = 0x0

ADC_CNV_CNFG.NUM_OF_SLOT = 0x01

ADC_CNV_CNFG.NUM_OF_AUX_SLOT = 0x02

ADC_CNV_CNFG.NUM_OF_ALT_AUX_SLOT = 0xXX (Don’t care since ADC_CTL.ALT_AUX_SLOT_SEL is set to 0x0)

ADC_CTL.ADC_CNV_CNFG.EXT_SOURCE_SEL = 0x0

ADC_CH_CNFG0.CH1_TBC = 0x0

ADC_AUX_CH_CNFG0.AUX_CH1_TBC = 0x2

ADC_AUX_CH_CNFG0.AUX_CH2_TBC = 0x7

ADC_CNFG.CHANNEL_TYPE = 0x00

AUX_SLOT_SCAN_RATE = 0x0

ADC_AUX_ALT_AUX_CNFG.ADC_AUX_SEL = 0x2 (Need to set for manual trigger mode only)

ADC_AUX_ALT_AUX_CNFG.ADC_AUX_ALT_AUX_SEL = 0x2 (Need to set for manual trigger mode only)

The Table 259 shows the behavior under auto trigger mode and manual trigger mode. To stop the auto trigger mode, user needs to set to manual trigger mode during the operation. Once FSM detects that user set to manual trigger mode, it will complete the existing group conversion before stopping the auto

trigger mode. Under manual trigger, it will always do round 1 conversion in every manual trigger. User will need to update the ADC_AUX_CH_CNFG0.AUX_CH1_TBC if another aux channel is needed for manual trigger

Table 259 Group channel behavior in auto trigger and manual trigger mode for example 1

Round	Conversion	Group channel under auto trigger mode (ADC_CNV_CNFG.TRIGGER_MODE = 0x1)	Group channel under manual trigger mode(ADC_CNV_CNFG.TRIGGER_MODE = 0x0)
1	1	Channel 0	Channel 0
	2	Map to AI0	Map to AI0
2	1	Channel 0	STOP
	2	Map to VDDA_DIV_2_GRP_1	
3	1	Channel 0	
	2	Map to AI0	
4	1	Channel 0	
	2	Map to VDDA_DIV_2_GRP_1	
5	1	Channel 0	
	2	Map to AI0	
6	1	Channel 0	
	2	Map to VDDA_DIV_2_GRP_1	

2. Group channel example 2

User wants to do conversion for Channel 0, Channel 7, AI0, and VDDA_DIV2_GRP_1 under group channel in auto trigger mode. The following will be set:

ADC_CTL.AUX_SLOT_SEL = 0x1

ADC_CTL.ALT_AUX_SLOT_SEL = 0x0

ADC_CNV_CNFG.GROUP_CH = 0x1

ADC_CNV_CNFG.NUM_OF_SLOT = 0x02

ADC_CNV_CNFG.NUM_OF_AUX_SLOT = 0x02

ADC_CNV_CNFG.NUM_OF_ALT_AUX_SLOT = 0xXX (Don’t care since ADC_CTL.ALT_AUX_SLOT_SEL is set to 0x0)

ADC_CTL.ADC_CNV_CNFG.EXT_SOURCE_SEL = 0x1 (Only applicable to auto trigger mode)

ADC_CH_CNFG0.CH1_TBC = 0x0

ADC_CH_CNFG0.CH2_TBC = 0x7 (0x7 does not need to be last channel)

ADC_AUX_CH_CNFG0.AUX_CH1_TBC = 0x2

ADC_AUX_CH_CNFG0.AUX_CH2_TBC = 0x7

ADC_CNFG.CHANNEL_TYPE = 0x00

AUX_SLOT_SCAN_RATE = 0

The following need to be set for manual trigger mode only:

ADC_AUX_ALT_AUX_CNFG.ADC_AUX_ALT_AUX_SEL = 0xX

The Table 260 shows the behavior under auto trigger mode and manual trigger mode. To stop the auto trigger mode, user needs to set to manual trigger mode during the operation. Once FSM detects that user set to manual trigger mode, it will complete the existing group conversion before stopping the auto trigger mode

Table 260 Group channel behavior in auto trigger and manual trigger mode for example 2

Round	Conversion	Group channel under auto trigger mode (ADC_CNV_CNFG.TRIGGER_MODE = 0x1)	Group channel under manual trigger mode(ADC_CNV_CNFG.TRIGGER_MODE = 0x0)
1	1	Channel 0	Channel 0
	2	Channel 7	Channel 7
2	1	Channel 0	STOP
	2	Map to AI0	
3	1	Channel 0	
	2	Map to VDDA_DIV_2_GRP_1	
4	1	Channel 0	
	2	Channel 7	
5	1	Channel 0	
	2	Map to AI0	
6	1	Channel 0	
	2	Map to VDDA_DIV_2_GRP_1	

3. Group channel Example 3

User want to do conversion for Channel 0, AI0, AI7 under group channel in auto trigger mode for AUX channel and ALT_AUX channel:

ADC_CTL.AUX_SLOT_SEL = 0x1

ADC_CTL.ALT_AUX_SLOT_SEL = 0x1

ADC_CNV_CNFG.GROUP_CH = 0x1

ADC_CNV_CNFG.NUM_OF_SLOT = 0x01

ADC_CNV_CNFG.NUM_OF_AUX_SLOT = 0x01

ADC_CNV_CNFG.NUM_OF_ALT_AUX_SLOT = 0x01

ADC_CTL.ADC_CNV_CNFG.EXT_SOURCE_SEL = 0x0

ADC_CH_CNFG0.CH1_TBC = 0x0

ADC_AUX_CH_CNFG0.AUX_CH1_TBC = 0x2

ADC_AUX_CH_CNFG0.ALT_AUX_CH1_TBC = 0x6

ADC_CNFG.CHANNEL_TYPE = 0x00

AUX_SLOT_SCAN_RATE = 0

The following need to be set for manual trigger mode only:

ADC_AUX_ALT_AUX_CNFG.ADC_AUX_SEL = 0x2

ADC_AUX_ALT_AUX_CNFG.SEL_SRC_N_GND = 0x2

ADC_AUX_ALT_AUX_CNFG.ADC_AUX_ALT_AUX_SEL = 0x2

The Table 261 shows the behavior under auto trigger mode and manual trigger mode. To stop the auto trigger mode, user needs to set to manual trigger mode during the operation. Once FSM detects that user set to manual trigger mode, it will complete the existing group conversion before stopping the auto trigger mode

Table 261 Group channel behavior in auto trigger and manual trigger mode for example 3

Round	Conversion	Group channel under auto trigger mode (ADC_CNV_CNFG.TRIGGER_MODE = 0x1)	Group channel under manual trigger mode(ADC_CNV_CNFG.TRIGGER_MODE = 0x0)
1	1	Channel 0	Channel 0
	2	Map to AI0	Map to AI0
2	1	Channel 0	STOP
	2	Map to AI7	
3	1	Channel 0	
	2	Map to AI0	
4	1	Channel 0	
	2	Map to AI7	
5	1	Channel 0	
	2	Map to AI0	
6	1	Channel 0	
	2	Map to AI7	

30.3.6.5 Single channel control scheme

There are groups and single channel handling being defined in the ADC_CNV_CNFG.GROUP_CH. This section describes the behavior of the single channel under auto trigger or manual trigger mode under different user setup.

The following section shows 3 examples of the behavior in single channel under auto trigger and manual trigger mode.

1. **Single channel example 1**

   User want to do conversion for channel 7 under single channel in auto trigger and manual trigger mode. The following will be set:

   ADC_CTL.AUX_SLOT_SEL = 0x0

   ADC_CTL.ALT_AUX_SLOT_SEL = 0x0

   ADC_CNV_CNFG.GROUP_CH = 0x0

   ADC_CNV_CNFG.NUM_OF_SLOT = 0x1

   ADC_CNV_CNFG.NUM_OF_AUX_SLOT = 0xXX

   ADC_CNV_CNFG.NUM_OF_ALT_AUX_SLOT = 0xXX

   ADC_CNV_CNFG.EXT_SOURCE_SEL = 0xX (Don’t care since it configures in single channel mode)

   ADC_CH_CNFG0.CH1_TBC = 0x7

   The following need to be set for manual trigger mode only:

   ADC_AUX_ALT_AUX_CNFG.ADC_AUX_ALT_AUX_SEL = 0x0

ADC_AUX_ALT_AUX_CNFG. SEL_SRC_N_GND= 0x0

ADC_AUX_ALT_AUX_CNFG.ADC_AUX_ALT_AUX_SEL = 0x0

The Table 262 shows the behavior under auto trigger mode and manual trigger mode. To stop the auto trigger mode, user needs to set to manual trigger mode during the operation. Once FSM detects that user set to manual trigger mode, it will complete the existing group conversion before stopping the auto trigger mode

Table 262 Single channel behavior in auto trigger and manual trigger mode for example 1

Round	Conversion	Single channel under auto trigger mode (ADC_CNV_CNFG.TRIGGER_MODE = 0x1)	Single channel under manual trigger mode(ADC_CNV_CNFG.TRIGGER_MODE = 0x0)
1	1	Channel 7	Channel 7
2	1	Channel 7	STOP
3	1	Channel 7	

2. Single channel example 2

User want to do conversion for AI0 under single channel in auto trigger and manual trigger mode for AUX channel. The following will be set:

ADC_CTL.AUX_SLOT_SEL = 0x1

ADC_CTL.ALT_AUX_SLOT_SEL = 0x0

ADC_CNV_CNFG.GROUP_CH = 0x0

ADC_CNV_CNFG.NUM_OF_SLOT = 0xX

ADC_CNV_CNFG.NUM_OF_AUX_SLOT = 0xX

ADC_CNV_CNFG.NUM_OF_ALT_AUX_SLOT = 0xXX

ADC_CNV_CNFG.EXT_SOURCE_SEL = 0xX (Don’t care since it configures in single channel mode)

ADC_CH_CNFG0.CH1_TBC = 0xXX

ADC_AUX_CH_CNFG0.AUX_CH1_TBC = 0x2

ADC_AUX_CH_CNFG0.ALT_AUX_CH1_TBC = 0xXX

AFE_CTL_AFE_EN_0 = 0xX

The following need to be set for manual trigger mode only:

ADC_AUX_ALT_AUX_CNFG.ADC_AUX_SEL = 0x2

ADC_AUX_ALT_AUX_CNFG. SEL_SRC_N_GND= 0x2

ADC_AUX_ALT_AUX_CNFG.ADC_AUX_ALT_AUX_SEL = 0x2

The Table 263 shows the behavior under auto trigger mode and manual trigger mode. To stop the auto trigger mode, user needs to set to manual trigger mode during the operation. Once FSM detects that user set to manual trigger mode, it will complete the existing group conversion before stopping the auto trigger mode

Table 263 Single channel behavior in auto trigger and manual trigger mode for example 2

Round	Conversion	Single channel under auto trigger mode (ADC_CNV_CNFG.TRIGGER_MODE = 0x1)	Single channel under manual trigger mode(ADC_CNV_CNFG.TRIGGER_MODE = 0x0)
1	1	AI0	AI0

(table continues...)

Table 263 (continued) Single channel behavior in auto trigger and manual trigger mode for example 2

Round	Conversion	Single channel under auto trigger mode (ADC_CNV_CNFG.TRIGGER_MODE = 0x1)	Single channel under manual trigger mode(ADC_CNV_CNFG.TRIGGER_MODE = 0x0)
2	1	AI0	STOP
3	1	AI0	

3. Single channel example 3

User wants to do conversion for AI7 under single channel in manual trigger mode for ALT_AUX channel

ADC_CTL.AUX_SLOT_SEL = 0x0

ADC_CTL.ALT_AUX_SLOT_SEL = 0x1

ADC_CNV_CNFG.GROUP_CH = 0x0

ADC_CNV_CNFG.NUM_OF_SLOT = 0xXX

ADC_CNV_CNFG.NUM_OF_AUX_SLOT = 0xXX

ADC_CNV_CNFG.NUM_OF_ALT_AUX_SLOT = 0xXX

ADC_CNV_CNFG.EXT_SOURCE_SEL = 0xX(Don't care since it configures in single channel mode)

ADC_CH_CNFG0.CH1_TBC = 0xXX

ADC_AUX_CH_CNFG0.AUX_CH1_TBC = 0xXX

ADC_AUX_CH_CNFG0.ALT_AUX_CH1_TBC = 0x06

AFE_CTL_AFE_EN_0 = 0xX

The following need to be set for manual trigger mode only:

ADC_AUX_ALT_AUX_CNFG.ALT_AUX_SEL_SRC = 0x6

ADC_AUX_ALT_AUX_CNFG.ALT_AUX_SEL_SRC_GND = 0x6

ADC_AUX_ALT_AUX_CNFG.ADC_AUX_ALT_AUX_SEL = 0x3

The Table 264 shows the behavior under auto trigger mode and manual trigger mode. To stop the auto trigger mode, user needs to set to manual trigger mode during the operation. Once FSM detects that user set to manual trigger mode, it will complete the existing group conversion before stopping the auto trigger mode.

Table 264 Single channel behavior in auto trigger and manual trigger mode for example 3

Round	Conversion	Single channel under auto trigger mode (ADC_CNV_CNFG.TRIGGER_MODE = 0x1)	Single channel under manual trigger mode(ADC_CNV_CNFG.TRIGGER_MODE = 0x0)
1	1	AI7	AI7
2	1	AI7	STOP
3	1	AI7	

30.3.7 Offset and gain correction

When ADC conversion is completed, the ATOP subsystem will apply offset and gain correction to the ADC result before updating the result to the ADC_DATA[x], ADC_AUX_DATA[x] and ADC_ALT_AUX_DATA[x] register depending on which channel is being converted by the ADC.

There are calibration registers available to correct the component specific variations. The PDL implements functions for copying the calibration values to the calibration registers of the ATOP. It is recommended to call the initialization API in the application. For more information, see the Peripheral Driver Library API Reference Manual.

30.3.8 Conversion timing

The conversion timing follows the below rules:

- After the core complete one conversion, a minimum initialization time of 1 clk cycle (10ns for 100 MHz) is needed before triggering another conversion
- The ADC core takes exactly 3 clk cycles (30ns for 100 MHz) to complete a conversion
- Flip-flops are used for signals entering and exiting the analog part of the ADC to synchronize their timing. Hence a latency of 1 clk cycle needs to be considered for the signal propagation
- The minimum sample time is 1 clk cycle
- Once the information about the type of conversion is available (SE or DIFF) the ADC core need to perform its dedicated initialization phase for at least 1 clk cycle
- When idling, the core is initializing
- If the AFE is used, the conversion the timing needs to take into consideration the phase of the AFE

Refer Start of conversion (SOC) and end of conversion (EOC) generation for diagrams showing the conversion timing.

30.3.9 Signed and unsigned translation

The ADC converted and compensated data is unsigned: there is a possibility to automatically convert the unsigned data into signed data. This is configured through ADC_SIGN_UNSIGN_CFG and AUX_SIGN_UNSIGN_CFG register. If the bit is set to ‘0’, the ADC_DATA will be expressed as unsigned data or if the bit is set to ‘1’, the ADC_DATA will be expressed as signed data in Two's complement binary format. Data provided to the filter will be in signed format. Figure 250 shows the flow of the adc_result_o data flow.

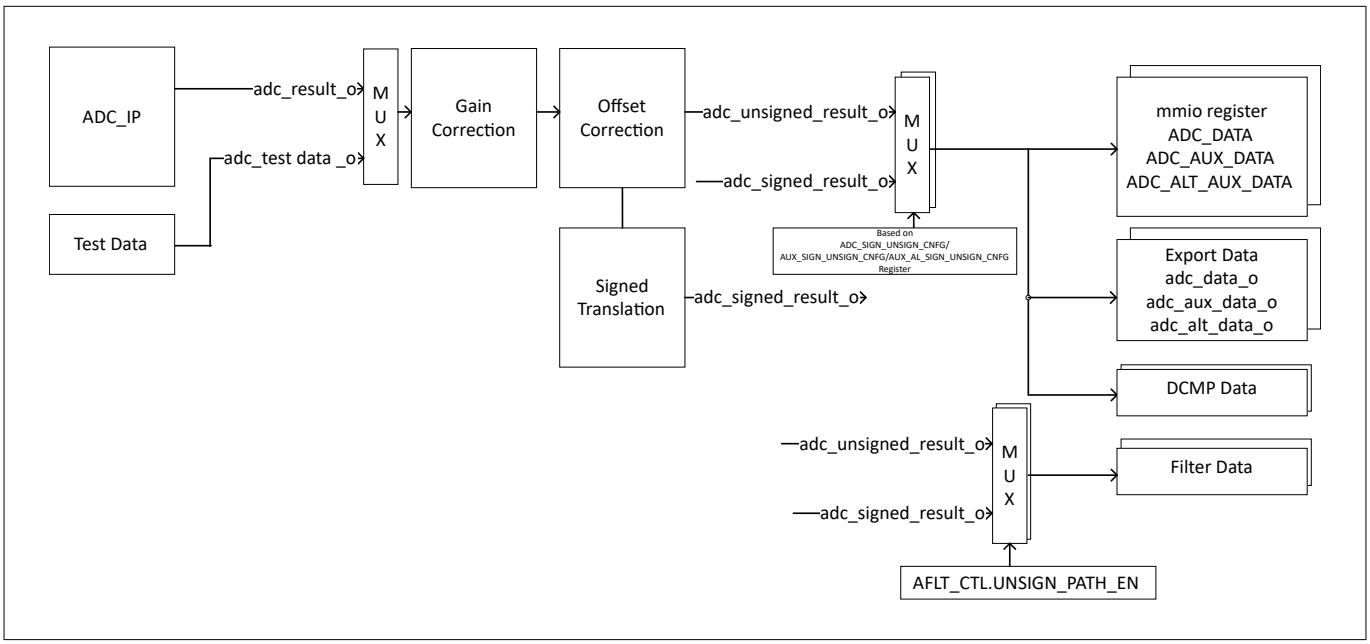


Figure 250 Flow of ADC output data

The Table 265 tabulated the signed/unsigned translation operation.

Table 265 Signed/Unsigned translation

ADC_SIGN_UNSIGN_BITS_X	adc_result_o (After Gain/ Offset correction)	adc_translate_result_o (express as Two's complement binary format)	Decimal value
0	0 – 4095	0x000 – 0xFFF	0 to 4095
1	2048 – 4095	0x000 – 0x7FF	0 to 2047
1	0 – 2047	0x800 – 0xFFF	-2048 to -1

30.3.10 Single ended or differential setup

The ADC in ATOP SS supports single ended or differential setup. The user enabled the ADC channel through ADC_CNFG.CHANNEL_EN bits for ADC_GRP_0. For example, if the user set the following bits:

ADC_CNFG.CHANNEL_EN = 0x05

ADC_CHANNEL_TYPE=0x01

This means that channel 0 and channel 3 are enabled. Channel 0 is configured as differential setup (AIN0P/ AIN0N), and channel 3(AIN0N/GND) is configured as single ended setup. Refer to the IO connections for the details of IO mapping.

30.3.11 Handling of debug state

At the assertion of the debug signal the ATOP SS enters in the debug state. In this state, the ADC needs to complete the existing ADC conversion (Manual/Auto) before the clock is being cut off to pause the operation and resumes the operation when the debug signal is released.

There are four groups of ADCs and the subsystem ensures each group complete the existing ADC conversion. The subsystem does stop any ADC conversion upon entering debug state. For arbitrary mode, the subsystem will complete the conversion from the existing FIFO before entering debug state. Each ADC group has their individual debug freeze input trigger signal. The entering of the debug state will be individually controlled based on the debug freeze input trigger signal that is assigned to the respective group.

Table 266 tabulates the behavior of the debug mode based on ADC_CTL.DBG_FREEZE_EN bit and debug freeze input signal.

Table 266 Behavior of Debug mode for ADC

ADC state	DBG_FREEZE_EN bit	debug freeze input trigger signal
ADC operation continues	0	0
ADC operation continues	0	1
ADC operation pauses after completing the existing ADC conversion	1	1
ADC operation continues	1	0

When activating the debug freeze input trigger, the debug freeze input trigger duration shall be greater than the duration of the ADC conversion in group channel mode under auto trigger mode.

30.3.12 ADC mask handling

The ADC_CH_SAMPLING_MASK bits are defined in ADC_SAMPLING_MASK register that are used to identify which channel will be masked to prevent ADC conversion, whereas ADC_CH_SAMPLING_HWUNMASK bits are defined in ADC_SAMPLING_MASK register that are used to trigger unmasking operation in manual group channel operation in the next trigger. There are a few ways to use the ADC_SAMPLING_MASK register for masking operation. They are described in the following subsection.

30.3.12.1 Firmware mask operation on group mode

This method is for firmware to control the masking of the ADC conversion. This can be done in group channel (ADC_CNV_CNFG.GROUP_CH) under manual trigger mode (ADC_CNV_CNG.TRIGGER_MODE). The following points describe the procedure:

1. Firmware setup slot definition, group channels and trigger mode through ADC_CNV_CNFG register
2. Firmware setup mask through ADC_SAMPLING_MASK register
3. Firmware triggers ADC conversion through ADC_TRIGGER register
4. Upon trigger, ADC conversion starts. Channel that is mask through ADC_SAMPLING_MASK register will not be converted

30.3.12.2 Hybrid mask operation on group mode with manual trigger

This method is for firmware to control the masking of the ADC conversion. This can be done in group channel (ADC_CNV_CNFG.GROUP_CH) under manual trigger mode for group (ADC_CNV_CNG.TRIGGER_MODE). The following points describe the procedure:

1. Firmware setup slot definition, group channels and trigger mode through ADC_CNV_CNFG register
2. Firmware setup mask through ADC_SAMPLING_MASK register
3. Upon trigger, ADC conversion starts. Channel that is masked through ADC_SAMPLING_MASK register will not be converted
4. Firmware triggers ADC_CH_SAMPLING_HWUNMASK bit that corresponds to ADC_SAMPLING_MASK bit. This action triggers the hybrid unmasking operation. If ADC_CH_SAMPLING_HWUNMASK is triggered and does not correspond to ADC_SAMPLING_MASK bit, the design will ignore this trigger action
5. Upon ADC trigger pulse (Through adc_soc_i<x> or ADC_TRIGGER depending on SRC_EOS bit that is defined in ADC_CNV_CNFG register), ADC conversion will be performed on the respective channel that is masked because ADC_CH_SAMPLING_HWUNMASK is performed in step 4. After conversion, the respective channel will be masked again when next ADC_TRIGGER occurred
6. If firmware unset mask through ADC_SAMPLING_MASK register, the hybrid unmasking operation will ease

The Figure 251 shows an example of the hybrid mask operation.

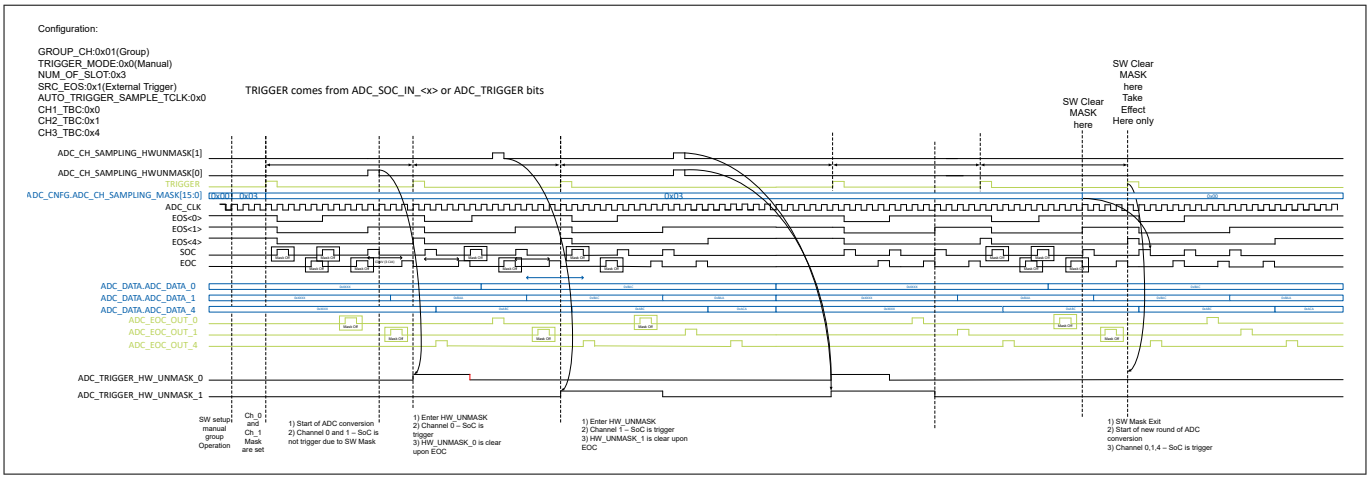


Figure 251 Hybrid mask operation

30.3.13 Digital comparator (DCMP)

As shown in the Figure 232, Digital Comparator (DCMP) modules are part of the ADC_GRP_IF. The number of DCMP modules available in each ADC_GRP_IF is summarized in Table 267.

Table 267 Summaries of DCMP per ADC_GRP_IF

ADC_GRP_IF	No of DCMP modules
ADC_GRP_IF_0	4
ADC_GRP_IF_1	2
ADC_GRP_IF_2	2
ADC_GRP_IF_3	4

The block diagram of the DCMP is shown in Figure 252. Each DCMP is connected to one of the converted ADC data channels. The selection is done through the configuration register.

The purpose of the DCMP is to generate an event (DCMP_OUT) at the time the input (or the elaborated input) crosses the programmed threshold. The DCMP_TRIG input signals are used to start the comparison window when the accumulated/averaged mode is selected through DCMP_CNFG.MODE bits. Different modes of operation are present in DCMP. DCMP_TRIGGER.DCMP_TRIG/DCMP_TRIG[] can be used as a trigger source if accumulated or averaged mode are selected. In Mode 1 and Mode 2, DCMP_TRIGGER.DCMP_TRIG/DCMP_TRIG[] (External trigger) is used to reset the accumulated computation and start the comparison of the DCMP.

DCMP_CNFG.INV_DCMP_EV_CNFG is used to configure whether the immediate comparison block output is inverted or not inverted. User should not change DCMP_CNFG registers (DCMP_CNFG.MODE/ DCMP_CNFG.SAMPLE/DCMP_SEL_SRC_CH/DCMP_INV_DCMP_EN_CNFG) and change the ADC_SIGN_UNSIGN_BITS/AUX_SIGN_UNSIGN_CNFG/AUX_ALT_SIGN_UNSIGN_CNFG on the fly. The correct procedure is to disable the DCMP, change the mode and enable the DCMP. This is to prevent unexpected behavior from the DCMP.

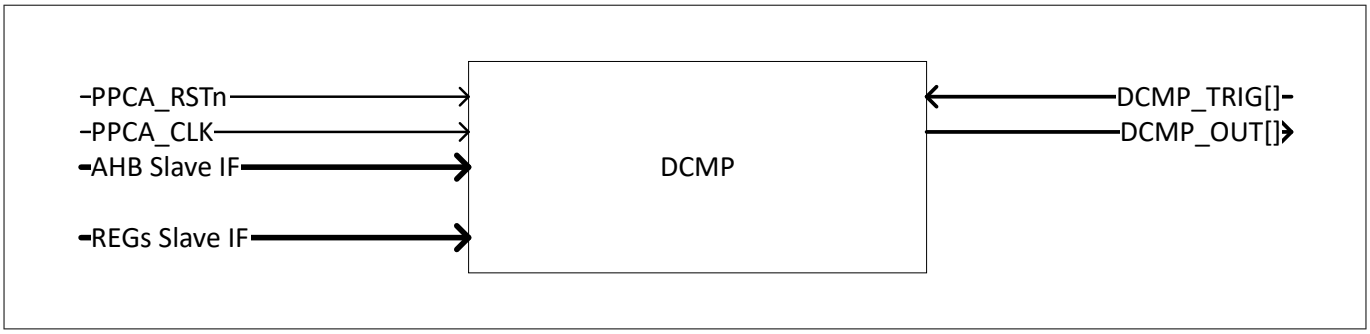


Figure 252 Block diagram of DCMP

The DCMP features:

- Configurable comparison channel (DCMP_CNFG.SEL_SRC_CH)
- Configurable comparison method (CCMP_CNFG.MODE):
  - Immediate (Mode 0):
    - A(n) > TH when INV_DCMP_EV_CNFG = 0
    - A(n) <= TH when INV_DCMP_EV_CNFG = 1
  - Integration of comparison (Mode 1):
    - ∑ ((A(n) > TH) - (A(n) ≤ TH)) > TH_ACC when INV_DCMP_EV_CNFG = 0
    - ∑ ((A(n) <= TH) - (A(n) > TH)) > TH_ACC when INV_DCMP_EV_CNFG = 1
    - When accumulated value < 0, it will cap at 0
    - When accumulated value = 255, it will cap at 255
  - Accumulate of comparison (Mode 2):
    - ∑ (A(n) > TH) > TH_ACC when INV_DCMP_EV_CNFG = 0
    - ∑ (A(n) <= TH) > TH_ACC when INV_DCMP_EV_CNFG = 1
    - When accumulated value = 255, it will cap at 255
  - Averaged (Mode 3):
    - ROUND(1/N ∑A(n)) > TH

Configurable digital threshold (DCMP_TH)

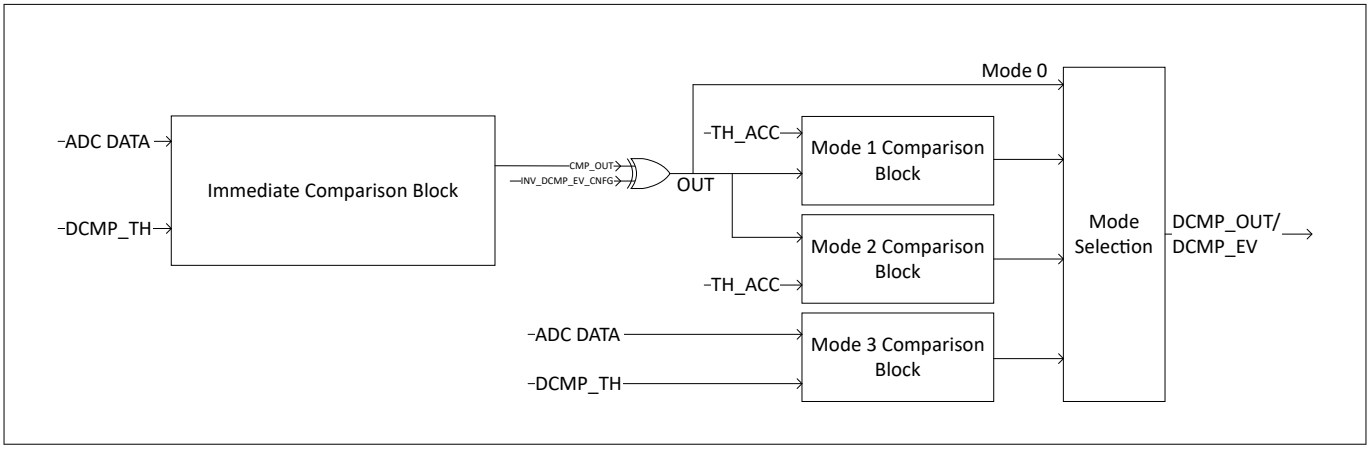


Figure 253 Overview diagram of DCMP

The Table 268 summarizes the usage of the DCMP_TRIGGER.DCMP_TRIG bit and DCMP_TRIG[] on the operation of the DCMP.

Table 268 DCMP operation mode usage behavior

Mode	Reset (Result & Accumulated comp)/ Start trigger	Note
0	-	Immediate comparison upon data is ready
1	DCMP_TRIGGER.DCMP_TRIG/ DCMP_TRIG[]	Trigger signal to start comparison
2	DCMP_TRIGGER.DCMP_TRIG/ DCMP_TRIG[]	Trigger signal to start comparison
3	DCMP_TRIGGER.DCMP_TRIG/ DCMP_TRIG[]	Based on the average sample window

The Table 269 illustrates the ADC DCMP mapping to which ADC. ADC_DCMP XY, where 'X' denotes which ADC where the digital comparator is connected to, and 'Y' denotes the comparator number.

Table 269 ADC_DCMP connection mapping

ADC DCMP	Mapping to ADC
ADC_DCMP_00	ADC_GRP0 ADC
ADC_DCMP_01	ADC_GRP0 ADC
ADC_DCMP_02	ADC_GRP0 ADC
ADC_DCMP_03	ADC_GRP0 ADC
ADC_DCMP_10	ADC_GRP1 ADC
ADC_DCMP_11	ADC_GRP1 ADC
ADC_DCMP_20	ADC_GRP2 ADC
ADC_DCMP_21	ADC_GRP2 ADC
ADC_DCMP_30	ADC_GRP3 ADC
ADC_DCMP_31	ADC_GRP3 ADC
ADC_DCMP_32	ADC_GRP3 ADC
ADC_DCMP_33	ADC_GRP3 ADC

30.3.13.1 Invert comparator output configuration

When INV_DCMP_EV_CNFG is set to '0', the Figure 254 and Figure 255 shows the behavior of the DCMP_OUT in signed ADC and unsigned ADC operation, respectively.

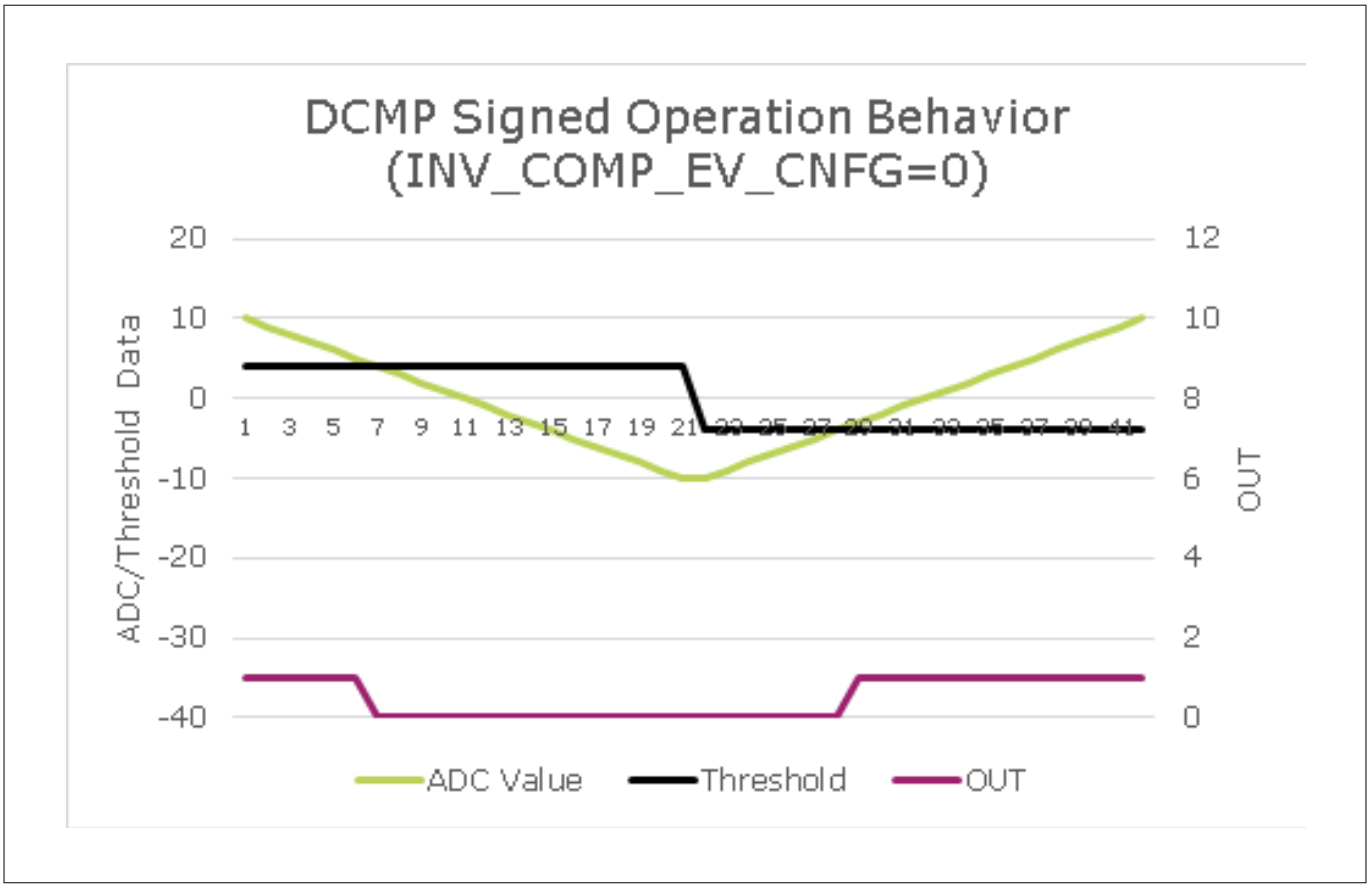


Figure 254 DCMP_OUT behavior in signed ADC data when INV_COMP_EV_CNFG = 0

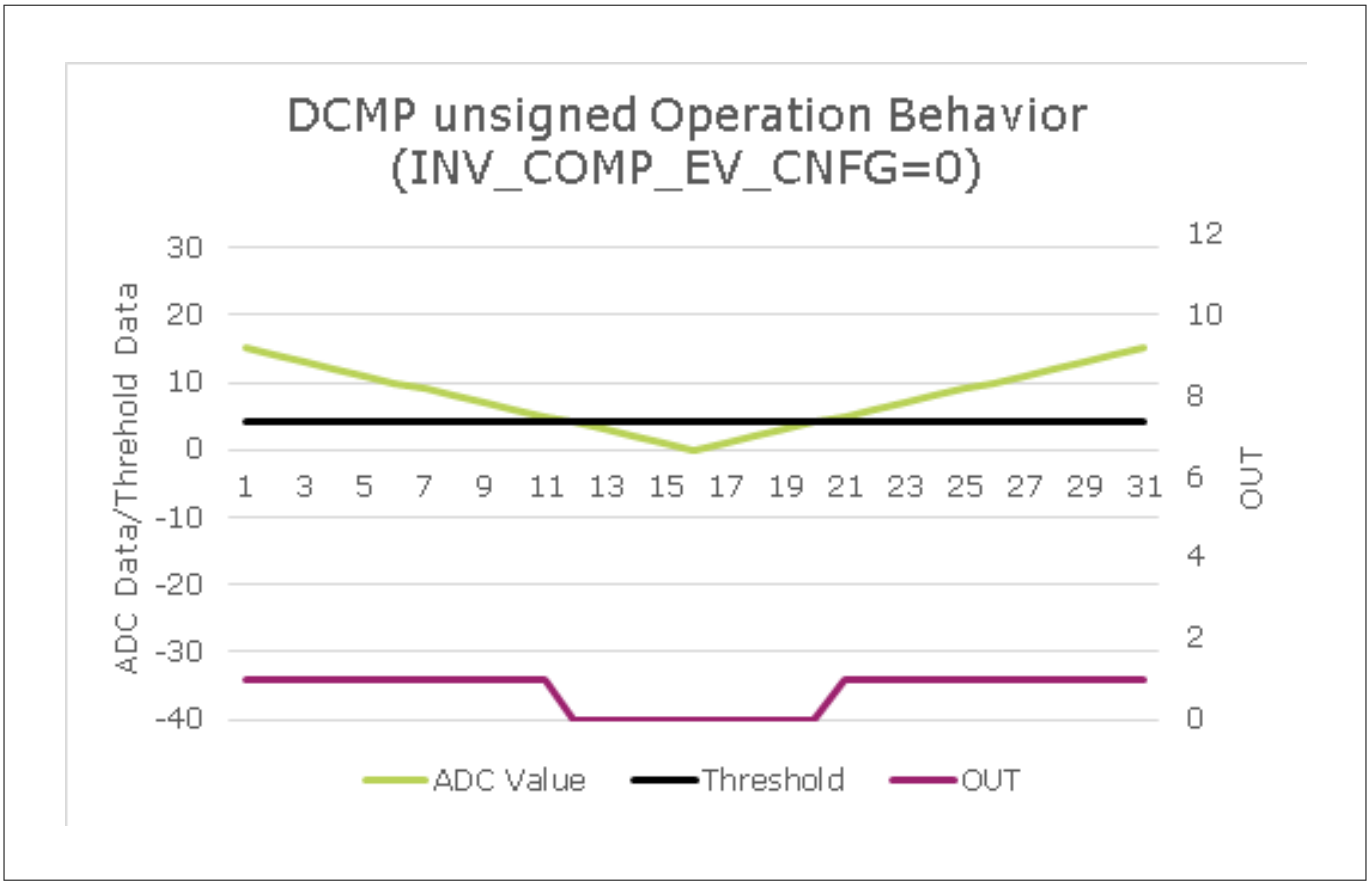


Figure 255 DCMP_OUT behavior in unsigned ADC data when INV_COMP_EV_CNFG = 0

When INV_DCMP_EV_CNFG is set to '1', the Figure 256 and Figure 257 shows the behavior of the DCMP_OUT in signed ADC and unsigned ADC operation, respectively.

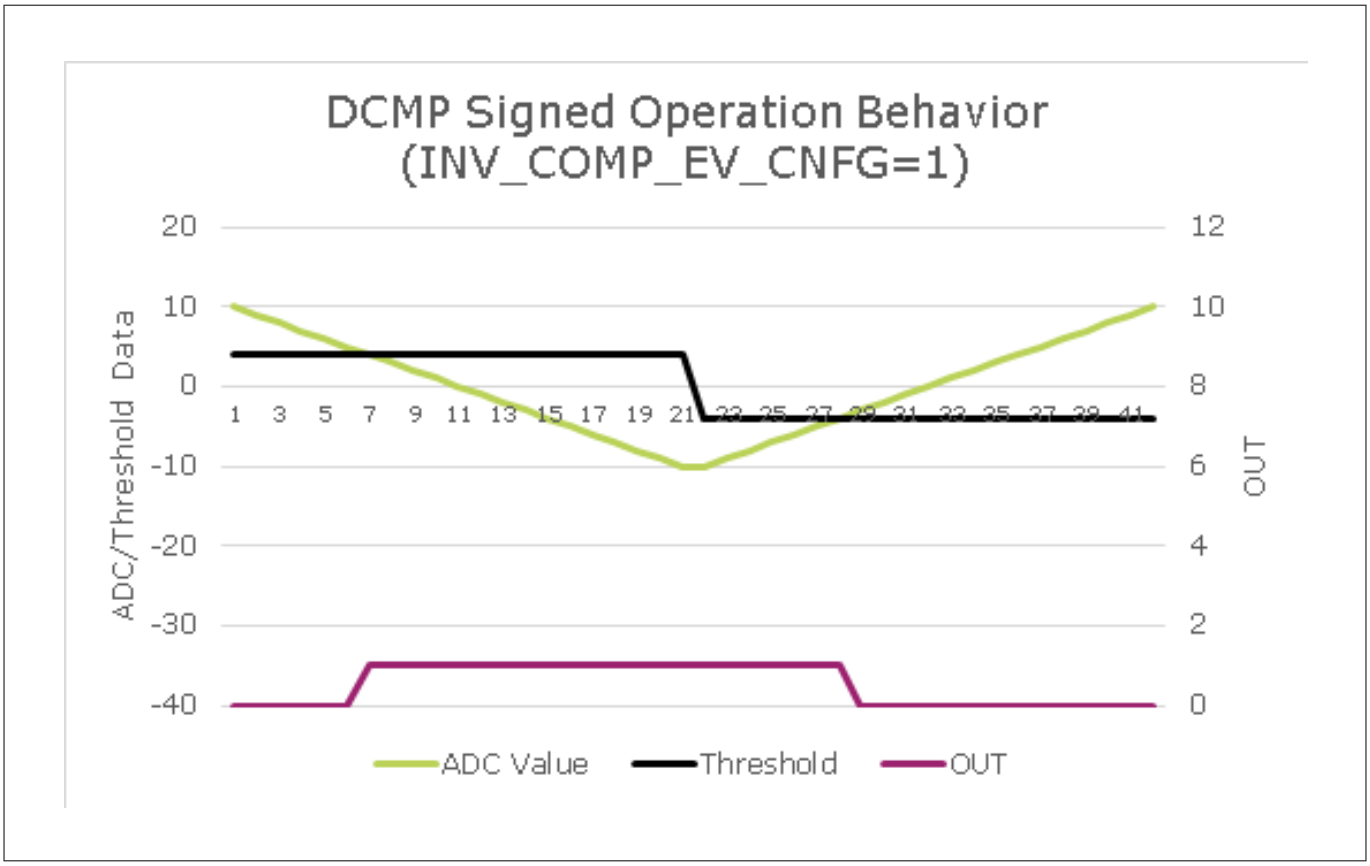


Figure 256 DCMP_OUT behavior in signed ADC data when INV_COMP_EV_CNFG = 1

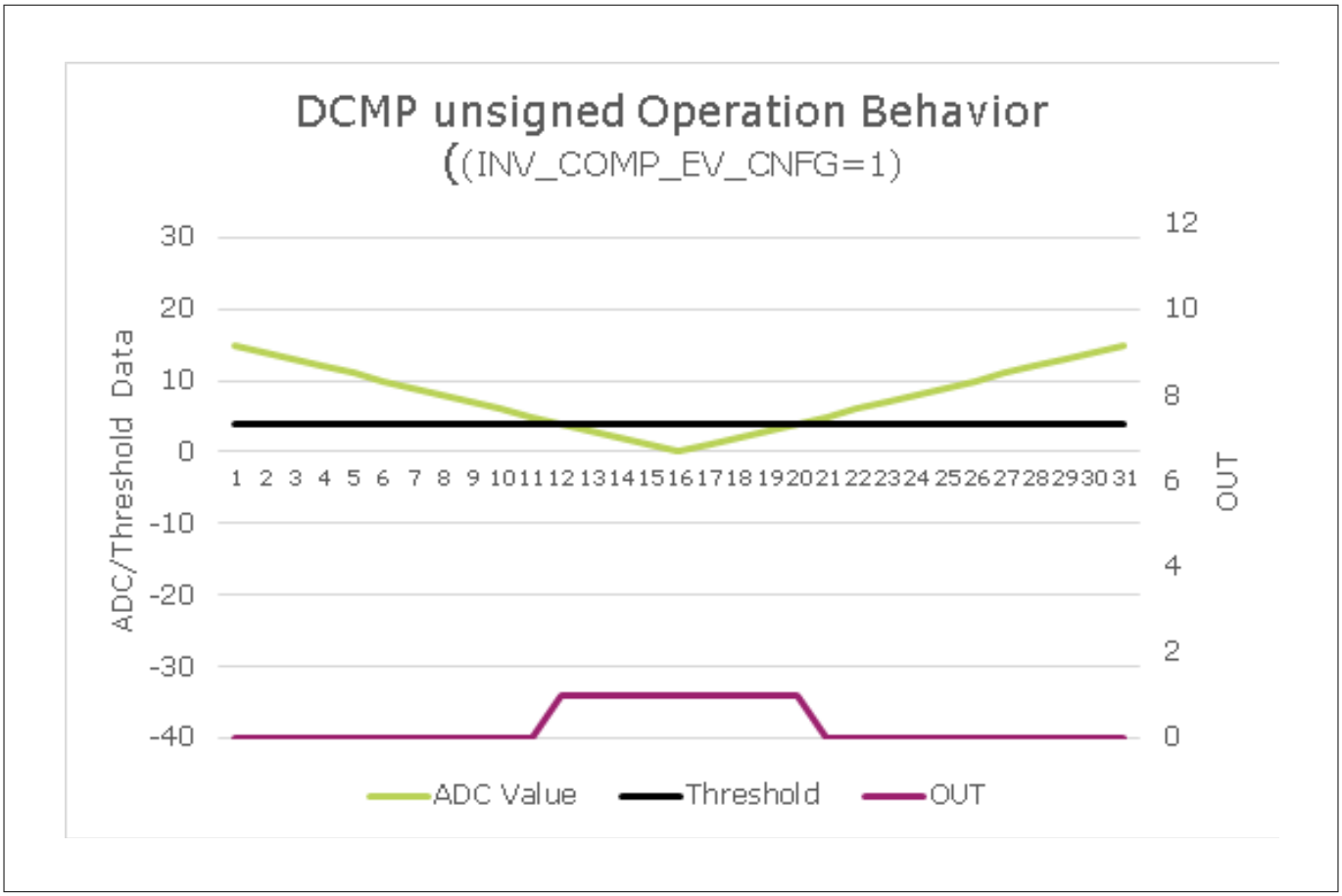


Figure 257 DCMP_OUT behavior in unsigned ADC data when INV_COMP_EV_CNFG = 1

30.3.13.2 Mode 1 operation - Accumulate

The Figure 258 shows the mode 1 operation behavior. MODE 1 Comp result is the accumulated result of the OUT. If OUT = 1, 1 will be added to MODE 1 Comp result, if OUT = 0, -1 will be added to MODE 1 Comp result. MODE 1 Comp result will be cap at 0 or 255 in the summation operation. MODE 1 DCMP result will be 1 if the MODE 1 Comp result is greater than DCMP_ACC_TH. DCMP_TRIGGER.DCMP_TRIG/DCPM_TRIG[] is used to reset the MODE 1 Comp result and start the comparison operation.

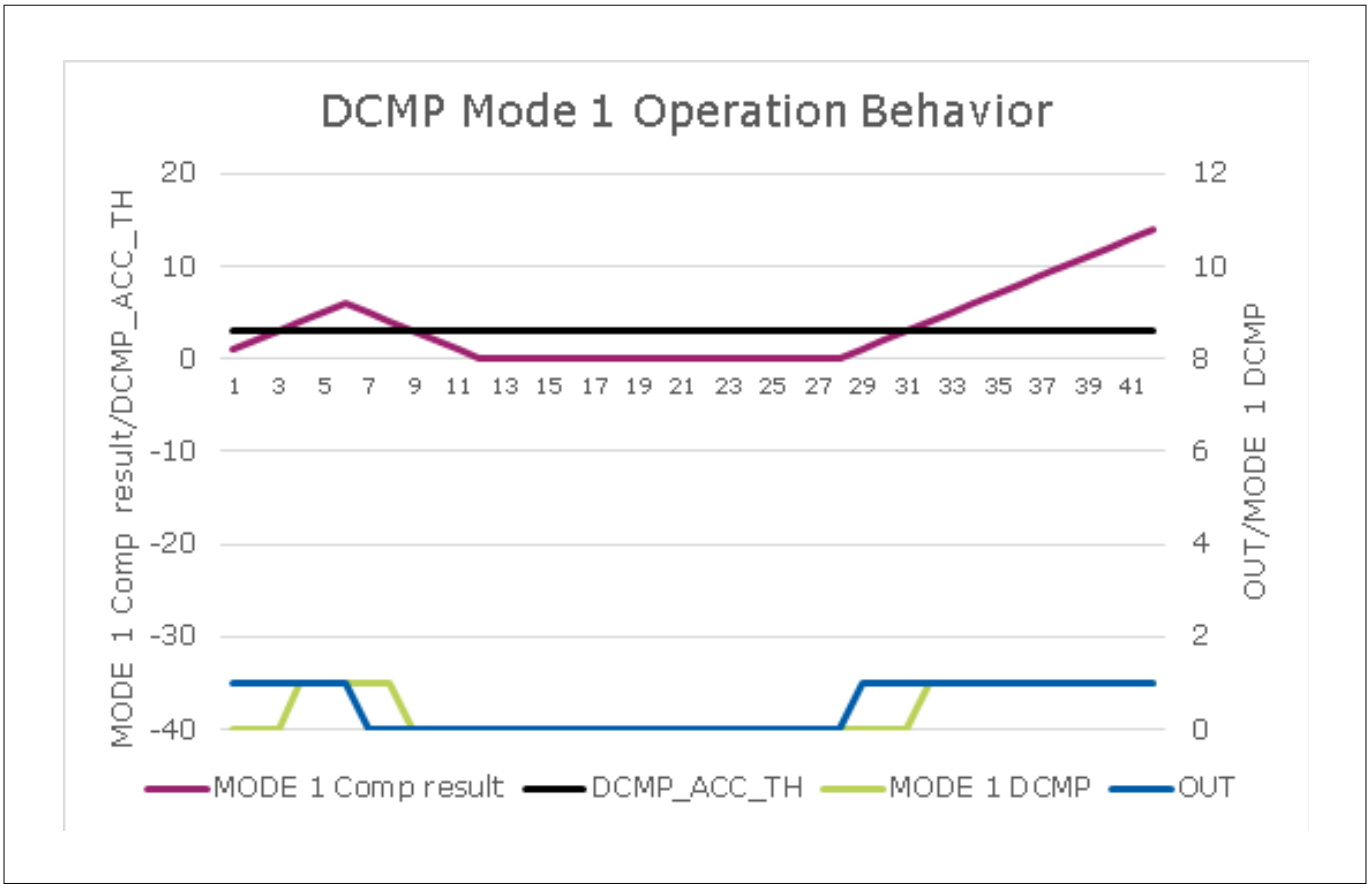


Figure 258 DCMP mode 1 operation

30.3.13.3 Mode 2 operation - Increment

The Figure 259 shows the mode 2 operation behavior. MODE 2 Comp result is the accumulated result of the OUT. If OUT = 1, 1 will be added to MODE 2 Comp result, if OUT = 0, 0 will be added to MODE 2 Comp result. MODE 2 Comp result will be capped at 0 or 255 in the summation operation. MODE 2 DCMP result will be 1 if the MODE 2 Comp result is greater than DCMP_ACC_TH. DCMP_TRIGGER.DCMP_TRIG/DCMP_TRIG[] is used to reset the MODE 2 Comp result and start the comparison operation.

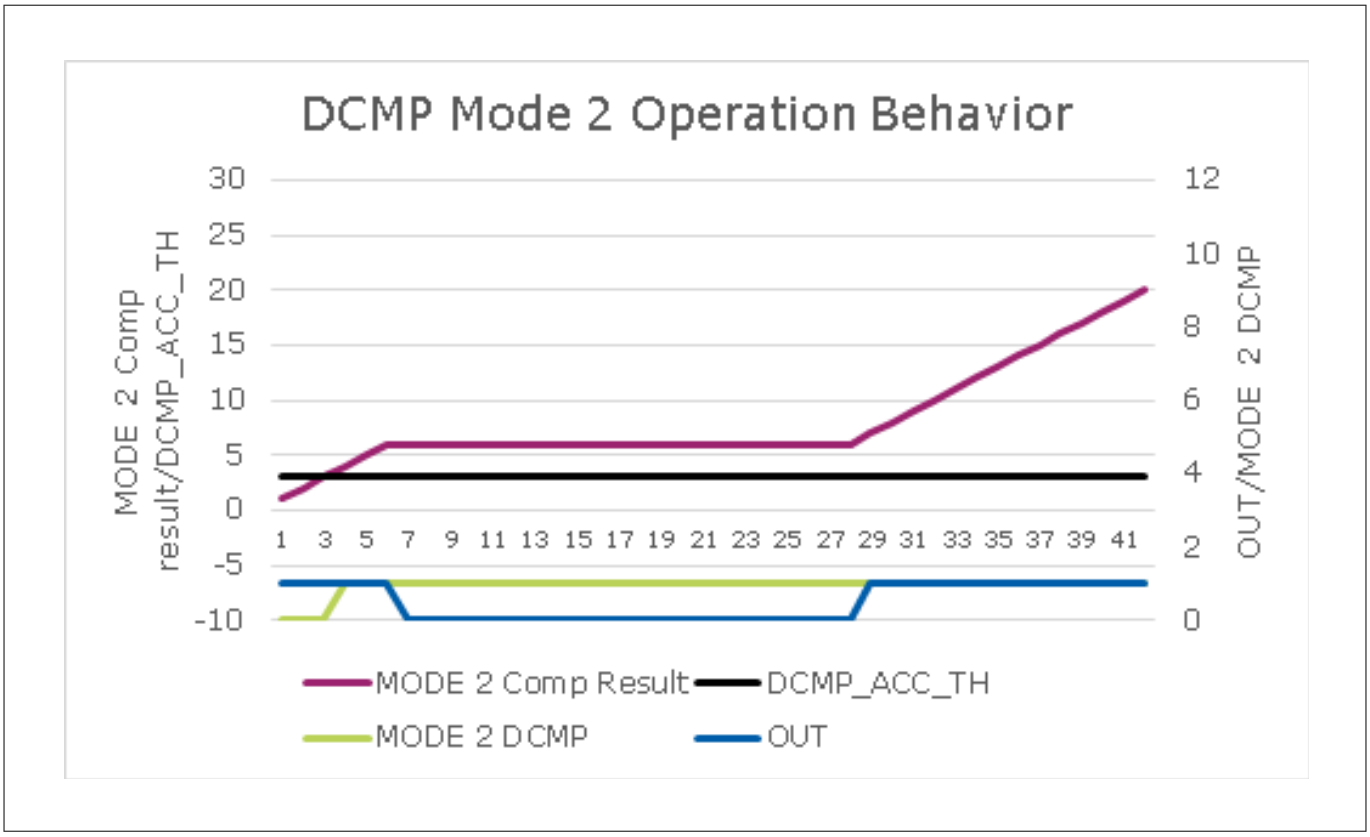


Figure 259 DCMP mode 2 operation

30.3.13.4 Mode 3 operation

The Table 270 shows the mode 3 operation behavior. MODE 3 operation will only start upon DCMP_TRIGGER.DCMP_TRIG (mmio register) or DCMP_TRIG[] signal. Upon trigger, the average comparison will start. It will stop when the window is reached. The operation is based on the sample window that is defined in DCMP_CNFG.SAMPLE. If the user changes the DCMP_CNFG.SAMPLE when DCMP_EN is set to '1', then the user must issue DCMP_TRIGGER.DCMP_TRIG or DCMP_TRIG[] (External trigger).

Table 270 Mode 3 example when DCMP_CNFG.SAMPLE = 0 (4 points)

No	ADC DATA	Internal computation	Remark
0	10	0	
1	20	20/4	Trigger by DCMP_TRIGGER.DCMP_TRIG/DCMP_TRIG[]
2	30	50/4	
3	26	76/4	
4	24	100/4	Upon computation, the result will be compared with DCMP_TH to reflect DCMP_EV.

(table continues...)

Table 270 (continued) Mode 3 example when DCMP_CNFG.SAMPLE = 0 (4 points)

No	ADC DATA	Internal computation	Remark
5	26	100/4	Internal computation maintained at 100/4 even though there is new ADC data
6	32	32/4	Trigger by DCMP_TRIGGER.DCMP_TRIG//DCMP_TRIG[]
7	22	54/4	
8	32	86/4	
9	12	98/4	Upon computation, the result will be compared with DCMP_TH to reflect DCMP_EV.

30.3.14 Digital filter

ADC_FILTS modules are part of the ADC_GRP_IF. The number of ADC_FILTS available in each ADC_GRP_IF is summarized in the Table 271.

Table 271 Summaries of ADC_FILTS per ADC_GRP_IF

ADC_GRP_IF	No of ADC_FILTS modules
ADC_GRP_IF_0	2
ADC_GRP_IF_1	1
ADC_GRP_IF_2	1
ADC_GRP_IF_3	2

The Table 272 illustrates the ADC filter mapping to which ADC in the ADC_GRP.

The abbreviation of the naming of the ADC Filter, ADC_FILT XY, where ‘X’ denotes which ADC where the filter is connected to, and ‘Y’ denotes the filter number.

Table 272 ADC_FILT connection mapping

ADC Filter	Mapping to ADC	Remarks
ADC_FILT_00	ADC_GRP0_ADC	Can be connected only to Group 0 ADC
ADC_FILT_01	ADC_GRP0_ADC	Can be connected only to Group 0 ADC
ADC_FILT_10	ADC_GRP1_ADC	Can be connected only to Group 1 ADC
ADC_FILT_20	ADC_GRP2_ADC	Can be connected only to Group 2 ADC

(table continues...)

Table 272 (continued) ADC_FILT connection mapping

ADC Filter	Mapping to ADC	Remarks
ADC_FILT_30	ADC_GRP3_ADC ADC_GRP2_ADC	Depending on the AFLT_CNFG.SEL_SRC_ADC_FILT bit, can be connected to ADC Group 2 or 3
ADC_FILT_31	ADC_GRP3_ADC ADC_GRP1_ADC	Depending on the AFLT_CNFG.SEL_SRC_ADC_FILT bit, can be connected to ADC Group 1 or 3

The block diagram of the ADC_FILTS is shown in Figure 260. The purpose of the ADC_FILTs modules are to support local ADC data filtering, mainly when ADC is configured to support a fast sampling rate. The ADC Filters are embedded in ADC_GRP_IF module.

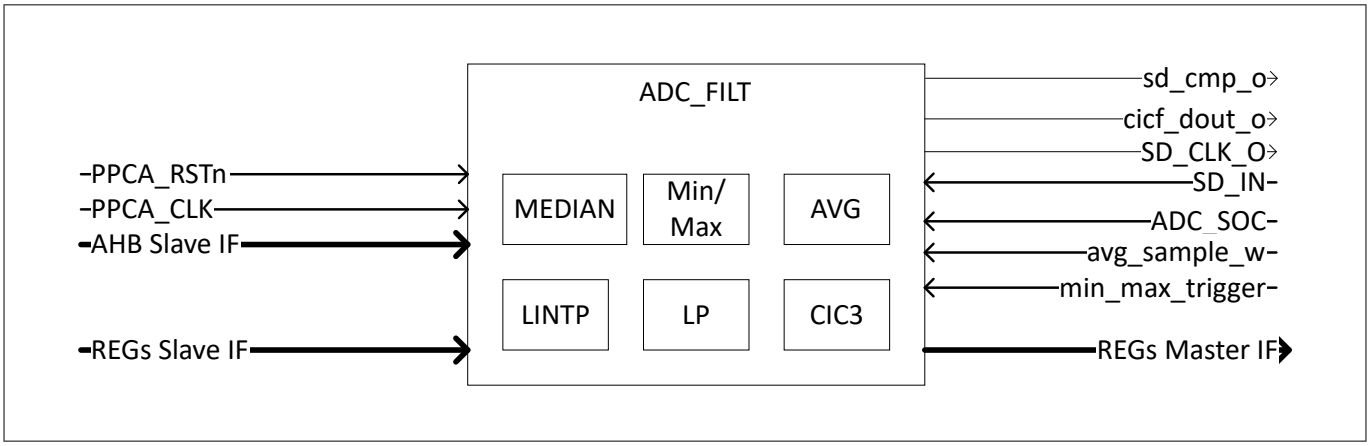


Figure 260 Block diagram of ADC_FILT

The ADC_FILT modules features:

- Median Filter (three sample window)
- Linear Interpolation Filter (LINTP)
- Low Pass Filter (LP),
- Cascade Integrator 3rd order Comb filter (CIC3),
- Configurable Average Filter (AVG)
- Configuration register (CNFG)
- Detection of minimum of the input data stream
- Detection of maximum of the input data stream

Two ADC_FILT modules are available with ADC_0 and ADC_3 and one with ADC 1 and ADC 2. The ADC_FILT can be connected to one of the available ADC converted data channels. The selection is done through the configuration register (AFLT_CNFG.SEL_SRC_CH). All the filters defined in ADC_FILT works with ADC signed data (User needs to set ADC_SIGN_UNSIGN_CNFG, AUX_SIGN_UNSIGN_CNFG, AUX_ALT_SIGN_UNSIGN_CFG register to signed).

***Note**: The user should avoid disabling and enabling of different filters at the same write cycle.*

Figure 261 shows the filters chain configuration in the ADC_FILT module. Users can select the type of filter through the AFLT.AFLT_CNFG register. Table 273 tabulates the filter configuration. The MIN/MAX module will be sourced by the input or output of the Median Filter.

For MEDIAN, LINTP and LP Filter, the output data is less than 32-bit. DATA_O is the ADC_FILT output and it is left aligned and padded the unused bits with 0 as the output of the MEDIAN, LINTP, and LP filter are routed to DATA_O and DATA_O is of 32-bit width. For the case of CIC and AVG filter, the 32-bit data will be routed to DATA_O. For AHB reading of the data of the filter, please refer Filter result format for the data format.

For the filters ADC_FILT_00, ADC_FILT_01, ADC_FILT_10 and ADC_FILT_20 the data is coming from the same ADC_GRP. For the filters ADC_FILT_30 and ADC_FILT_31 option is available to select data from another ADC_GRP also. The firmware designer needs to configure the data path according to the application need.

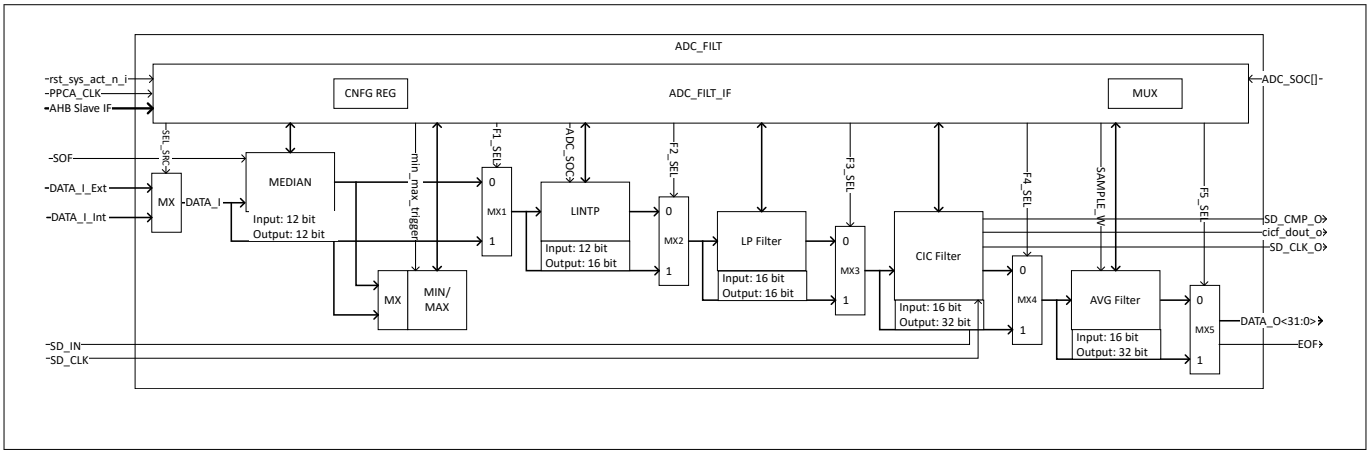


Figure 261 Filter chain configuration in ADC_FILT

Table 273 Filter chain configuration table

Output	F1_SEL		F2_SEL		F3_SEL		F4_SEL		F5_SEL	
	0	1	0	1	0	1	0	1	0	1
MX1_OUT	MEDIAN	INPUT								
MX2_OUT			LINTP	MX1OUT						
MX3_OUT					LPF	MX2OUT				
MX4_OUT							CIC	MX3OUT		
MX5_OUT									AVG	MX4OUT

User can make use of the MX1 to MX5 to bypass the data to different filter block in the filter chain. Please ensure that at least one of the filters is enabled in the filter module if the user wants to bypass the entire filter chain to send data from ADC to the next peripheral if a dedicated datapath is used. The Figure 262 illustrates an example bit mapping of the input of the average filter from a 12-bit bit width source when user set '1' to the following register bit AFLT_CNFG.SEL_MEDIAN, AFLT_CNFG.SEL_LIF, ALFT_CNFG.SEL_LPF, AFLT_CNFG.SEL_CIC. This is applicable to LP and CIC filter where the input of the filter is 16-bit width.

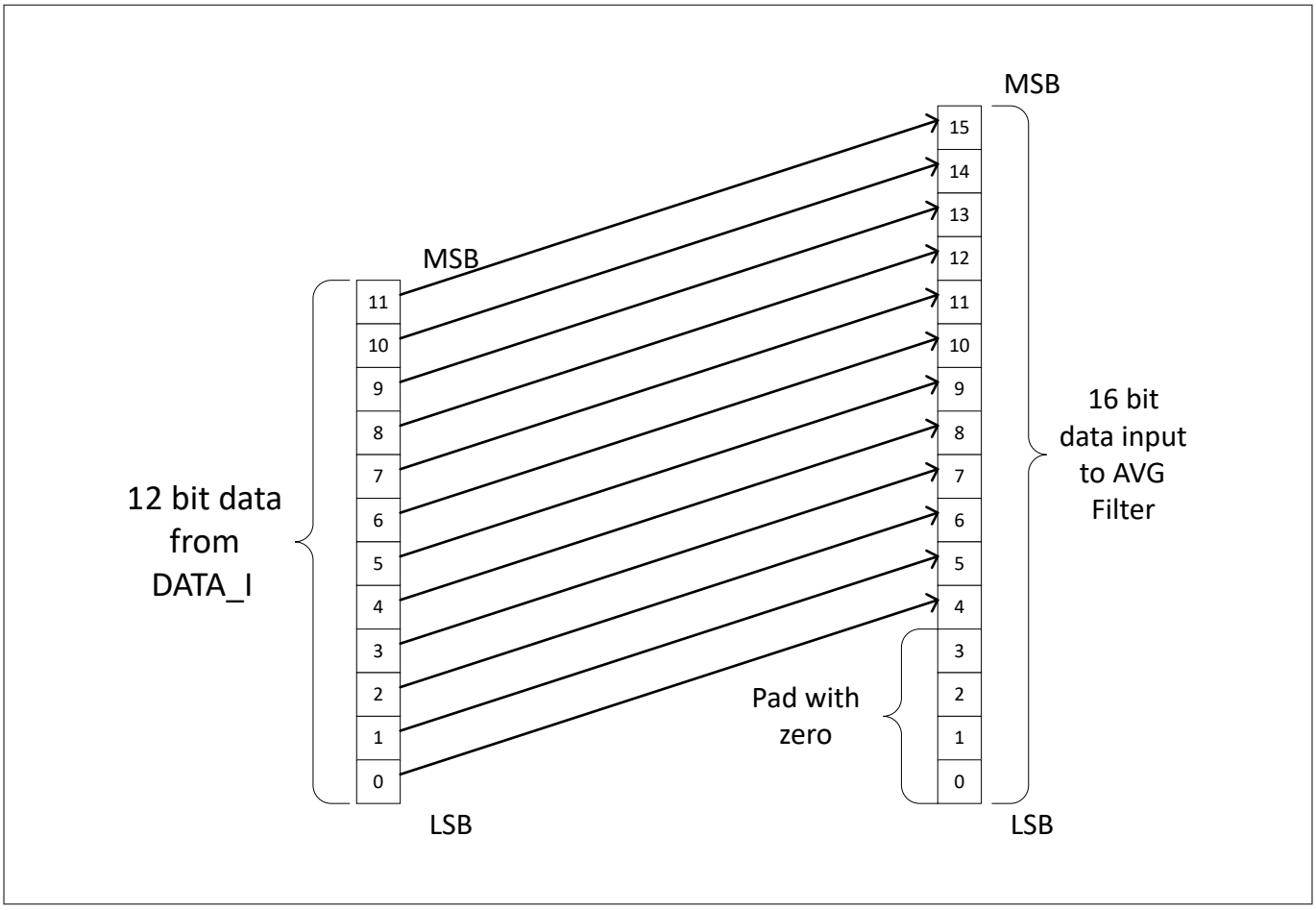


Figure 262 Bit width mapping of 16-bit width filter input from 12-bit data source

30.3.14.1 Median filter

The block diagram of the median filter is shown in Figure 263. The Median Filter removes the noise (glitch) from the sample stream. The median filter works on the input data stream over a window of three samples. The median filter extracts the middle value of the three samples. The median filter only supports a window of three samples.

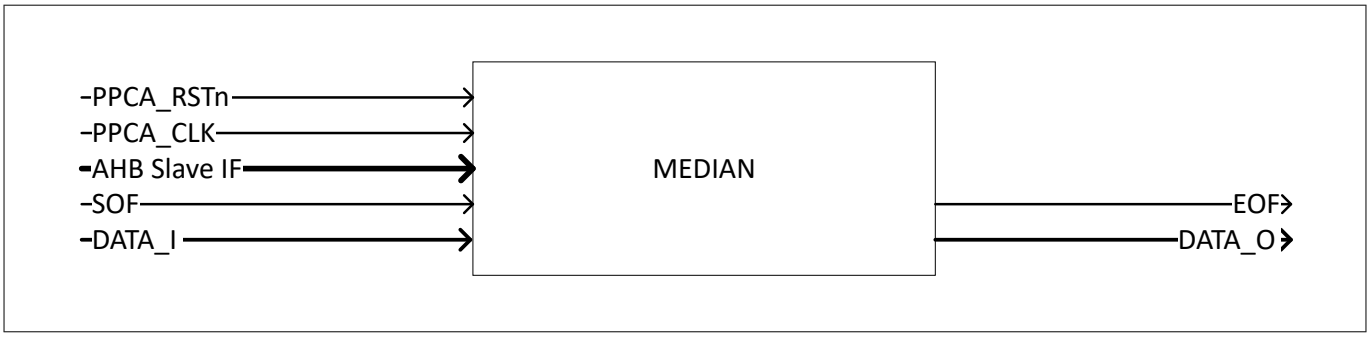


Figure 263 Block diagram of median filter

$$\begin{aligned} y(0) &= x(0) \\ y(1) &= median(x(0), x(0), x(1)) \\ y(n) &= median(x(n-2), x(n-1), x(n)) \end{aligned} \tag{6}$$

Figure 264 illustrate the behavior of the median filter where the window is set to 3 and the expected output y(n) based on a set of x(n).

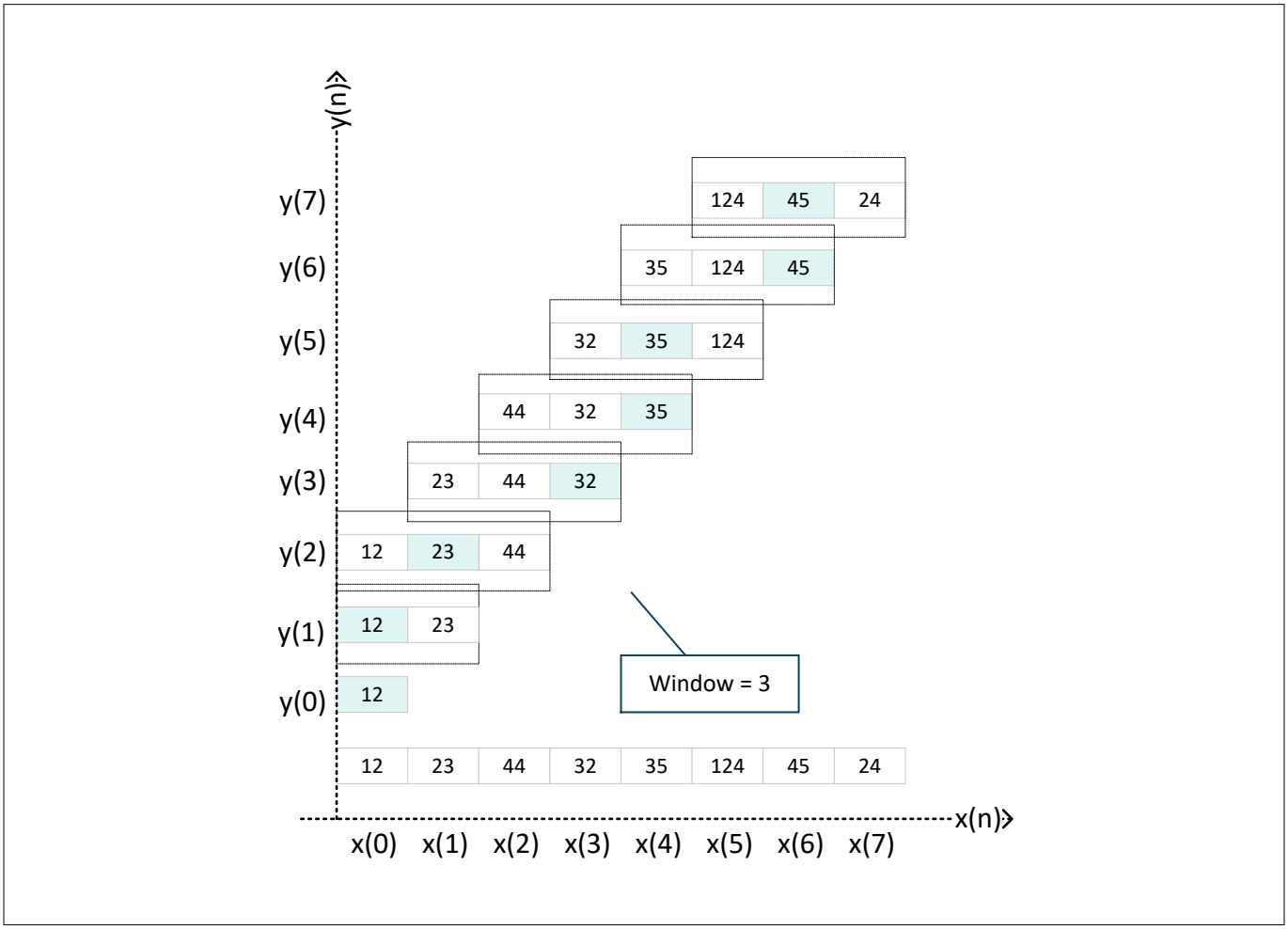


Figure 264 Behavior of median filter

30.3.14.2 Linear interpolator filter

The block diagram of the Linear Interpolator Filter (LINTP) is shown in Figure 265. The linear interpolator filter is used to recover the phase delay between the sampling point and the provided ADC trigger when ADC is configured to work in a continuous sampling mode.

The linear interpolator filter features:

- 12-bit input (the output of the 12-bit ADCs or MEDIAN Filter)
- 16-bit output (where the last four LSB bits are the fractional bit)
- Max 2 clock cycles execution latency upon ADC data is ready
- AHB slave interface (for filter reset, control and for data result read)
- Register slave interface (i.e. Data input bus, DATA_I, synched to SOF signal)
- Register master interface (i.e. processed Data Output bus, DATA_O synched to EOF)
- INTP_EXTP_SEL to switch between interpolator and extrapolator computation in the filter

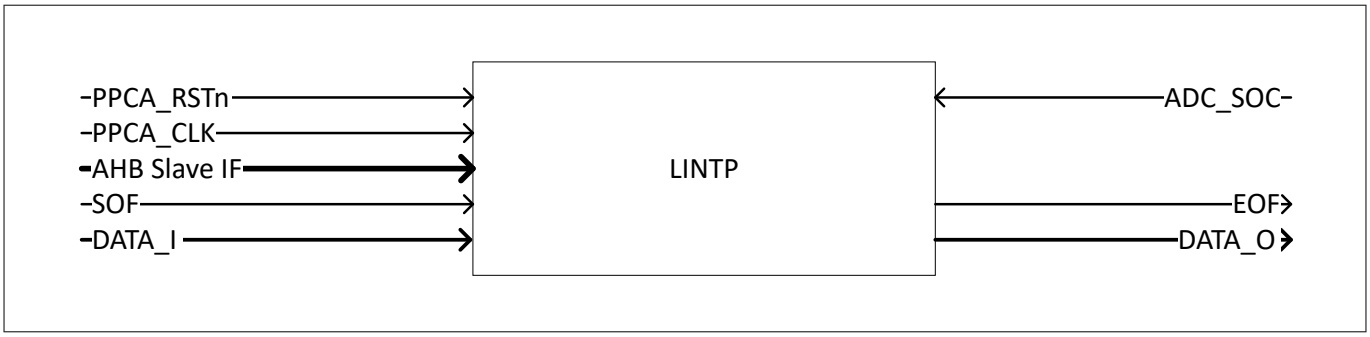


Figure 265 Block diagram of LINTP filter

When LIF_CIFG.INTP_EXTP_SEL is set to 0, the equation for the LINTP filter is shown in Figure 266.

$$y(trigger) = Fract_{Phase} * (y(n+1) - y(n)) + y(n)$$
$$Fract_{Phase} = \frac{M_TRIGGER_POINT}{2^{(M_FACTOR+1)}}$$

Figure 266 LINTP interpolator

M_TRIGGER_POINT is the location of the interpolation trigger and 2^(M_FACTOR+1) defines the maximum number of clock cycles between start of ADC conversion. Please be aware that 2^(M_FACTOR+1) support ranges from 2 to 256.

If 2^(M_FACTOR+1) is less than M_TRIGGER_POINT, y(trigger) will be equal to y(n+1). When LIF_CIFG.INTP_EXTP_SEL is set to 1, the equation for the LINTP filter is shown in Figure 267:

$$y(trigger) = Fract_{Phase} * (y(n+1) - y(n)) + y(n+1)$$
$$Fract_{Phase} = \frac{M_TRIGGER_POINT}{2^{(M_FACTOR+1)}}$$

Figure 267 LINTP extrapolator

The design will update the location of the interpolation trigger, x(trigger) to the LIF_CNFG.M_TRIGGER_POINT.

The linear Interpolator filter diagram is depicted in Figure 268:

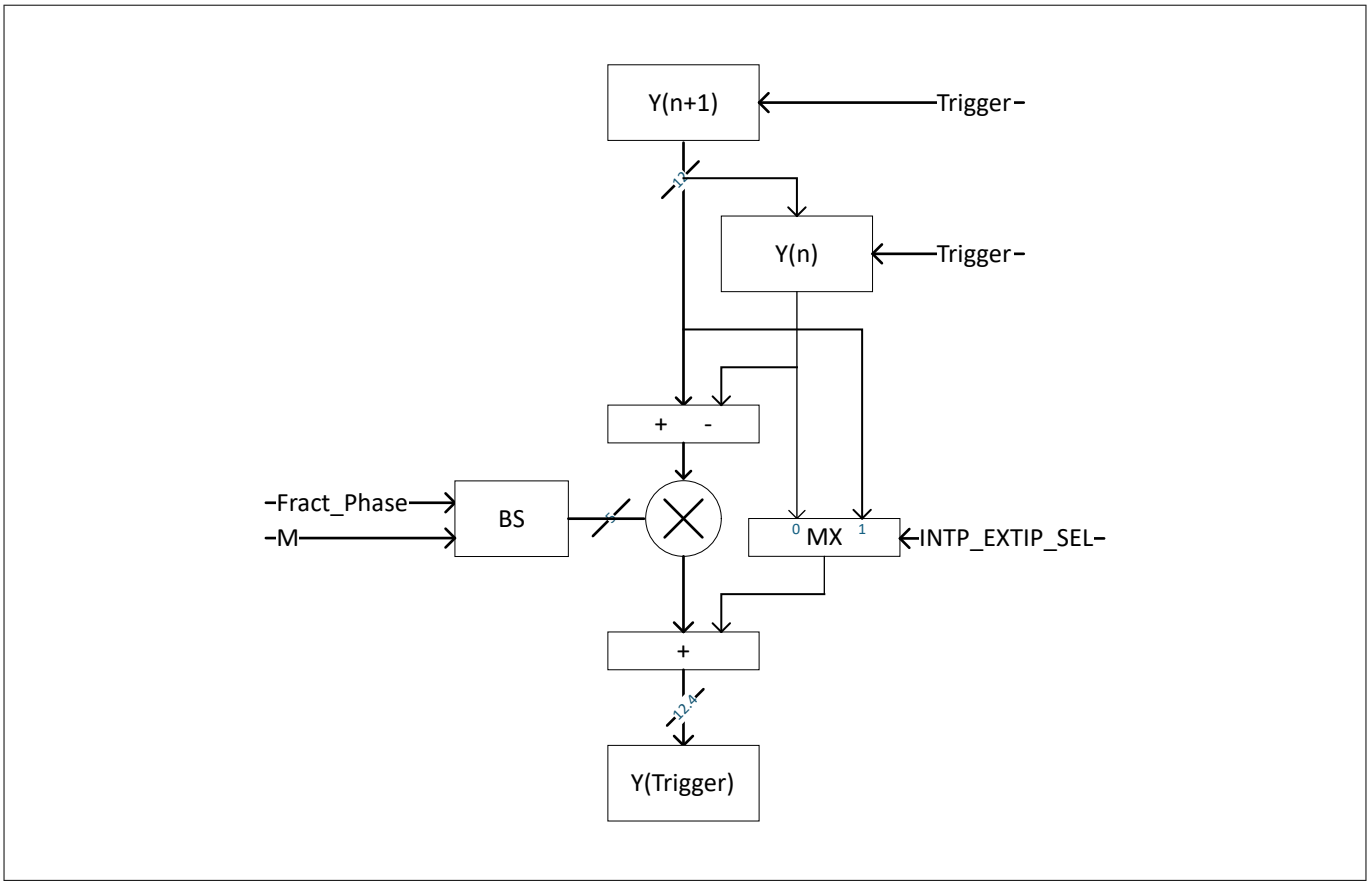


Figure 268 LINTP filter diagram

Linear interpolation and extrapolator filter operation diagram is shown in Figure 269 and Figure 270.

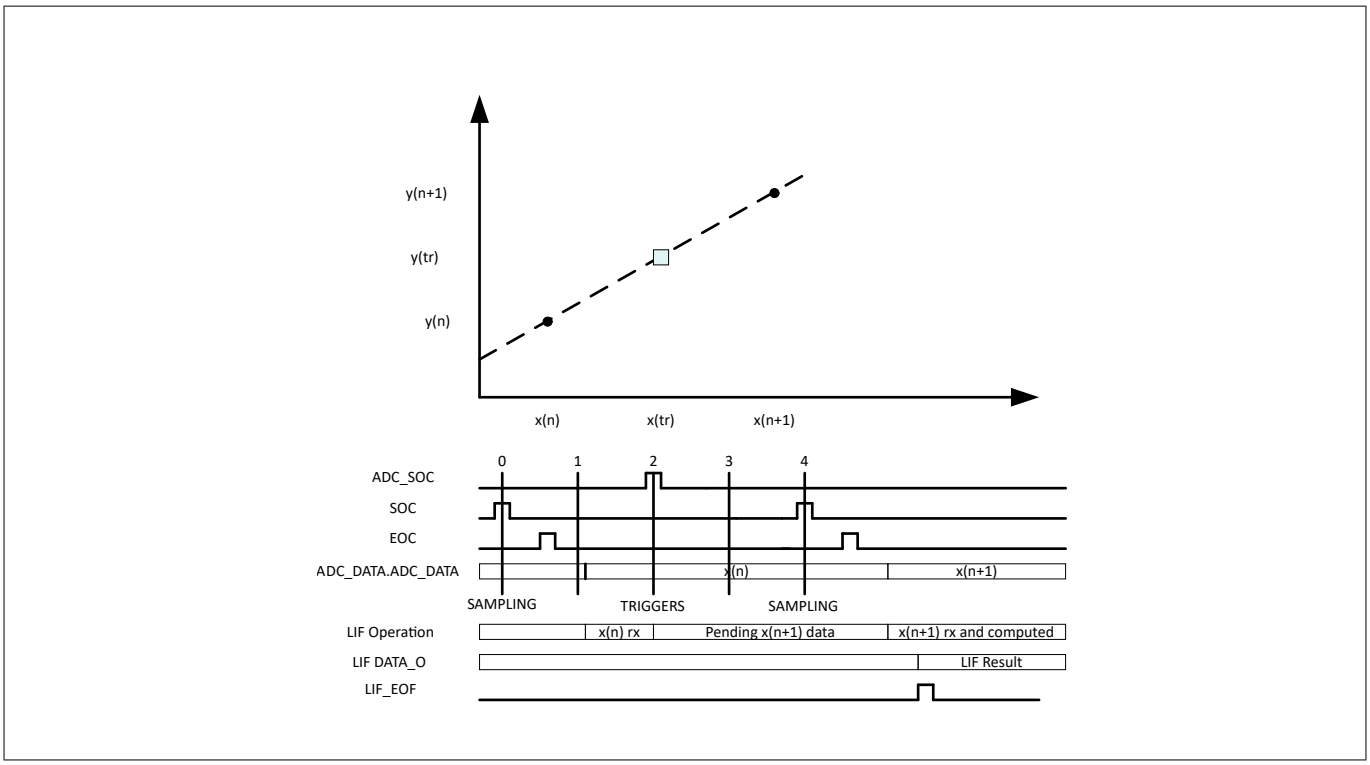


Figure 269 LINTP interpolation filter operation diagram

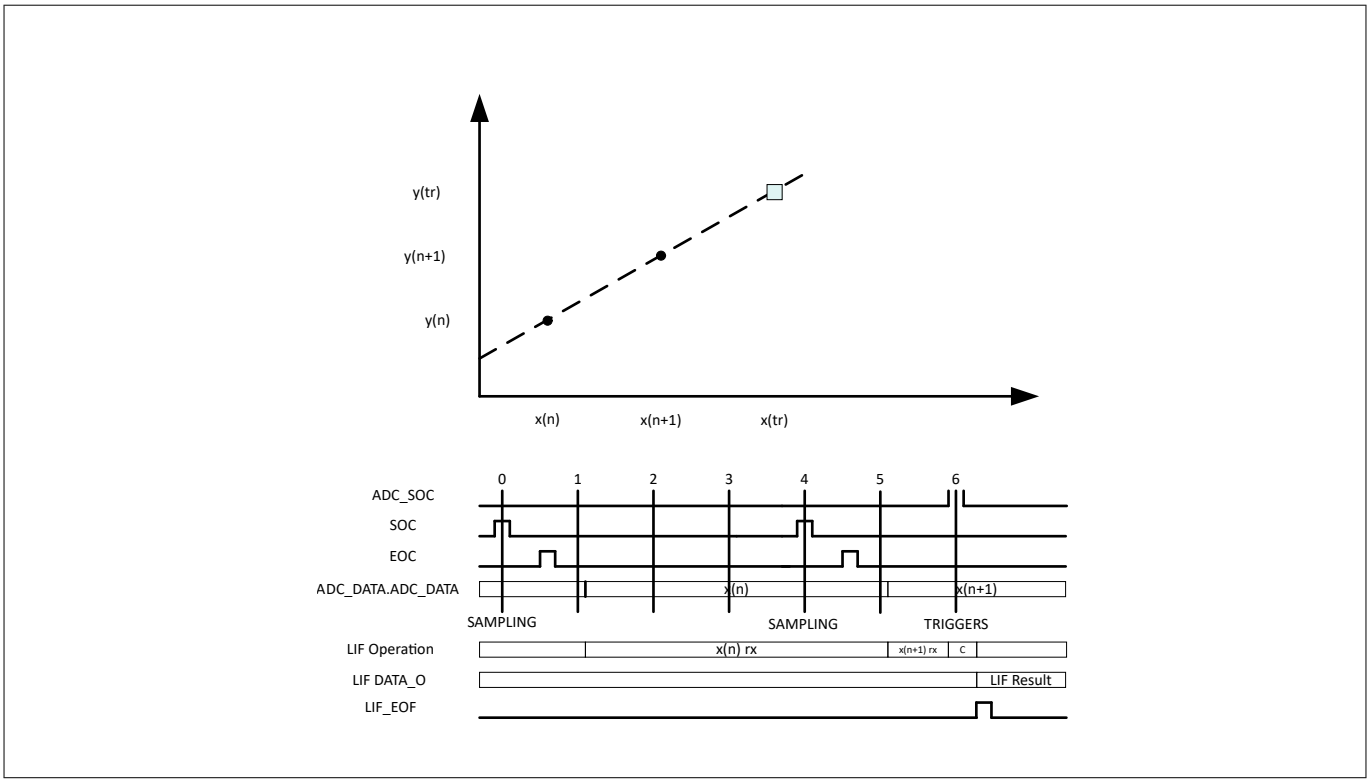


Figure 270 LINTP extrapolation filter operation diagram

Figure 271 and Figure 272 illustrates the behavior of the LINTP based filter where the sampling and trigger impact on the LINTP filter under interpolator operation. The sampling is triggered from the FSM that starts the SOC and the TRIGGER is from the ADC_SOC. This is based on the auto-trigger mode. For manual trigger mode where ADC_CNV_CNFG.SRC_EOS bit is configured as external EOS, it is not possible to have independent sampling and trigger for the LINTP as it will be coming from the same source, ADC_SOC. For arbitrary trigger mode, it is not possible to have independent sampling and trigger for the LINTP as ADC_SOC is used to request for ADC conversion.

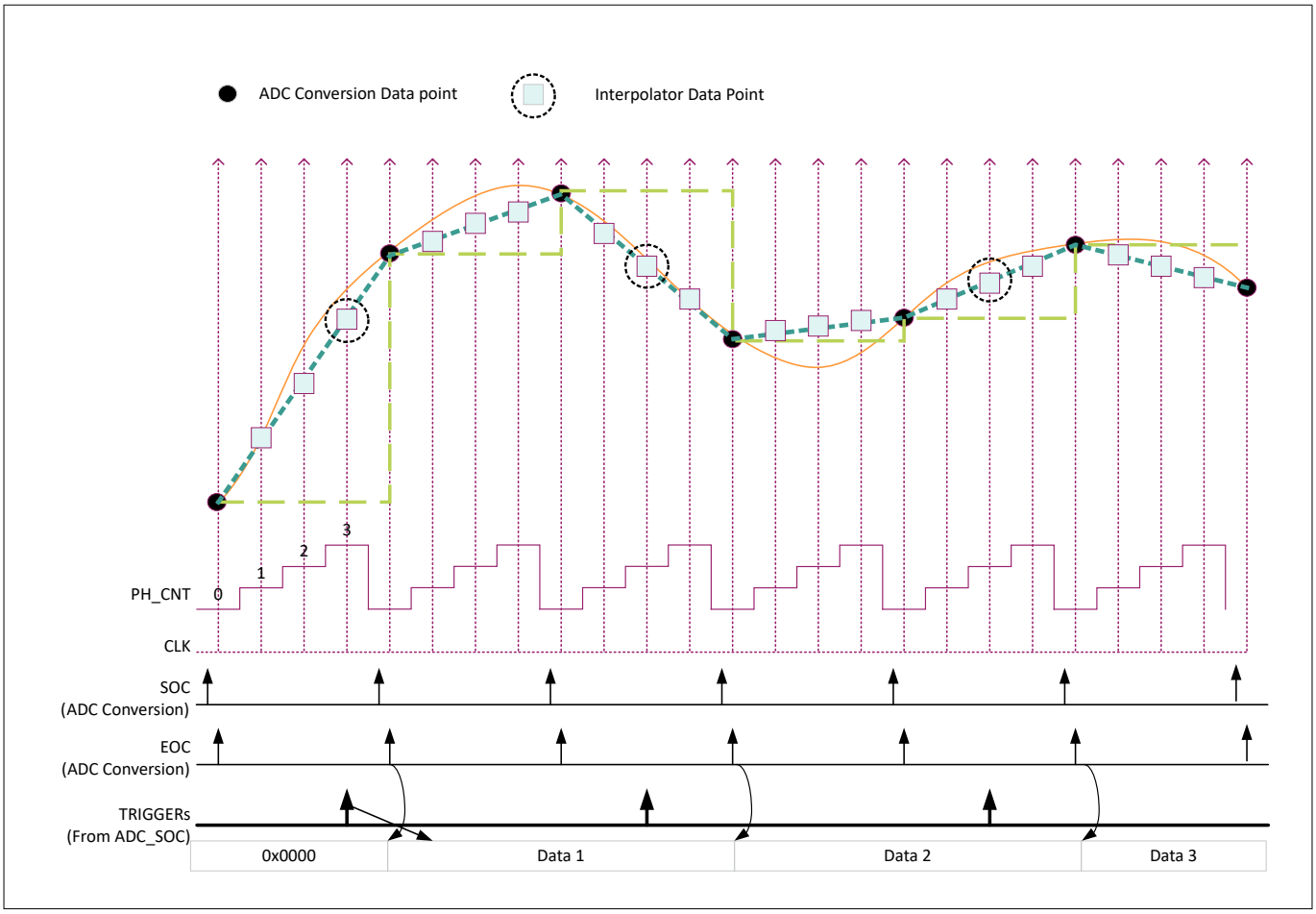


Figure 271 Behavior of LINTP filter

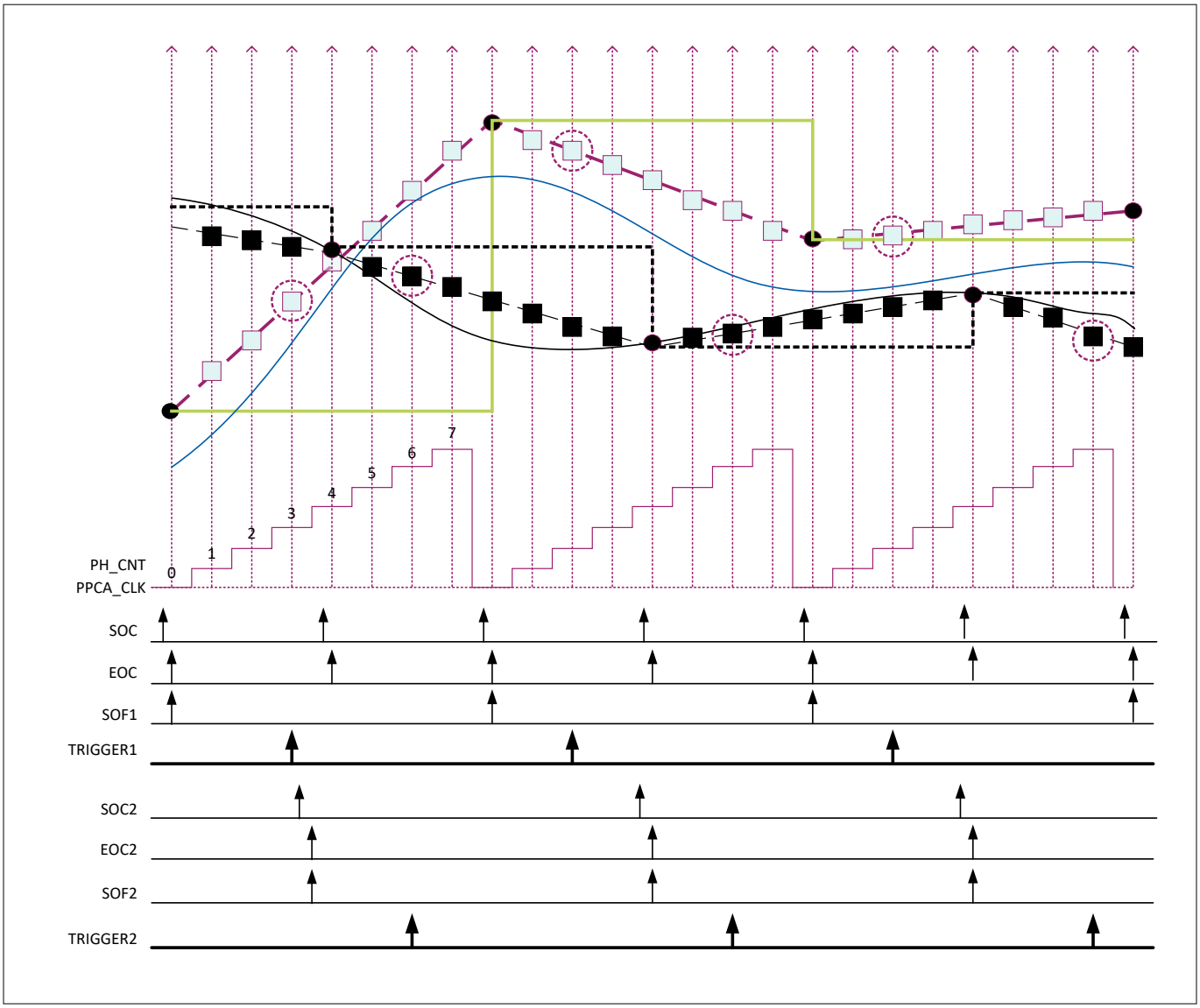


Figure 272 Behavior of LINTP filter with 2 ADC

The output of the LINTP Filter is aligned to the SOF. In the corner case where sampling triggers when the first ADC conversion has not been completed, the LINTP filter will compute based on the first ADC conversion as x(n+1) and x(n) = 0.

30.3.14.3 Low pass filter

The block diagram of the low pass filter (LP) is shown in Figure 273. The low pass filter features:

- 16-bit input/output (the output of the 12-bit ADCs left aligned or the output of the LINTP)
- 16-bit architecture
- Programmable cutoff frequency (through 8-bit coefficient register)
- Start of Filter (SOF), End of Filter (EOF)
- Max 2 clock cycles execution latency upon ADC data is ready
- Saturation operators supported
- AHB slave interface (for filter configuration, control and for data result read)

- Register slave interface (i.e. Data input bus, DATA_I, synched to SOF signal)
- Register master interface (i.e. processed Data Output bus, DATA_O synched to EOF)

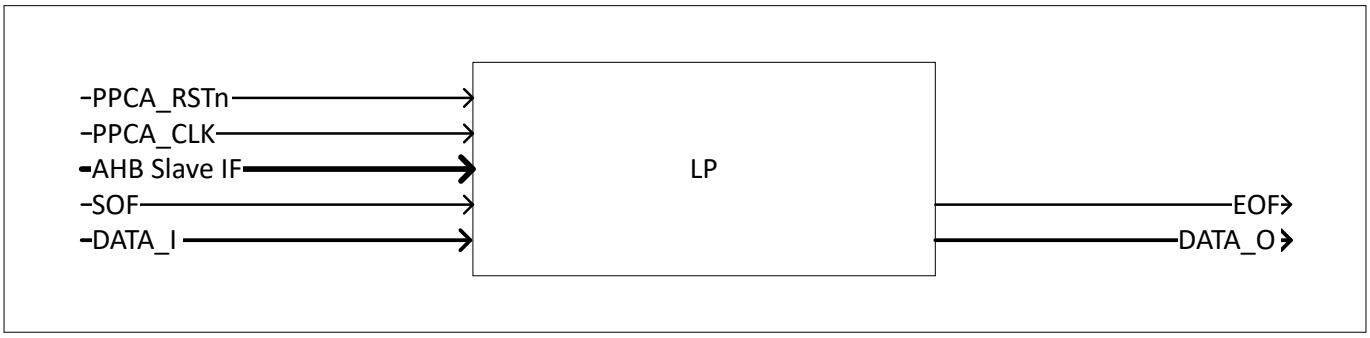


Figure 273 Block diagram of low pass filter

The equation for the low pass filter is shown in Figure 274:

$$y(n) = Fix\{q_acc(n),\ 2^{16}\}$$
$$\text{q_acc(n)} = \left(LPF_ALFA.BW * \left((x(n)\ <<\ 4) - y(n-1)\right)\right) + q_acc(n-1)$$

Figure 274 Low pass filter equation

The low pass filter block diagram is depicted in Figure 275:

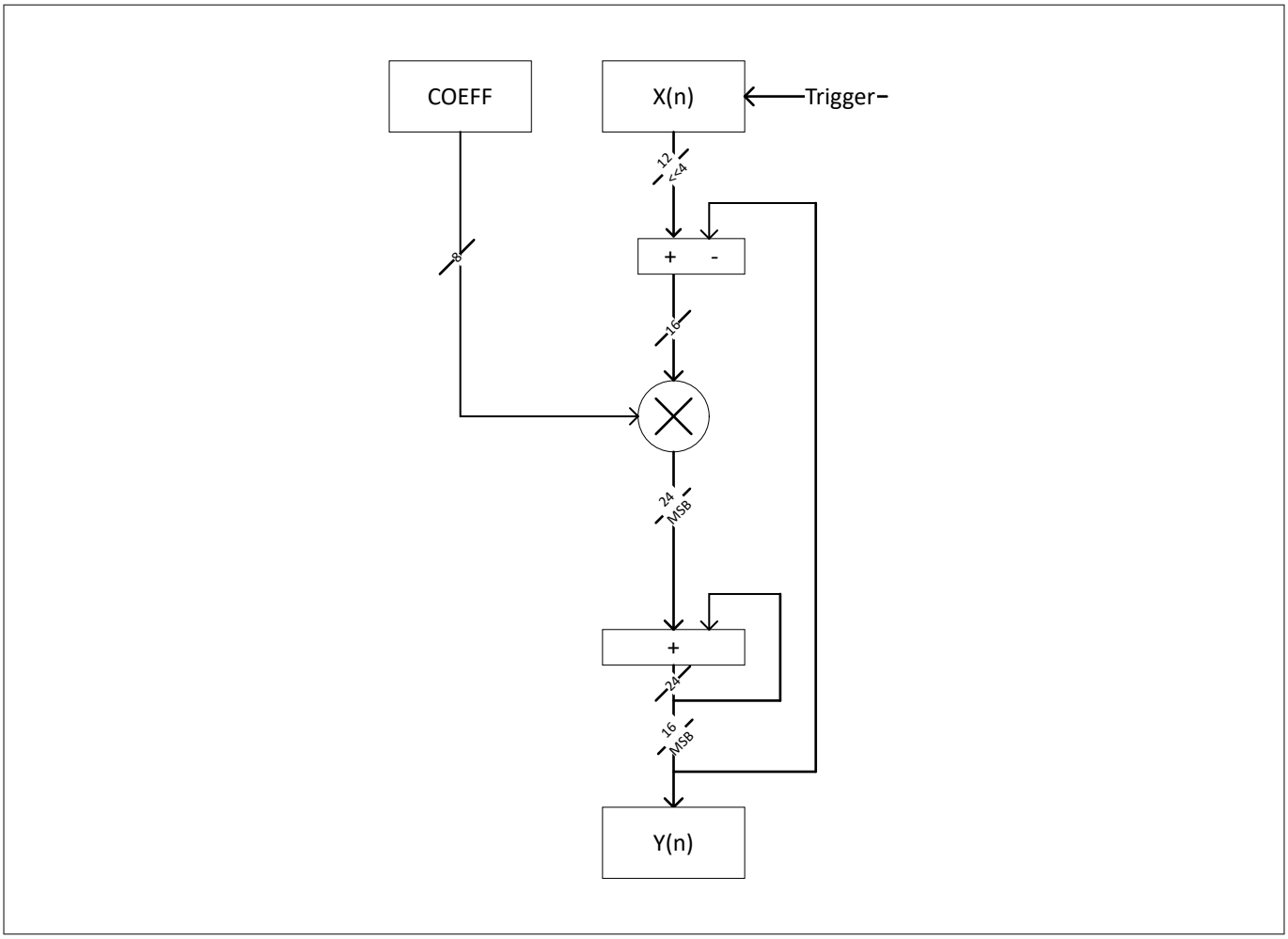


Figure 275 Low pass filter diagram

30.3.14.4 CIC3 filter

The block diagram of the CIC3 filter is shown in Figure 276. This module implements a Cascade Integrator Comb filter 3rd order. The CIC3 filter features:

- 16-bit input (the output of the 12-bit ADCs or the 16-bit LP Filter)
- 32-bit signed output
- Optional external input source (SD_IN, single bit) with selectable input sampling rate
- Configurable Decimation Factor (11-bit) supports from 4 to 2047
- Configurable Decimation Offset (related to a start trigger)
- Configurable Scaler (From 0 to 31)
- Start of Filter (SOF), End of Filter (EOF)
- AHB Slave Interface (for filter configuration, control and for data result read)
- Register Slave Interface (i.e. Data input bus, DATA_I, synched to SOF signal)
- Register Master Interface (i.e. processed Data Output bus, DATA_O synched to EOF)
- Digital comparator to compare CIC output to upper and lower threshold

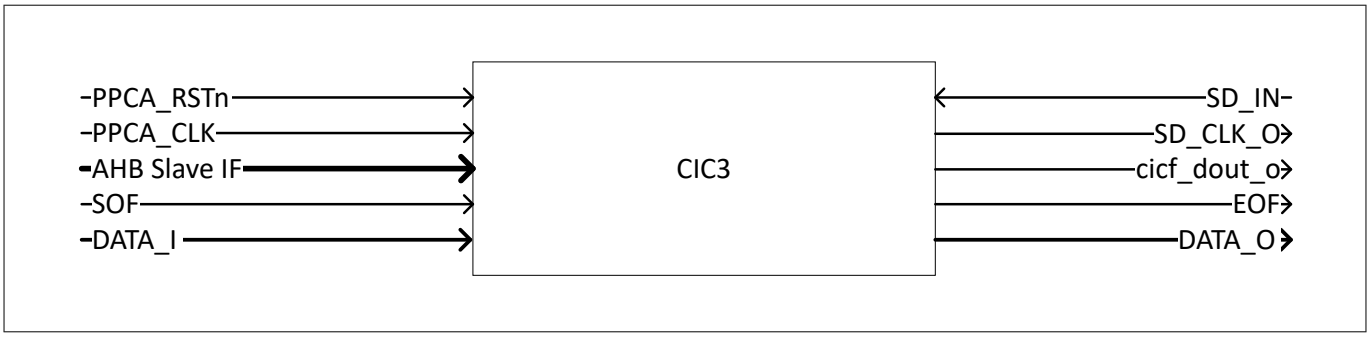


Figure 276 Block diagram of CIC3 filter

The equation for the CIC3 filter is shown in Figure 277.

$$H(z) = \left(\frac{1 - z^{-M}}{1 - z^{-1}}\right)^{N}$$

Where M = Decimation factor, N = 3

Figure 277 CIC3 filter equation

Table 274 CIC3 filter parameter

CIC3 filter parameter	Value	Description
N	3rd order	Order of the filter
M	4 to 2047	Decimation ratio, programmable from 4 up to 2047, that is for FS = 25 MHz, Fmin = FS/2047, Fmax=FS/4
Word_Width	34	Maximum word output size: For Bin =1, Bout=3*log2(2047) +1 = 34 For Bin =16, Bout=3*log2(64) +16 = 34 Bout = N Log_2(M) + Bin < 34
CIC_SCALE	0 to 31	CIC Filter Output Scale: CIC_SCALE = Word_Width – Bout

The CIC_CNFG.CICF_DEC_SCALE is implemented as a 5-bit value. Range from 0 – 31. It can be calculated as:

Scale (BIN16 case) = (3*log_2(CIC3_M_{MAX})+Bin) – (3*log2(CIC3_M)+Bin) where CIC3_M_{MAX} is 64 for BIN = 16

Scale (BIN1 case) = (3*log_2(CIC3_M_{MAX})+Bin) – (3*log2(CIC3_M)+Bin) where CIC3_M_{MAX} is 2047 for BIN = 1

Where CIC3_M is the programmed decimation ratio, CIC3_M $_{MAX}$ is the maximum decimation ratio depending on Bin and Bout are the number of bits on which the result will be mapped to CICF_DOUT.

The decimation ratio shall be selected as per Table 275.

Table 275 CIC3 decimation ratio support based on signal input width

SEL_SRC	Signal Input Width	Decimation Ratio (CIC_M)	Description
0	16	4 to 64	Allowed Decimation Ratio Range per 16-bit input signal
1	1	4 to 2047	Allowed Decimation Ratio Range per 1-bit input signal

The typical flow diagram implementing the CIC3 filter is depicted in Figure 278.

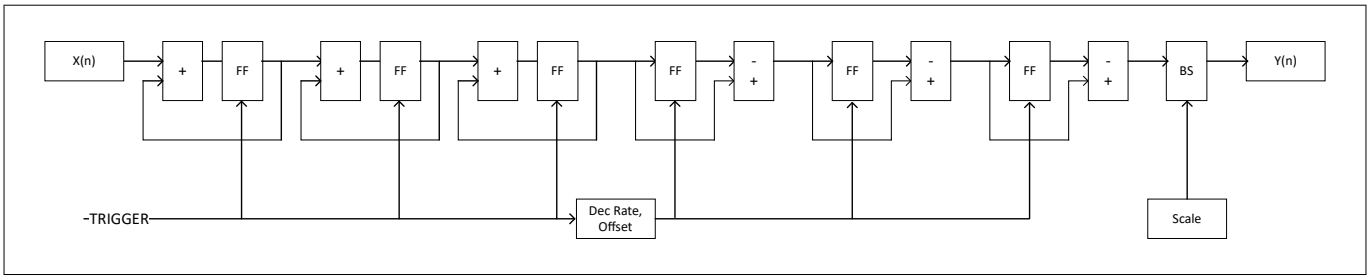


Figure 278 CIC3 filter flow diagram

To support Sigma-Delta demodulation for Isolated External Sensor device exporting a bit stream, the CIC Filter can be feed directly from external pins. The 0/1 based single bit stream is converted to a signed integer format by replacing 0 into a “-1” (2 complement) while 1 is unchanged. From this point (DATA_IN) all numbers are evaluated as signed integers.

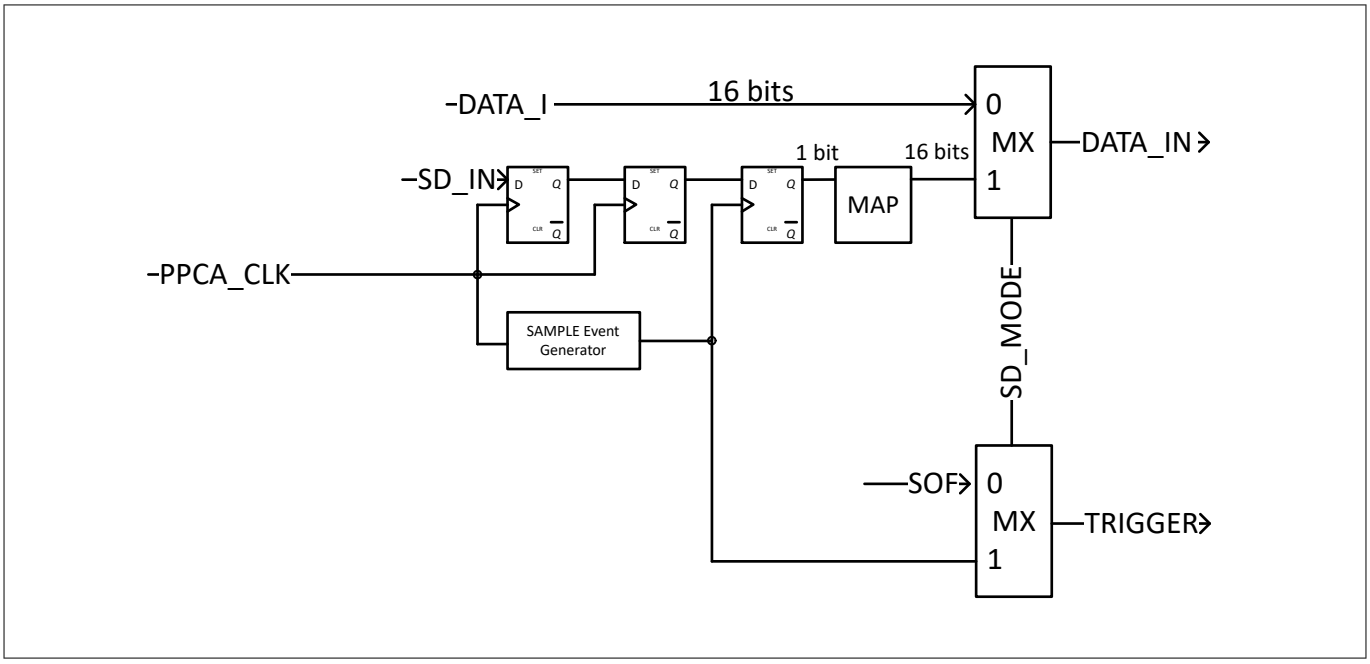


Figure 279 SD_IN handling diagram

30.3.14.5 CIC comparator

The CIC filter has a comparator that has two set of threshold registers to compare. The two set of registers are namely CICF_UB_TH (Upper Boundary THreshold) and CICF_LB_TH (Lower Boundary THreshold). The output result of the comparator can be connected to EPU.

The Table 276 defines the decision criteria for the CIC comparator. The application use case for the CIC comparator will work for decimation factor range from 4 to 32. Anything more than 32, the CICF_CMP_EN will be set to '0'.

Table 276 CIC comparator decision criteria

Decision Criteria	sd_cmp_o
CICF_LB_TH < CICF_DATA_O < CICF_UB_HT	0
CICF_LB_TH >= CICF_DATA_O >= CICF_UB_HT	1

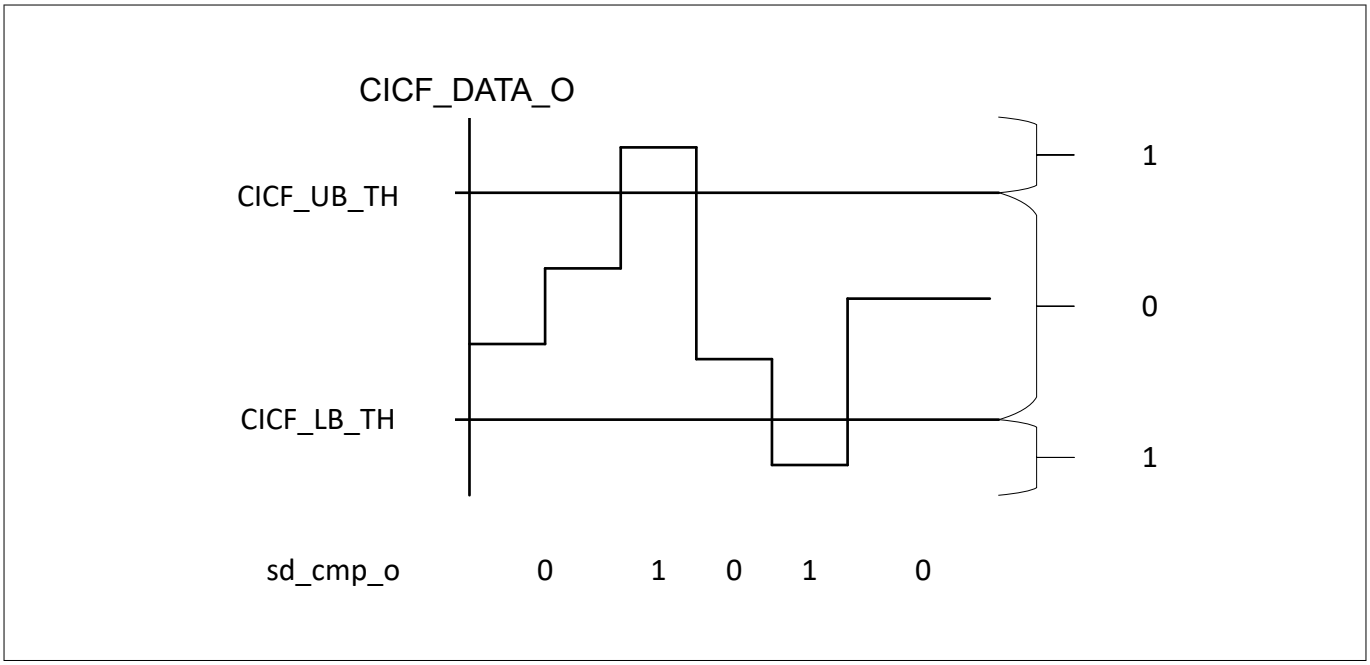


Figure 280 Illustration of sd_cmp_o behaviour

30.3.14.6 Average/Integrator filter

The block diagram of the AVG filter is shown in Figure 281. This module implements a configurable average/ integrator filter. The average computation is handled by the firmware. The subsystem will handle the accumulation operation. Here, the options are available to select between fixed or dynamic sample window, linear, or squared samples and the number of accumulated samples. It can be selected using the AVGF_CNFG register.

- 16-bit input (the most significant bits of the input bus)
- Configurable Accumulation Sample Window
- Configurable Weight
- Selectable Accumulation of input data samples or squared input data sample
- Start of Filter (SOF), End of Filter (EOF)
- AHB Slave Interface (for filter configuration, control and for data result read)
- Register Slave Interface (i.e. Data input bus, DATA_I, synched to SOF signal)
- Register Master Interface (i.e. processed Data Output bus, DATA_O synched to EOF)

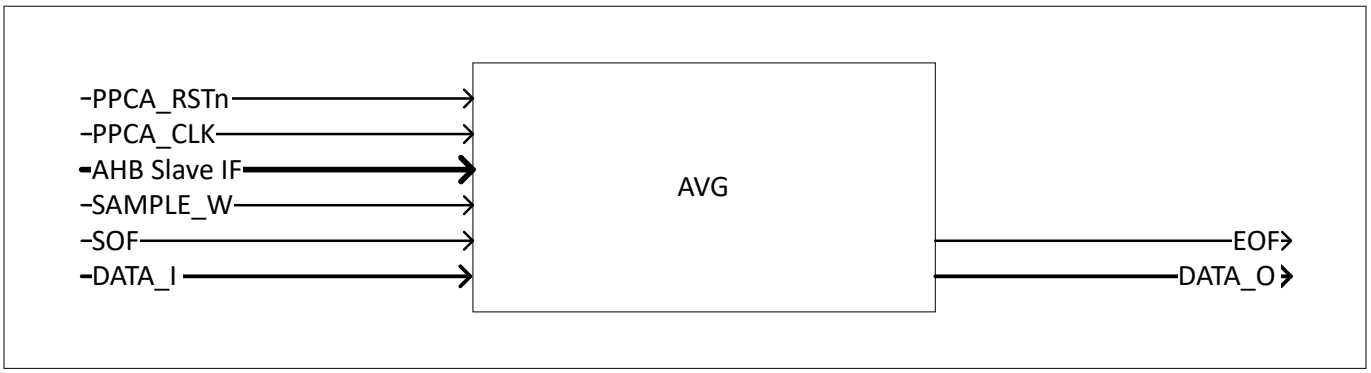


Figure 281 Block diagram of AVG filter

The equation for the AVG filter is shown below:

$$y\left(n\right) = \sum_{i=0}^{M} x(i)^{K} \tag{7}$$

Where M is configurable, and K is either 1 or 2.

The Average Filter diagram is depicted in Figure 282.

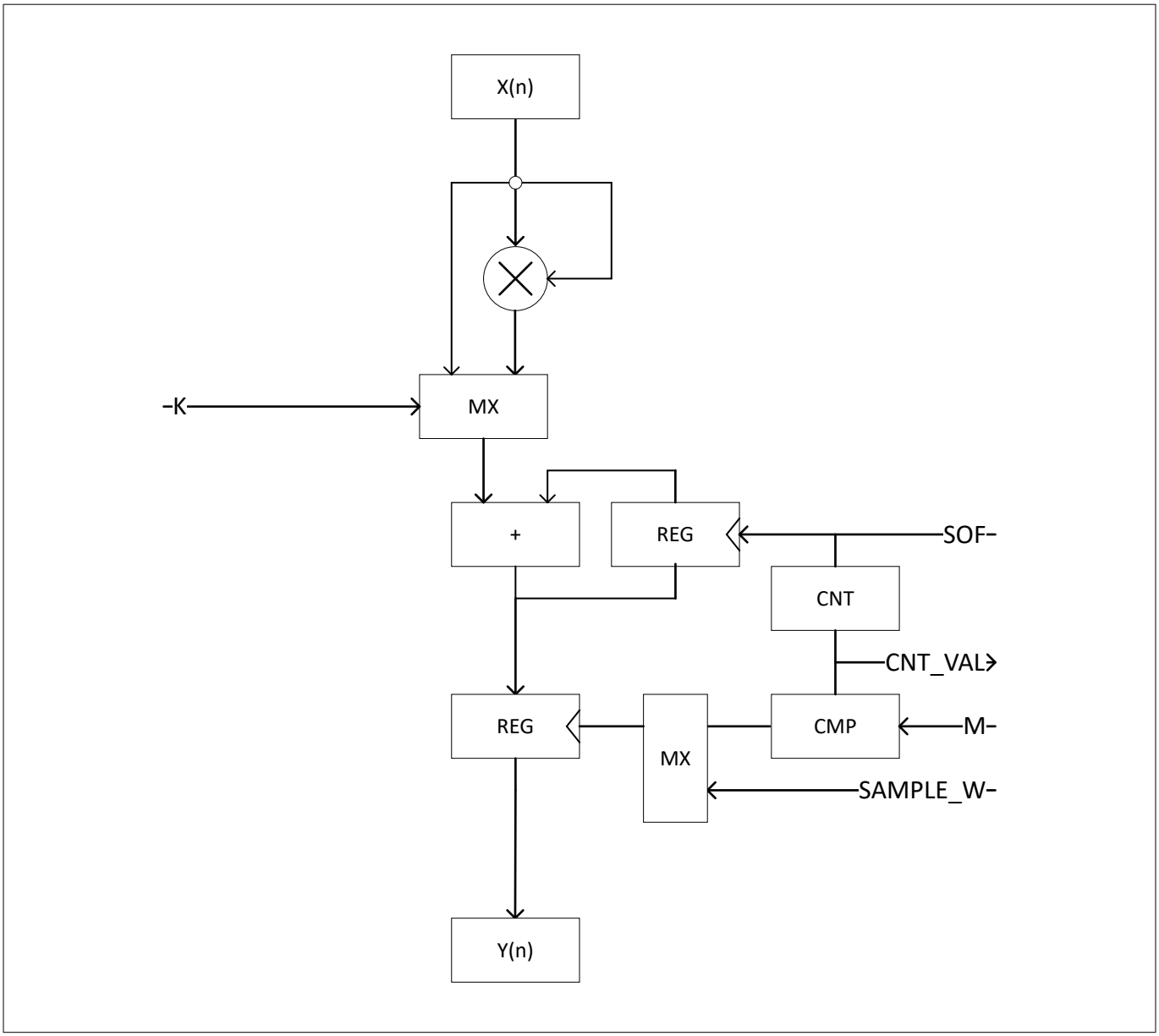


Figure 282 Average filter diagram

The table Table 277 shows an example when SEL_W_TYPE = 0 case, K=1 (AVGF_CNFG.K bit is set to 0B), M = 2. The subsystem updates AVGF_CTL.ACCM_SAMPLE_RPT with the value of M when an EOF event occurred.

Table 277 Average filter example when SEL_W_TYPE = 0

No	X	Y	Remark
0	10	10	Trigger by SOF_TRIGGER bit, i=0
1	20	30	i=1
2	30	60	EOF will be generated, i=2
3	26	60	
4	24	24	Trigger by SOF_TRIGGER bit, i=0

(table continues...)

Table 277 (continued) Average filter example when SEL_W_TYPE = 0

No	X	Y	Remark
5	26	50	i=1
6	32	82	EOF will be generated, i=2

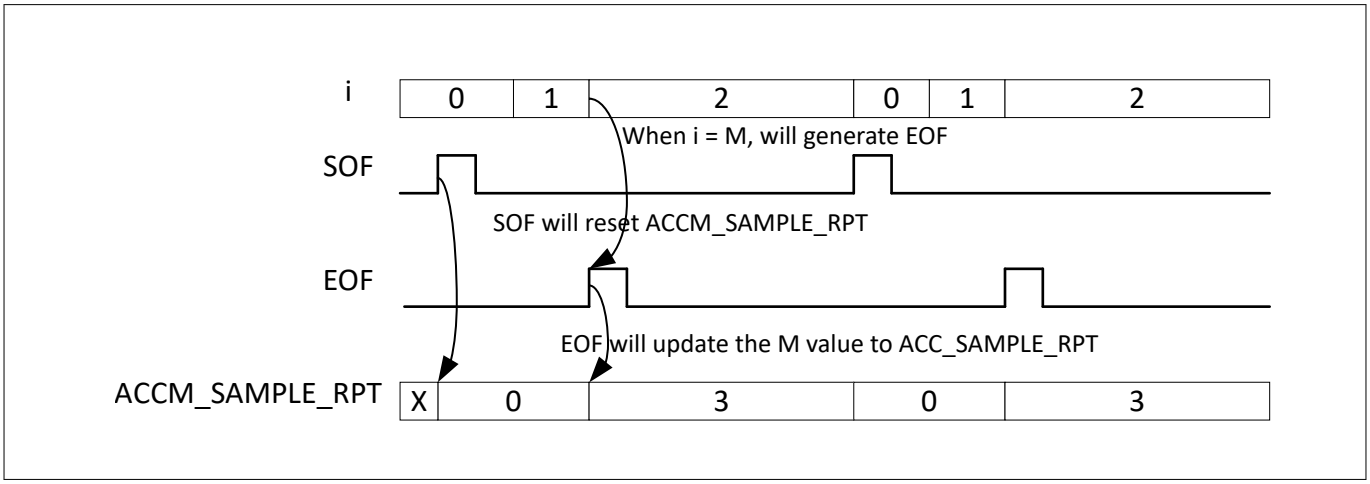


Figure 283 Behavior when SEL_W_TYPE = 0

The table Table 278 illustrate an example when SEL_W_TYPE = 1 case, K=1 (AVGF_CNFG.K bit is set to 0B). The subsystem updates AVGF_CTL.ACCM_SAMPLE_RPT with the value of M when an EOF event occurred.

Table 278 Average filter example when SEL_W_TYPE = 1

i	X	Y	Remark
0	10	10	SOF – Trigger by SAMPLE_W signal going high, i=0, M=1
1	20	30	i=1, M=2
2	30	60	i=2, M=3
3	26	86	i=3, M=4
4	24	110	EOF – Trigger by SAMPLE_W signal going low, i=4, M=5
5	26	110	
6	32	32	SOF – Trigger by SAMPLE_W signal going high, i=0, M=1

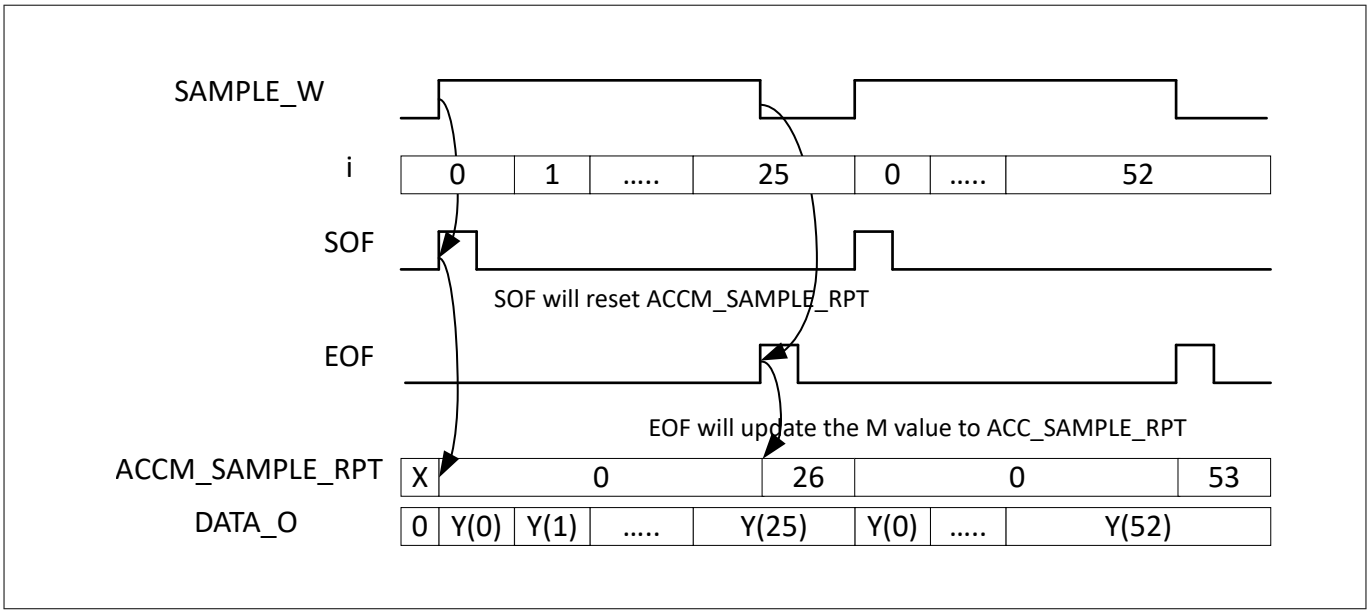


Figure 284 Behavior when SEL_W_TYPE = 1

30.3.14.7 Min/Max detection

This module provides a detection of the minimum and/or the maximum of an input data stream. Figure 285 shows an example of the min/max detection.

- Selectable input data stream: between input or output of the median filter (Through AFLT.AFLT_CNFG.SOURCE_MIN_MAX_SAMPLE_SEARCH)
- Configurable search window: Continuous or Framed (Through AFLT.AFLT_CNFG.MIN_MAX_SAMPLE_SEARCH)
- Search window triggered by firmware through upon read or min_max_trigger_i signal
- Once the search window is started, if min_max_trigger_i is triggered, the current min/max value in ADC_MIN_MAX register will be cleared upon the restart of search window. Once it is cleared, software/ firmware will be reading based on the current search window if a read operation is performed

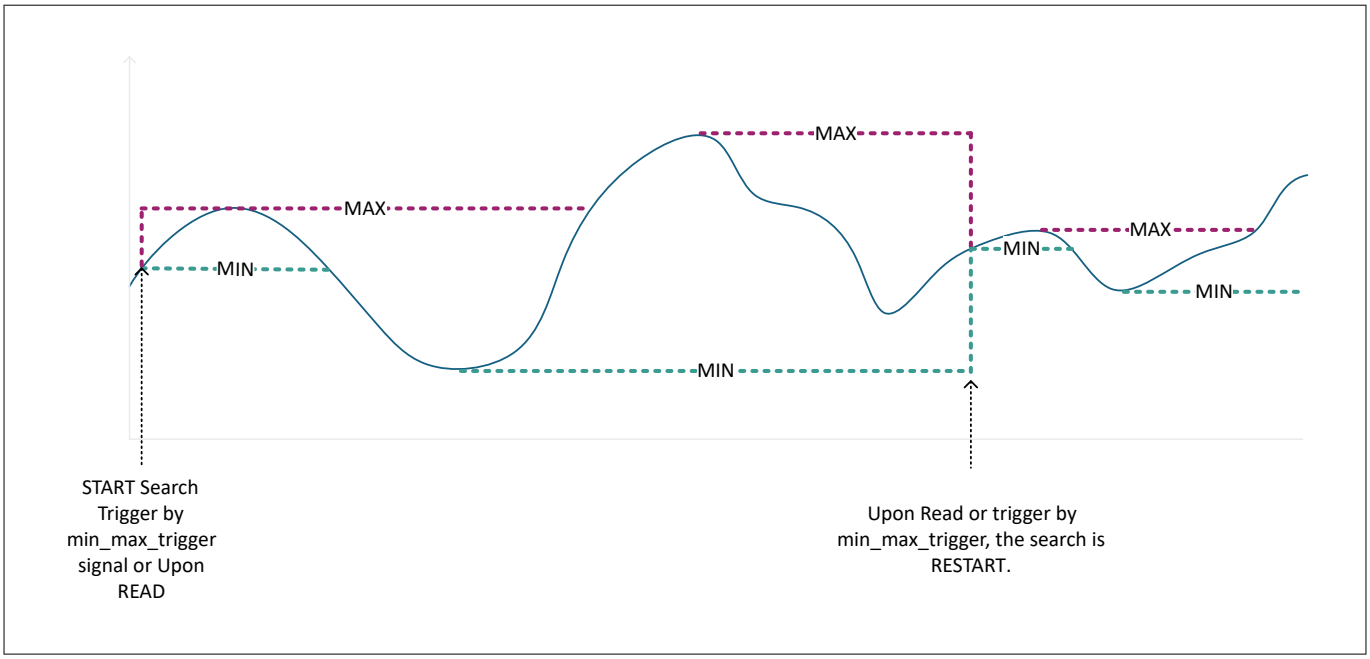


Figure 285 Example of Min/Max detection configure in open search

The table Table 279 shows an example when the MIN_MAX_SAMPLE_SEARCH window is set to 16 points and the behavior of restart upon reading of the ADC_MIN_MAX register. This behavior is applied to an open, and 64/256 points search window. The main difference is that for an open search window, it will continue to update the min or maximum value as the search window is infinite. It will reset only upon reading of the ADC_MIN_MAX register.

Table 279 Example of the MIN_MAX_SAMPLE_SEARCH window set at 16 points

No of ADC Conversion	ADC Value	Min Value	Max Value	Remark
0	0	0	0	Order of the filter
1	40	40	40	Start Trigger by mix_max_trigger signal
2	60	40	60	
3	90	40	90	
4	80	40	90	
…..	xx	40	90	Assuming 5th to 11 ADC conversion is between Min and Max value
12	20	20	90	
13	40	20	90	
14	69	20	90	
15	68	20	90	
16	78	20	90	
17	88	20	90	

(table continues...)

Table 279 (continued) Example of the MIN_MAX_SAMPLE_SEARCH window set at 16 points

No of ADC Conversion	ADC Value	Min Value	Max Value	Remark
18	100	20	90	Maximum value does not change as the search window configuration is based 16 points
19	60	60	60	Read of ADC_MIN_MAX register will cause restart of search. Thus, Min and Max will be the same value.

30.3.14.8 Filter module dedicated data path

There is a dedicated data path from the filter to the HWFILT3P3Z instances or to CLB. The following figure shows the relationship between the dedicated data path from the filter to the HWFILT3P3Z or the CLB. For ADC_FILT_30 and ADC_FILT_31, the ADC_1 ADC data can be routed to ADC_FILT_31 and ADC_2 ADC data can be routed to ADC_FILT_30. The Figure 286 show the HW filter dedicated datapath.

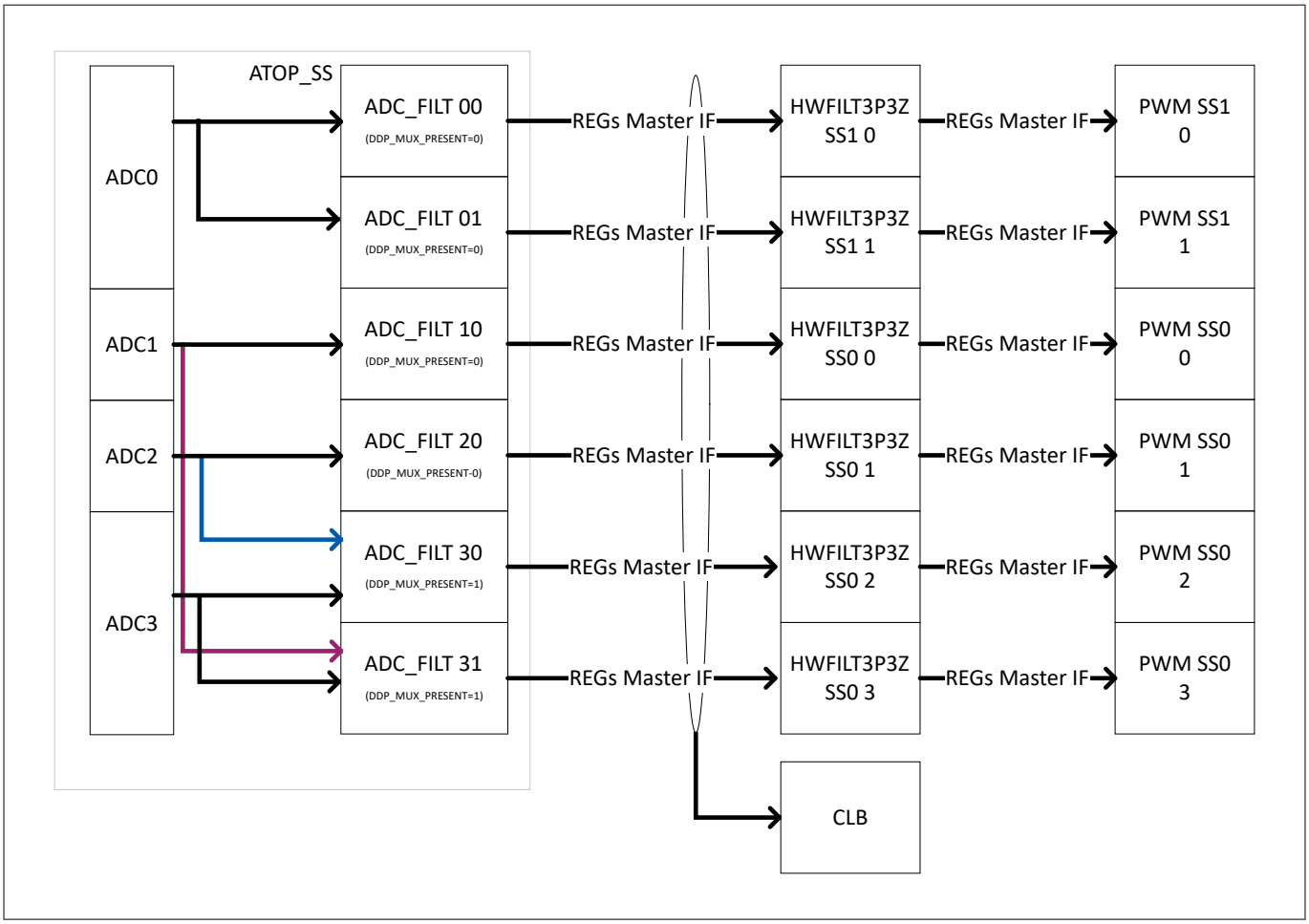


Figure 286 HW filter dedicated data path

The table Table 280 shows the mapping of ADC filter to HWFILT3P3Z in the PPCA.

Table 280 Mapping of ADC Filter to HWFILT3P3Z hardware in PPCA

ADC Filter	Mapping to HWFILT3P3Z
ADC_FILT 00	HWFILT3P3Z Group1 Filter 0
ADC_FILT 01	HWFILT3P3Z Group1 Filter 1
ADC_FILT 10	HWFILT3P3Z Group0 Filter 0
ADC_FILT 20	HWFILT3P3Z Group0 Filter 1
ADC_FILT 30	HWFILT3P3Z Group0 Filter 2
ADC_FILT 31	HWFILT3P3Z Group0 Filter 3

The selection of the adc data to be to be fed to ADC_FILT_30 depend on the AFLT_CNFG.SEL_SRC_ADC_FILT and the status of whether ADC2 and ADC3 are enabled. The table Table 281 illustrates the condition on the switching of ADC data to the ADC_FILT_30 that need be taken care.

Table 281 Data source condition for ADC_FILT_30

ADC3_EN	ADC2_EN	AFLT_CNFG.SEL_SRC_ADC_FILT	Data Source
0	0	1	ADC_2
0	1	1	ADC_2
1	0	0	ADC_3
1	1	0	ADC_3
1	1	1	ADC_2
1	0	1	ADC_2

The selection of the adc data to be to be fed to ADC_FILT_31 depend on the AFLT_CNFG.SEL_SRC_ADC_FILT and the status of whether ADC1 and ADC3 are enabled. The table Table 282 shows the condition on the switching of ADC data to the ADC_FILT_31 that need be taken care.

Table 282 Data source condition for ADC_FILT_31

ADC3_EN	ADC1_EN	AFLT_CNFG.SEL_SRC_ADC_FILT_31	Data Source
0	0	1	ADC_1
0	1	1	ADC_1
1	0	0	ADC_3
1	1	0	ADC_3
1	1	1	ADC_1
1	0	1	ADC_1

The figure Figure 287 illustrates the data mapping and padding of the data stream where the data stream is less or equal to 16 bits.

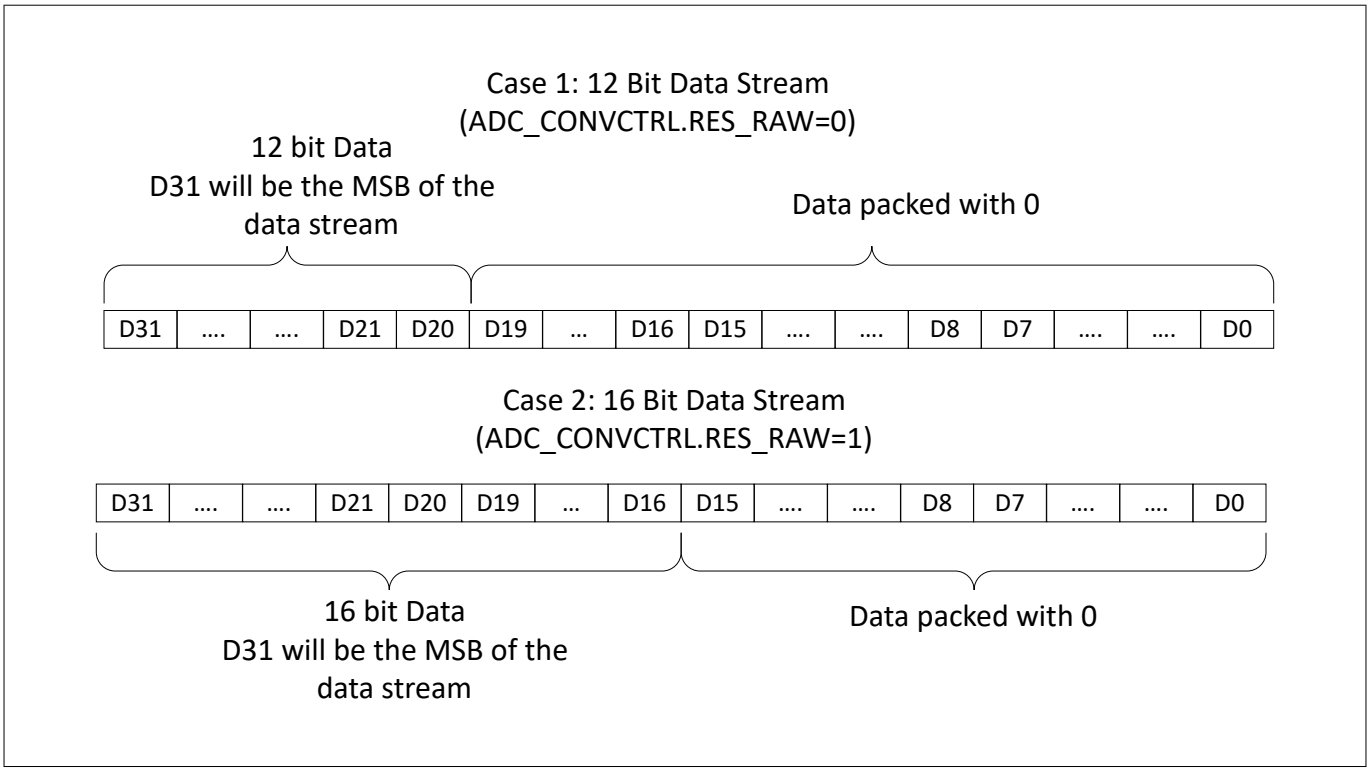


Figure 287 DATA[31:0] Data alignment and padding for data stream of less than 16 bit

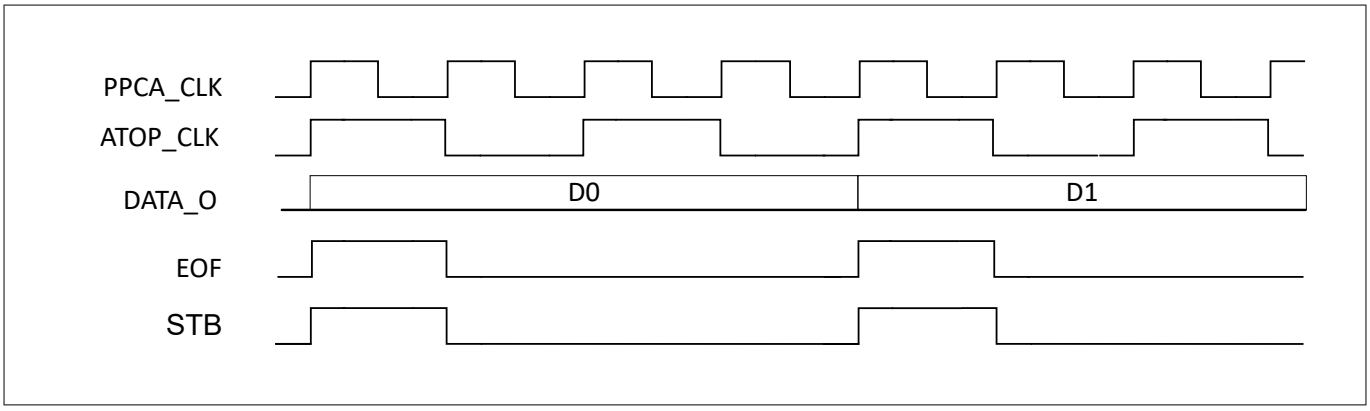


Figure 288 Behavior of STB signal

30.3.15 ADC result

To ease the data manipulation in the FW, the data read through the AHB bus will be sign extended to 32 bits for the following registers:

1. ADC_DATA
2. ADC_AUX_DATA
3. ADC_ALT_AUX_DATA
4. LPF_DOUT
5. MED_DOUT
6. LIF_DOUT
7. CIFC_DOUT

The table Table 283 shows the mapping of the channel's selector of the ADC_CORE channels to data channel number for ADC with eight passive channels.

Table 283 Data channel number mapping to channel of the ADC Core for ADC with 8 passive channels

ADC IP channels selector	Subsystem data channel number	Remarks
8	0	If AFE_EN is '1', AFE channel will be mapped to data channel 0
0	0	If AFE_EN is '0', passive channel 0 will be mapped to data channel 0
1	1	
….	….	
7	7	

The register SIGN_EXTENDED bits are implemented for ADC_DATA, ADC_AUX_DATA, ADC_ALT_AUX_DATA, LPF_DOUT, MED_DOUT and LIF_DOUT. When the data is negative for the sign format, the SIGN_EXTENDED bit will be 1. When the data is positive in the sign format or the unsigned data option is selected, the SIGN_EXTENDED bit will be 0. The diagram Figure 289 shows the different format for the AHB read of the ADC related data based on 12/16 bits stream.

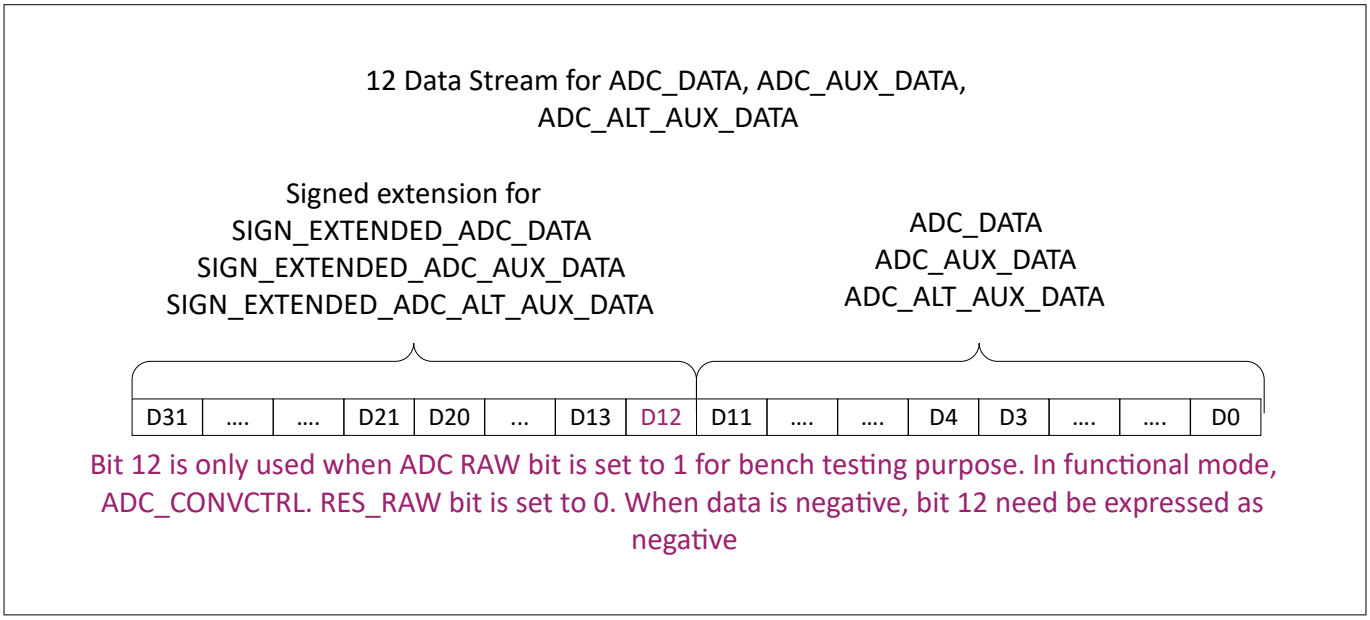


Figure 289 Sign_extended for ADC related data stream

30.3.15.1 Filter result format

As there are different bit widths for different type of filter in the ADC_FILTS, the table Table 284 tabulate the different input and output bit width for the different type of filter in the filter module.

Table 284 Bit width of filter input and output

Type of filter	Input bit width	Output bit width	AHB-Read handling
Median filter	12	12	Yes
LIP	12	16	Yes
LP	16	16	Yes

(table continues...)

Table 284 (continued) Bit width of filter input and output

Type of filter	Input bit width	Output bit width	AHB-Read handling
CIC3	16	32	No
Average	16	32	No
MIN/MAX	12	12	No

The filter data read through the AHB bus will be based on 32 bits even though the output data for medium, low pass, and linear interpolator filter are less than 32 bits. The diagram Figure 290 shows the different format for the AHB read of the filter data based on 12/16/32 bits stream.

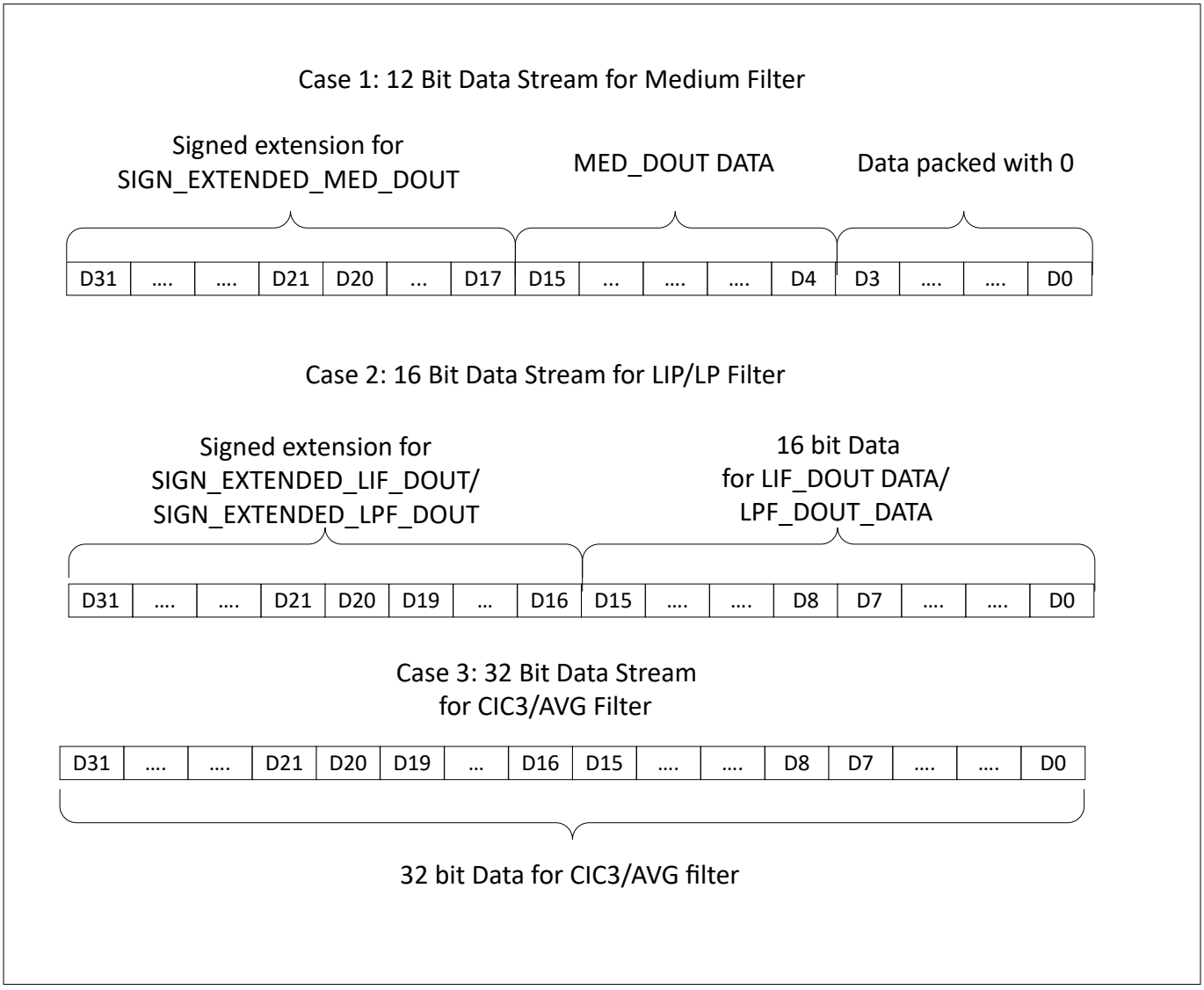


Figure 290 FIL_DOUT[31:0] Data alignment and padding for data stream of less than 16 bit

The table Table 285 shows the behavior for the medium filter behavior due to the padding of zero on the last 4 significant bit when it is being read through the AHB bus.

Table 285 12 bit actual data its sign extended 32 bit data

Dec	12 Bit Data	32 bit Read from AHB Bus	Dec value in 32 bit Read from AHB Bus
0	0x000	0x0000 0000	0
2047	0x7FF	0x0000 7FF0	32,752
-2048	0x800	0xFFFF 8000	-32,768
-1	0xFFF	0xFFFF FFF0	-16

30.3.16 ADC interrupt error condition

For the following conditions the bit ADC_INTR.TR_ERROR will be asserted:

- Receiving internal trigger when external source is selected
- Receiving external trigger in auto trigger mode
- Receiving external trigger in manual trigger mode when internal source selected
- Receiving trigger before start-up calibration is completed
- Receiving trigger from invalid CHANNEL
- Receiving trigger when adc_busy is set

30.4 IDAC

The IDAC is belonging to the ADC_GRP0. IDAC modules are used to generate a known current to be used to sense an external resistance connected to AI0 and AI1 pads. As the IDAC is connected to AI0 and AI1 which has shared the same connection to GPI module and DAC_R2R output buffer. When a resistor is connected to the AI0 and AI1 pad respectively, the DAC_R2R output buffer must be disabled.

Another function of the IDAC is for analog input pin float detection. Through the DCSG<0> MUX, the IDAC can be used to check all the AIN<11:0>P and AIN<11:0>N pins.

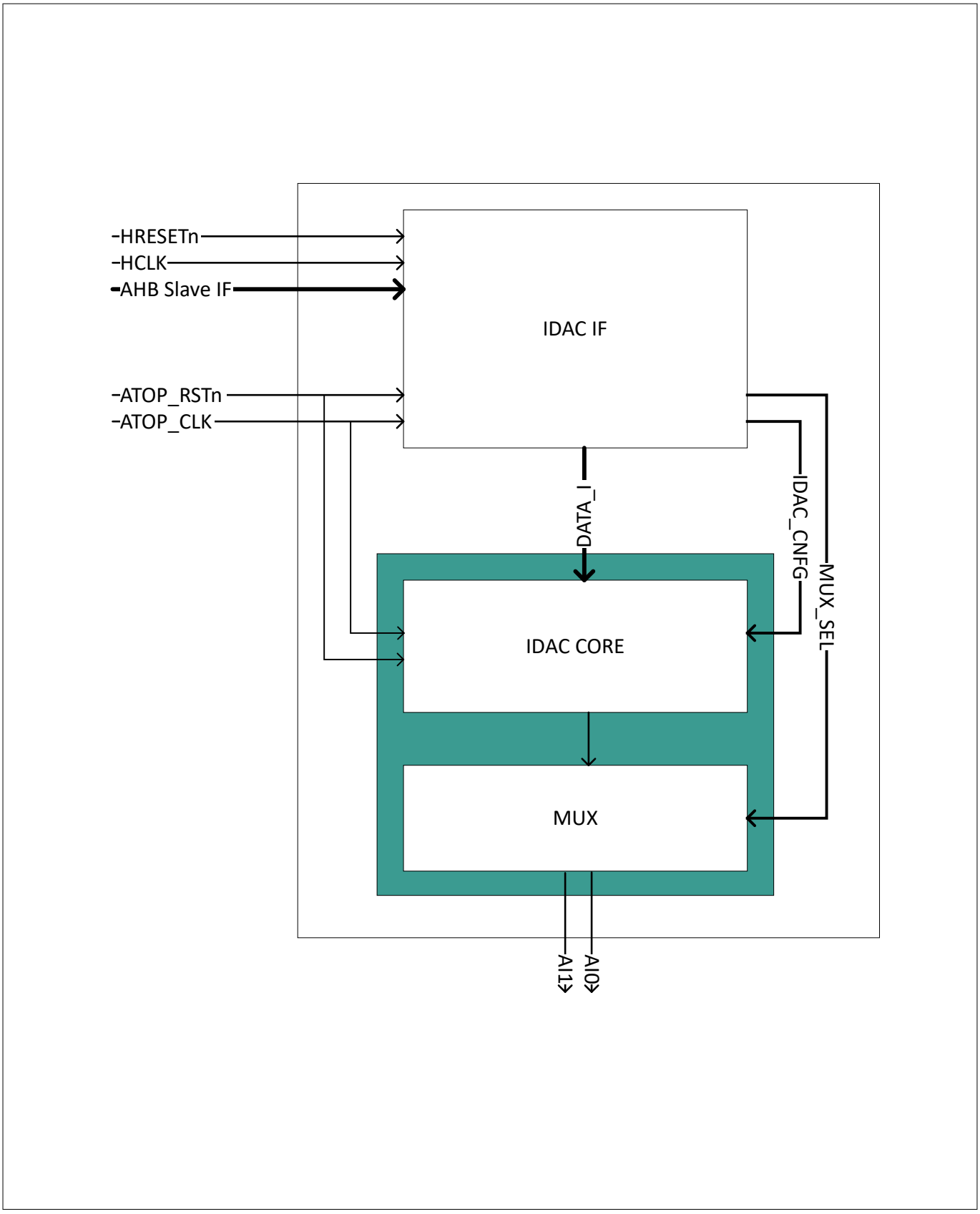


Figure 291 IDAC block diagram

30.4.1 Using the IDAC for PMBus detection

In this scenario, the user is using the IDAC to source 10uA current into the external resistance that will be placed on the motherboard. The value of an external resistor will denote the power rating requirement of the motherboard. The Figure 292 illustrates the application setup on the use of IDAC with ADC0 to measure voltage across the external resistor.

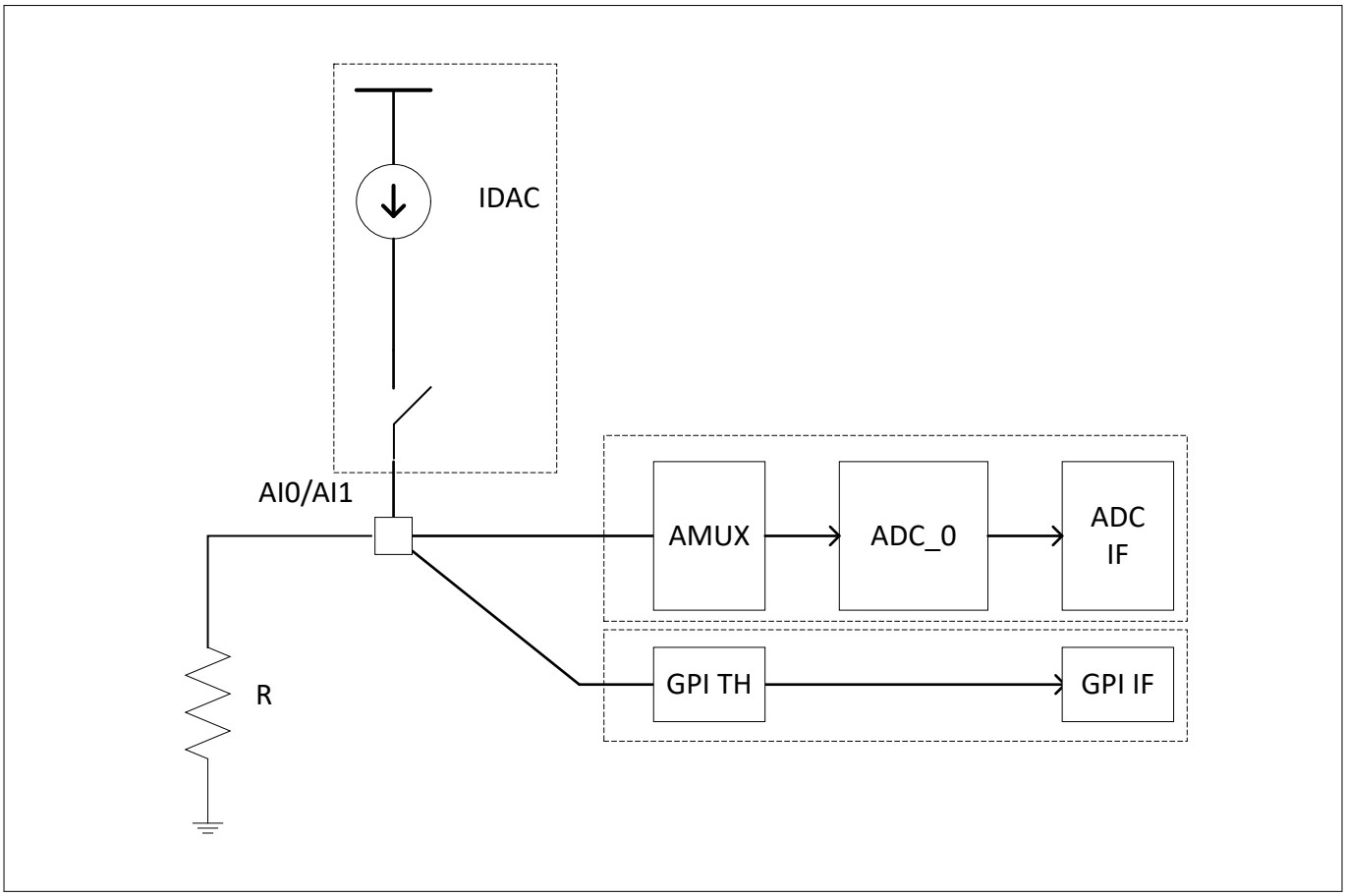


Figure 292 IDAC PMBus resistance measurement

When the voltage is generated across the resistor, ATOPSS can measure the voltage across the resistor through ADC0. The following table illustrated the voltage will be generated across the 7 range of resistance for the PMBus application. There is +/-1% variation and +/-10% variation on the current source of the IDAC.

	Resistance (Ohm) +/- 1% Tolerance on the resistor			**Voltage (V) (VDDA:2.97 to 3.60)**		
	Min	**Typ**	**Max**	**Min**	**Typ**	**Max**
Motherboard	277,200	280,000	282,800	2.495	2.800	3.111
	119,790	121,000	122,210	1.078	1.210	1.344
	67,419	68,100	68,781	0.607	0.681	0.757
	39,798	40,200	40,602	0.358	0.402	0.447
	23,463	23,700	23,937	0.211	0.237	0.263
	13,167	13,300	13,433	0.119	0.133	0.148
	5,702	5,760	5,818	0.051	0.058	0.064

30.4.2 Using the IDAC for input pin float detection

In this scenario, the system and user can use IDAC to check whether the input pin is floating. The diagram Figure 293 shows the block diagram of the IDAC with the DCSG Mux for the purpose of input pin float detection.

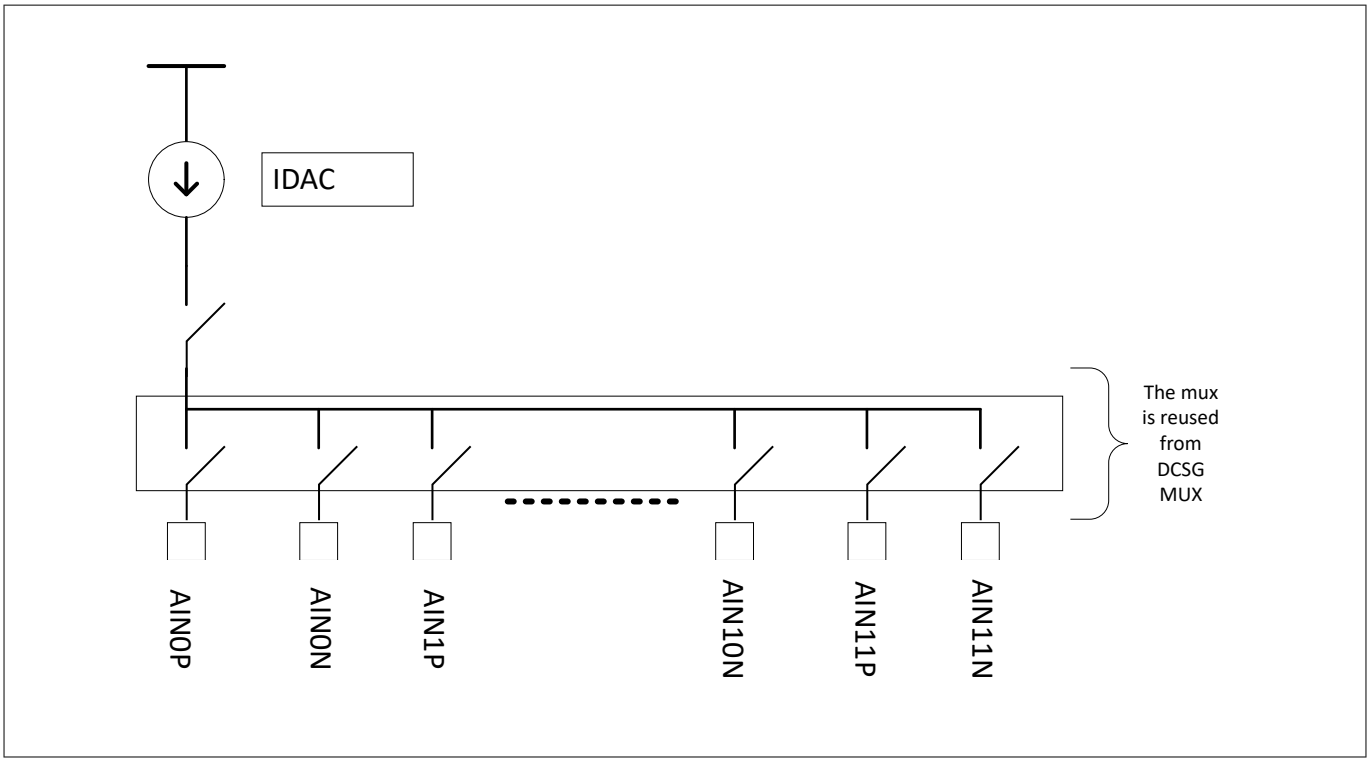


Figure 293 IDAC input pin float detection block diagram

Condition that causes the input pin floating is shown in the figure Figure 294.

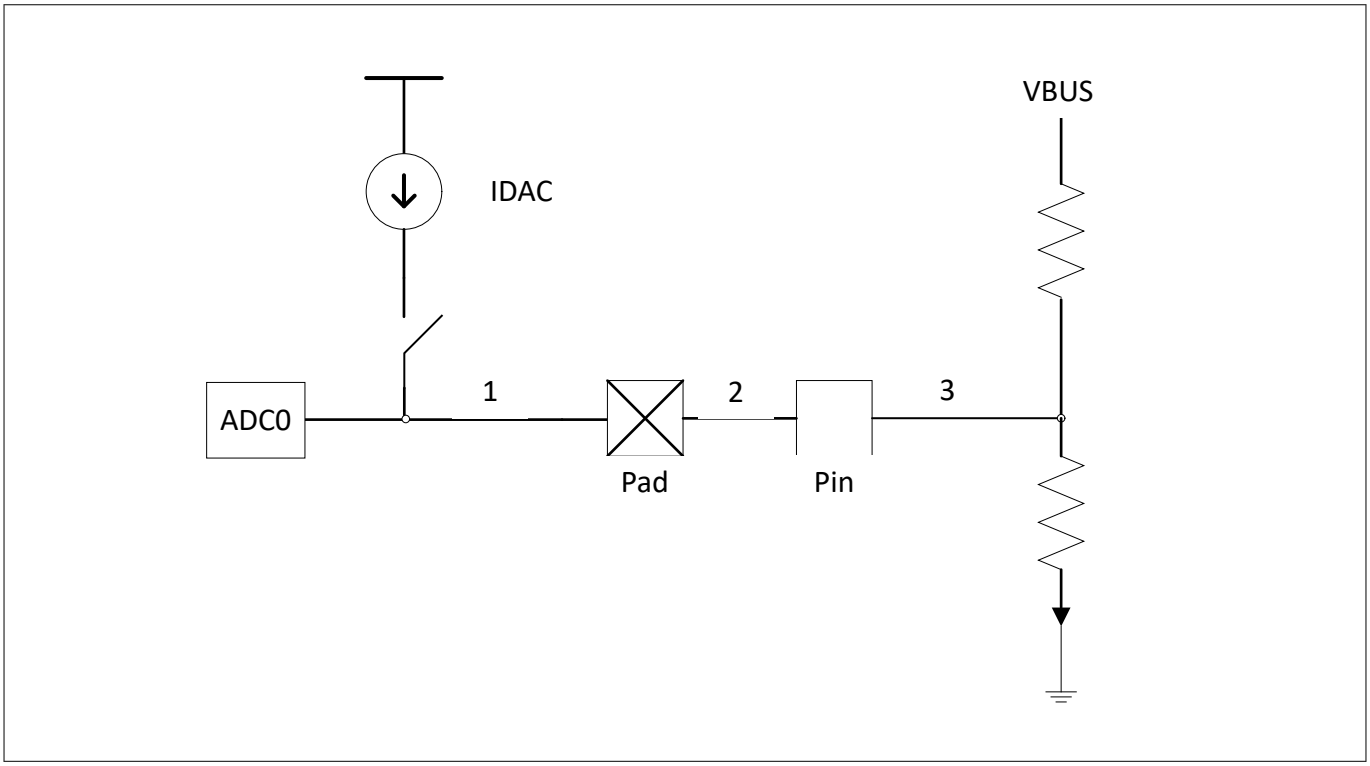


Figure 294 Floating input pin scenario

The table Table 286 summarized the condition where the ADC0 is to measure the floating input pin condition with the use of IDAC.

Table 286 Input pin float detection conditions

Condition No	Causes	Behavior
No 1	Connection from the pad to ADC0 and IDAC is open	ADC0 Code will be FS code
No 2	Bonding wire between the pad and pin is open	ADC0 Code will be FS code
No 3	PCB routing is open between pin and the resistor divider	ADC0 Code will be FS code

30.5 Analog Reference (AREF)

The AREF module generates the reference voltage and reference current for all the analog modules that reside in the ATOP SS. There is a single AREF for the entire ATOP. It is located in ADC_GRP0. Therefore, to use the PPCA analog resources, the ADC_GRP0 has to be always enabled.

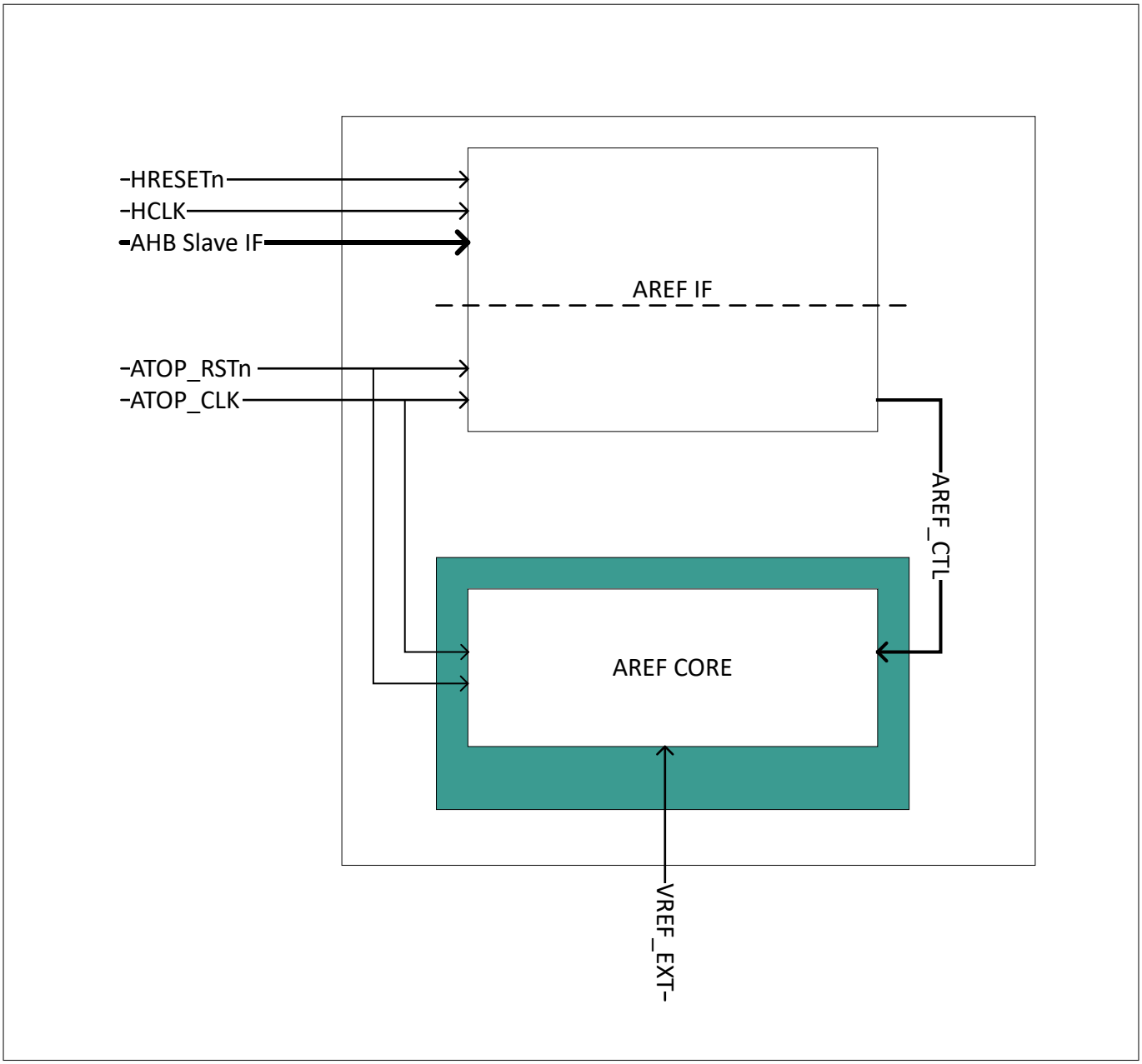


Figure 295 AREF block diagram

As AREF is providing the reference current and voltage to the analog block, AREF is the last IP block to be powered off in ATOP SS. The (AREF) provides a precision voltage reference and current references to the ATOP SS analog blocks.

There are calibration registers available to correct the component specific variations. The PDL implements functions for copying the calibration values to the calibration registers of the AREF. It is recommended to call the initialization API in the application. For more information, see the Peripheral Driver Library API Reference Manual.

30.6 General Purpose Input (GPI)

The GPI modules are belonging to the ADC_GRP0. These are used to convert an analog input into digital output by assuming a fixed analog threshold(0.3 VDDA, 0.7 VDDA). Analog inputs are coming from the AI0 and AI1.

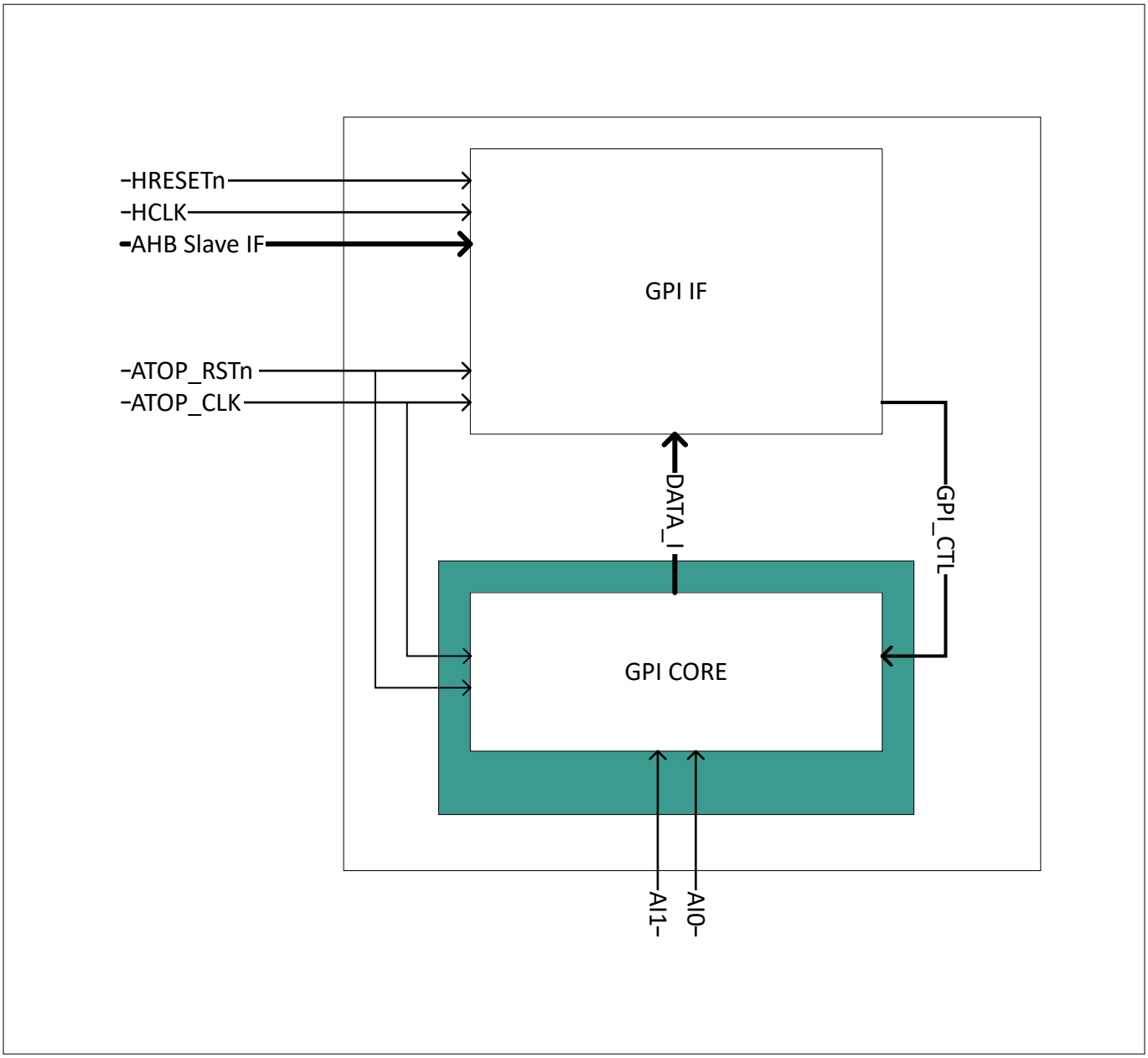


Figure 296 GPI block diagram

30.6.1 Using GPI

In this scenario, the user is using the GPI to detect the logic level of AI0 or AI1. As shown in the figure Figure 297, the GPI is reading values from analog circuit as digital signals. For that GPI compares the analog input with the threshold values and determine the logic state based on the voltage level.

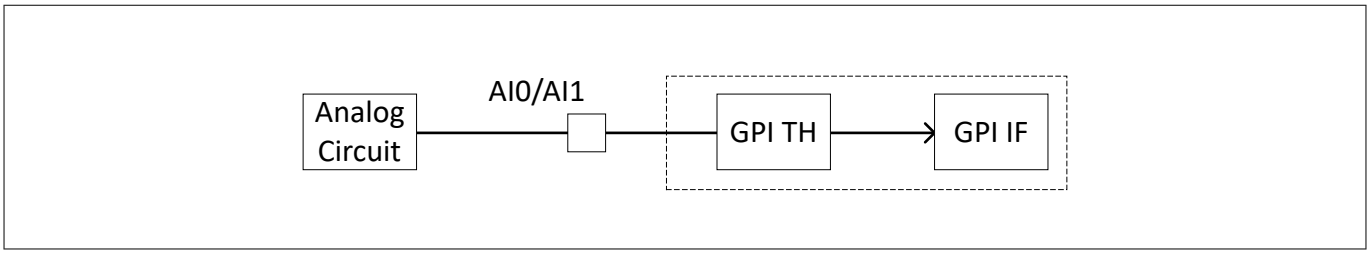


Figure 297 GPI use case diagram

30.7 Temperature sensor (TSNS)

ATOP SS includes an on-chip temperature sensor (TSNS) that is used to measure the internal die temperature. It allows approximate temperature measurement in ATOP SS. The TSNS module is used to work with AREF and ADC0. The ADC0 will convert the voltage across the TSNS module and it is used for temperature measurement.

TSNS comprised of a single current source input (i.e., the 1.0uA Ibias current) and a single BJT in diode configuration. This provides about 0.5 degrees C/LSB in resolution without amplification.

This block has a single Ibias connected to it (used to forward bias the diode connected BJT with the resulting forward bias voltage measured by the ADC). When the temp sensor is disabled, the Ibias line is left floating. The temperature sensor should be enabled only when a temperature measurement is taken.

Temperature can be calculated as given in the following equation:

Temperature = (ADC_COUNT $*A + 2^{10} *$ B)$/2^{16}$

Here, the ADC_COUNT is the number returned by the ADC. The constants A, and B are stored in the SFLASH_SAR_TEMP_COEF_A, and SFLASH_SAR_TEMP_COEF_B registers during factory calibration.

The PDL contains a function to convert the measured voltage to temperature. For more information, see the Peripheral Driver Library API Reference Manual.

30.8 Differential Comparator Slope Generation (DCSG)

30.8.1 Features

Analog Top subsystem (ATOP SS) includes two DCSG groups, each DCSG can integrate up to six differential/ single ended analog compactor (DCSG slice [i]) with programmable threshold.

The DCSG group 0 instantiates a DCSG module configured to support six differential/single ended analog comparator (DCSG Slice[i]) with programmable threshold, a R2R DAC and an AHB Slave interface. DCSG group0 supports two HW slope generation function.

The DCSG group 1 instantiates a DCSG module configured to support three differential/single ended analog comparator (DCSG Slice[i]) with programmable threshold, a R2R DAC and an AHB slave interface. DCSG group 1 supports an HW slope generation function.

30.8.2 Architecture

30.8.2.1 Block diagram

Figure 298 shows the DCSG_GRP block diagram.

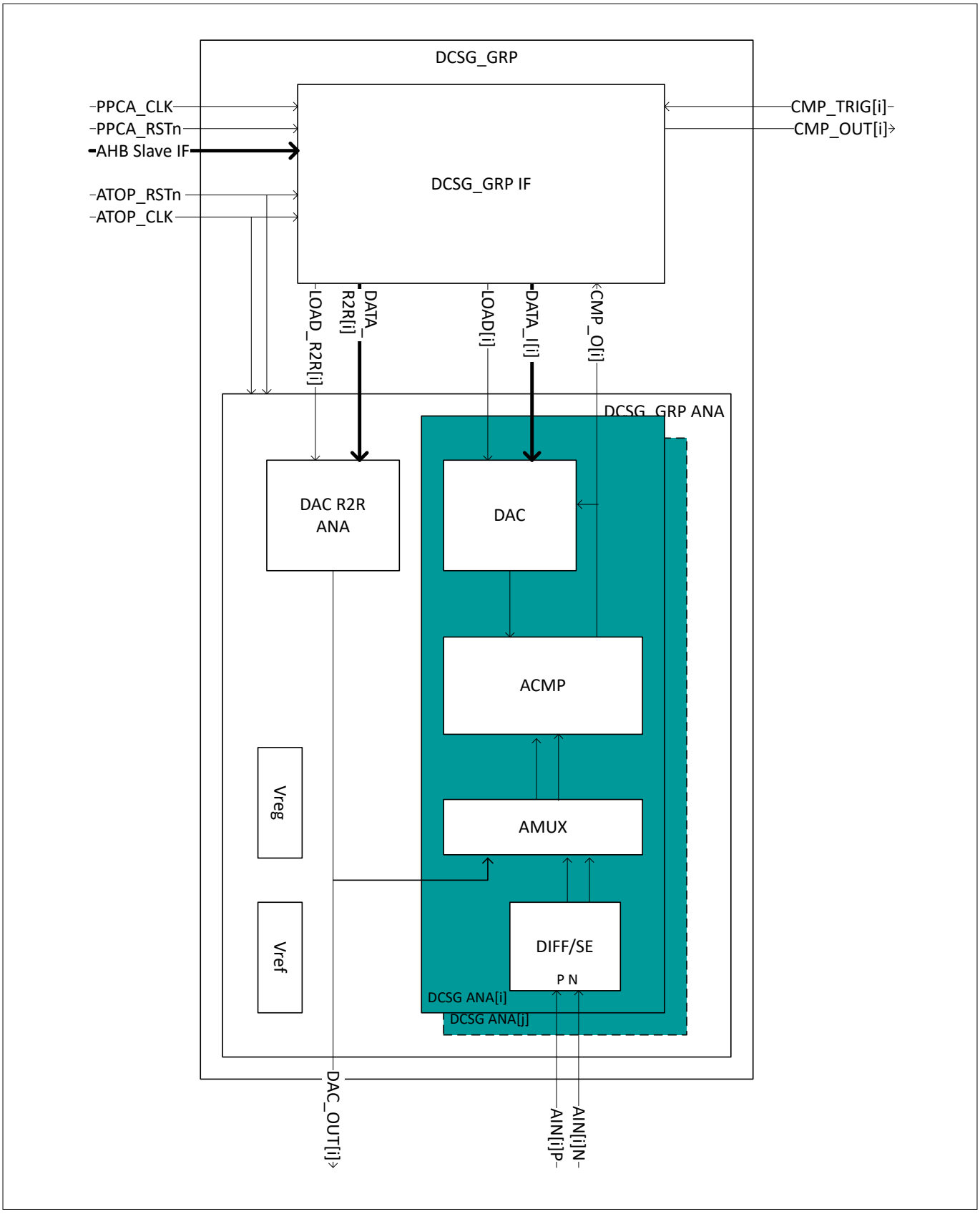


Figure 298 DCSG_GRP block diagram

30.8.3 Functionality description

30.8.3.1 DCSG group

Each DCSG_GRP subsystem is partitioned into two sections:

- DCSG_GRP IF
- DCSG_GRP ANA

The DCSG_GRP IF section integrates the digital sections of the instantiated mixed-signal/analog Hard IPs interfacing the AHB slave bus.

The DCSG_GRP ANA section integrates the analog sections of the DCSG Slices and R2R DAC mixed signal/analog hard IPs.

The DCSG_GRP_0 instantiates a DCSG module configured to support six differential/Single ended Analog Comparator (DCSG Slice[i]) with programmable threshold supported by a dedicated embedded 12-bit DAC, a R2R DAC and an AHB Slave interface. Two DCSG Slice supports an HW slope generation function.

Figure 299 shows DCSG_GRP_0 configuration.

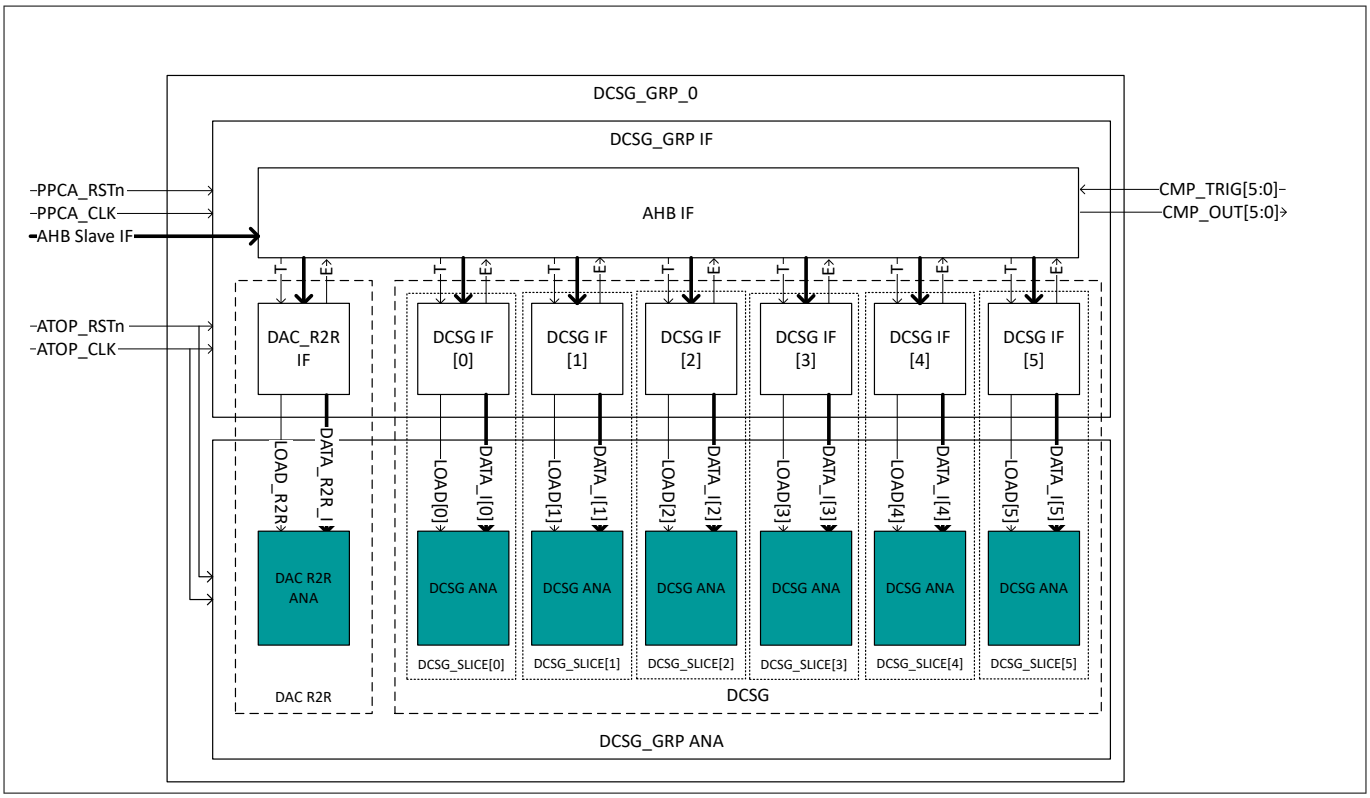


Figure 299 DCSG_GRP_0 configuration diagram

***Note**: ATOP_CLK for DCSG is is fed from PPCA_CLK. It is operating at the same frequency of PPCA_CLK.*

The DCSG_GRP_1 instantiates a DCSG module configured to support three differential/Single ended analog comparator (DCSG Slice[i]) with programmable threshold supported by a dedicated embedded 12-bit DAC, a R2R DAC and an AHB Slave interface. One DCSG Slice supports an HW slope generation function.

***Note**: The ATOP_CLK for DCSG is fed from PPCA_CLK. It is operating at the same frequency of PPCA_CLK.*

Figure 300 shows DCSG_GRP_1 configuration.

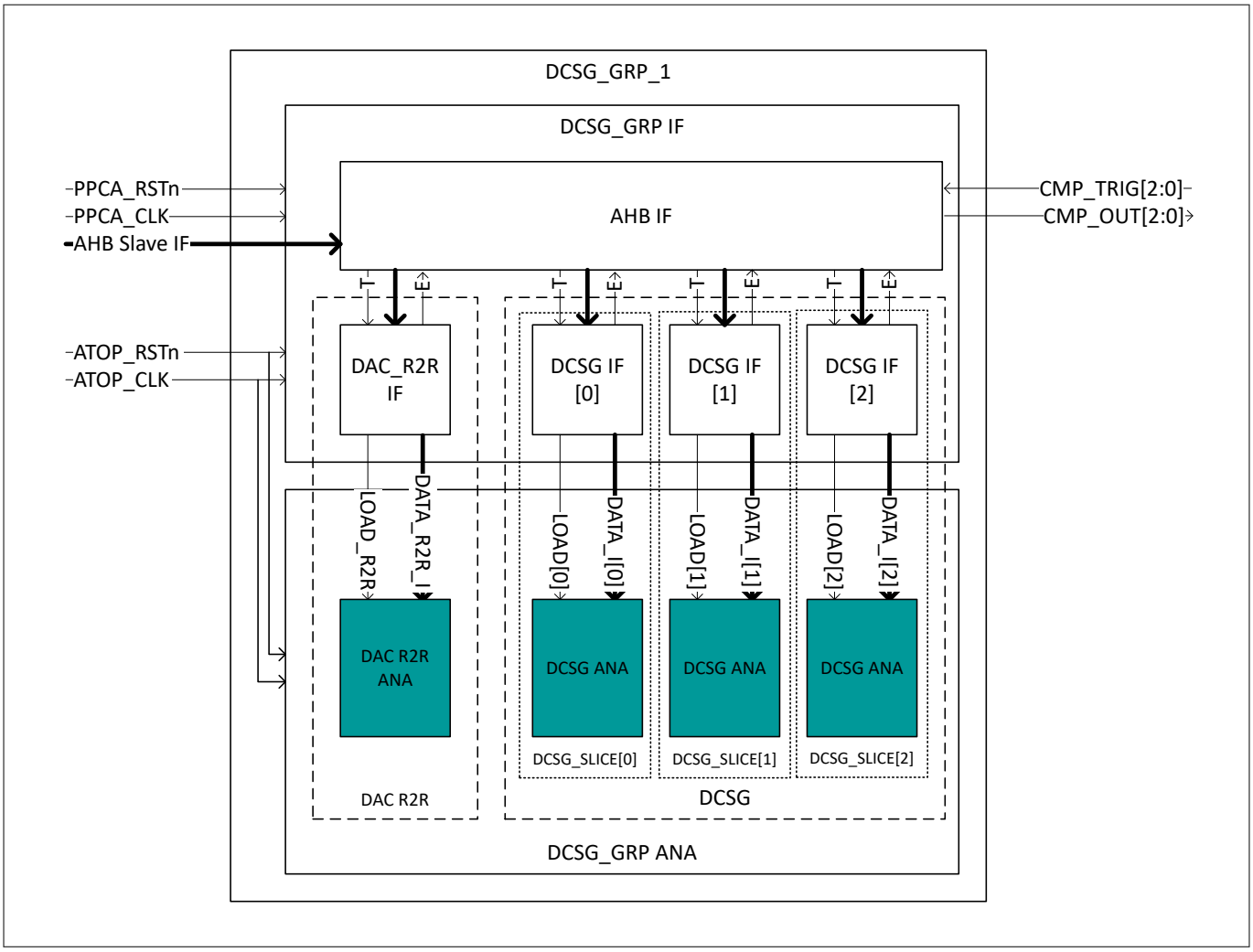


Figure 300 DCSG_GRP_1 configuration diagram

The DCSG_ANA section implements the DCSG analog functions:

- DAC
- Analog comparator
- Channels input analog selector (AMUX)

The DCSG_IF implements:

- AHB Slave IF
- Configuration register (CNFG), to select between synchronous CMP_OUT
- Control register (CTL), to enable/disable the corresponding Slice
- Clock domains (PPCA_CLK and ATOP_CLK) adapter
- DAC control interface, optionally including the slope generator function
- Counter to trigger the DAC refresh
- The logic to manage the interleave option to get rid of the required blanking window
- DCSG group0 has two HW slope generation function and DCSG group1 has one HW slope generation function

30.8.3.1.1 Differential mode handling for DCSG_TH

As the DCSG supports single ended and differential ended operation, there is some post processing that needs to be be performed on DCSG_TH value by DCSG before providing to the DCSG analog interface through grpX_dacp_ctrl_i<10:0> and grpX_dacn_ctrl_i<10:0>.

The Table 287 tabulated the operation needed for post processing setup.

Table 287 DCSG Post processing

No.	mmio register		Remarks
	TYPE_CH_SRC	DCSG_TH	
1	DIFF_ER	0~2047	
2	DIFF_ER	2048~4095	
3	DIFF_ER	0~4095	For Init
4	DIFF_FR	0~4095	
5	DIFF_FR	0~4095	For Init
6	SINGLE_P	0~4095	
7	SINGLE_P	0~4095	
8	SINGLE_N	0~4095	
9	SINGLE_N	0~4095	
10	DIFF_RR	0~2047	
11	DIFF_RR	2048~4095	
12	A_VS_B	0~2047	
13	A_VS_B	2048~4095	
14	A_VS_B	0~4095	For Init

30.8.3.2 Slope generator function

The slope generator function implements a hardware machine to provide a DAC value computed between two programmable thresholds, with programmable increment/decrement steps triggered by the input signal TRIGGER, and a register storing the DAC programmed threshold causing comparator to trigger status.

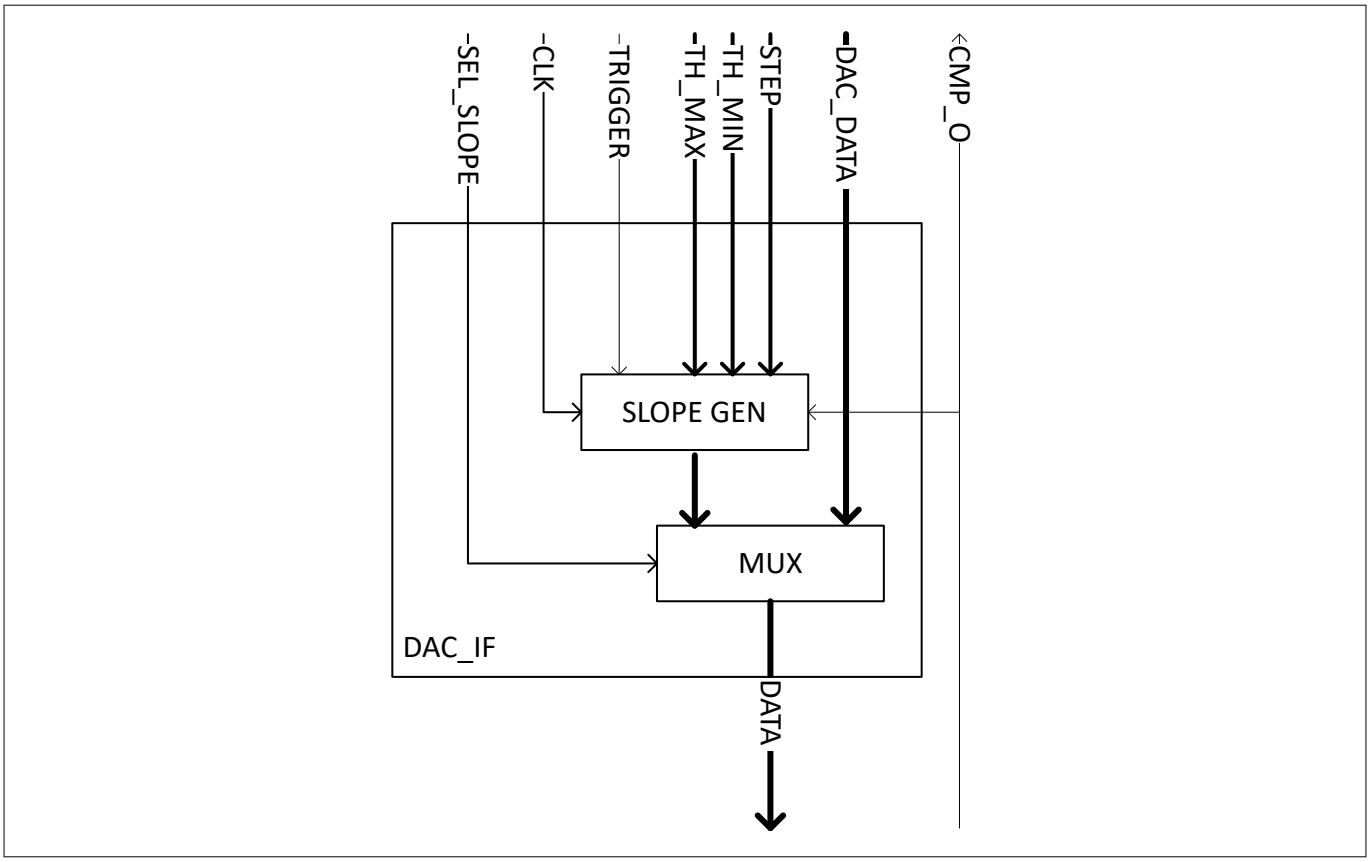


Figure 301 Slope generator function block diagram

***Note**: Slope generation function shall not be used when DCSG_CNFG.SEL_SRC is set to "1". and the operation of the slope generation must be completed before the timing set in the DCSG_CNFG.CFG_REFRESH_RATE setup.*

An implementation example of the slope generator flow diagram is depicted in Figure 302. The slope generator will be triggered by the signal DCSG_SLGEN_CTRL.SLGEN_EN_TRIGGER and then disabled when the comparator ds_slopegen signal is triggered.

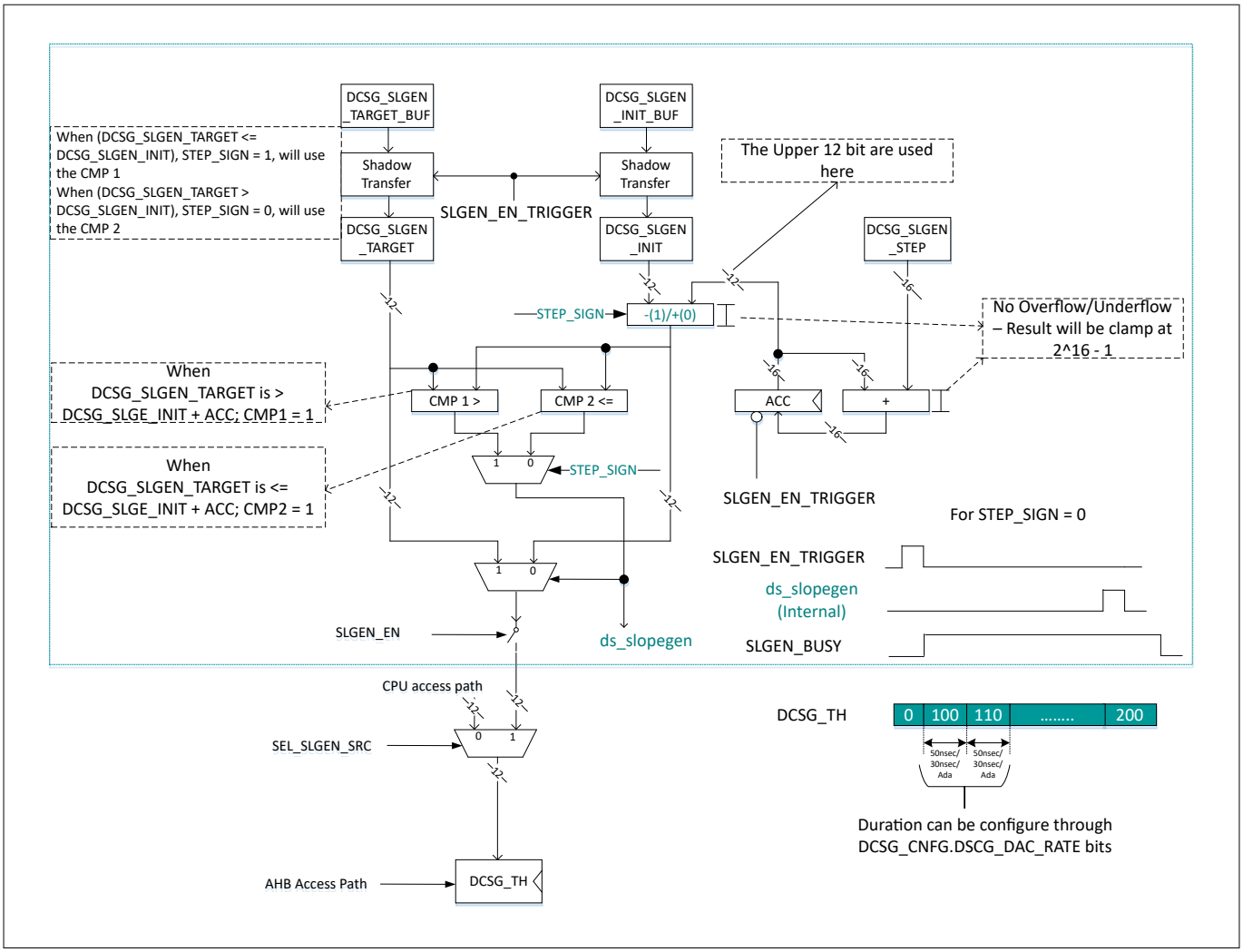


Figure 302 Implementation of slope generator

***Note**: The trigger signal will also be used to trigger the update of the shadow register when DCSG_SLGEN_CNFG.EN_SHADOW_TRANSFER is set to "1".*

The following are the requirements for the slope generator:

- The slope generator allows to use a fractional 4-bit steps
- The value of the accumulator will be clamped at 2^16-1(65,535)
- When STEP_SIGN = 1, CMP 1 is used (DCSG_SLGEN_TARGET <= DCSG_SLGEN_INIT)
- When STEP_SIGN = 0, CMP 2 is used (DCSG_SLGEN_TARGET > DCSG_SLGEN_INIT)
- The duration of the threshold adjustment for the slope generation can be configured from 20nsec to 100nsec based on the setting of DCSG_CNFG.DCSG_DAC_RATE bits
- Auto reset from Target to Init value upon toggle of ds_slopegen
- If SLGEN_EN_TRIGGER is toggled before the completion of the slope generation, the slope generation will restart the slope generation operation

There are two scenarios for the slope generator function block:

The first scenario is when the DSCG_SLGEN_TARGET value is greater than DCSG_SLGEN_INIT. An example for this scenario is tabulated in Table 288

The second scenario is when the DSCG_SLGEN_TARGET value is lesser than DCSG_SLGEN_INIT. An example for this scenario is tabulated in Table 289

***Note**: The initial value of DCSG_TH should be set to a value of DCSG_SLGEN_INT before triggering the slope generator function.*

Table 288 Case 1: When Target > Init

	Target	Init	Step	ACC	Init + ACC	CMP1	CMP2	STEP_ SIGN	ds_slo pe gen	DCSG_ TH (12Bits)	SLGEN _EN _TRIGG ER
	0	0	0	0	0	0	1	0	0	100	0
0	200	100	192	0	100	1	0	0	0	100	1 (Pulse)
1	200	100	192	192	112	1	0	0	0	112	0
2	200	100	192	384	124	1	0	0	0	124	0
3	200	100	192	576	136	1	0	0	0	136	0
4	200	100	192	768	148	1	0	0	0	148	0
5	200	100	192	960	160	1	0	0	0	160	0
6	200	100	192	1152	172	1	0	0	0	172	0
7	200	100	192	1344	184	1	0	0	0	184	0
8	200	100	192	1536	196	1	0	0	0	196	0
9	200	100	192	1728	208	0	1	0	1	200	0
10	200	100	192	1920	220	0	1	0	1	200	0

Table 289 Case 2: When Target < Init

	Target	Init	Step	ACC	Init - ACC	CMP1	CMP2	STEP_ SIGN	ds_slo pe gen	DCSG_ TH (12Bits)	SLGEN _EN _TRIGG ER
	0	0	0	0	0	0	1	0	0	200	0
0	100	200	192	0	200	0	1	1	0	200	1 (Pulse)
1	100	200	192	192	188	0	1	1	0	188	0
2	100	200	192	384	176	0	1	1	0	176	0
3	100	200	192	576	164	0	1	1	0	164	0
4	100	200	192	768	152	0	1	1	0	152	0
5	100	200	192	960	140	0	1	1	0	140	0
6	100	200	192	1152	128	0	1	1	0	128	0
7	100	200	192	1344	116	0	1	1	0	116	0
8	100	200	192	1536	104	0	1	1	0	104	0
9	100	200	192	1728	92	1	0	1	1	100	0
10	100	200	192	1920	80	1	0	1	1	100	0

Figure 303 shows the behavior of the slope generator and its control relationship with the DCSG block. Please take note that CSG_SLGEN_MATCH<11:0> will store the value of DCSG_TH register that causes a toggle of the CMP_0 based on the STEP_SIGN. This value (CSG_SLGEN_MATCH<11:0>) will be kept for the current slope generation cycle. It will be updated after the next SLGEN_EN_TRIGGER is toggled, and the next match event occurs. CSG_SLGEN_MATCH<12> will be the signal to inform the SW whether the CSG_SLGEN_MATCH<11:0> is a valid result. For example, if the STEP_SIGN is equal to 0, the FSM will look for a 1 to 0 transition to store the value. If the STEP_SIGN is equal to 1, the FSM will look for the 0 to 1 transition to store the value. Bit 12 of CSG_SLGEN_MATCH will indicate that the data is validated for the slope generation cycle. CSG_SLGEN_MATCH <12> will be cleared when the next SLGEN_EN_TRIGGER is toggled.

***Note**: DCSG_TH register can be accessed through the AHB bus and slope generation function. When slope generation function is enabled through DCSG_SLGEN_CTL.SLGEN_EN bit, firmware should avoid writing to DCSG_TH register to prevent unexpected behavior when slope generation function is in operation.*

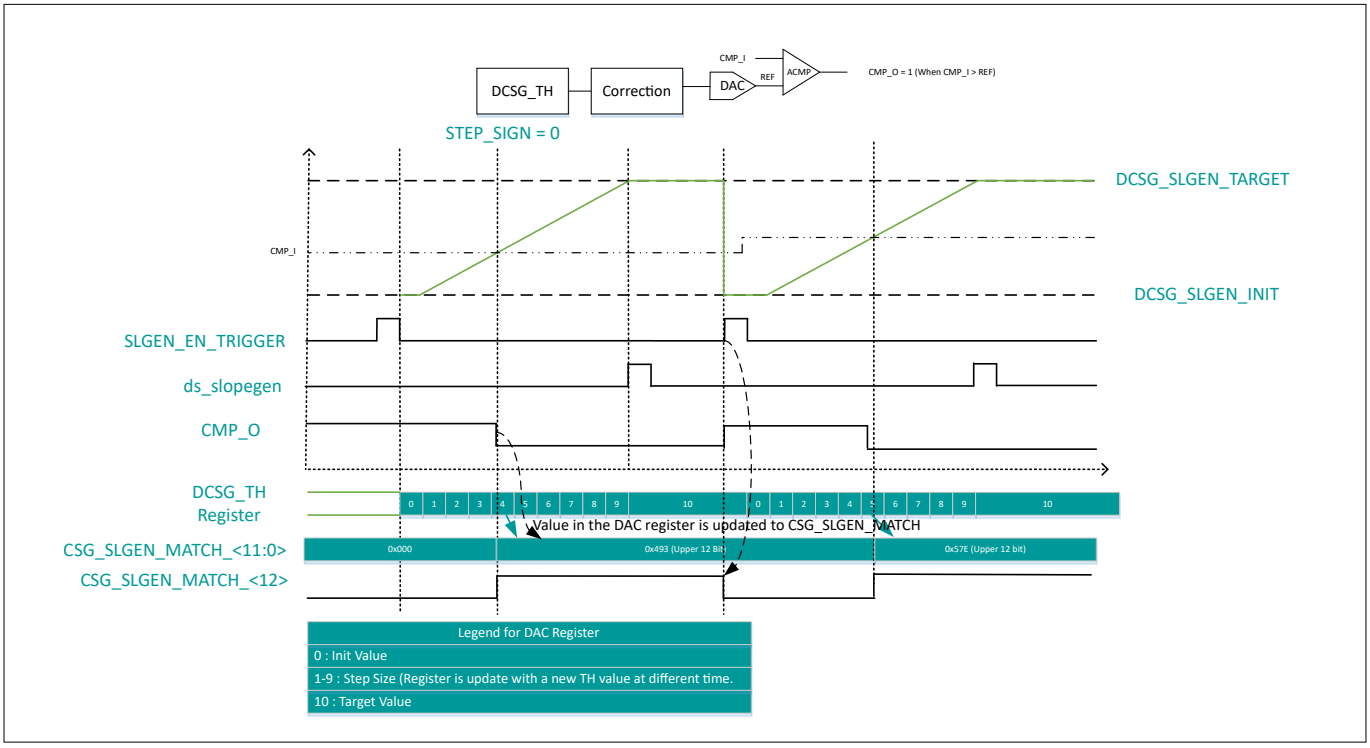


Figure 303 Example of slope generator behavior

Figure 304 shows the behavior of the slope generator when SLGEN_EN_TRIGGER is toggle in the mid of the slope generator operation. It will restart the slope generation from the initial value (DCSG_SLGEN_INIT).

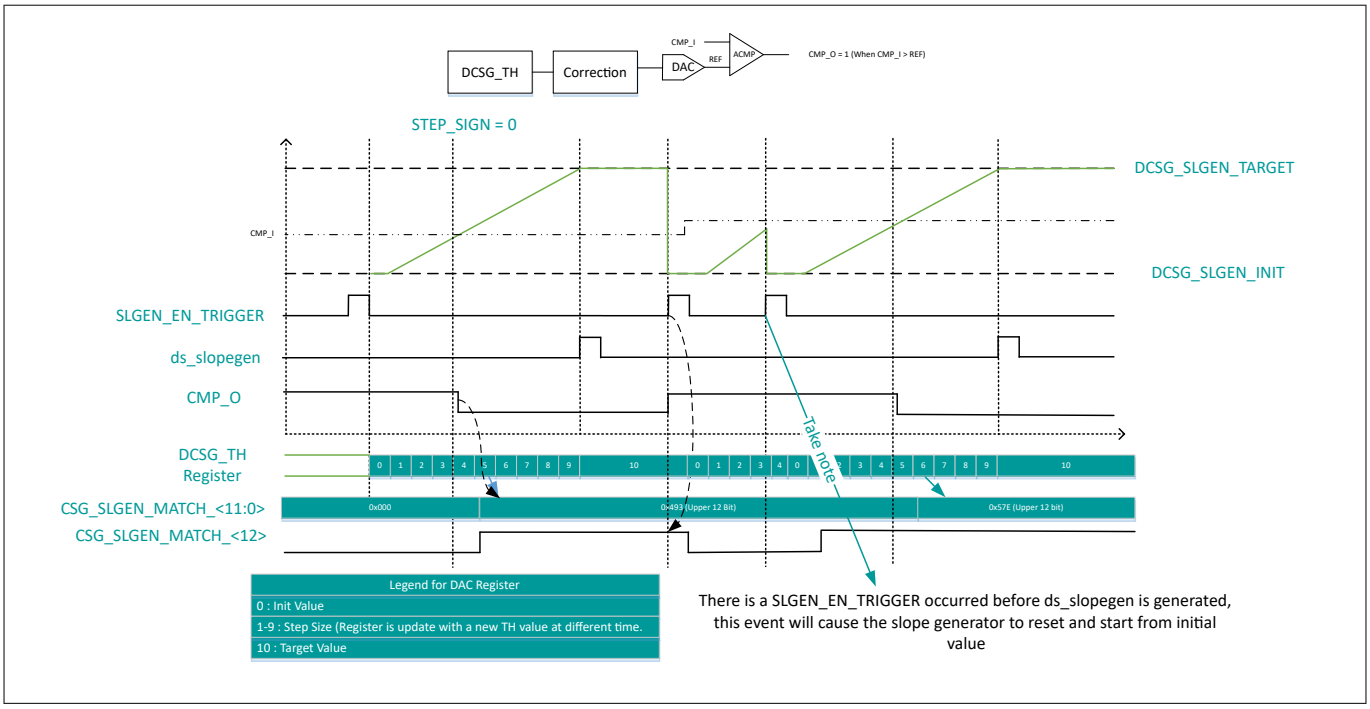


Figure 304 Example of slope generator behavior upon SLGEN_EN_TRIGGER

30.8.3.2.1 Slope Generator Handling of Glitch from the CMP_O

In certain scenarios, there may be a glitch from the CMP_O due to comparator trip prematurely because of a large current transient occurring when the power switches controlled by the PWM are initially turned on. This will result in false toggle of the CMP_0 which will cause the CSG_SLGEN_MATCH to store the wrong value of DCSG_TH register. The slope generator will handle this momentary glitch by checking the transition to be valid for at least 2 clock cycle. This covers the scenario where the glitch is less than one clock cycle (If the frequency is 200 MHz, then it is around 5 nsec).

DCSG_SLGEN_CTL.SIN_CLK_BLANK bit is used to enable the handling of the glitch feature.

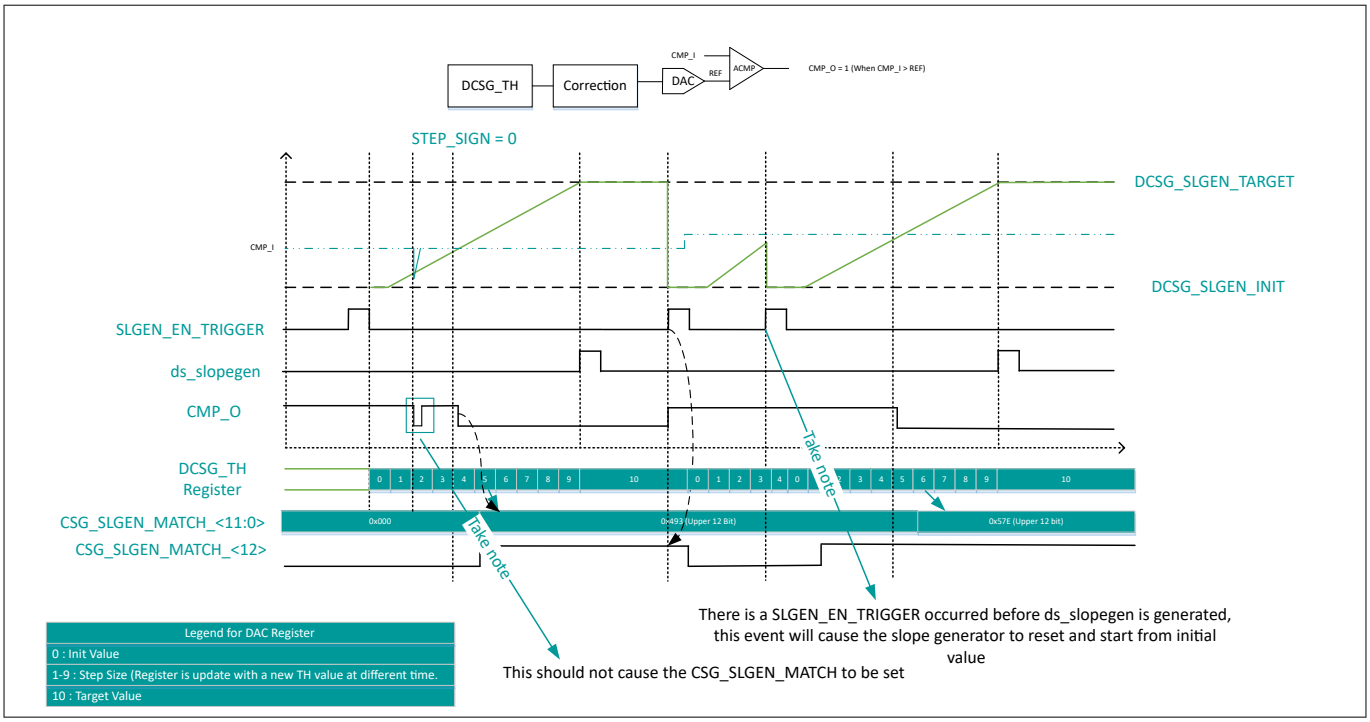


Figure 305 False trigger of CMP_0 that result in CSG_SLGEN_MATCH result

30.8.3.2.2 Handling of debug state

At the assertion of the debug signal the ATOP SS enters in the debug state. In this state, the slope generation is supposed to freeze the progress of the slope generation and resumes the operation when the debug signal is released.

Each DCSG group has their individual tr_debug_freeze_i. The enter of the debug state will be individual control based on the tr_debug_freeze_i signal that is assigned to the respective group.

The table tabulates the behavior of the debug mode based on DCSG_SLEN_CTL.DBG_FREEZE_EN bit and tr_debug_freeze_i signal.

Table 290 Behavior of debug mode for DCSG

	DBG_FREEZE_EN bit	tr_debug_freeze_i[NUM_TOTAL_GRP-1:0]
SLGEN operation continues	0	0
SLGEN operation continues	0	1
SLGEN operation pauses immediately	1	1
SLGEN operation continues	1	0

30.8.3.2.3 DCSG_TH timing requirement

Depending on the delta value of step, the analog module of the DCSG has a different timing requirement for the updating of the DCSG_SLICE.DCSG_TH.

For example, if the delta step is greater than 400 LSB, the timing requirement is 50 nsec to achieve a setting resolution to +/- 8 LSB. In the slope generation, the worst scenario for the maximum delta change will be the absolute value of DCSG_SLGEN_TARGET – DCSG_SLGEN_INIT regardless of whether when STEP_SIGN is 0 or 1

when the SLGEN_EN_TRIGGER is triggered. To allow the feasibility of a different response based on their application, the slope generation supports the following update duration based on DCSG_CNFG.DCSG_DAC_RATE. The following are some examples for the setting for DCSG_CNFG.DCSG_DAC_RATE.

***Note**: The following examples are based on INIT_SETTING=0, SEL_SRC=0, and CFG_LOW_SUPPLY=0. If INIT_SETTING, SEL_SRC, CFG_LOW_SUPPLY set at other value than 0, it will affect the init timing of the DCSG_IP, which has an impact on the DCSG_BUSY_O signal upon initialization. The subsequent threshold_wr_i after INIT will be blanking time (defined in INIT_SETTING) plus 2 clock cycles.*

DCSG_CNFG.DCSG_DAC_RATE 50 nsec interval: The slope generator will update the DCSG_SLICE.DCSG_TH register at every 50 nsec upon the start of the slope generator operation. Figure 306 shows the control of updating the DCSG_SLICE.DCSG_TH register at 50 nsec interval. Note that DCSG.init_i is generated by the FSM at the start of the slope generator operation.

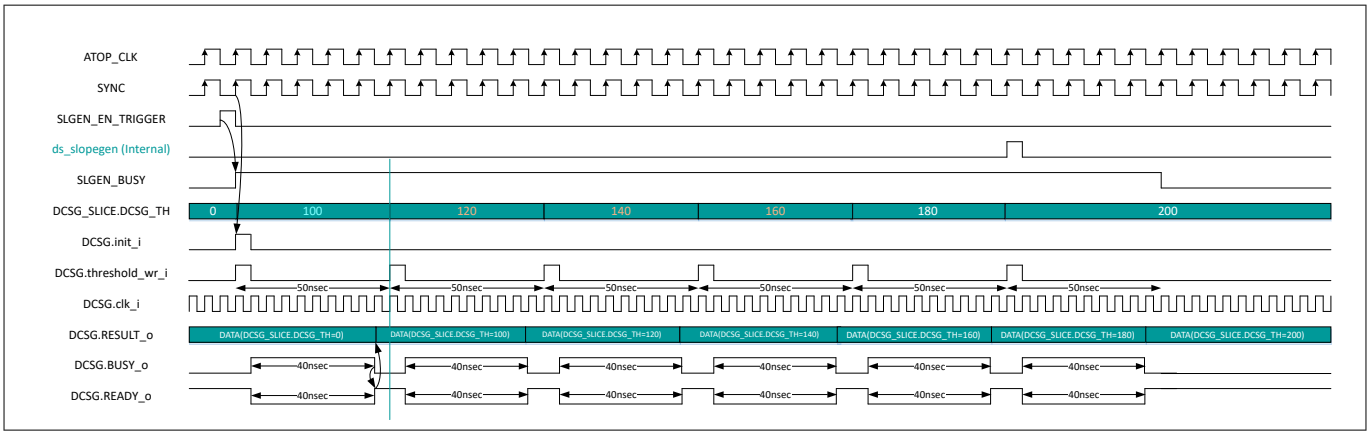


Figure 306 50 nsec interval update based on 200 MHz clock (DCSG_DAC_RATE = 5, INIT_SETTING=0, SEL_SRC=0)

DCSG_CNFG.DCSG_DAC_RATE 30 nsec interval: the slope generator will update the DCSG_SLICE.DCSG_TH register at every 30 nsec upon the start of the slope generator operation except the first update. The reason is because DCSG.init_i requires 50 nsec operation. Figure 307 illustrate the control of updating the DCSG_SLICE.DCSG_TH register at 30 nsec interval. Note that DCSG.init_i is generated by the FSM at the start of the slope generator operation.

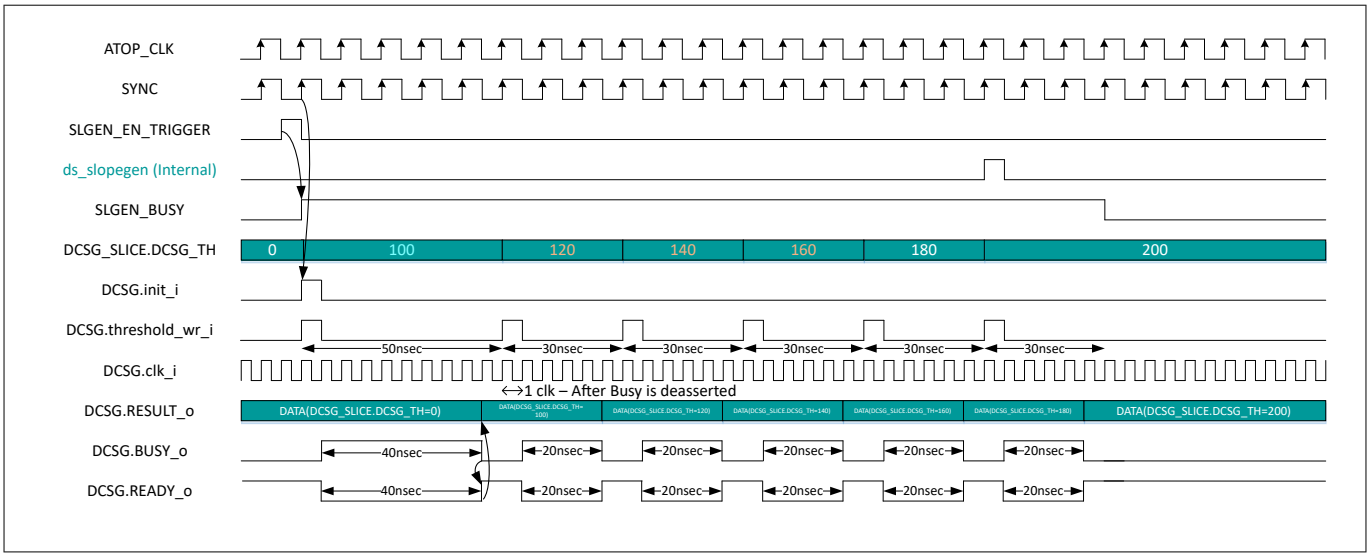


Figure 307 30 nsec interval update based on 200 MHz clock (DCSG_DAC_RATE=3, INIT_SETTING=0, SEL_SRC=0)

DCSG_CNFG.DCSG_DAC_RATE 60 nsec interval: The slope generator will update the DCSG_SLICE.DCSG_TH register at every 60 nsec upon the start of the slope generator operation. Figure 308 shows the control of updating the DCSG_SLICE.DCSG_TH register at 60 nsec interval. Note that DCSG.init_i is generated by the FSM at the start of the slope generator operation.

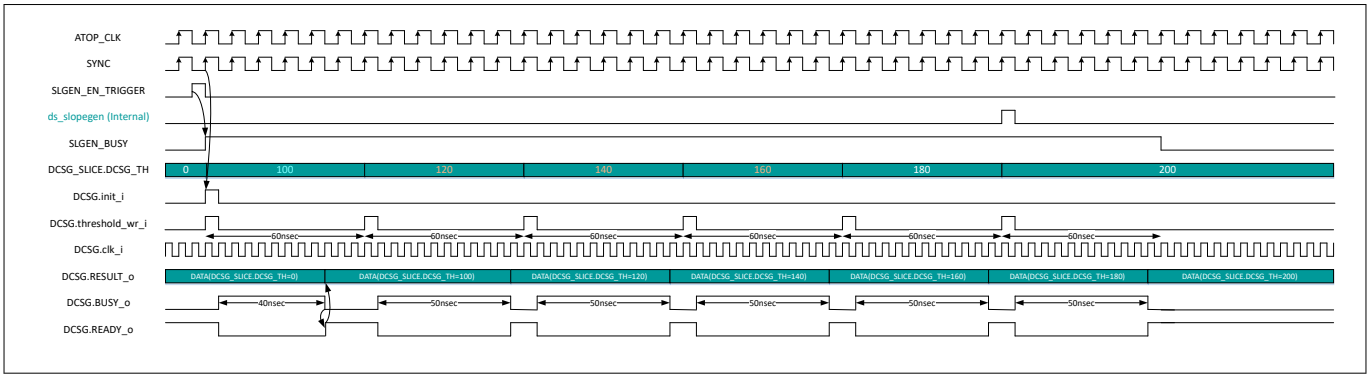


Figure 308 60 nsec interval update based on 200 MHz clock (DCSG_DAC_RATE=6, INIT_SETTING=0, SEL_SRC=0)

DCSG_CNFG.DCSG_DAC_RATE 20 nsec interval: The slope generator will update the DCSG_SLICE.DCSG_TH register at every 20 nsec upon the start of the slope generator operation except the first update. The reason is because DCSG.init_i requires 50 nsec operation. The following figure illustrates the control of updating the DCSG_SLICE.DCSG_TH register at a 20 nsec interval.

Note:
- *DCSG_DAC_RATE of 2, 1 and 0 will result in 20 nsec interval write operation. The only major difference in the three setting is that the data will be provided earlier by the DCSG.*
- *DCSG_DAC_RATE setting of 0, the ready signal is always high as the comparator is providing result at every clock cycle. There is a need for FSM to check for the transition at every clock cycle after DCSG.threshold_wr_i is asserted.*

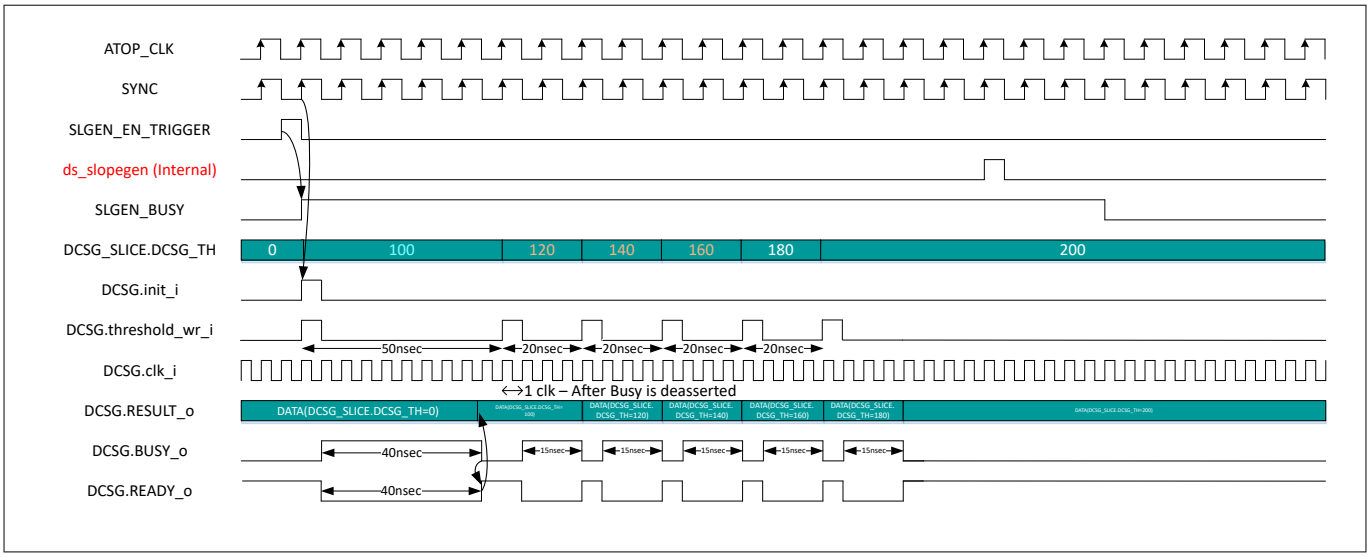


Figure 309 **20 nsec interval update based on 200 MHz clock (DCSG_DAC_RATE=2, INIT_SETTING=0, SEL_SRC=0)**

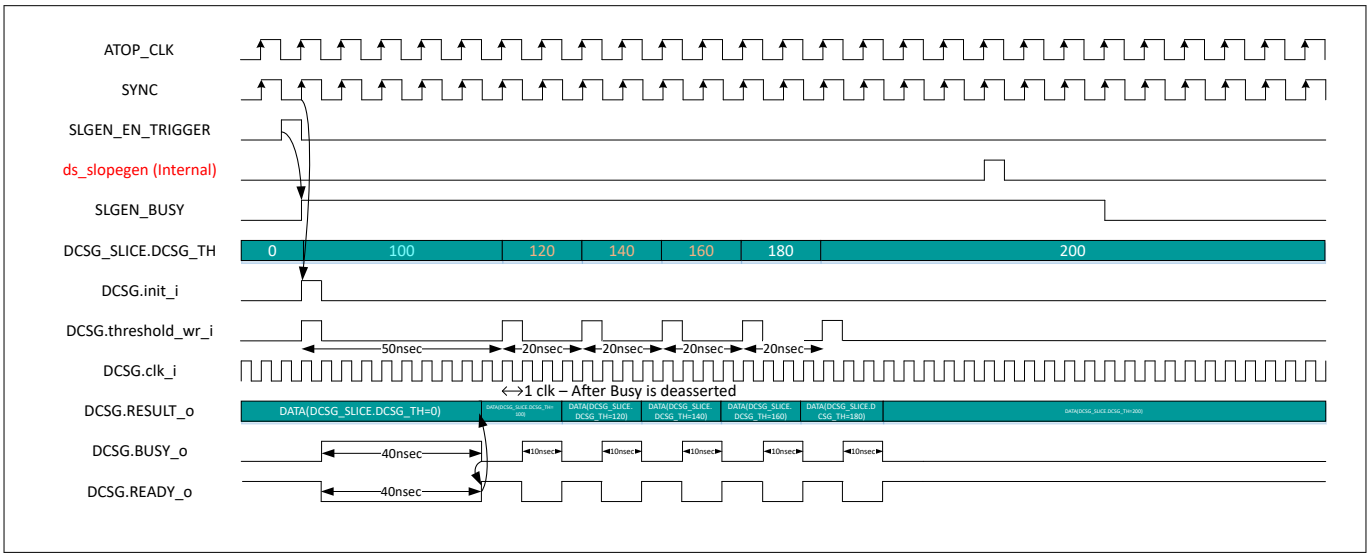


Figure 310 **20 nsec interval update based on 200 MHz clock (DCSG_DAC_RATE=1, INIT_SETTING=0, SEL_SRC=0)**

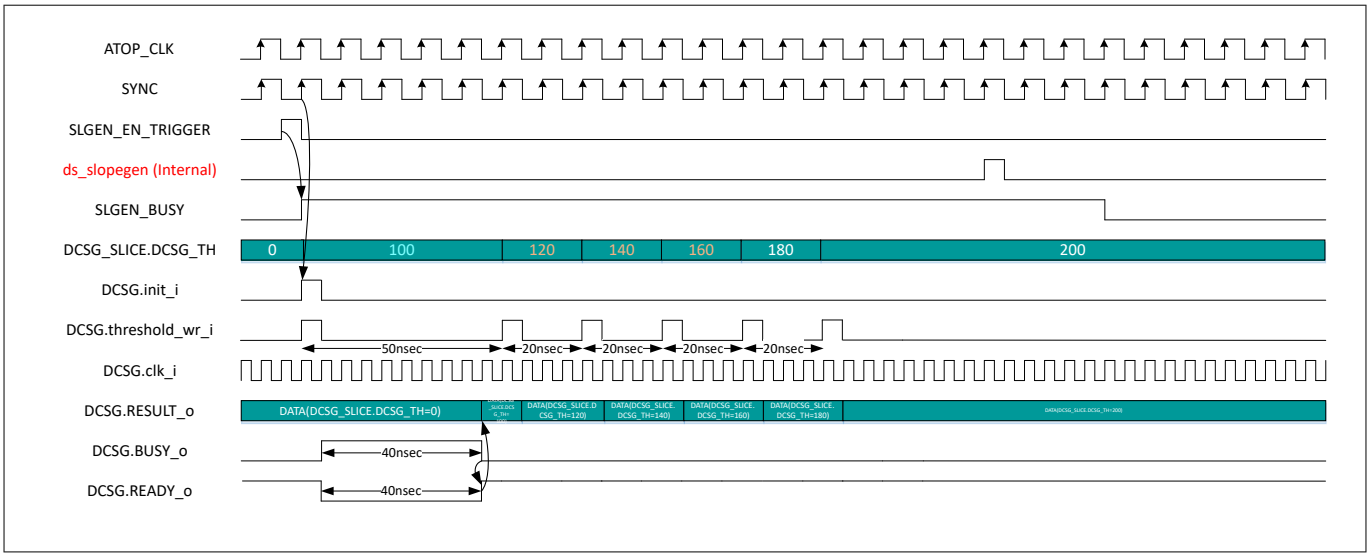


Figure 311 **20 nsec Interval update based on 200 MHz clock (DCSG_DAC_RATE=0, INIT_SETTING=0, SEL_SRC=0) special handling**

30.8.3.2.4 AUTO_RESET feature

In the slope generation, the worst scenario for the maximum delta change will be the absolute value of DCSG_SLGEN_TARGET – DCSG_SLGEN_INIT regardless whether when STEP_SIGN is 0 or 1 when the SLGEN_EN_TRIGGER is triggered, slope generation has the AUTO_RESET feature where it will set the slope generator to DCSG_SLGEN_INIT value upon the ds_slopegen is triggered.

***Note**: AUTO_RESET feature will support all the interval that is defined in DCSG_CNFG.DCSG_DAC_RATE*

Figure 312 shows the behavior of the slope generator and its control relationship with the DCSG block for the AUTO_RESET feature.

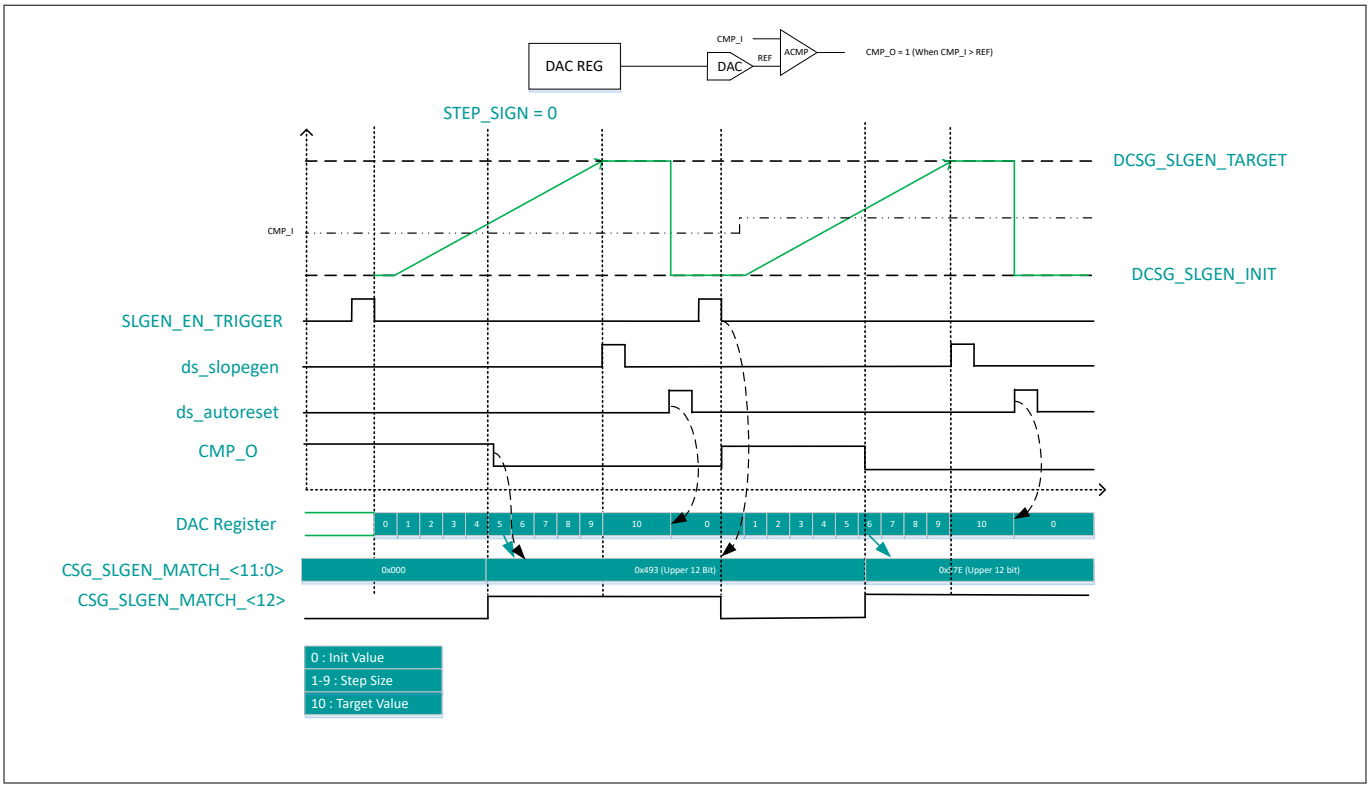


Figure 312 AUTO_RESET behavior

Figure 313 shows an example of AUTO_RESET with 30 nsec interval setup, it can be observed that AUTO_RESET is activated at the end target value.

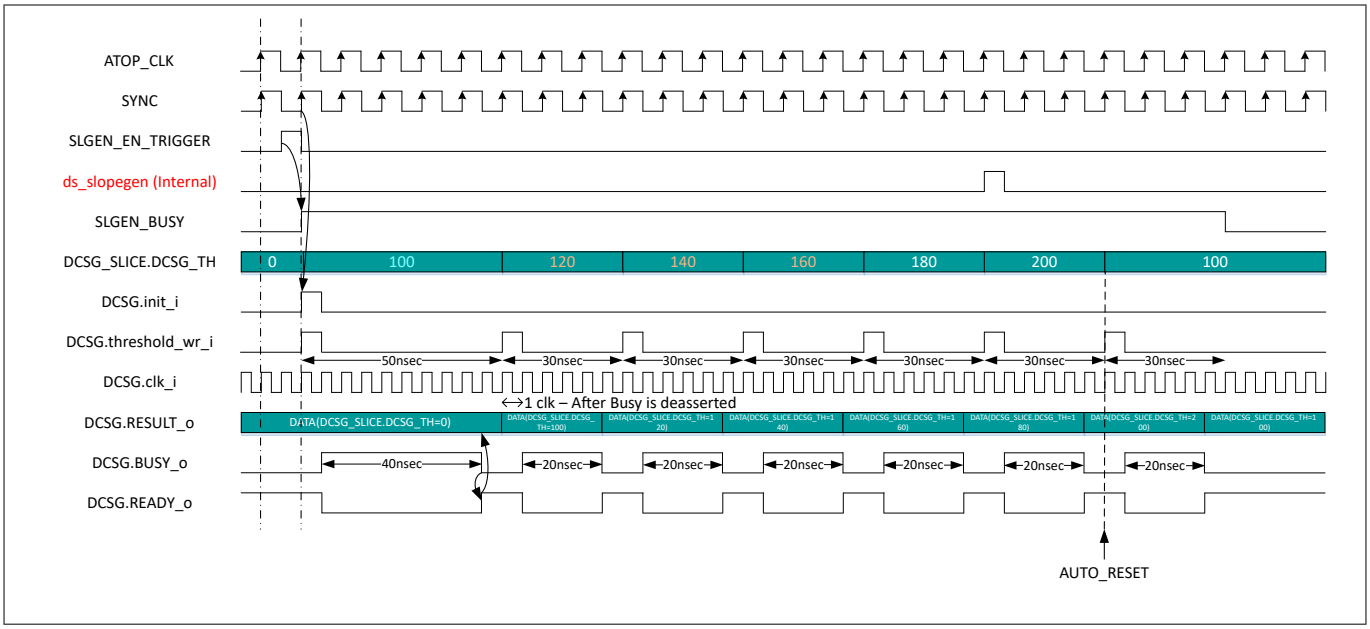


Figure 313 Example of AUTO_RESET control with 30 nsec interval setup based on 200 MHz clock (DCSG_DAC_RATE=3)

30.8.3.2.5 SLGEN_MAP feature

SLGEN_MAP feature allows SLGEN to be mapped to DCSG_IF[x]. The user can configure DCSG_SLGEN_CNFG.SLGEN_MAP to map SL_GEN. This feature provides convenient mapping options to the customer.

***Note**: Software needs to configure the SLGEN_MAP for the mapping.*

30.8.3.3 DCSG interleave blanking feature

The DCSG slice embedded DAC need to be refreshed at least every 50 usec (configured through DCSG_CNFG.CFG_REFRESH_RATE, the refresh duration is around 50 nsec). This means that during this interval, the DCSG is unable to monitor the analog input pad that it has been assigned to. This period is also known as DCSG blanking time.

***Note**:*
- *The refresh duration can be configured from 5 usec to 50 usec based on 200 MHz PPCA clock*
- *DCSG_CNFG.CFG_REFRESH_RATE for the all the DCSG slice need to be set with the same DCSG_CNFG.CFG_REFRESH_RATE setting.*

For application that require the DCSG to monitor a period longer than DCSG_CNFG.CFG_REFRESH_RATE, there is a need to use two DCSG to monitor the same analog pad to overcome the blanking period. User can configure DCSG_BLANK_CTL.DCSG_BLANK_EN to enable interleave blanking support for DCSG_GRP.

When DCSG_ BLANK _EN is set to ‘1’ for DCSG_GRP_0, DCSG<5> will be used as the DCSG for the interleave operation for DCSG<0:4>. When DCSG_ BLANK _EN is set to ‘1’ for DCSG_GRP_1, DCSG<2> will be used as the DCSG for the interleave operation for DCSG<0:1> in DCSG_GRP_1.

The DCSG DIGITAL IF implements a delay counter to shift the start of the refresh counter. The delay counter should be synchronized with the master DCSG refresh counter. Updating the DAC will restart the counter and delay.

Figure 314 shows the block diagram on how to get rid of the required DCSG blanking time by using a redundant DCSG for DCSG_GRP0 case. For DCSG_GRP1, the similar concept is used as well. DCSG<5> connection will be set up in DCSG<2> for DCSG_GRP1.

***Note**:*
- *During interleave operation, the output of the DCSG<5> will be assigned to DEFAULT value (CMP_FLAG from the comparator output)*
- *DCSG interleave blanking feature will not apply to DCSG when SLGEN_EN is set to 1B*

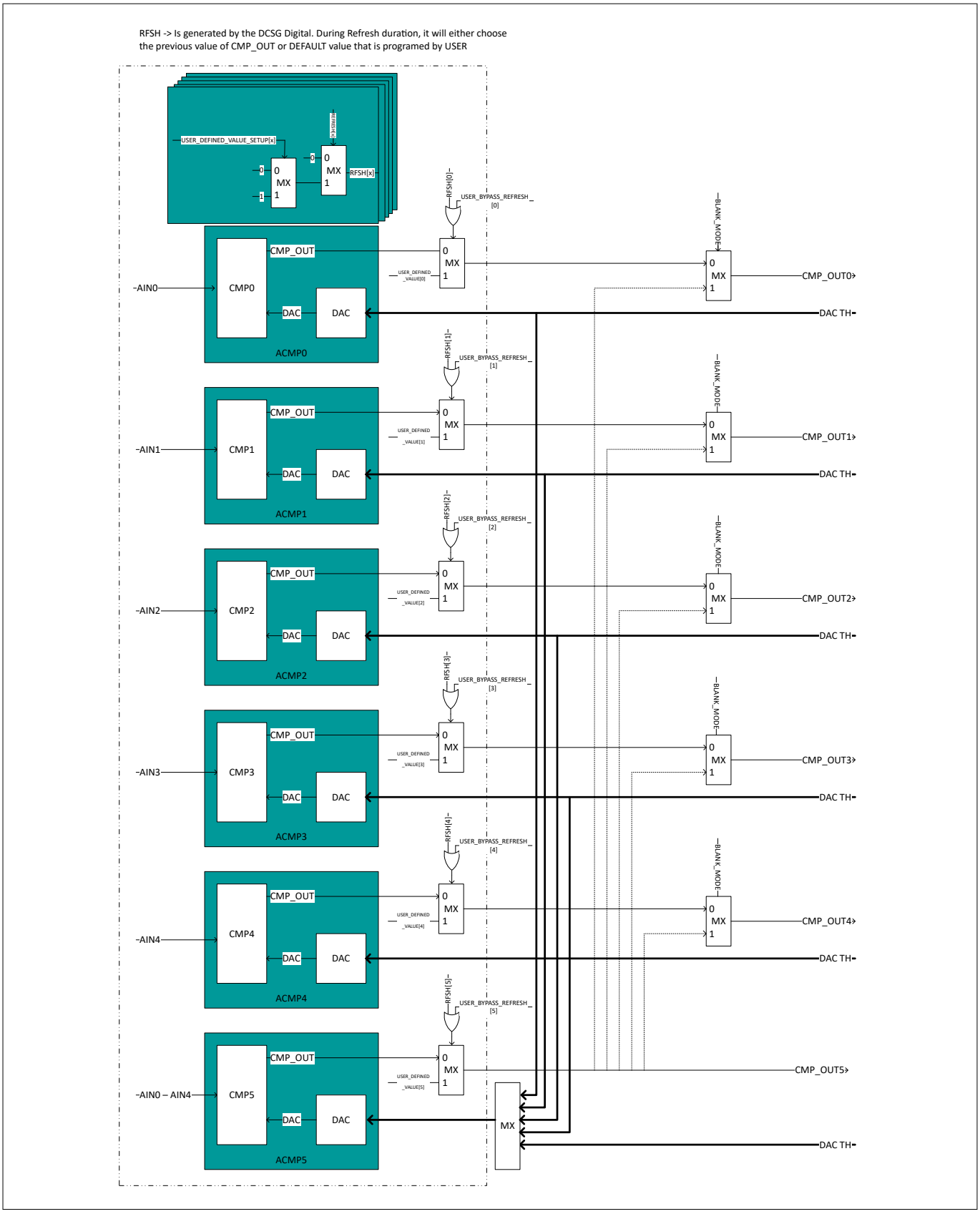


Figure 314 Block diagram for blanking control for DSCG_GRP0

30.8.3.3.1 CFG_REFRESH_RATE consideration

The CFG_REFRESH_RATE is set to 50 usec based on 200 MHz clock. The CFG_REFRESH_RATE register allows firmware designer to adjust the refresh rate based on the PPCA clock setup. Table 291 tabulate the setting based on the clock frequency. When the clock frequency is 200 MHz, firmware will set to “0”, when the clock frequency is 150 MHz, firmware will set to “1”. This will ensure that at different frequencies, the CFG_REFRESH_RATE is maintained at 50 nsec.

Table 291 Refresh rate setting based on PPCA frequency for DCSG

PPCA clock frequency (MHz)	CFG_REFRESH_RATE (50 usec)	CFG_REFRESH_RATE (37.5 usec)	CFG_REFRESH_RATE (17.5 usec)
200	0 (10000*5 nsec)	1 (7500*5 nsec)	2
150	1 (7500*6.67 nsec)	2	3
70	3 (3500*14.29 nsec)	4	5
50	4 (2500*20 nsec)	5	6

30.8.3.3.2 Interleave blanking enable procedure

The following described the procedure for FW to enable the interleave blanking operation.

1. Enable by setting DCSG_BLANK_CTL.DCSG_BLANK_EN to “1”.
2. Enable the corresponding DCSG<4:0> that user wants to use in the application for DCSG_GRP_0 after the setup configuration is done. For DCSG_GRP_1, enable the corresponding DCSG<1:0>.
3. Once step 2 is completed, user enable DCSG<5> for interleave operation for DCSG_GRP_0. For DCSG_GRP_1, enable DCSG<2>.
4. The interleave operation will start operation once the above steps are completed.

30.8.3.4 DCSG Control

30.8.3.4.1 Slope generator control behavior

Figure 315 illustrates the control behavior interaction between the slope generation subsystem and interface data of DCSG. The slope generation subsystem generates DCSG.threshold_wr_i pulse signal whenever the subsystem wants to change the threshold of the DCSG.

While DCSG is updating the threshold of the DAC and performing the comparison after the DAC reference voltage has settled, BUSY_O will be asserted by DCSG. Once BUSY_O is deasserted, the subsystem writes a new threshold to the DCSG. To read the RESULT_O, subsystem can do it one cycle after BUSY_O is deasserted or upon trigger of READY_O. The data will remain in RESULT_O till the next trigger of READY_O.

Once BUSY_O is deasserted, the subsystem writes the next threshold. The period of DCSG.threshold_wr_i is governed by DCSG_DAC_RATE.

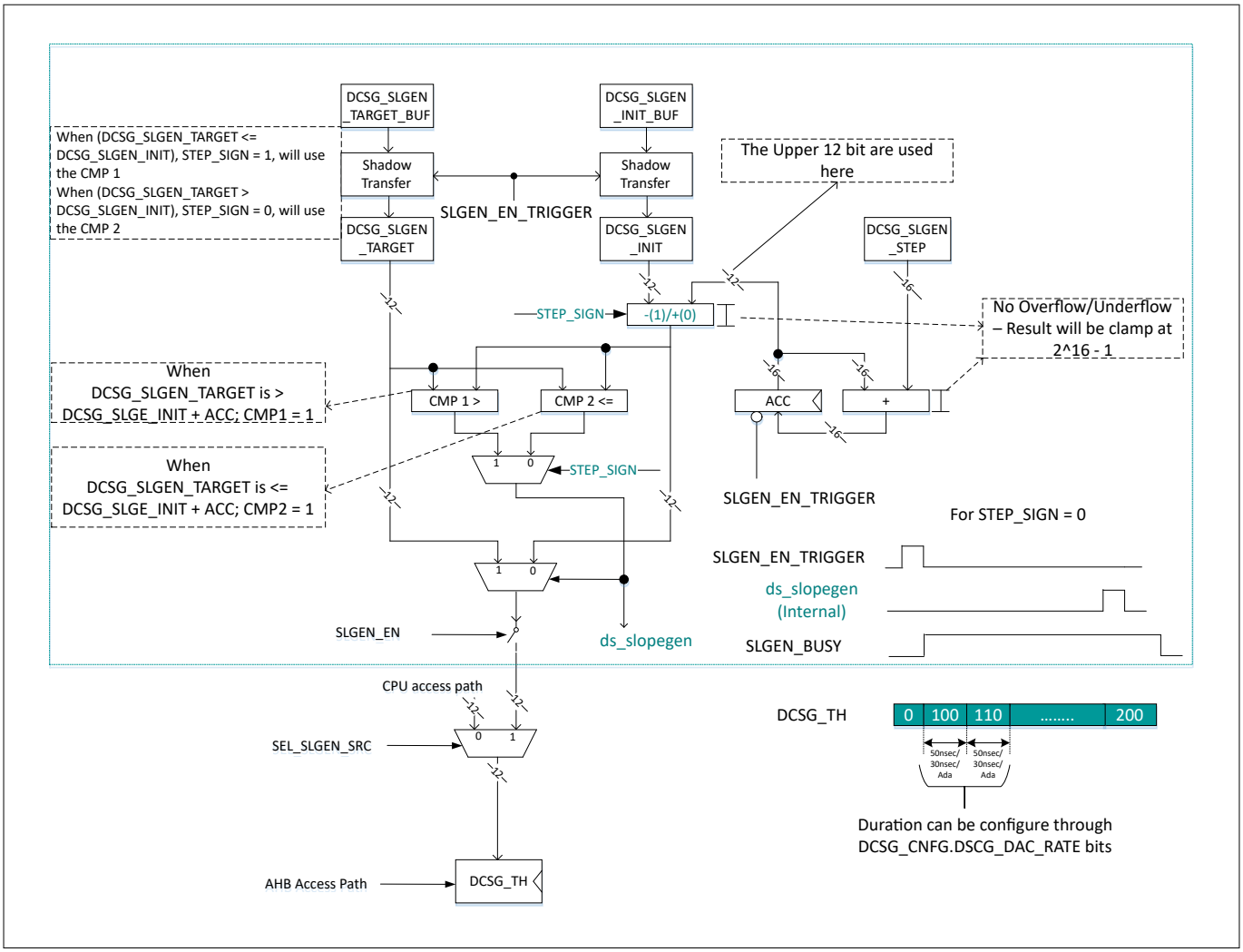


Figure 315 Implementation of slope generator

FSM needs to control the DCSG.threshold_wr_i based on the DCSG_DAC_RATE setting.

30.8.3.4.2 A vs B Control Behavior

To enable A vs B mode, DCSG_CNFG.TYPE_CH_SRC needs to be set to 101B. This will configure the DCSG to support A vs B operation.

Hysteresis operation

In the A_VS_B mode, the user can compare the two analog inputs from the user system. When the two analog inputs are equal and if there are noise couplings on the input, the output of the DCSG will toggle, which will be perceived as comparator oscillation as show as Figure 316.

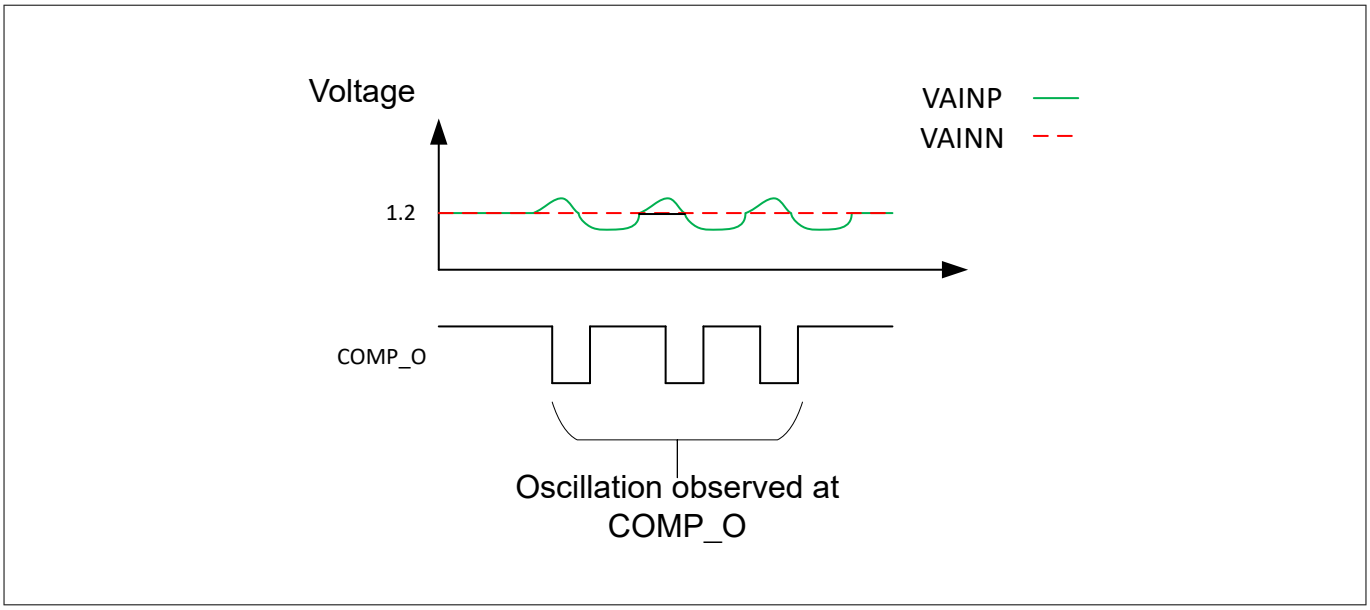


Figure 316 Oscillation behavior on A_VS_B mode operation

To prevent the oscillation behavior, DCSG_CNFG.HYST will be used to define the level of hysteresis to be added into cfg_threshold_i<11:0> to control the DCSG in A_VS_B mode to make it harder for the COMP_O to be toggle. For example, if a user sets DCSG_CNFG.HYST to be “2”, then 2 LSB will be added or subtracted depending on the status of the COMP_O.

The following are the rule that governs whether to add or subtract the term ROUNDDOWN(2^(DCSG_CNFG.HYST-1)) for the hysteresis operation.

- When the COMP_O is high, the subsystem will subtract ROUNDDOWN(2^(DCSG_CNFG.HYST-1)) from 2047
- When the COMP_O is low, the subsystem will add ROUNDDOWN(2^(DCSG_CNFG.HYST-1)) to 2047
- The subtraction operation will occur when there is a COMP_O transition from low to high
- The addition operation will occur when there is a COMP_O transition from high to low

Debounce operation

Other than DCSG_CNFG.HYST bit field, DCSG_CNFG.DEB is also used in hysteresis operation. The purpose of the DCSG_CNFG.DEB is for the subsystem to monitor the transition of the DCSG.COMP_O signal coming from the DCSG analog. It is used to monitor a valid transition on DCSG.COMP_O before the subsystem make the decision to modify the cfg_threshold_i of the DCSG. The following diagram illustrates the behavior of the DCSG_CNFG.DEB for the case where DCSG.CNG_DEB is set to ‘1’ and ‘0’. The similar behavior concept is applied to other settings of DCSG.CNG_DEB.

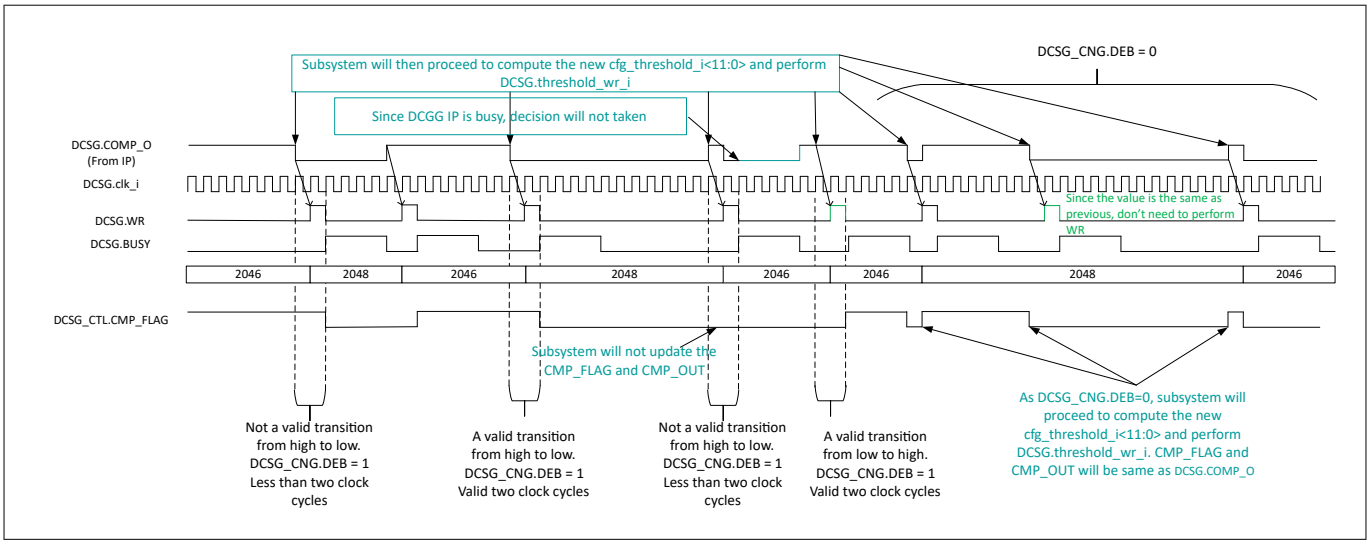


Figure 317 DCSG_CNFG.DEB behavior for DCSG_CNG.DEB = "1" and DCSG_CNG.DEB = "0"

Hysteresis control scheme

The following figures illustrate the hysteresis control operation for DCGG_CNFG.DEB is being set to "1", "2" and "0" scenario. The same control concept will apply to other setting of DCSG_CNFG.DEB.

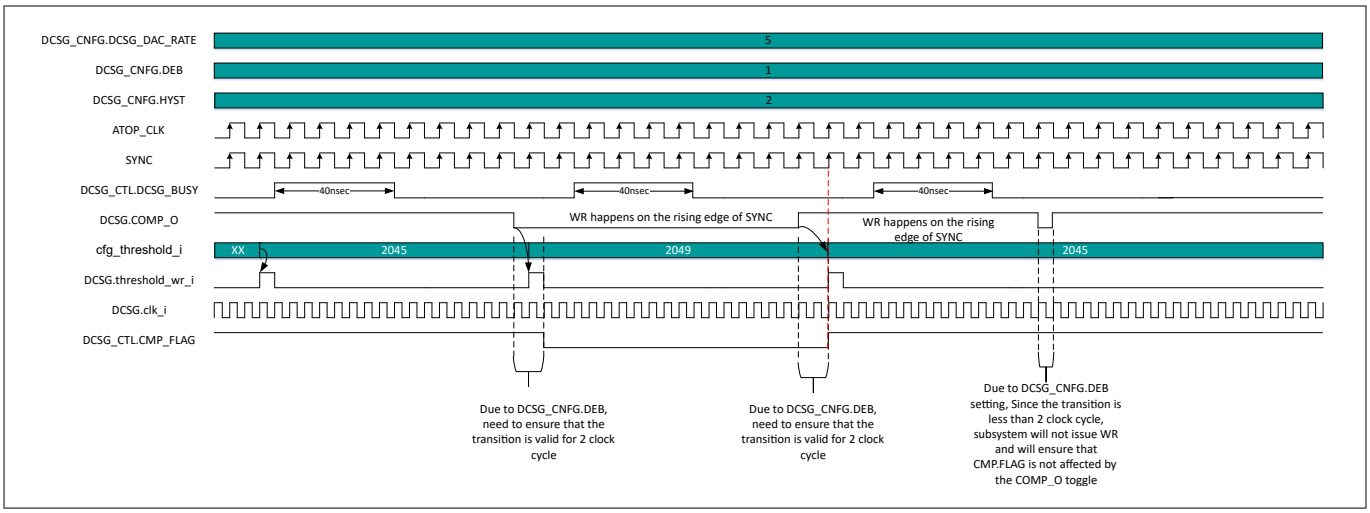


Figure 318 Subsystem hysteresis control behavior when DCSG_CNFG.DEB = 1

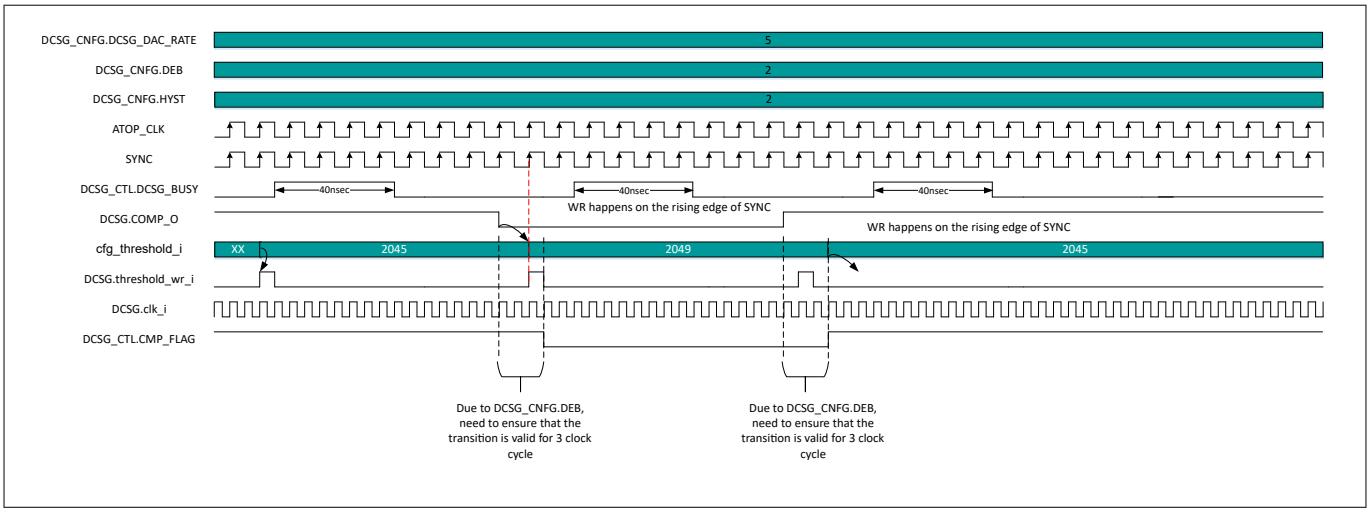


Figure 319 Subsystem hysteresis control behavior when DCSG_CNFG.DEB = 2

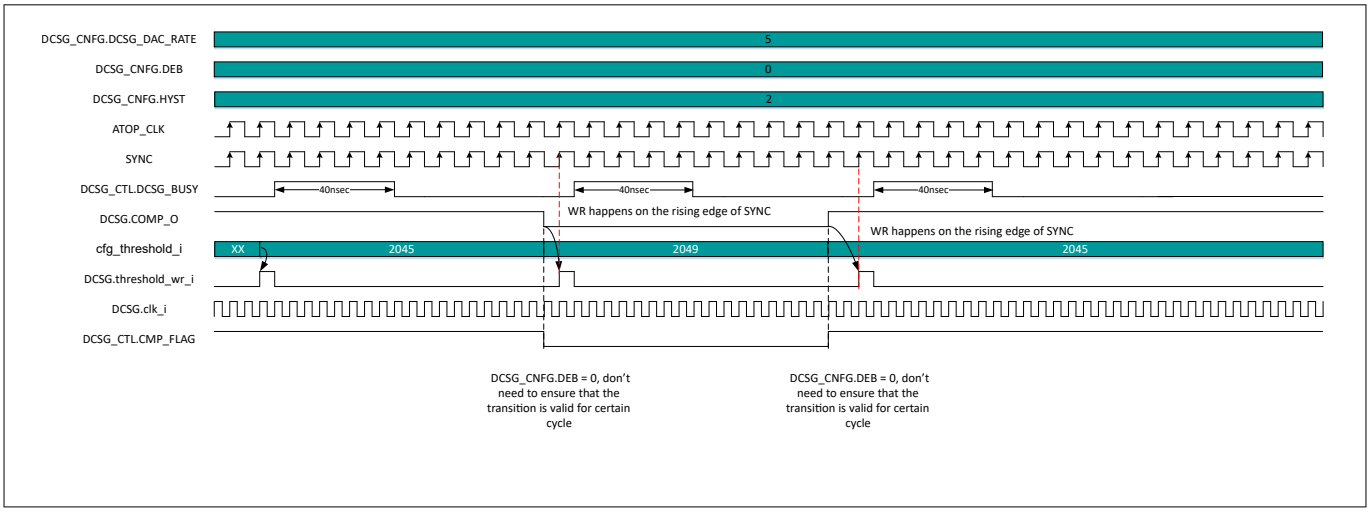


Figure 320 Subsystem hysteresis control behavior when DCSG_CNFG.DEB = 0

Be aware that this control scheme needs to be handled in the DCSG interleave blanking feature which is described in section DCSG interleave blanking feature.

30.8.3.4.3 Firmware control of DCSG

Figure 321 shows the control behavior interaction between the FW and DCSG.

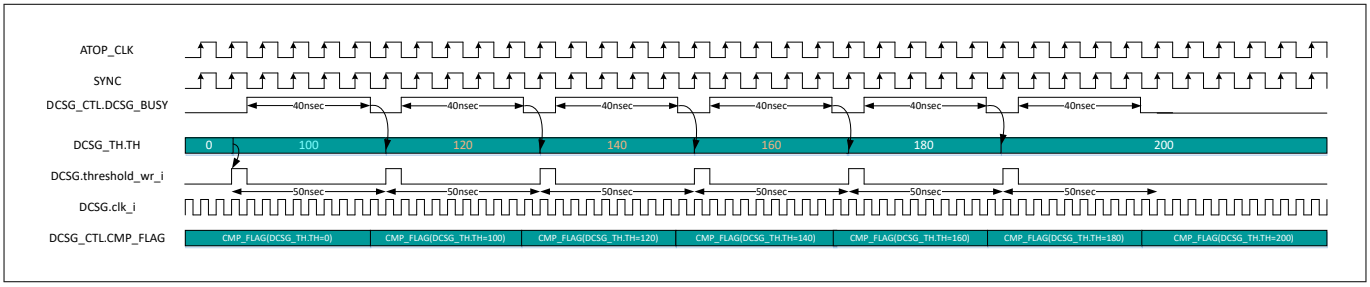


Figure 321 Firmware control of DCSG (DCSG_DAC_RATE = 5)

30.8.3.5 R2R DAC

R2R DAC module implements a Digital Analog Converter to support AC signal monitoring and/or generation of a reference Voltage. DAC signal is buffered and exported to the pins as well as to the DCSG module and ADC_0 (Channel 7).

R2R DAC features:

- 12-bit resolution with 10-bit accuracy
- 2MHz refresh rate
- 50pF load driver
- Single ended
- VDAC_OUT (max) = VREF or VDDA/2

The R2R DAC module comprises two sections:

- **DAC_ANA**: This section implements the DAC analog functions:
  - The DAC
  - The splitter or inverter
- **DAC_IF**: The DAC_IF implements:
  - The AHB Slave IF
  - The configuration register (CNFG)
  - The control register (CTL), to enable/disable the DAC function
  - The data register (DATA), to set the DAC output value
  - The clock domains (PPCA_CLK and ATOP_CLK) adapter
  - The DAC control interface

30.8.3.5.1 Using the R2R DAC

Figure 322 shows the use cases of R2R DAC. There are two R2R DAC in ATOPSS. They are connected to the AI0 and AI1 respectively. In the use case, the R2R DAC is used with DCSG and IDAC, which tabulates the relationship of the R2R DAC in the four-use case scenario.

Note:

- *EN_VREF_TO_BUF and EN_DAC_O_EN_BUF bit in the DAC_CNFG register select the source of the reference voltage to the buffer. At any time, only one of these bits needs to be set to 1. Do not set both bits to "1"*
- *User should disable the R2R DAC (DAC_CTL.DAC_EN) before switching mode (DAC_CNFG.R2RDAC_CNFG). This is to prevent unexpected behavior*
- *R2RDAC_startup and R2RDAC_BUF_startup time when powering up the R2RDAC and Buffer. User shall wait for it to be powered up before operating the R2R DAC and buffer. This is to prevent unexpected behavior*
- *R2R_DAC is under DCSG_GRP_0 and DCSG_GRP_1. Thus, we need to enable the DCSG under each group to use it. This is due to DAC digital interface block requires DCSG clock supply to propagate the selected internal DAC value to the analog block based on the DAC configuration. Hence at least one of the DCSG in the same DCSG group must be enabled to provide the required clock*
- *The R2R DAC must be enabled prior to enabling the R2R DAC buffer to ensure that they do not switch at the same time*

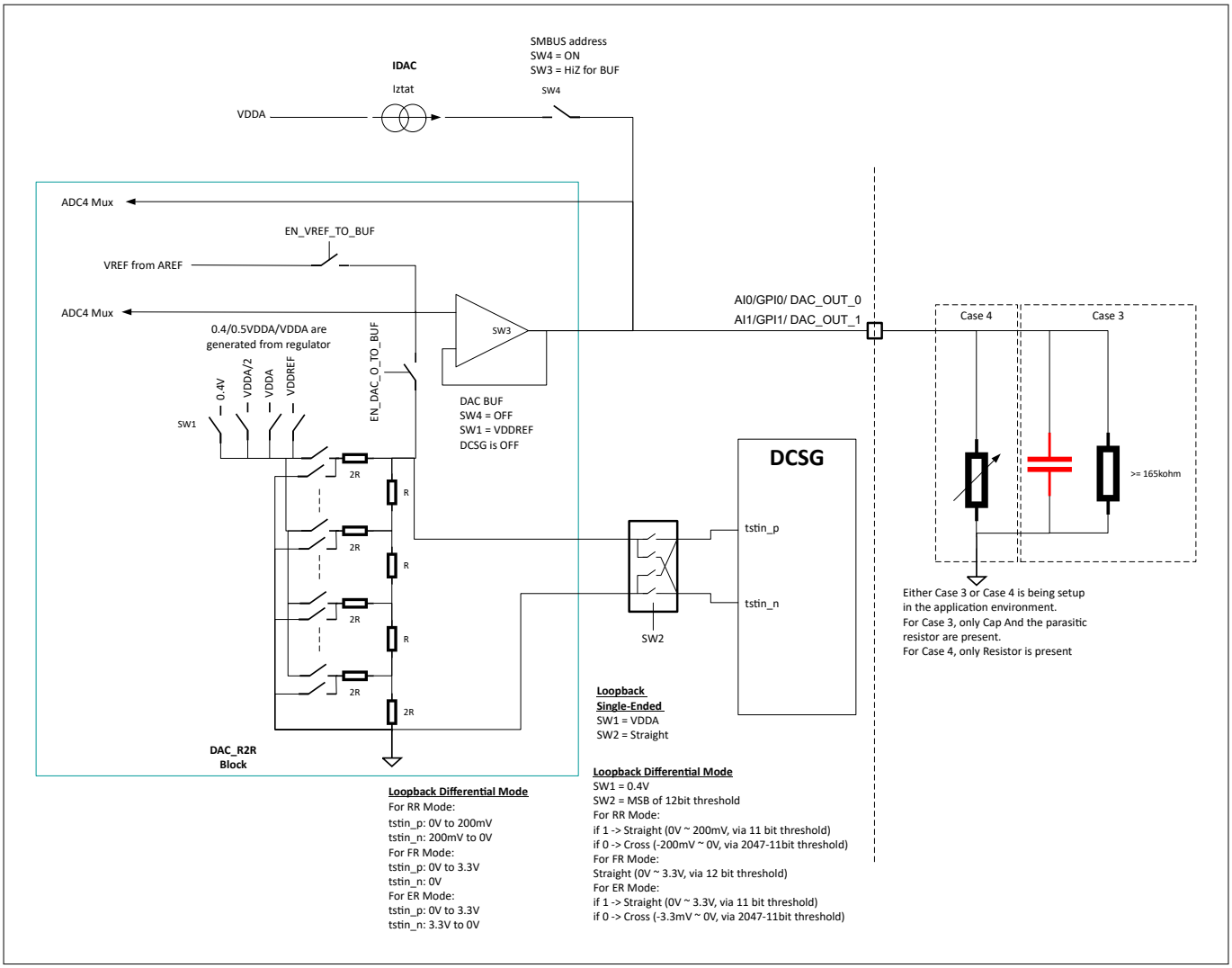


Figure 322 Use case of R2R DAC

Table 292 Use case scenario

Case	Scenario	SW1 (DAC_CNFG.R2RDAC_CNFG)	SW2 (DCSG_CNFG.SEL_SRC)	SW3 (DAC_CTL.DAC_BUFF_EN)	SW4 (IDAC_CNFG)	DCSG	Remarks
1	Loop Back Differential Case	0.4 V or VDDA	Straight and Cross Configuration	Off (HiZ)	Open	On. DCSG is enabled	Refer to Figure 322
2	Loop Back Single Ended	VDDA	Straight Configuration	Off (HiZ)	Open	On. DCSG is enabled	-

(table continues...)

Table 292 (continued) Use case scenario

Case	Scenario	SW1 (DAC_CNFG.R2RDAC_CNFG)	SW2 (DCSG_CNFG.SEL_SRC)	SW3 (DAC_CTL.DAC_BUFF_EN)	SW4 (IDAC_CNFG)	DCSG	Remarks
3	Drive External Load (CAP only)	- VDDA/2 - VREF	Open	On	Open	Don't care (Either on or off)	DCSG is working independently from the Buffer
4	PM Bus Measurement (Res only)	Don't care	Open	Off (HiZ)	Open	Don't care (Either on or off)	DCSG is working independently from the IDAC

30.8.3.5.2 Buffer offset correction

For an application where a buffer is used to drive external load which is is described in section Using the R2R DAC. It is important that the offset of the buffer is taken into consideration when the buffer is enabled (DAC_CTL.DAC_BUFF_EN = 1 and DAC_CNFG.BUF_OFFSET_DIS = 0). The offset correction for buffer covers from -64 to 63 LSB.

The formula is expressed below:

grpX_r2r_dac_ctrl_i<11:0> = DAC_GAIN * DAC_DOUT + Signed DAC_OFFSET + Signed BUFFER_OFFSET

Table 293 shows the mapping for the DAC_GAIN and DAC_OFFSET registers to be used based on the R2RDAC_CNFG_x setting.

Table 293 BUFFER_OFFSET register mapping based on R2RDAC_CNFG mode

R2RDAC_CNFG	BUFFER_OFFSET	Remarks
R2RDAC_CNFG_0	By-Pass	Not a valid use case
R2RDAC_CNFG_1	BUFFER_OFFSET_1	Not a valid use case
R2RDAC_CNFG_2	BUFFER_OFFSET_1	DAC_BUFF_EN = 1
R2RDAC_CNFG_3	BUFFER_OFFSET_2	DAC_BUFF_EN = 1
R2RDAC_CNFG_4	BUFFER_OFFSET_1	Not a valid use case. In application, DAC_BUFF_EN will be set to 0.
R2RDAC_CNFG_5	BUFFER_OFFSET_1	Not a valid use case. In application, DAC_BUFF_EN will be set to 0.
R2RDAC_CNFG_6	BUFFER_OFFSET_1	DAC_BUFF_EN = 1
R2RDAC_CNFG_7	BUFFER_OFFSET_2	DAC_BUFF_EN = 1

The user has the option to disable BUFFER offset correction through the DAC_CNFG register.

***Note**: The dynamic range of the BUFFER range starts from r2rdac_buffer_out_range (Min) to r2rdac_buffer_out_range (Max), for more detail of the range please refers to the HSIP_Op_Conditions. SW needs to ensure that the DAC_DOUT.DOUT program is within the dynamic range of the BUFFER output.*

30.9 Registers summary

Table 294 ADC registers summary

Register	Function
ADC_CTL	ADC Control Register
ADC_CNFG	ADC Configuration Register
ADC_AUX_ALT_AUX_CNFG	ADC Aux and ALT Aux Configuration Register
ADC_CNV_CNFG	ADC Conversion Configuration Register
ADC_SAMPLING_MASK	ADC Sampling Mask register
ADC_TRIGGER	ADC Channel Trigger register
ADC_CH_CNFG0	ADC Channels group 0 to be converted Configurations registers
ADC_CH_CNFG1	ADC Channels group 1 to be converted Configurations registers
ADC_AUX_CH_CNFG0	ADC Auxiliary Channels to be converted configuration registers
ADC_ALT_AUX_CH_CNFG	ADC AUX Alternate Channel to be converted configuration register
ADC_DATA	ADC Converted Data Channels
ADC_AUX_DATA	ADC Auxiliary Data Channels
ADC_ALT_AUX_DATA	ADC Alternate Auxiliary Data Channels
ADC_INTR	Interrupt request register
ADC_INTR_SET	Interrupt set request register
ADC_INTR_MASK	Interrupt mask register
ADC_INTR_MASKED	Interrupt masked request register
ADC_SIGN_UNSIGN_CNFG	ADC Channel Signed and Unsigned configuration Register
AUX_SIGN_UNSIGN_CNFG	AUX Channel Signed and Unsigned configuration Register
AUX_ALT_SIGN_UNSIGN_CNFG	AUX Alternate Channel Signed and Unsigned configuration Register
AFE_CTL	AFE Control register
AFE_CNFG	AFE Configuration register

Table 295 AFLT registers summary

Register	Function
AFLT_CTL	Low Pass Filter Control Register
AFLT_CNFG	ADC Filter Module Configuration register
LPF_ALFA	Low Pass Filter Bandwidth configuration Register
LPF_DOUT	Low Pass Filter Data Output
CICF_CNFG	CIC3 Configuration register
CICF_DOUT	CIC3 Filter Data Output
CICF_UB_TH	CIC3 Upper bound threshold register
CICF_LB_TH	CIC3 Lower bound threshold register
AVGF_CTL	Average Filter CTL register
AVGF_CNFG	Average Filter Configuration Register (N,M,K)
AVGF_DOUT	Average Filter Data Output
LIF_CNFG	Linear Interpolator Filter Configuration register
LIF_DOUT	Linear Interpolator Filter Data Output
MED_CNFG	Median Filter Configuration register
MED_DOUT	Median Filter Data Output
ADC_MIN_MAX	ADC Min Max detect value Register

Table 296 DCMP registers summary

Register	Function
DCMP_CTL	Digital Comparator Control register
DCMP_CNFG	Digital Comparator Configuration register
DCMP_TRIGGER	Digital Comparator Trigger register
DCMP_TH	Digital Comparator Threshold Configuration Register
DCMP_ACC_TH	Digital Comparator Threshold Configuration Register

Table 297 IDAC registers summary

Register	Function
IDAC_CTL	IDAC Control register
IDAC_CNFG	IDAC Configuration register

Table 298 AREF registers summary

Register	Function
AREF_CTL	AREF Control register
VDDA_STATUS	Analog Voltage Status
AREF_ANA_CTL	Analog Control

Table 299 GPI registers summary

Register	Function
GPI_CTL	GPI Control register

Table 300 DCSG registers summary

Register	Function
DCSG_CTL	Analog Comparator Control Register
DCSG_CNFG	Analog Comparator Configuration Register
DCSG_TH	Analog Comparator Threshold Configuration Register
DCSG_TEST	Analog Comparator Test Configuration Register
DCSG_SLGEN_CTL	Slope Generation Control Register
DCSG_SLGEN_CNFG	Slope Generation Configuration Register
DCSG_SLGEN_INIT	Slope Generation Initial Start Value Register
DCSG_SLGEN_TARGET	Slope Generation Final Target Value Register
DCSG_SLGEN_INIT_BUF	Slope Generation Initial Start Value Buffer Register
DCSG_SLGEN_TARGET_BUF	Slope Generation Final Target Value Buffer Register
DCSG_SLGEN_STEP	Slope Generation Step Increment/Decrement Register
DCSG_SLGEN_MATCH	Slope Generation Match Value
DCSG_BLANK_CTL	Analog Comparator Blanking Control Register
DAC_CTL	DAC Control Register
DAC_CNFG	DAC Configuration Register
DAC_DOUT	DAC Output Data Register

***Note**: When DCSG_CTL.DCSG_BUSY is '1', the user shall not perform a write operation to the DCSG_TH register.*

31 PPCA Custom Logic Block (CLB)

CLB module implements dedicated hardware support for specific functions or control algorithms, like Digital Current Observer used in the Totem Pole PFC Controller, or Matrix Converter Commutation, requiring a very high computational resource.

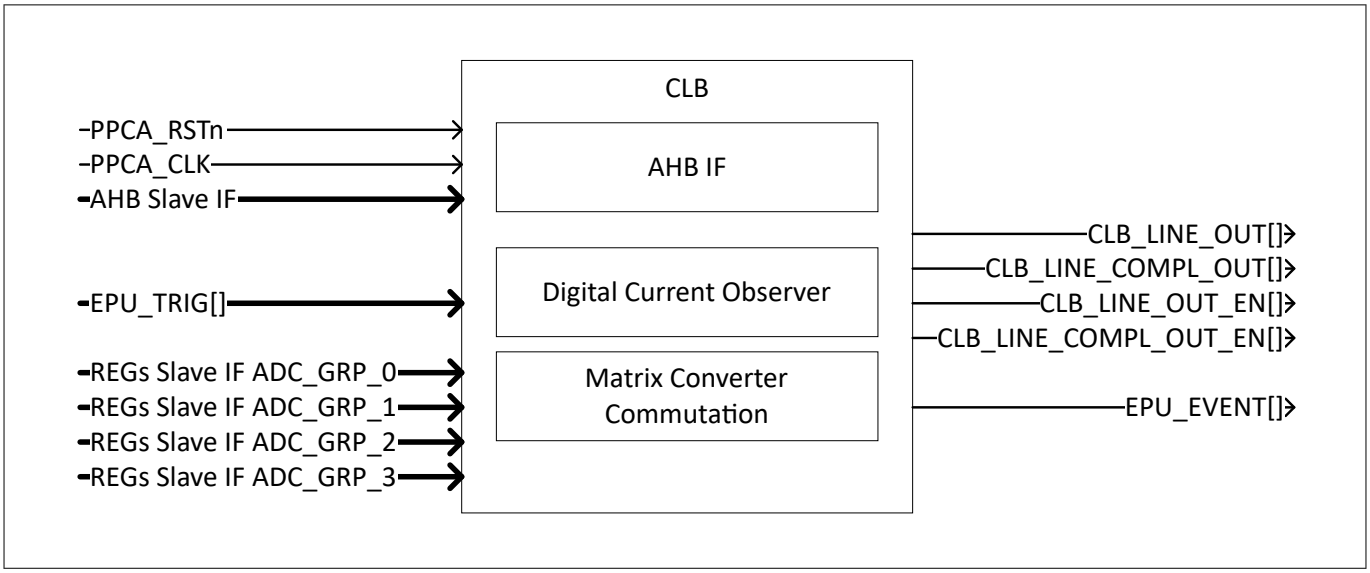


Figure 323 CLB

31.1 Features

The CLB supports the following functions:

- Digital Current Observer for Totem-Pole PFC
- Matrix Converter Commutation logic for matrix converters
- Support for Trajectory control and Spread-pulse start-up
- AHB Slave interface to configure and control the above functions

31.2 Architecture

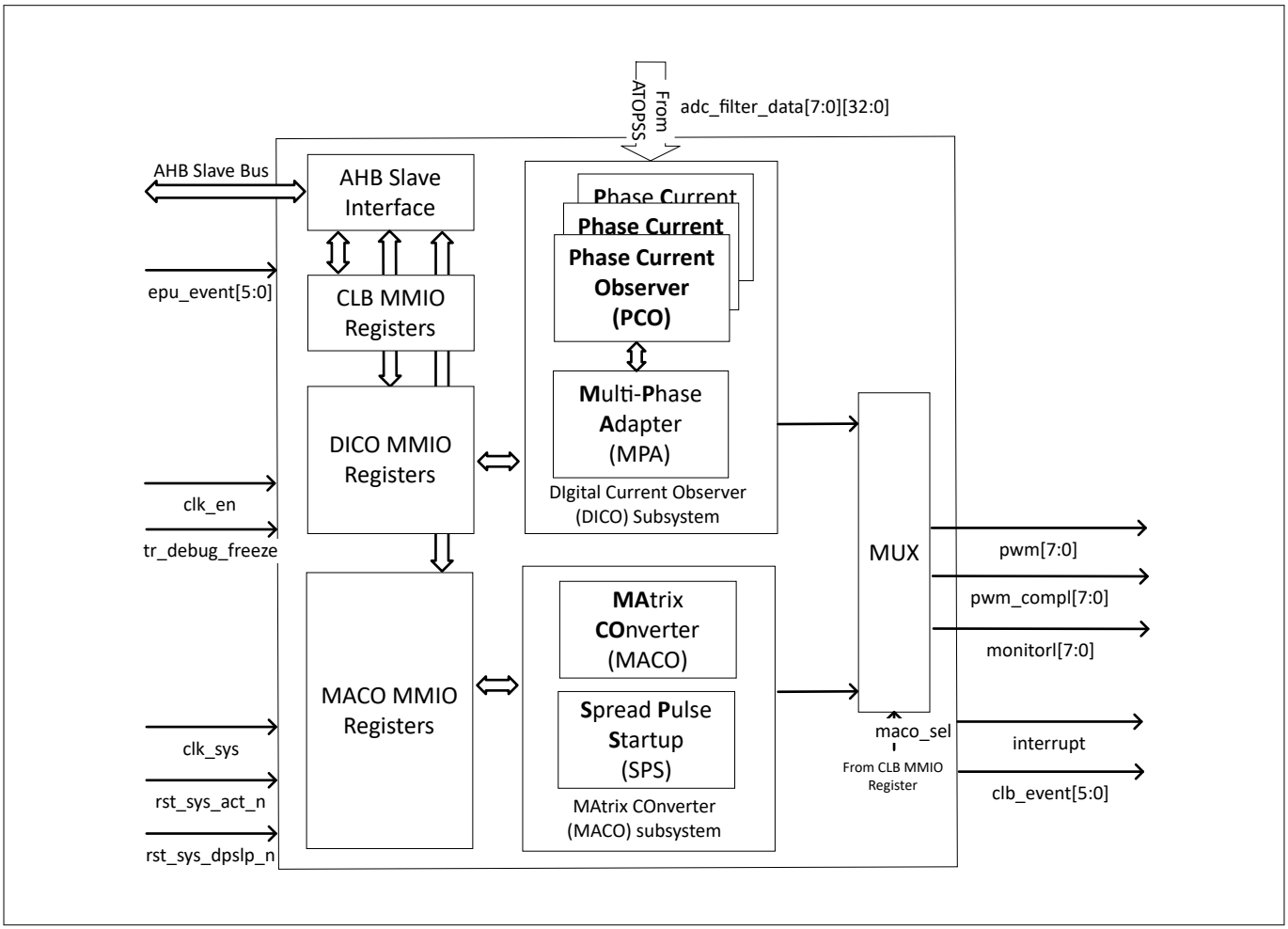


Figure 324 Block diagram of CLB

Custom Logic Block (CLB) shown in Figure 324 embeds two subsystems: DIgital Current Observer (DICO) and MAtrix COnverter (MACO). DICO is for Totem-Pole Power Factor Correction (TP PFC), while MACO is designated for use with Matrix Converter control algorithm based on a Trajectory Control and a Commutation Matrix. CLB is a pure digital block and the input signals related to the sensed current and voltage are digital signals. These are provided by ADCs (and then by ADC_FILTs) from ATOPSS. Details of the ATOPSS signal are described in ADC Filter data selection.

The DICO facilitates a multi-phase totem-pole PFC implementation based on Hysteretic Control, accommodating up to three phases. There are three Phase Current Observer (PCOs) correspond to three individual controlled phases. Each PCO is tasked with managing the PFC for its respective phase. The Multiphase Adapter (MPA) is responsible for ensuring the proper interleaving of phase-currents to enhance delivered power, improve power factor, diminish current ripple, and mitigate input harmonics. It is important to note that the MPA is not active in a single-phase TP PFC topology, where only one PCO is active.

The MACO is used for controlling the matrix converter, which is typically constructed using a matrix of bidirectional switches to directly control the power flow and voltage conversion between the AC input and DC output.

CLB handles debug signals, see Debug handling for details. CLB can monitor internal signals for testing and debugging, see Monitor for details. When “maco_sel” is asserted (HIGH or 1), CLB outputs the drive signals (“pwm” and “pwm_compl”) and monitoring information of MACO. Otherwise, CLB selects DICO's.

Each block of the CLB is synchronously reset when disabled so that it starts from a clean initial state the next time it is enabled. For PCO and MPA of DICO, the zero-crossing period is equivalent to being disabled.

DICO and MACO can support mask and kill events. They handle such events slightly differently. How DICO handles these two events is explained in Digital current observer (DICO) subsystem, and how MACO handles them is in Matrix commutation (MACO).

31.3 Digital current observer (DICO) subsystem

The Digital Current Observer (DICO) subsystem supports the implementation of the current loop of Totem-Pole PFC AC-DC conversion and consists of three phase current observers (PCOs) and a multiphase adapter (MPA). The details of PCO and MPA are in Phase current observer (PCO) and Multi-phase adapter (MPA) respectively.

31.3.1 Phase current observer (PCO)

The totem-pole PFC using sliding mode current control (or hysteretic control) is depicted in Figure 325. This is the application scenario of PCO. The PFC Controller in Figure 325 corresponds to the PCO. PCO takes sensored values (Vac, iL and Vo), IMIN and IMAX as inputs, and generates complementary signal pair PWMs1 and PWMs2 for controlling the high frequency switches S1 and S2. LS1 and LS2 are low frequency switches. LS2 is ON and LS1 is OFF when Vac > 0 (reverse when Vac<0).

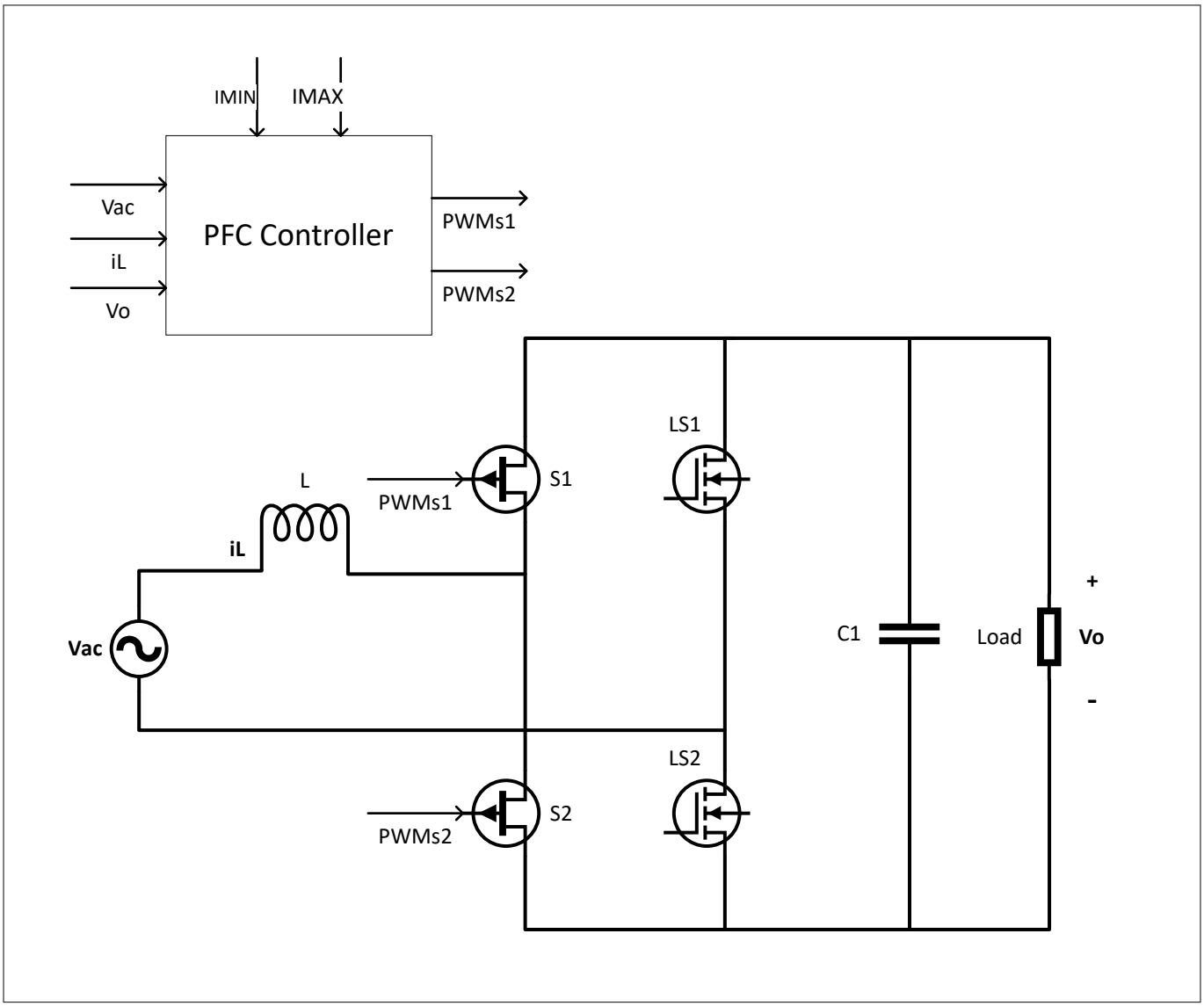


Figure 325 Topology of Totem-Pole PFC using Sliding Mode Current Control (or Hysteretic Control)

The operational mode of the totem-pole PFC is shown in Figure 326. Note that Vo is also often represented as Vbulk. Since totem-pole PFC is a boost converter, Vo > |Vac|.

PCO is disabled and reset when IMIN=IMAX=0. When IMAX <=IMIN, the current estimator is stopped/frozen (not reset) while other blocks continue to run.

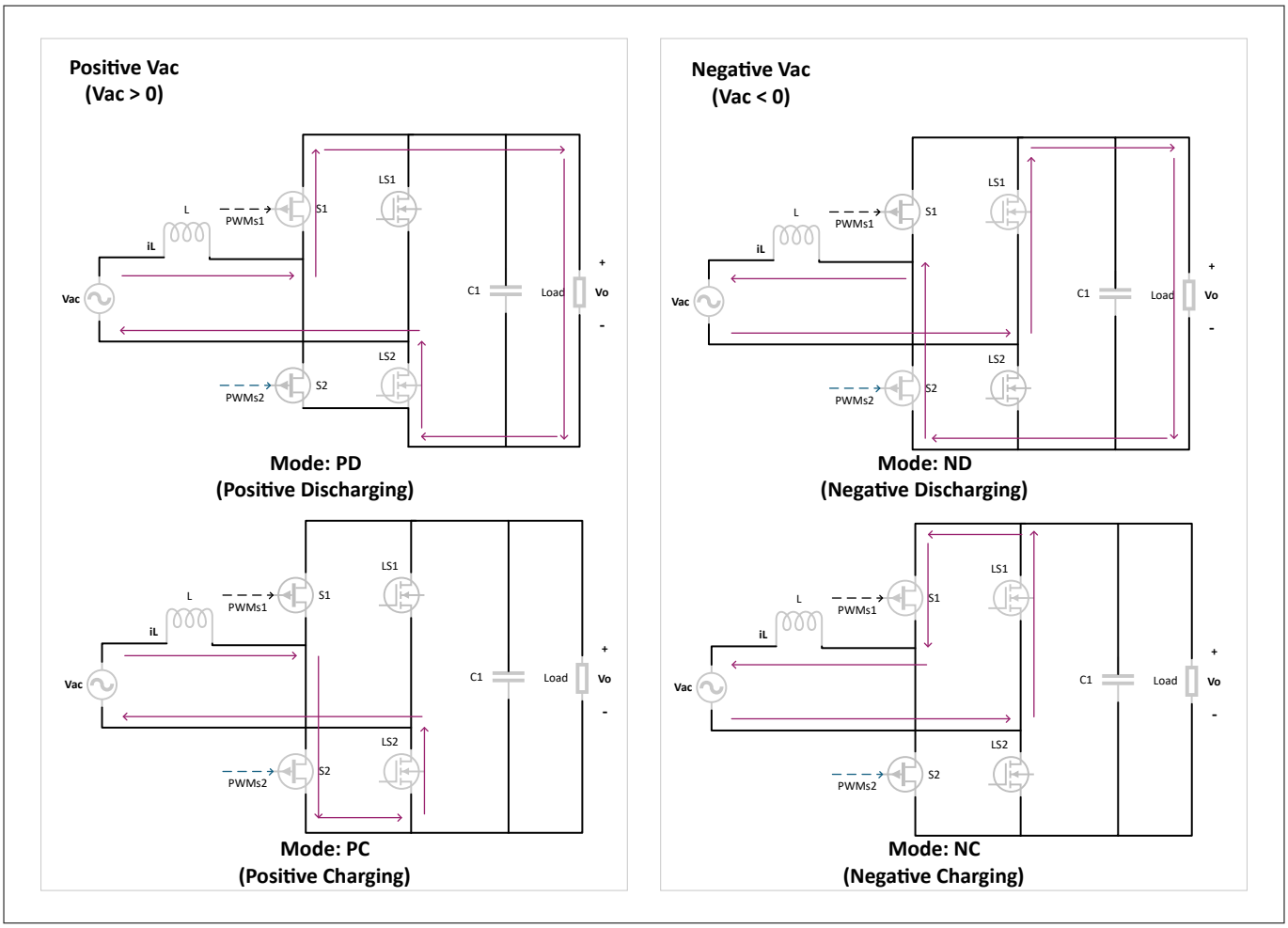


Figure 326 Operational modes of the totem-pole PFC

The PCO consist of three sub-blocks:

1. Current Estimator
2. Modulator
3. Error Observer

The block diagram of PCO is shown in Figure 327. The Current Estimator estimates the inductor current by integrating the inductor voltage. The estimated inductor current is then fed into the Error Observer for error estimation and current correction, forming a feedback loop. Concurrently, the estimated inductor current is input to the Modulator to regulate the generation of the PWM signal, clipping the inductor current in the range of [IMIN, IMAX].

When the PCO is disabled, all its registers (not MMIO registers) and outputs are synchronously reset to 0, so that PCO can be re-enabled from a reset state.

All the registers (not MMIO registers) in the PCO are synchronously reset during the zero-crossing range of Vac. The zero-crossing range is defined by the user, such as [-5V, 5V], and vac_pol is generated by the voltage loop in software.

The arithmetic units in PCO - including subtractors, adders, and accumulators - are designed to prevent overflow. We apply a saturation operation to the result of the accumulator. For a 9-bit value x, the saturation operation equals to $min(-128, max(127, x))$

Except for counters and 1-bit signals, all other signals are signed.

The PWM Modulator introduces a dead time when the [“pwm”, “pwm_compl”] signal transitions between [0, 1] and [1, 0]. During this dead time, the [“pwm”, “pwm_compl”] signal is forced to [0,0]. In the absence of dead time, this would cause a shoot-through condition in real systems where both switches are turned ON, leading to a fault scenario. This is due to the propagation delay and phase delay that occur in real systems.

The Error Observer has a configuration for setting the number of switching periods, "num_period". If the "num_period" period is reached, the Error Observer triggers an interrupt and synchronously resets the counter and both accumulators in Error Observer. The counter and both accumulators in Error Observer are enabled every four clock cycles, which helps to prevent saturation from occurring.

If the CPU cannot handle the PCO interrupt (from Error Observer) in time, no action is needed and the interrupt that cannot be handled can be ignored. The robustness of the PI controller allows us to miss some error information from Error Observer. In this case, the interrupt trigger frequency can be reduced through configuration. Users can set interrupts to be issued every 1/2/3/4 switch period via “num_period”. A freeze bit “frozen_info” is introduced to ensure the atomic read operation of “il_accu”, “iest_ind_accu” and “count” by CPU.

IMIN and IMAX in the modulator are adjusted by the correction (“imin_imax_corr”) from the phase synchronizer to maintain proper out-of-phase. After applying the pending correction (indicated by “imin_imax_corr_pending”), the PCO clears the pending bit.

“x.y-bit” represents that the signal is x+y bits, where “x” bits are the integer bits and “y” bits are the fractional bits. The integer part is 12 bits, and the fractional part is 4 bits. Only the integer part of the estimated current is fed into Modulator. In Error Observer, only the integer part is written into the MMIO registers. In other words, the fractional part is only used in the accumulation path.

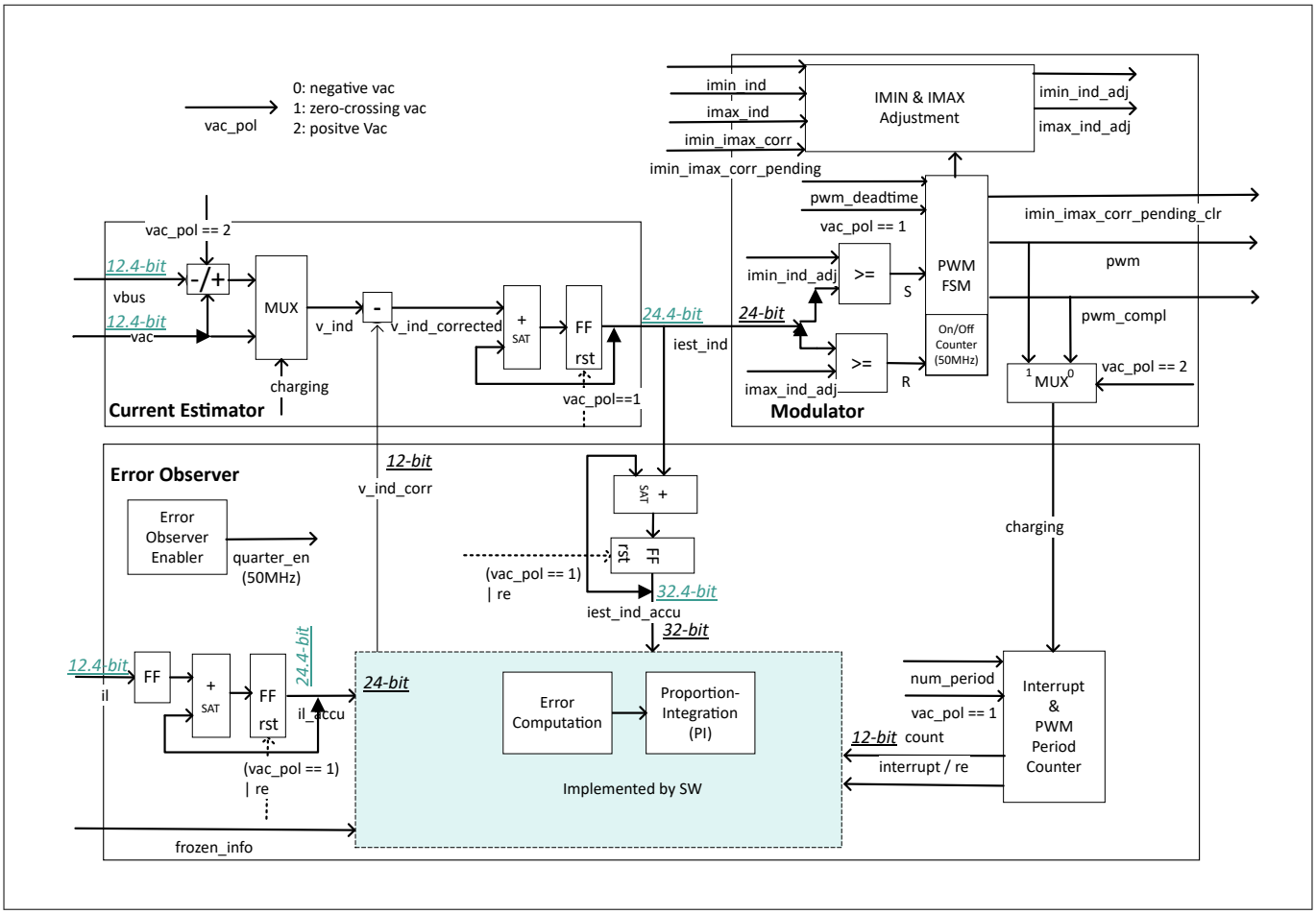


Figure 327 Block diagram of PCO

31.3.1.1 Current estimator

The inductor voltage in different operation modes is shown in Table 301.

Table 301 Inductor voltage in different operating modes

Operation Mode (shown in Figure 326)	Inductor Voltage (v_ind)
Positive Discharging (PD)	Vac-Vbulk
Positive Charging (PC)	Vac
Negative Discharging (ND)	Vac+Vbulk
Negative Charging (NC)	Vac

Based on the correlation between voltage and current in an inductor $L \quad (di/dt) = v$, the product of the inductor's current and its inductance (L) can be estimated as $Li = \int_0^t v$ In the digital integer implementation, $iest_{ind} = \sum v_{ind} = L * i * f_{sys}$, where $L$is the inductance i is the inductor current and F_{sys} is the system frequency. During the dead time phase, the accumulation behavior is the same as the discharge phase. On zero crossing, the accumulator is synchronously reset.

31.3.1.2 Modulator

The modulator is responsible to generate the PWM signal and its complementary based on IMIN (“imin_ind”), IMAX (“imax_ind”), and the corrected estimated inductor current (“iest_ind”) obtained from the Current Estimator.

A five-state FSM is used to produce PWM signals for controlling the high frequency switches S1 and S2. The PWM FSM behaves like an RS flip-flop but supports de-asserting of output during zero-crossing and transition deadtime. The transition of the FSM is shown in Figure 328.

The correction information (“imin_imax_corr” and “imin_imax_corr_pending”) is from Phase Synchronizer (Phase synchronizer (PS)). The correction is applied only when “imin_imax_corr_pending” is true.

1. If CNFG.ONLY_CORR_CHARG = 0, the correction is applied to the upcoming IMIN or IMAX, whichever comes first
2. If CNFG.ONLY_CORR_CHARG = 1, the correction is applied to the upcoming IMAX when Vac is positive (vac_pol == 2) or the upcoming IMIN when Vac is negative (vac_pol == 0)

When the duration of Q or NQ is greater than “max_on_count”, the PWM FSM is forced to transit. By default (after reset), “max_on_count” contains the maximum value. The counting frequency is 50 MHz, a quarter of the system frequency. The duration of Q and NQ is at least “min_on_count”. By default (after reset), “min_on_count” is 0.

If Vac is positive (vac_pol == 2), the output signal ''pwm'' represents the charging signal. Conversely, if Vac is negative or zero (vac_pol != 2), the charging signal is represented by the ''pwm_compl'' output signal. The charging signal indicates that when it is 1, the inductor is in the charging stage.

After zero crossing or when PCO is enabled, the PCO starts with charging the inductor.

The adjustment of IMIN (“imin_ind”) and IMAX (“imax_ind”) are shown in Table 302. When an adjusted IMIN or IMAX is hit, hardware issues a pending clear.

Table 302 Adjusted IMIN and IMAX

imin_imax_corr_pending, FSM State	Adjust IMIN and IMAX
0, DO NOT CARE	imin_ind_adj=imin_ind imax_ind_adj=imax_ind
1, Q	imin_ind_adj=imin_ind imax_ind_adj=imax_ind + imin_imax_corr
1, NQ	imin_ind_adj=imin_ind - imin_imax_corr imax_ind_adj=imax_ind

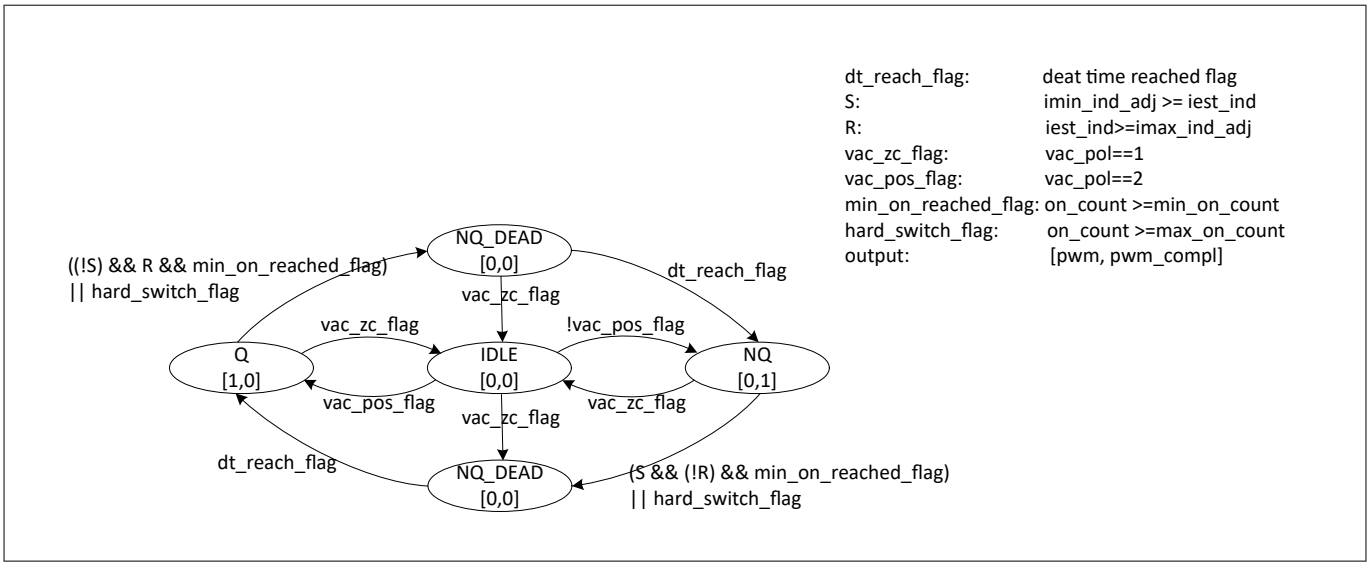


Figure 328 FSM of PCO Modulator

31.3.1.3 Error observer

Since components (switches, inductors, etc.) are not ideal, the estimated inductor current is different from the real one. This difference (also called the error) is measured and used to correct the estimated inductor current.

The sensored and estimated inductor currents are accumulated separately. Their accumulated values (“il_accu” and “iest_ind_accu”) over one or multiple switching cycles are written into MMIO registers, and their corresponding difference is the error over the period. At the same time, the number of cycle counts corresponding to this period is also written to MMIO register (“count”). Based on these data, the CPU uses the PI controller to calculate the correction to the inductor voltage to minimize the error in the next switching cycles. An interrupt is triggered at the rising edge of the charging signal every “num_period” switch periods.

***Note**: The Error Observer is enabled every four system clock cycles via the internal enabler called Error Observer Enabler. Since the current and voltage sensed by the ADC have different gain amplifications, the accumulated sensed and estimated inductor currents need to be rescaled on the software side before calculating the error.*

The two accumulators and counter are also synchronously reset during the zero-crossing phase (vac_pol==1). Besides, they are synchronously reset when an interrupt is triggered (that is the rising edge of charging PWM every “num_period” switch periods). The Error Observer can be configured to issue interrupts every 1/2/3/4 switch cycles. Each switching cycle is determined by measuring the time between the rising edge to the subsequent rising edge of the charging signal.

A freeze bit “frozen_info” is introduced to ensure the atomic read operation of “il_accu”, “iest_ind_accu” and “count”. When the freeze bit is asserted, "il_accu", "iest_ind_accu" and "count" cannot be updated. To avoid missing any possible updates, shadow registers are introduced, ensuring that "il_accu", "iest_ind_accu" and "count" are updated immediately after the freeze bit is invalid.

31.3.2 Multi-phase adapter (MPA)

Multi-Phase Adapter (MPA) is used for multi-phase totem-pole PFC to achieve low ripple, high Power Factor (PF) and low Total Harmonic Distortion (THD). In CLB, MPA has two features:

- It controls the insertion (enablement) and removal (disablement) of the 2nd phase (the 2nd PCO) and the 3rd phase (the 3rd PCO). This function is implemented by a block called Phase Enabler (illustrated in Phase

enabler (PE)). In Figure 329, it is a three-phase interleave totem-pole PFC, which can be regarded as three single-phase totem-pole PFC circuits supplying power to the load

- After enabling PCOs, MPA detects, adjusts, and maintains synchronized out-of-phase between phase-currents, that is the inductor current iL in each phase. The phase synchronization is achieved by tuning the IMIN and IMAX of each PFC controller (PCO). The IMIN and IMAX are corresponding to “imin_ind” and “imax_ind” shown in Figure 327, respectively. This function is implemented by a block called Phase Synchronizer (illustrated in Phase synchronizer (PS))

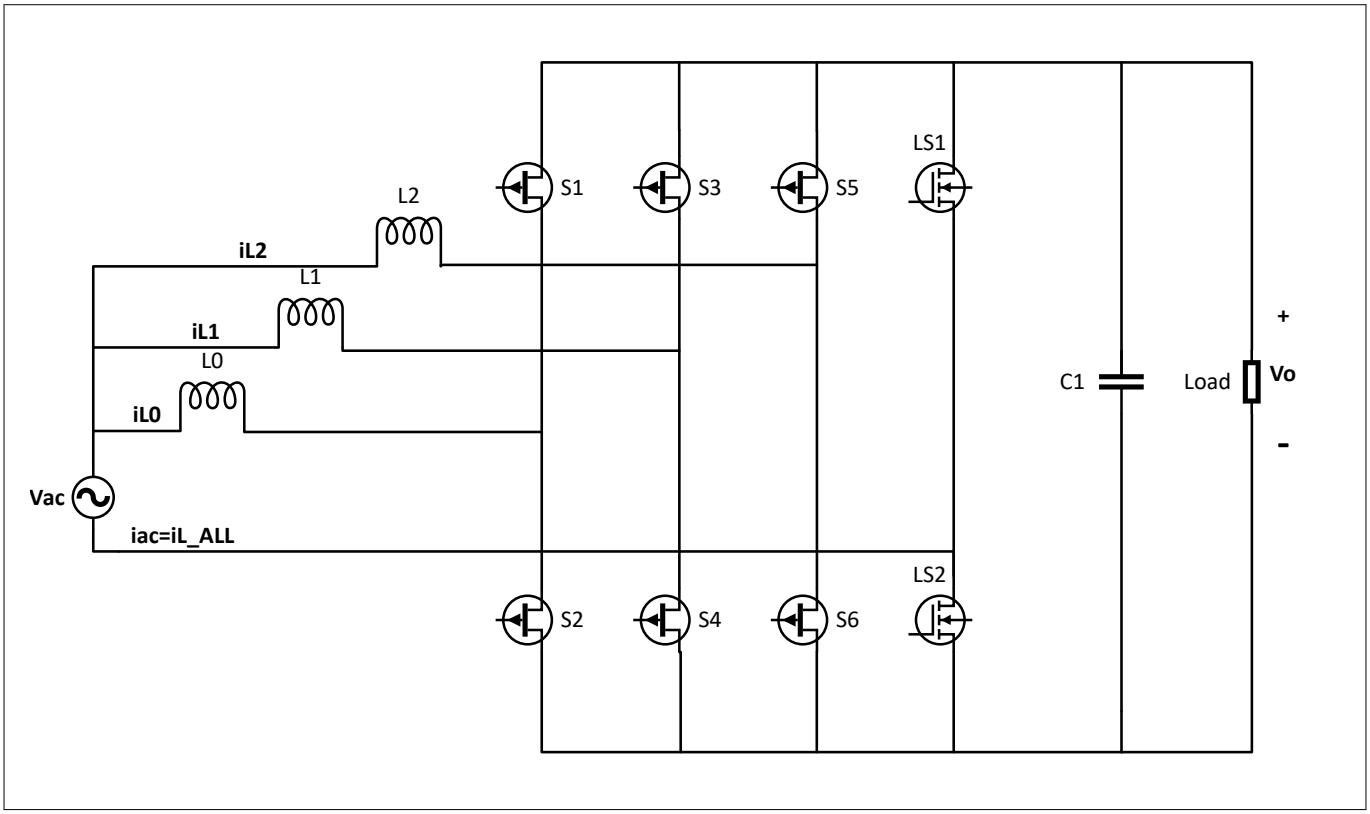


Figure 329 Multi-Phase (Three-Phase) Interleaved totem-pole PFC

31.3.2.1 Phase enabler (PE)

Phase Enabler (PE) function support the dynamic phase insertion during the AC Cycle without affecting the THD. Basically it estimates the time required for the insertion of new phase-current (iL1 or iL2 in Figure 329) to reach IMIN + N * IEXT, where N is either 1 for the 2nd PCO or 2 for the 3rd PCO, with IEXT set to $|IMAX - IMIN|/2$ for two-phase totem-pole PFC or IEXT set to $|IMAX - IMIN|/3$ for three-phase totem-pole PFC. Then it starts the new phase, PCO_1 (the 2nd PCO) or PCO_2 (the 3rd PCO), at an appropriate time, reducing the overall ripple, so increasing power factor, and reducing Total Harmonic Distortion (THD).

The block diagram of PE is shown in Figure 330. The Finite-State Machine (FSM) is shown in Figure 331. The description of each state is available in Table 303. Computations supported by software (i.e. CPU) are marked in blue.

When the system enters deep sleep mode, all outputs and registers in PE are reset regardless of the state of the FSM.

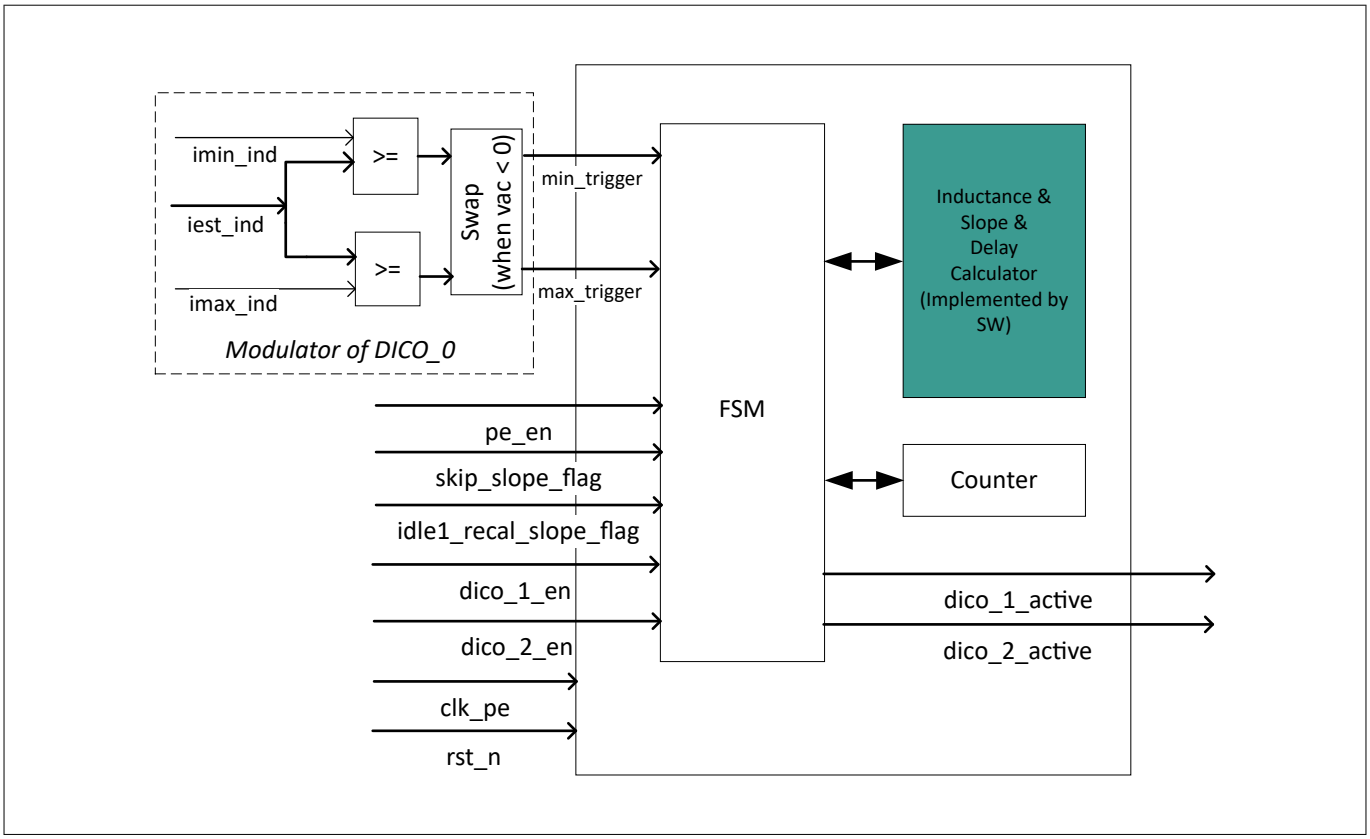


Figure 330 Block diagram of Phase Enabler

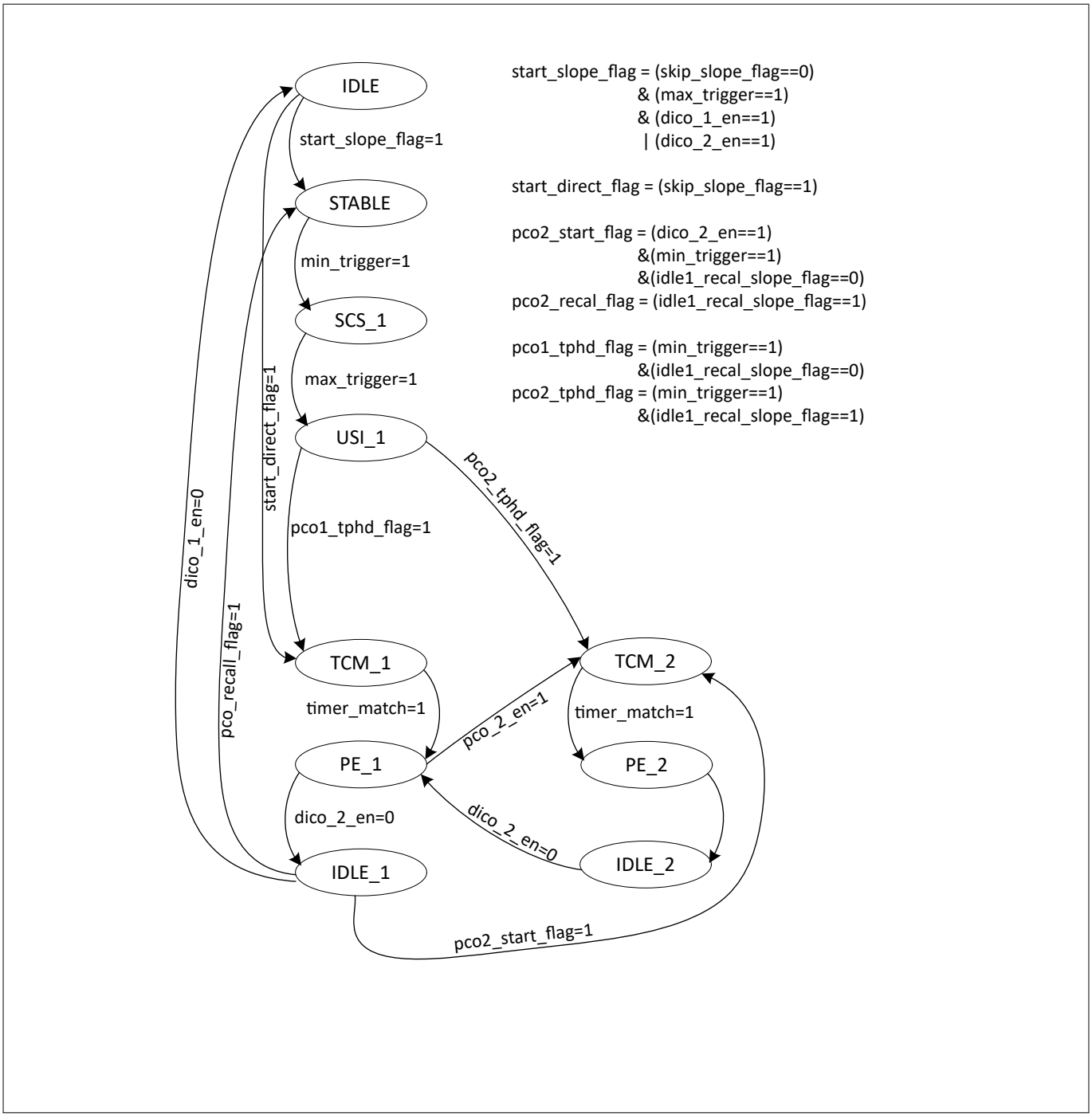


Figure 331 FSM of Phase Enabler

Table 303 State of Phase Enabler

State	Description
IDLE	Initial state The PE is only active when at least two PCOs are required. Set the 2nd and 3rd phase/PCO to be inactive when in IDLE. If skip_slope_flag == 1, transiting to TCM_1. Otherwise, when iL0 reaching IMAX, max_trigger=1, transiting to STABLE.
STABLE	Stable state After iL0 reaching IMAX for the first time, DICO_0 is considered to have entered a stable state. When iL0 reaching IMIN, min_trigger=1, transiting to SCS_1
SCS_1	SCS: Slope Calculation Start The timer starts counting from 0 and the operating frequency is clk_pe. Content to be written into MMIO Register by HW: IMIN When iL0 reaching IMAX, max_trigger=1, transiting to USI_1
USI_1	USI: Up Slope and Inductance The timer count is recorded as T_{UP} and re-started counting from 0. $Inductance \quad L = \left(\left\|V_{AC}\right\| * T_{UP}\right)/\left(\left\|I_{MAX[n]} - I_{MIN[n]}\right\|\right)$ $Up-slope: \quad S_{UP} = \left(\left\|I_{MAX[n]} - I_{MIN[n]}\right\|\right)/T_{UP} = \left\|V_{AC}\right\|/L$ Content to be written into MMIO Registers by HW: T_{UP}, IMAX Trigger interrupt When iL0 reaching IMIN, min_trigger=1, if idle1_recal_slope_flag == 1, transiting to TCM_2, otherwise transiting to TCM_1 *Note: The inductance calculated here considers non-ideal factors.*

(table continues...)

Table 303 (continued) State of Phase Enabler

State	Description
TCM_1	TCM: Timer Count Match The timer count is recorded as T_{DN} and re-started counting from 0. $Down-slope: \quad S_{DN} = \lvert V_{AC} - V_o \rvert / L$ $T_{cross} = \lvert IMAX[n] - IMIN[n] \rvert / S_{UP} + \lvert IMAX[n] - Iav \rvert / S_{DN}$ $T_{av} = \lvert Iav \rvert / S_{UP}$ where $I_{av} = IMIN + IEXT$ Calculation of the delay value TPHD: $If \quad T_{cross} > T_{av}$ $TPHD = T_{cross} - T_{av} = ((\lvert IMAX[n] - IMIN[n] \rvert - \lvert Iav \rvert)/S_{UP}) + ((\lvert IMAX[n] - Iav \rvert)/S_{DN}),$ $else:$ $TPHD = ceil\left(\left(T_{av} - T_{cross}\right)/T_{cycle}\right) \times T_{switch} + T_{cross} + T_{av}$ where $T_{switch} = (\lvert IMAX[n] - IMIN[n] \rvert / S_{UP}) + (\lvert IMAX[n] - IMIN[n] \rvert / S_{DN})$ Content to be written into MMIO Registers by HW: T_{DN}, IMIN Trigger interrupt Software writes TPHD to the MMIO Register PE_TPHD_1 and sets the corresponding pending bit (valid is renamed to pending) in the same MMIO REG. When the timer count reaches the delay value in PE_TPHD_1, timer_match=1, transiting to the state PE_1 When transition to the state PE_1, clear the corresponding pending bit.
PE_1	PE: Phase Enable Set the 2nd phase/DICO to be active (Set the 3rd Phase/DICO to be inactive). If the 3rd DICO is not enabled (dico_2_en=0), the state transits to IDLE_1. Otherwise, the state transits to SCE_2.
IDLE_1	IDLE State for the case of two phases/DICOs. This state is created for clock gating purposes. When dico_1_en = 0, transiting to IDLE. When dico_2_en=1 and min_trigger=1, if idle1_recal_slope_flag == 1, transiting to STABLE, otherwise transiting to TCM_2.

(table continues...)

Table 303 (continued) State of Phase Enabler

State	Description
TCM_2	TCM: Timer Count Match Same as SCE_1: calculating new TPHD using pre-computed S_{UN}, S_{DN} . $T_{av} = \lvert Iav \rvert / S_{UP}$, where $I_{av} = IMIN + 2 * IEXT$ Calculation of the delay value $TPHD$ If $T_{cross} > T_{av}$, $TPHD = T_{cross} - T_{av} = ((\lvert IMAX[n] - IMIN[n] \rvert - \lvert Iav \rvert)/S_{UP}) + (\lvert IMAX[n] - Iav \rvert / S_{DN})$ Else: $TPHD = ceil(((T_{av} - T_{cross})/T_{cycle}) * T_{switch} + (T_{cross} - T_{av}))$, where $T_{switch} = \lvert IMAX[n] - IMIN[n] \rvert / S_{UP} + \lvert IMAX[n] - IMIN[n] \rvert / S_{DN}$ Nothing need to be written into MMIO Registers by HW. Trigger interrupt. Software writes TPHD to the MMIO Register PE_TPHD_2 and sets the corresponding pending bit in the same MMIO Reg. When the timer count reaches the delay value in PE_TPHD_2, timer_match=1, transiting to the state PE_2 When transition to the state PE_2, clear the corresponding pending bit.
PE_2	PE: Phase Enable Set the 3rd Phase/PCO to be active. Direct transiting to IDLE_2.
IDLE_2	IDLE State for the case of three phases/PCOs. This state is created for clock gating purposes. When dico_2_en = 0, transiting to IDLE_1.

31.3.2.2 Phase synchronizer (PS)

In a multiple-phase totem-pole PFC (CLB supports up to three phases), a specific phase offset must be maintained between the phase-currents. In the case of two-phase totem-pole PFC, this phase offset is 180 degrees, and for three-phase, it amounts to 120 degrees.

Phase Synchronizer (PS) is employed to uphold the phase shift, accomplishing this by adjusting the IMIN and IMAX of each PCO. PS provides the correction term. The final adjustment of IMIN and IMAX is performed in the Modulator of PCO. The block diagram of PS is shown in Figure 332. Its Phase Corrector (PC) block diagram is shown in Figure 332. The description of each sub-block and IO-ports is available in Table 304 and Table 305, respectively.

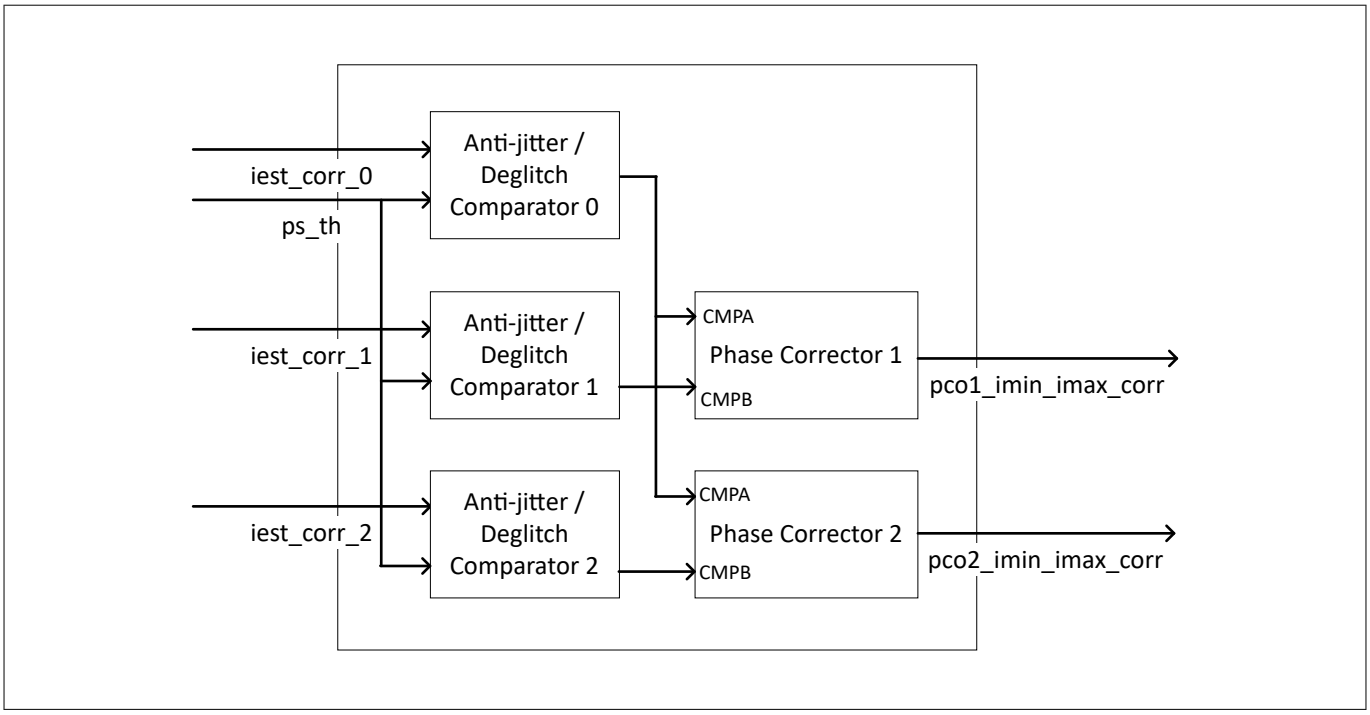


Figure 332 Block diagram of Phase Synchronizer

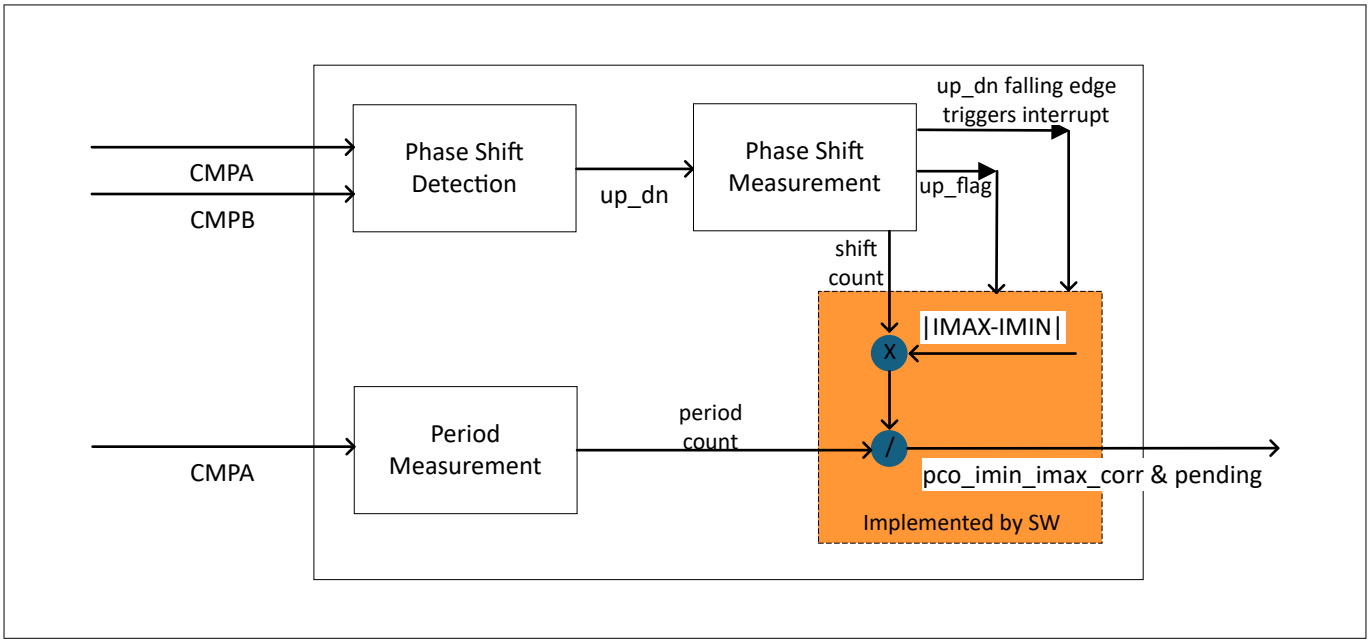


Figure 333 Block diagram of Phase Corrector

Table 304 Block description of Phase Synchronizer and Phase Corrector

Sub-Block	Description
Anti-Jitter/Deglitch Comparator	It does comparison between iest_corr_x and ps_th. It outputs 1 when iest_ind_x >= ps_th, otherwise 0. Since the update frequencies of iest_corr_x and ps_th are inconsistent and the update amplitudes are different, the comparison results might suffer from jitter. Figure 334 shows the solution to eliminate jitter when obtaining the comparison result, CMP.
Phase Corrector	It is for computing the correction of IMIN or IMAX.
Phase Shift Detection	It generates UP or DN, that is up_dn, which indicates the phase shift to be corrected.
Phase Shift Measurement	It measures the phase shift to be corrected. The measurement is achieved by counting the number of system clocks of up_dn. The interrupt is triggered at the falling edge of up_dn.
Period Measurement	It measures the period of HIGH and LOW of CMPA. The measurement is achieved by counting the system clock cycles between the rising/falling edge to falling/rising edge of CMPA.

Table 305 IO-ports of Phase Synchronizer and Phase Corrector

Block	Signal	Description
Phase Synchronizer	iest_corr_0, iest_corr_1, iest_corr_2	The estimated inductor current of each phase from PCOs.
	ps_th_0	Comparison threshold, computed by CPU and write into the MMIO register. Two-phase totem-pole PFC: $ps_th = (IMIN + IMAX)/2$ Three-phase totem-pole PFC: $ps_th = (IMIN + IMAX)/2$ $(recommended)$ or $ps_th = ((2 * IMIN)/3) + (IMAX/3)$ *Note: Different thresholds make the software part different.*

(table continues...)

Table 305 (continued) IO-ports of Phase Synchronizer and Phase Corrector

Block	Signal	Description
	dico_1_active, dico_2_active (Not shown in the block diagram)	Signals from PE. If dico_1_active = 0 and dico_2_active = 0, one-phase totem-pole PFC, PS is inactive. If dico_1_active = 1 and dico_2_active = 0, two-phase totem-pole PFC, Phase Corrector 0 is active. If dico_1_active = 1 and dico_2_active = 1, three-phase totem-pole PFC, entire PS is active. Combination of dico_1_active = 0 and dico_2_active = 1 is illegal. dico_1_active and dico_2_active are also for gating internal clocks for Phase Correctors.
	pcoX_min_max_corr	pcoX_min_max_corr is the correction term to IMIN or IMAX in PCO. The calculation of pcoX_min_max_corr is implemented by SW. The calculation of pcoX_min_max_corr requires period_count, phase_shift_count, up_flag, IMIN and IMAX.
	clk_ps (Not shown in the block diagram)	Gated clock for Phase Synchronizer
	rst_n (Not shown in the block diagram)	Reset signal for Phase Synchronizer
Phase Corrector	CMPA, CMPB	Inputs from Anti-Jitter/Deglitch Comparators to Phase Corrector. CMPA is the master signal (as reference for the adjustment) and CMPB is the slave signal.
	up_dn	up_dn (UP or DOWN) indicates the phase shift need to corrected.
	up_flag	up_flag indicates that up_dn is UP or DOWN.
	shift count	The number of system clock cycles of up_dn
	up_dn falling edge triggered interrupt	Interrupt signal, the falling edge of up_dn

(table continues...)

Table 305 (continued) IO-ports of Phase Synchronizer and Phase Corrector

Block	Signal	Description
	period count	The number of system clock cycles of HIGH and LOW of CMPA. If $ps_th = (IMIN + IMAX)/2$, it is the half period of CMPA.
	pco_imin_max_corr,	The correction term to IMIN and IMAX.
	pco_imin_imax_corr_pending	Indicate a pending correction to be applied.

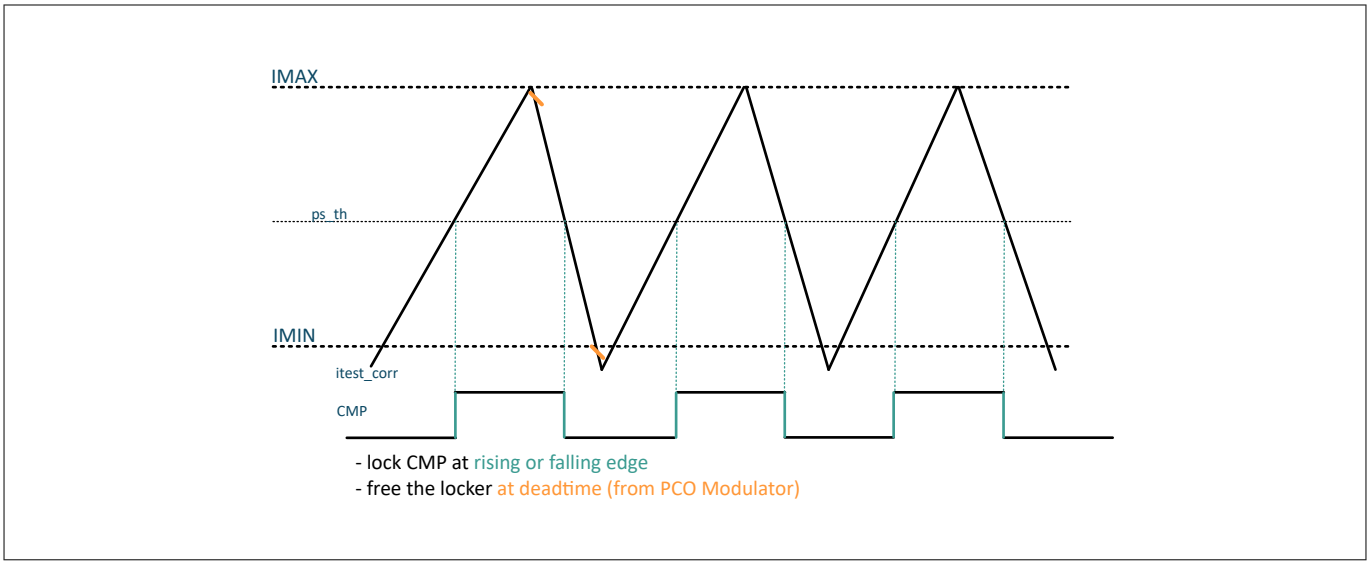


Figure 334 Solution of Anti-jitter in Comparator

The adjustment of IMIN and IMAX for the **two-phase** totem-pole PFC is shown in Figure 334.

- In two-phase totem-pole PFC, the two currents need to be 360/2=180 degrees out of phase
- I_0 is the master signal. I_{ls} and I_{rs} are slaves whose IMIN/IMAX are adjusted out of phase with the master
- CMPx is equals to Ix >= TH
- The CMPA of master current I_0 sets the period, from rising/falling edge to next falling/rising edge. If ps_th=(IMIN+IMAX)/2, half a period is always measured
- The UP or DN values is obtained by performing a logical AND operation between CMPx and CMPy or !CMPx and !CMPy (that is (CMPx & CMPy) | (!CMPx & !CMPy))
- If CMPx is equal to 1 before the rising edge of CMPx & CMPy, then the value of CMPx & CMPy is UP; otherwise, it is DN
- If !CMPx is equal to 1 before the rising edge of !CMPx & !CMPy, then the value of CMPx & CMPy is UP; otherwise, it is DN
- The interrupt is triggered at the falling

- The UP value indicates that CMPx is ahead of CMPy (or that CMPy is behind CMPx), while the DN value indicates that CMPx is behind CMPy. UP or DN is also called phase shift, with the excess or shortfall beyond 180 degrees
- The correction to IMIN and IMAX are calculated with (also summarized in Table 306, realized by CPU)
- - Positive Vac:
    - Left shift: up_dn is UP. $correction = +((UP_COUNT)/(PRD_COUNT))*|IMAX - IMIN|$
    - Right shift: up_dn is DN. $correction = -((DN_COUNT)/(PRD_COUNT))*|IMAX - IMIN|$
  - Negative Vac:
    - Left shift: up_dn is UP. $correction = +((UP_COUNT)/(PRD_COUNT))*|IMAX - IMIN|$
    - Right shift: up_dn is DN. $correction = -((DN_COUNT)/(PRD_COUNT))*|IMAX - IMIN|$

UP_COUNT or DN_COUNT is the number of clock cycles of UP or DN. PRD_COUNT is the number of clock cycles of a phase-synchronizer-period (the measured period count multiplied by 2).

Table 306 IMIN/IMAX correction for two-phase TP PFC

Vac Polarity	Left/Right Shift (relative to correct out-of-phase)	UP/DN	Correction to IMIN/IMAX
Positive	Left	UP	$+((UP_COUNT)/(PRD_COUNT))*\|IMAX - IMIN\|$
	Right	DN	$-((DN_COUNT)/(PRD_COUNT))*\|IMAX - IMIN\|$
Negative	Left	UP	$+((UP_COUNT)/(PRD_COUNT))*\|IMAX - IMIN\|$
	Right	DN	$-((DN_COUNT)/(PRD_COUNT))*\|IMAX - IMIN\|$

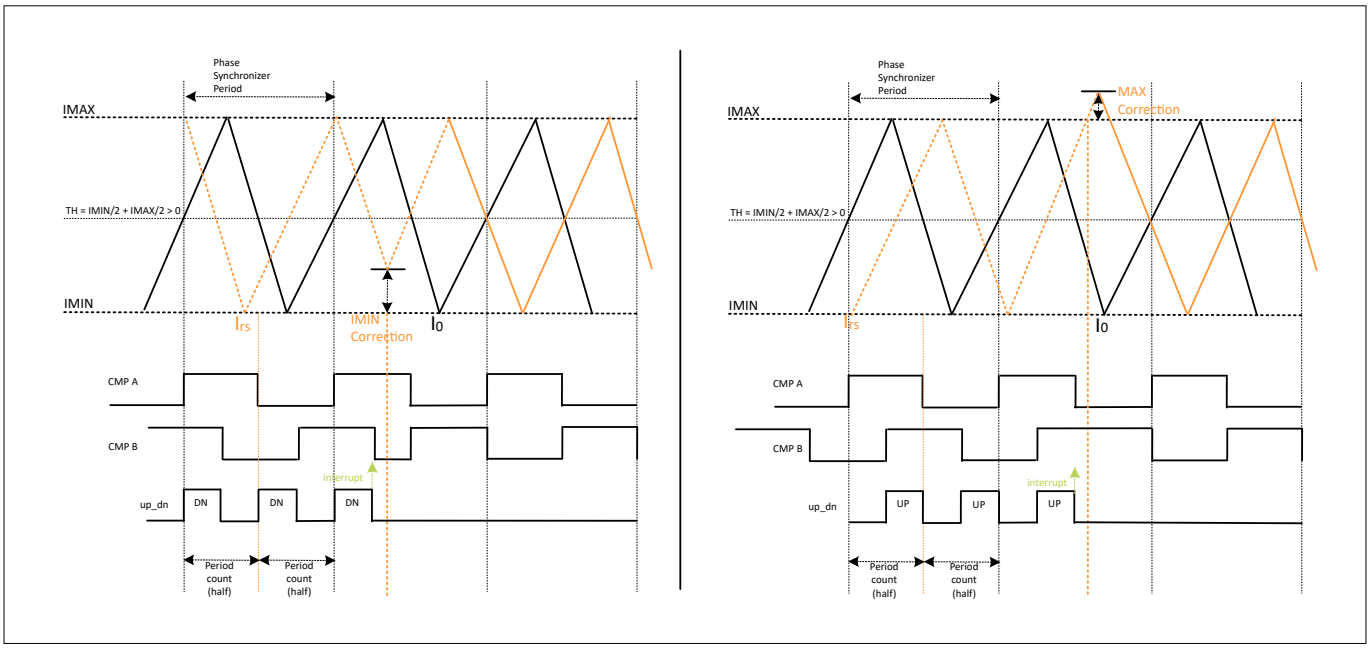


Figure 335 Out-of-phase of two-phase

31.3.3 Mask and kill event

This section illustrates how DICO handles the mask and kill event. The source of the mask and kill event is shown in Table 307

Table 307 Source of the mask and kill event

Event	CPU (via MMIO Reg)	EPU (via epu_event)
Mask Event	No	Yes
Kill Event	Yes	Yes

During mask and kill events, DICO is still be running, but its “pwm” and “pwm_compl” are switched to be driven by the MMIO register SAFE_REG.

When the mask event from EPU is de-asserted (that is 0 or LOW), DICO’s “pwm” and “pwm_compl” are still driven by SAFE_REG until the rising edge of charging signal comes. The charging signal here refers to the “pwm” and “pwm_compl” output by the PCO can drive the MOSFET charging inductor.

When the kill event from CPU is asserted (that is 1 or HIGH), DICO enters the kill state. Only when the kill event exit signal (KILL_EXIT in EVENT MMIO register) is asserted and kill event is de-asserted, DICO exits the kill state. However, DICO’s “pwm” and “pwm_compl” is still be driven by SAFE_REG until the rising edge of charging signal comes.

The user bits for extending DIOC’s output (“pwm” and “pwm_compl”) don’t have corresponding SAFE state.

31.3.3.1 ADC Filter data selection

The signal from ATOPSS to CLB is the ADC filter data, “adc_filter_data”. DICO has five input sensed signals: Vac, Vo, iL0, iL1 and iL2, each of them corresponds to a channel of the ADC filter data. The user is responsible for establishing the relationship between each sensed signal and its corresponding channel using the MMIO register CNFG, which includes options of SEL_VAC, SEL_VBUS, SEL_IL0, SEL_IL1, and SEL_IL2. CLB selects the ADC filter data of the corresponding channel to Vac, Vo, iL0, iL1 and iL2 through a multiplexer.

31.3.3.2 Incorrect setting handling

Incorrect DICO settings may cause PCO to damage the plant. Incorrect settings may be that IMIN is greater than IMAX, or the polarity setting is incorrect. To protect the plant, three strategies are proposed:

1. Set the maximum on-time and limit the on state of the MOSFETs. Update in real time whether the maximum on-time is reached, write the state to the status register, and output it to the EPU
2. Compare the real-time polarity with the polarity set by SW, write it to the status register, and output it to the EPU
3. Display the internal status of DICO to facilitate software monitoring. The status information consists of active state of each PCO, DICO mask state, DICO kill state and DICO safe state

31.4 Matrix commutation (MACO)

Matrix Converter is controlled with a Commutation Matrix of switches. Matrix converter control algorithm can be based on a Trajectory Control and a commutation matrix.

The switch control information, which is also known as the control pattern, is stored in registers. MACO includes three register regions, each dedicated to holding a control pattern. Each register region is divided into Transition, Modulation, and Not-Used segment, as shown in Figure 336. A register region has 20 rows, each row holds the control information in a format of

$[S0, S1, ...S11, ..., S15, HOLDTIME]$

Where $S0, S1, ..., S11, ..., S15$ represents the switch status, and $HOLDTIME$ denotes the duration of the status. The transition table can be empty, while the modulation table contains at least one row. $HOLDTIME$ starts from 0. The number of clock cycles for a row is $HOLDTIME + 1$. Each pattern has a corresponding SAFE state register (**16-bit**) (not shown in Figure 336), which only contains the switch status for the safe state. MACO has no user bits.

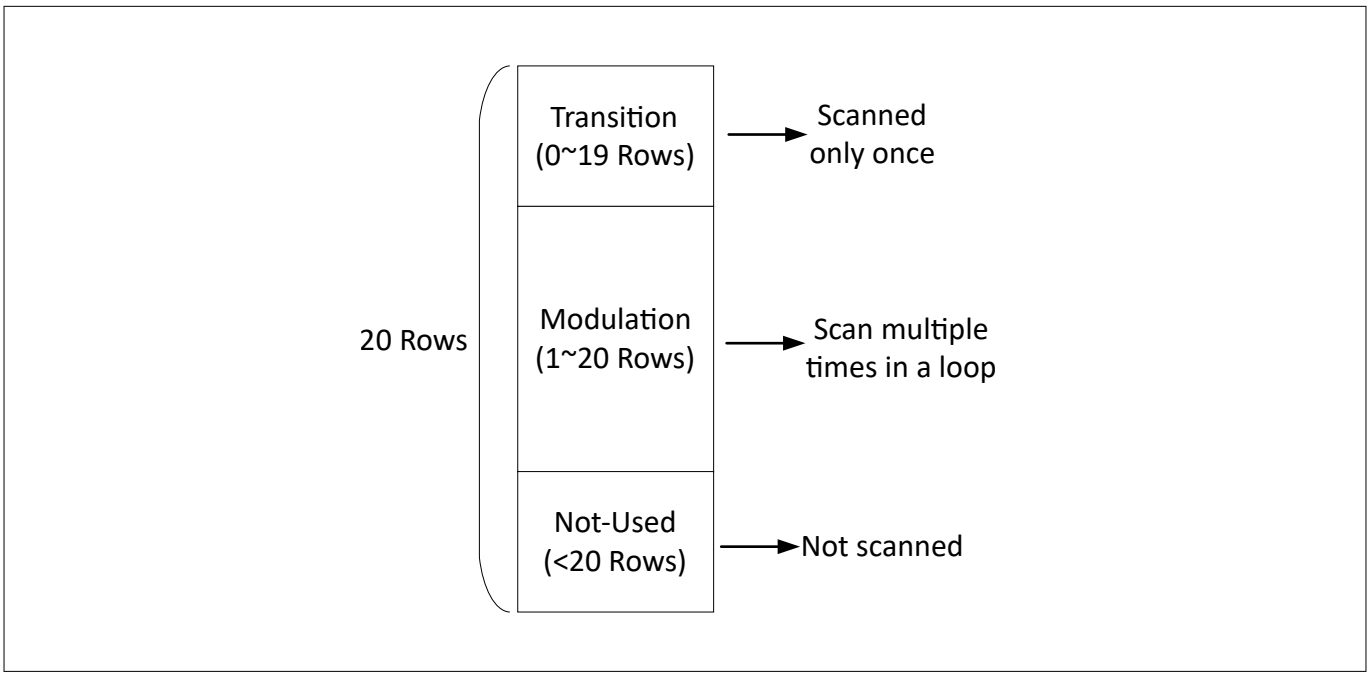


Figure 336 Register region for a control pattern

To set the number of rows for each pattern, the user specifies the number of rows for the transition table (“tran_row”) and the number of rows for the entire pattern (“all_row”). The number of rows in the modulation table “mod_row” is derived from “tran_row” and “all_row”, “mod_row” = “all_row”- “tran_row”. The valid value of “all_row” is limited in the range of [1, 20]. If the user sets a value outside of these limits, it is clipped by the hardware accordingly. “all_row” needs to be larger than “tran_row”. If “all_row” set by the user is not larger than “tran_row”, “all_row” is adjusted to 20. In these cases, no error or fault is raised.

MACO can start immediately after setting the enable bit (EN in the CTRL MMIO register), or it can start after the EN bit is enabled and the trigger signal CLB_TRIG_IN_4 is asserted. The selection is controlled by the WAIT_FOR_TRIGGER bit in CTRL MMIO register.

When transitioning to a new control pattern, the transition table is scanned first and just once. Following this, the modulation table is scanned. If “en_auto_transition” =1, the modulation table is also scanned only once and will automatically transition to the new control pattern. If “en_auto_transition” =0, the modulation table is scanned repeatedly in a cyclical manner until a transition event occurs, triggering to a new control pattern. The transition event can originate from either the MMIO register (“trans_event”) or the peripheral EPU (“CLB_TRIG_IN_0”). The new control pattern is set by the SEL_NXT_SECT in EVENT MMIO register. “en_auto_transition” in the CTRL MMIO register.

When a transition event is asserted (HIGH), MACO continues execution of the current pattern until the last row is completed before changing to the new control pattern.

MACO can be configured to select the current or previous or next register region/pattern when a transition occurs. Table 308 indicates the region that is previous or next to each region.

Table 308 Transition Scheme of Register Region/Pattern

Previous Pattern	Current Pattern	Next Pattern
2	**0**	1
0	**1**	2
1	**2**	0

MACO has mask and kill mechanisms. The sources of mask and kill events are the same as DICO, as shown in Table 307.

When the mask event(CLB_TRIG_IN_5) is asserted (HIGH), MACO immediately process to the SAFE state, which only contains the switch status. “pwm” and “pwm_compl” are driven by the SAFE state register of the current pattern. The transition to the safe state occurs in a synchronous manner.

The configuration “maco_func” is used to differentiate the behavior between Trajectory Control and Matrix Converter.

- For Trajectory Control (maco_func=1), only the output is masked (The output switch to SAFE state), MACO is still running. When the mask event is de-asserted (LOW), MACO switches back the pattern driver state. The mask event for Trajectory Control is a pure mask operation
- For the matrix converter (maco_func=0), not only does the output switch to a safe state, but also, on the exit mask event, MACO restarts from row 0 (The first row of transition table.). When the mask event is de-asserted (LOW), MACO resumes scanning the control pattern. If the transition event is not active, MACO resumes from the region/pattern it had scanned before the mask event occurred. On the other hand, if the transition event is active, MACO resumes scanning from the current or previous or next region/pattern. The resume starts from the first row of a pattern

The kill mechanism can be triggered by either the comparator through the EPU (“CLB_TRIG_IN_1”), such as in the case of overcurrent, or by the CPU through the MMIO register (“kill_event”), such as when the firmware wants to halt the PWM due to output overvoltage.

When the kill event comes (HIGH), MACO immediately moves to the SAFE state, which only contains the switch status. “pwm” and “pwm_compl” are driven by the SAFE state register of the current pattern. The transition to the safe state occurs in a synchronous manner.

When the kill event is de-asserted (LOW) and the kill event exit (KILL_EXIT in EVENT MMIO register) is asserted (HIGH), MACO resumes scanning the control pattern. If the transition event is not active, MACO resumes from the region/pattern it had scanned before the kill event occurred. On the other hand, if the transition event is active, MACO resumes scanning from the current or previous or next region/pattern. The resume starts from the first row of a pattern.

MACO exports the following status to the CPU: previous pattern, current pattern, the coming pattern to be scanned, scanning in progress, kill state, mask state, safe state.

31.5 Spread pulse startup (SPS)

Spread Pulse Startup controls the selection between Matrix Converter outputs and PWM outputs enabling the control algorithm to combine MACO and PWM features. In the CLB, it provides the CPU with information about when MACO should start. The corresponding diagram is shown in Figure 337.

In Figure 337, signals with uppercase names come from MMIO registers, signals with lowercase names come from IO ports.

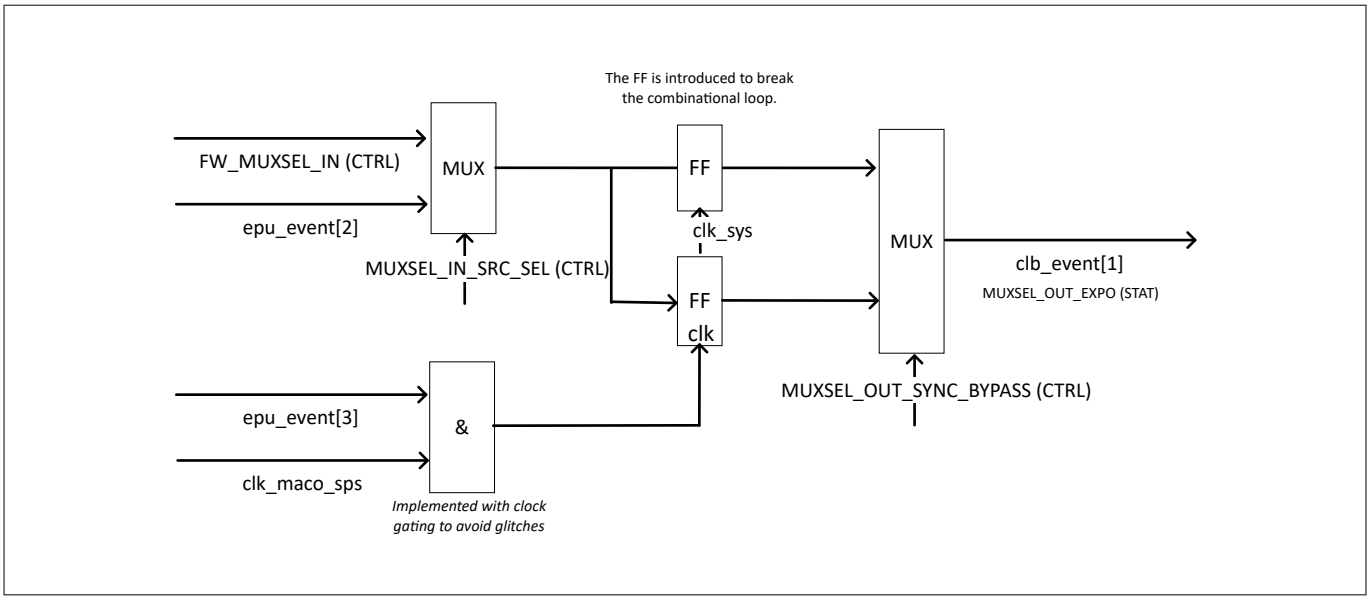


Figure 337 Spread Pulse Startup in CLB

31.6 Debug handling

CLB has two signals related to the debug/freeze feature.

1. The debug signal “tr_debug_freeze” in Figure 324, indicating whether to freeze the DICO subsystem and MACO subsystem. tr_debug_freeze signal is set by the debug600 block when a break point is hit
2. The configuration bit from MMIO register CTRL “DEBUG_FREEZ_EN”, indicating whether to enable the step-by-step debug feature

Three combinations cause different behavior:

1. When “DEBUG_FREEZ_EN” = 0,
   - **a.** set “tr_debug_freeze”, that is “tr_debug_freeze” = 1
     1. “pwm” and “pwm_compl” of DICO are set to the safe state defined in the SAFE registers. “switch_status” of MACO is set to the safe state
     2. CLB (DICO subsystem & MACO subsystem) is frozen. This is achieved by clock gating the clock “clk_sys”. MMIO register access is still possible
   - **b.** the debug signal “tr_debug_freeze” is de-asserted (0 or LOW) after assertion, that is the falling edge of “tr_debug_freeze”
     1. The enable bit of DICO and MACO are reset. Afterwards, the internal status and flip-flops of DICO and MACO are reset
2. When “DEBUG_FREEZ_EN” = 1,
   - **a.** set “tr_debug_freeze” =1, the behavior is the same as when “DEBUG_FREEZ_EN” = 0 and “tr_debug_freeze” = 1
   - **b.** set “tr_debug_freeze” =0,
     1. “pwm” and “pwm_compl” of DICO are still set to safe state. “switch_status” of MACO is still set to the safe state
     2. DICO and MACO continues to operate
3. When “debug_freeze_en” is de-asserted after assertion:
   - **a.** If “tr_debug_freeze” = 1, same behvior as when “debug_freeze_en” = 0
   - **b.** If “tr_debug_freeze” = 0, the enable bit of DICO and MACO are reset. Afterwards, the internal status and flip-flops of DICO and MACO are reset

In summary,

- At the falling edge of (“DEBUG_FREEZ_EN” | “tr_debug_freeze”), DICO and MACO are reset
- “DEBUG_FREEZ_EN” | “tr_debug_freeze” = 1 sets the output to safe state
- “tr_debug_freeze” = 1 freezes DICO and MACO
- “DEBUG_FREEZ_EN” enables the step-by-step debugging

To be noted that since DICO and MACO are disabled after debug, their outputs, “pwm” and “pwm_compl”, are still in safe state. CPU must restart CLB from scratch.

Debugging power control by single stepping is not common. On the contrary, for debugging, the monitoring function is more commonly used. When (“DEBUG_FREEZ_EN” | “tr_debug_freeze”) = 1, it indicates that the CPU is paused, possibly due to debugging or encountering a fault. In this case, the MOSFET driver must be switched to a safe state to protect the power plant. Besides, other components of the controller (such as ADC, etc.) might also in the halt state, and CLB cannot obtain the real-time status. As a result, CLB also needs to switch its outputs to the safe state. Since the duration of (“DEBUG_FREEZ_EN” | “tr_debug_freeze”) =1 is unknown and the actual status may be unknown, it is suggested to reset CLB, and the CPU needs to restart the CLB from scratch.

31.7 Safe state

CLB-DICO and CLB-MACO each have a safe state indicated by SAFE_REG. Their outputs, “pwm” and “pwm_compl”, are set to the safe state in the following cases:

1. During mask event
2. During kill event
3. When the debug signal (“tr_debug_freeze”) is asserted
4. When CLB, CLB-DICO or CLB-MACO is disabled
   a. The outputs of CLB-DICO are set to the safe state when CLB or CLB-DICO is disabled
   b. The outputs of CLB-MACO are set to the safe state when CLB or CLB-MACO is disabled

31.8 Monitor

To test or debug CLB, CLB introduces a monitoring function that can stream internal signals to the outside. The monitor bus is 8-bit wide.

When “maco_sel” (shown in Figure 324) is asserted (HIGH or 1), CLB outputs the internal signal of MACO, otherwise it output the internal signal of DICO.

DICO offers five signals to be monitored, while MACO offers two. They are selected individually by multiplexers. The monitoring signals are listed in Table 309.

Table 309 Monitor signals of DICO and MACO

Subsystem	Monitor Signals	Description
DICO	Saturation bit of the current estimator (1-bit) Part of FSM state of the modulator [2:1] (2-bit) Saturation bit of the error observer (1-bit) Current Correction minimum or maximum flag (1-bit) Part of Current Correction [4:2] (3-bit)	Phase Current Observer (PCO) monitor signal. PCO provides one each.
	Part of FSM state [3:1] (3-bit) Reduced AND of the counter (1-bit) Part of the counter [9:6] (4-bit)	Phase Enabler monitor signal
	CMP0 (1-bit) CMP1 (1-bit) CMP2 (1-bit) up_dn_0 (1-bit) up_dn_1 (1-bit) Reduced OR of the period counter (1-bit) Reduced OR of the shift counter 0 (1-bit) Reduced OR of the shift counter 1 (1-bit)	Phase Synchronizer monitor signal
MACO	Pattern index (2-bit) Row index (5-bit) Part of the counter [0] (1-bit)	MACO monitor signal option 1
	Counter (8-bit)	MACO monitor signal option 2

31.9 Interrupts

The interrupt requests come from eight different sources are summarized in Table 310

Table 310 Interrupt sources CLB

Bit number in INTR register	IRQ source	Description
0	PS_0	Phase Synchronizer interrupt 0: used to maintain the phase shift between the controlled inductor current of PCO_0 and PCO_1. The interrupt is triggered at the falling edge of up_dn.

(table continues...)

Table 310 (continued) Interrupt sources CLB

Bit number in INTR register	IRQ source	Description
1	PS_1	Phase Synchronizer interrupt 1: used to maintain the phase shift between the controlled inductor current of PCO_0 and PCO_2. The interrupt is triggered at the falling edge of up_dn.
2	PE_USI	Phase Enabler Up Slope and Inductance interrupt. (Up slope is also the charging slope of the inductor current of PCO_0)
3	PE_SCE1	Phase Enabler Scope Calculation End interrupt 1: it indicates the time to calculate the DOWN (discharging) slope of the inductor current of PCO_0. After slope calculation, PCO_1 will be started if TPHD_1 is set.
4	PE_SCE2	Phase Enabler Scope Calculation End interrupt 2: used for calculating TPHD_2 to start PCO_2.
5	PCO_0	Phase Current Observer 0 (Error observer) interrupt: used for correcting the estimated inductor current based on the measured and accumulated voltage
6	PCO_1	Phase Current Observer 1 (Error observer) interrupt: used for correcting the estimated inductor current based on the measured and accumulated voltage
7	PCO_2	Phase Current Observer 2 (Error observer) interrupt: used for correcting the estimated inductor current based on the measured and accumulated voltage

31.10 Registers summary

Table 311 CLB Registers

Register	Function
CTRL	CLB Access Control Register

Table 312 Digital Current Observer registers

Register	Function
CTRL	Control Register
CNFG	Configuration Register
EVENT	Event Register
SAFE	Safe State Register
IMAXL	Maximum Current multiplied by Inductance Configuration Register
IMINL	Minimum Current multiplied by Inductance Configuration Register
VAC_POL	VAC Ploarity Register
OBS_CTRL	Observer Control Register
PERIOD_MEAS	Register reporting the Measured Interval Time
IEST_IND_ACCU	Register Reporting the accumulated estimated current
IL_ACCU	Register Reporting the accumulated inductor current
IEST_CORR	Current Correction Configuration Register
PE_IMINL	IMINL for Up and Down Slope
PE_IMAXL	IMAXL for Up and Down Slope
PE_UP_COUNT	UP Count Register for Up and Down Slope
PE_DN_COUNT	DOWN Count Register for Up and Down Slope
PE_TPHD_1	TPHD Value (PCO_1) Register for PE
PE_TPHD_2	TPHD Value (PCO_2) Register for PE
PS_TH	Threshold Current multiplied by Inductance Register for PS
PS_CC_INFO	IMINL/IMAXL Correction Information Register for PS
PS_CC	Correction term of IMINL/IMAXL Register for PS
INTR	Interrupt Request Register
INTR_SET	Interrupt Set Request Register
INTR_MASK	Interrupt Mask Register
INTR_MASKED	Interrupt Masked Register
PCO_CORR_IEST	Register reporting the corrected estimated current
STAT	Status Register

Table 313 Matrix commutation registers

Register	Function
CTRL	Control Register

(table continues...)

Table 313 (continued) Matrix commutation registers

Register	Function
EVENT	Trigger Register
STAT	Status Register
SECT0_ROW	Row Register of Pattern 0
SECT1_ROW	Row Register of Pattern 1
SECT2_ROW	Row Register of Pattern 2
INTR	Interrupt Request Register
INTR_SET	Interrupt Set Request Register
INTR_MASK	Interrupt Mask Register
INTR_MASKED	Interrupt Masked Register
SECT0_PATTERN	Pattern Memory 0
SECT0_SAFE	Safe State Register of Pattern 0
SECT1_PATTERN	Pattern Memory 1
SECT1_SAFE	Safe State Register of Pattern 1
SECT2_PATTERN	Pattern Memory 2
SECT2_SAFE	Safe State Register of Pattern 2

32 PPCA Hardware filter (HWFILT)

The PPCA features two independent instances of hardware filters (HWFILT). Each instance can be allocated to either two of the PPCA CPUs. Hardware filter instance 0 (HWFILT3P3Z SS 0) has four 3 poles and 3 zeroes hardware filters ((HWFILT3P3Z), hardware filter instance 1 (HWFILT3P3Z SS 1) has two 3 poles and 3 zeroes hardware filters.

32.1 Overview

Each filter provides a configuration hardware filter function, three poles/three zeros with the following features:

- 3 poles and 3 zeroes
- 16-bit input, q0.15
- 24-bit output, q0.23
- 24-bits coefficients, q0.23
- 48-bit multiplier
- 58-bit internal accumulator
- 2-bit gain input and 3-bit attenuator output
- Saturation and rounding operators supported
- Arithmetic two's complement
- Max 9 CPU clock cycles execution latency
- Filter processing triggered by writing into one of the two data registers
- Generates a pulse event at the completion of filter computation
- Register slave interface (connected to ADC_FILT outputs)

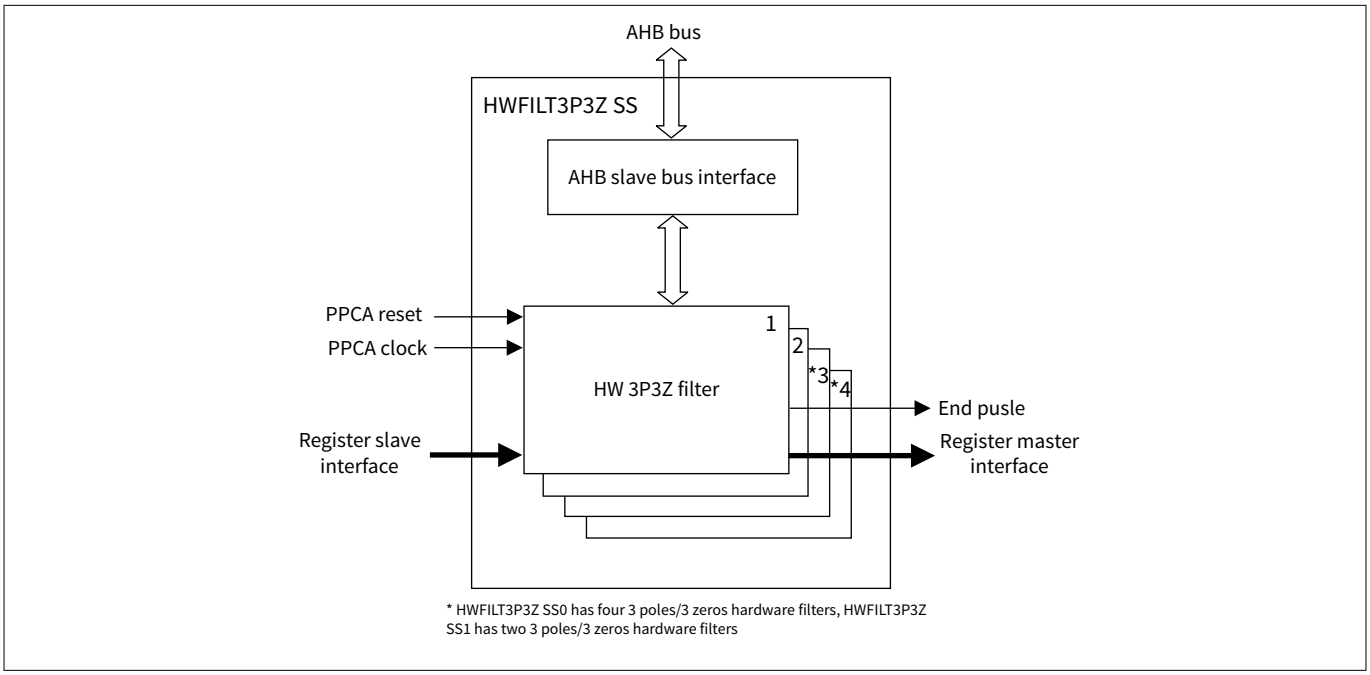


Figure 338 Hardware filter block diagram

All filter data registers are sign-extended to a full 32-bit width when read through the AHB interface.

Hardware filter equations:

$$
\begin{gathered}
Q_y = 24 \\
x(n) = \left(x_{ref} - xin(n)\right) < \ < Gin \\
acc(n) = \Sigma_{i=0}^{3}\left(CX_i \cdot x\left(n-i\right)\right) + \Sigma_{j=1}^{3}\left(CY_j \cdot y\left(n-j\right)\right) + 2^{-Q_y} \\
y\left(n\right) = \begin{cases} acc(n) > \ > \left(Q_y - 1\right), & -1 \le acc(n) < 1 \\ 1, & acc(n) \ge 1 \\ -1, & acc(n) < \ -1 \end{cases} \\
y_tmp(n) = (y(n) > \ > Gout) + Offset \\
yout\left(n\right) = \begin{cases} y_tmp(n), & LimMin \le y_tmp(n) \le LimMax \\ LimMax, & y_tmp(n) > LimMax \\ LimMin, & y_tmp(n) < LimMin \end{cases}
\end{gathered}
\tag{8}
$$

Figure 339 shows the hardware filter block diagram with used registers:

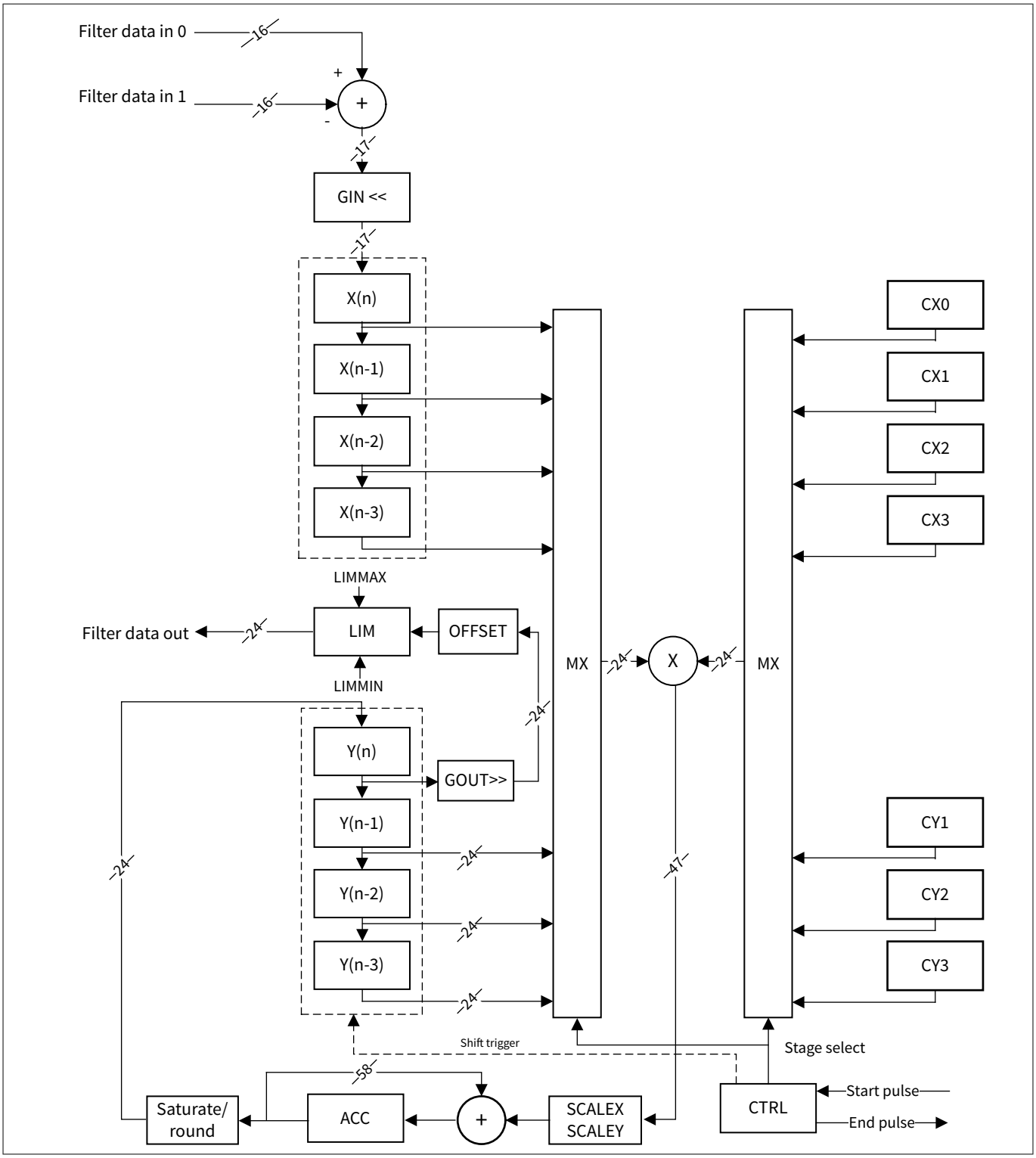


Figure 339 Hardware filter processing block diagram

32.2 Trigger filter processing

The filter processing can be triggered in the following ways:

1. If HWFILT3P3Z_SS_x_HWFILT3P3Zy_CNFG.EN_TRIG0 = 1, the processing is started by writing into HWFILT3P3Z_SS_x_HWFILT3P3Zy_DATA_IN0 register
2. If HWFILT3P3Z_SS_x_HWFILT3P3Zy_CNFG.EN_TRIG1 = 1
   a. If HWFILT3P3Z_SS_x_HWFILT3P3Zy_CNFG.SRC_SEL = 1, the processing is started by the trigger signal in the register slave interface. Writes to the HWFILT3P3Z_SS_x_HWFILT3P3Zy_DATA_IN1 register updates the register without triggering filter processing
   b. If HWFILT3P3Z_SS_x_HWFILT3P3Zy_CNFG.SRC_SEL = 0, the processing is started by writing into HWFILT3P3Z_SS_x_HWFILT3P3Zy_DATA_IN1 register

***Note**: HWFILT3P3Z_SS_x_HWFILT3P3Zy_CNFG.EN_TRIG0 and HWFILT3P3Z_SS_x_HWFILT3P3Zy_CNFG.EN_TRIG1 should not be set at the same time.*

32.3 Data path connection

The HWFILT3P3Z also supports data path connection, where an embedded register is feed directly from the output of the ADC_FILT module through the register interface.

The following figure shows the ADC filtering data passing to hardware filter block, and from hardware filter to TCPWM block, it could drive a new duty cycle of PWM without CPU intervention.

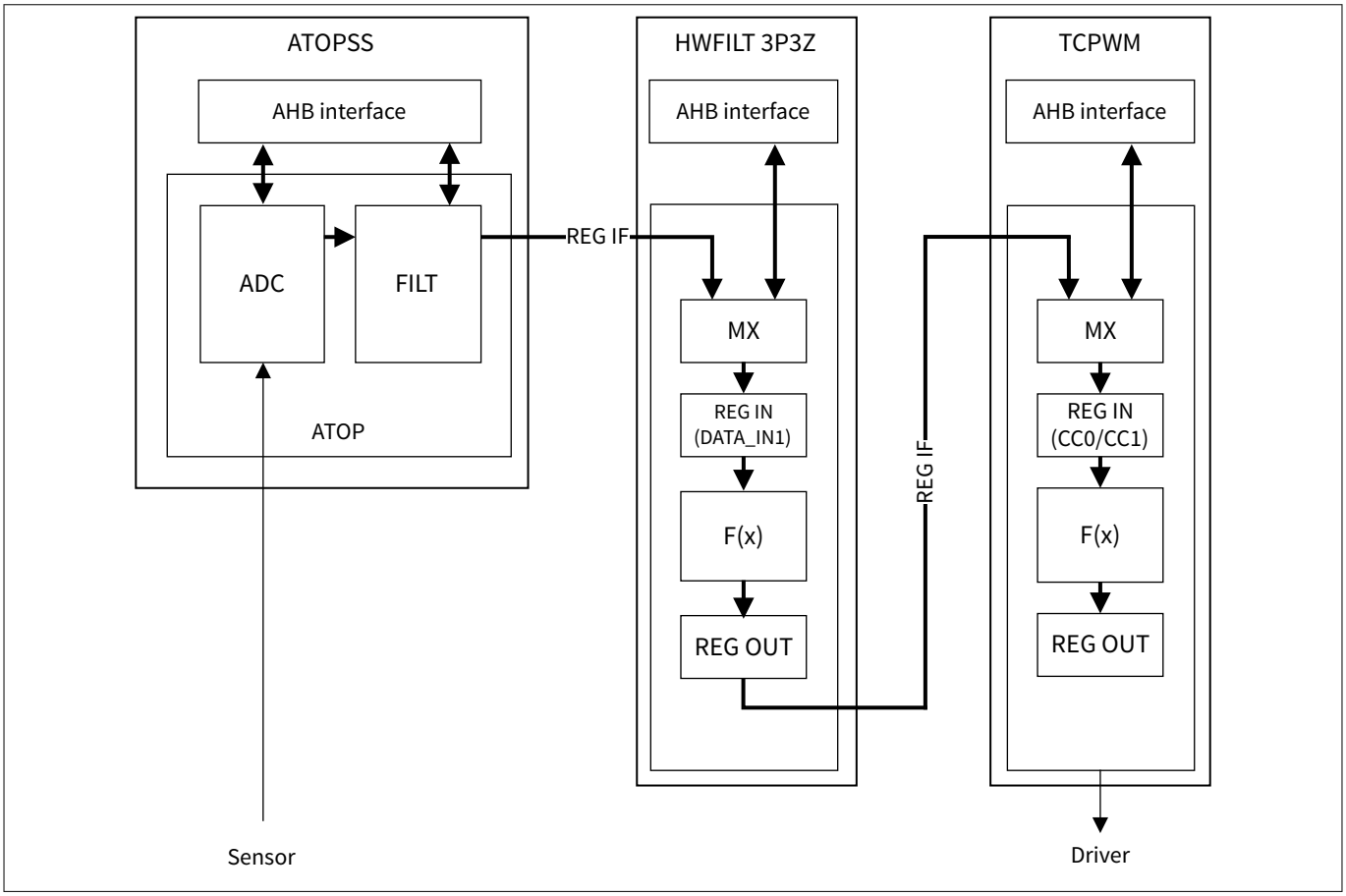


Figure 340 Example of data path connection

32.4 Registers summary

Table 314 Hardware filter registers summary

Register	Function
HWFILT3P3Z_SS_x_CTRL	Hardware filter module control register
HWFILT3P3Z_SS_x_HWFILT3P3Zy_CTRL	Filter control register
HWFILT3P3Z_SS_x_HWFILT3P3Zy_DATA_IN0	Filter data input 0 register
HWFILT3P3Z_SS_x_HWFILT3P3Zy_DATA_IN1	Filter data input 1 register
HWFILT3P3Z_SS_x_HWFILT3P3Zy_CNFG	Filter configuration register
HWFILT3P3Z_SS_x_HWFILT3P3Zy_CX0	Filter coefficient X0
HWFILT3P3Z_SS_x_HWFILT3P3Zy_CX1	Filter coefficient X1
HWFILT3P3Z_SS_x_HWFILT3P3Zy_CX2	Filter coefficient X2
HWFILT3P3Z_SS_x_HWFILT3P3Zy_CX3	Filter coefficient X3
HWFILT3P3Z_SS_x_HWFILT3P3Zy_CY1	Filter coefficient Y1
HWFILT3P3Z_SS_x_HWFILT3P3Zy_CY2	Filter coefficient Y2
HWFILT3P3Z_SS_x_HWFILT3P3Zy_CY3	Filter coefficient Y3
HWFILT3P3Z_SS_x_HWFILT3P3Zy_DATA_OUT	Filter data output register
HWFILT3P3Z_SS_x_HWFILT3P3Zy_OFFSET	Filter offset register
HWFILT3P3Z_SS_x_HWFILT3P3Zy_LIMMAX	Filter out maximum limit register
HWFILT3P3Z_SS_x_HWFILT3P3Zy_LIMMIN	Filter out minimum limit register
HWFILT3P3Z_SS_x_HWFILT3P3Zy_SCALECX	Scale X coefficients factor register
HWFILT3P3Z_SS_x_HWFILT3P3Zy_SCALECY	Scale Y coefficients factor register
HWFILT3P3Z_SS_x_HWFILT3P3Zy_GIN	Amplify input difference factor register
HWFILT3P3Z_SS_x_HWFILT3P3Zy_GOUT	Scale output register
HWFILT3P3Z_SS_x_HWFILT3P3Zy_VERSION	Filter version register

***Note**: In HWFILT3P3Z_SS_x_HWFILT3P3Zy, 'x' signifies hardware filter instance number (x=0 or 1), 'y' is the filter number in the respective filter instance (y = 0 - 3 for instance 0, y = 0 - 1 for instance 1).*

33 PPCA Timer, counter, and pulse-width modulator (TCPWM)

The timer, counter, and pulse-width modulator (TCPWM) block implements a 16- or 32-bit timer, counter, pulse width modulator (PWM), high-resolution PWM (HRPWM), and motion interface (MOTIF) functionality. The PPCA in PSC3M/P8, PSC3M/P7 MCU contains four TCPWM blocks. Each TCPWM block includes multiple counter groups. 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), and generate PWM signals. The TCPWM block works in Active and Sleep modes.

Compare to TCPWM block of the main CPU, PPCA offers four TCPWM instances with more 32-/16-bit counters and features, supporting MOTIF and less than 100 ps high-resolution pulse-width modulator (HRPWM). This chapter explains the features, implementation, and operational modes of PPCA TCPWM block.

33.1 Features

The TCPWM block has the following features[4]:

- Supports up to eight counter groups
- Each counter group consists up to 256 counters (counter group specific)
- Each counter can run in one of the following function modes:
  - Timer
  - Capture
  - PWM
  - PWM with dead time (PWM_DT)
- 16-bit or 32-bit counters (counter group specific)
- Up, down, and up/down counting modes
- Clock pre-scaling (1x, 2x, 4x, 8x, 16x, 32x, 64x, 128x)
- Up to two captures and compare functions (counter group specific)
- Double buffering of all compare/capture and period registers
- Shadow update of the fractional shift of the edges through PERIOD_BUFF, CC0_BUFF, CC1_BUFF and DT_BUFF 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 “HiZ”
- 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 (counter group 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

[4] See Table 315 to check which functions are supported and the configuration for counter group.

- Supports MOTIF (motion interface) extension block at the TCPWM input (counter group specific)
  - Supports Quadrature decoder mode
  - Supports Hall sensor mode
  - Supports multi-channel mode
- Support HRPWM (high resolution PWM generation) feature in 32-bit TCPWM counters
- HRPWM is limited to PWM and PWM_DT mode only
- Following PWM output modes are supported:
  - Right align
  - Left align
  - Center align
  - Asymmetric align with CC0
  - Asymmetric align with CC0 and CC1
  - Asymmetric center align with CC0 and CC1
- Parallel data path to update CC0/CC1 register directly through a data interface
- Fractional shift on the edges for PERIOD, CC0, CC1, and DT registers on counters with HRPWM feature
- Run time enable the HRPWM feature by software/firmware (HRPWM_CTRL.HRPWM_EN='1')
- Debug mode support

33.1.1 Supported features of PPCA TCPWM blocks

The following table lists the counter group supported features of TCPWM blocks in PPCA.

Table 315 Counter group features of TCPWM blocks

Features[1)]	TCWPM0			TCPWM1			TCPWM2	TCPWM3
	Group 0	Group 1	Group 2	Group 0	Group 1	Group 2	Group 0	Group 0
Number of counters per TCPWM group	4	4	4	2	2	4	4	2
Counter width in number of bits per TCPWM group (GRP_CNT_WIDTH)	32-bit	32-bit	16-bit	32-bit	32-bit	16-bit	16-bit	16-bit
Number of input triggers per counter only routed to one counter (TR_ONE_CNT_NR)	1			1			1	1

(table continues...)

Table 315 (continued) Counter group features of TCPWM blocks

Features[1)]	TCWPM0			TCPWM1			TCPWM2	TCPWM3
	Group 0	Group 1	Group 2	Group 0	Group 1	Group 2	Group 0	Group 0
Number of input triggers routed to all counters (TR_ALL_CNT_NR)	6			6			1	1
Second capture/ compare Unit (CC1) is present (GRP_CC1_PRESENT)	1	1	1	1	1	1	1	1
Advanced motor control features (AMC) are present (GRP_AMC_PRESENT)	1	1	1	1	1	1	1	1
Stepper motor control (SMC) features are present (GRP_SMC_PRESENT)	1	1	1	1	1	1	1	1
Parallel data path for CC0 and CC1 register present (GRP_DATA_IN_CC_PRESENT)	1	N/A	N/A	1	N/A	N/A	N/A	N/A
HRPWM feature extension is available for counter group (GRP_HRPWM_PRESENT)	1	1	N/A	1	1	N/A	N/A	N/A

(table continues...)

Table 315 (continued) Counter group features of TCPWM blocks

Features[1)]	TCWPM0			TCPWM1			TCPWM2	TCPWM3
	Group 0	Group 1	Group 2	Group 0	Group 1	Group 2	Group 0	Group 0
Number of micro ticks bits supported in HRPWM feature (GRP_HRPWM_WIDTH)	6	6	N/A	6	6	N/A	N/A	N/A
Number of TCPWM MOTIF instances units per counter group (GRP_MOTIF_NR)	N/A	N/A	1 MOTIF instance	N/A	N/A	1 MOTIF instance	N/A	N/A

1) 1: the feature is present; N/A: the feature is not present.

33.2 Architecture

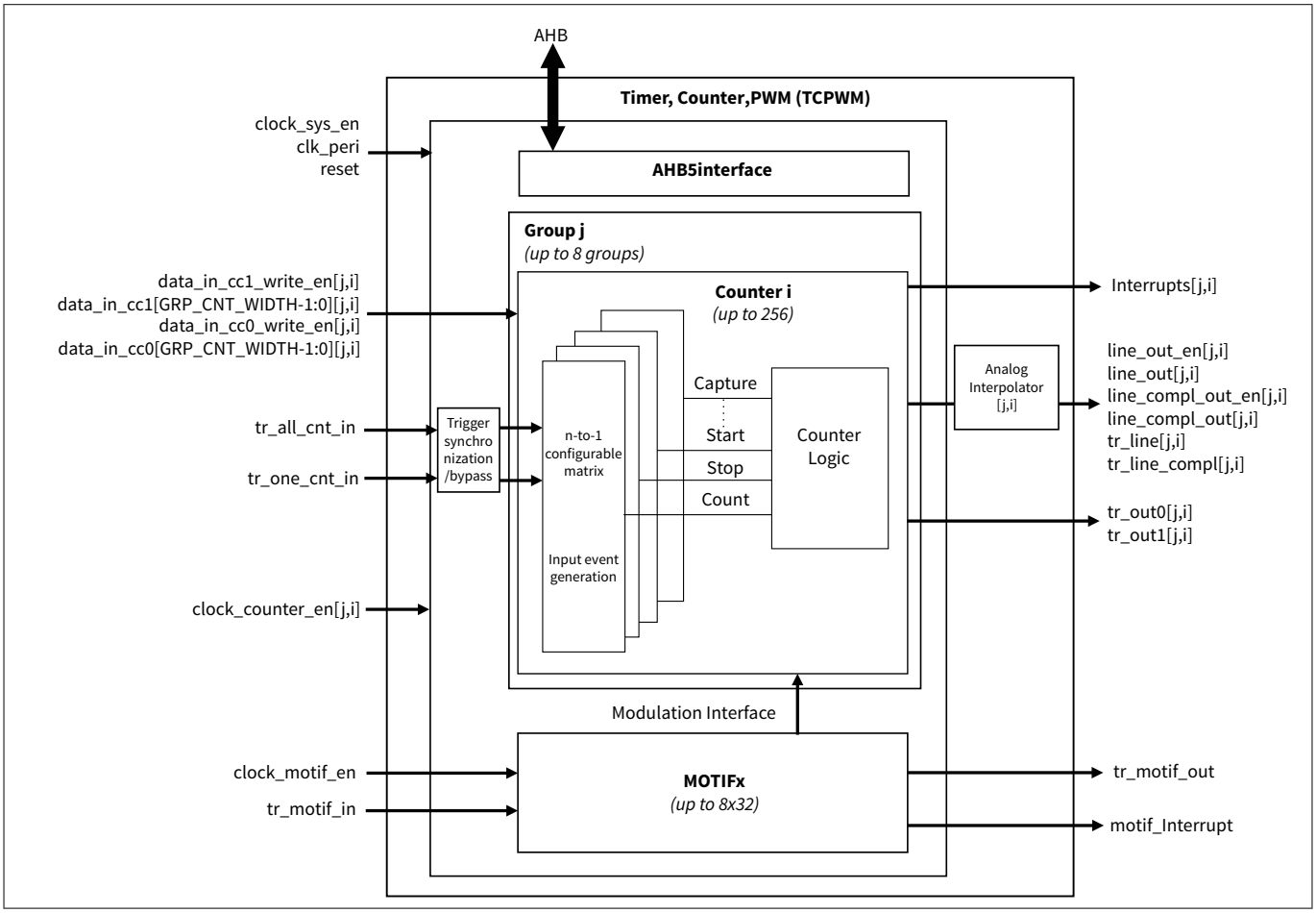


Figure 341 TCPWM block diagram

The TCPWM block has up to 8 groups, with up to 256 counters in each group and up to 8 x 32 MOTIF instances, each counter group can interface with up to 32 MOTIF instances (device specific). The bus interface is the AHB5 interface. Input triggers will be synchronized by 2DFF synchronizer (Default enable) before they are detected by each counter and MOTIF module. The 2DFF synchronizer can be bypass for an input trigger going into the counter. The above figure illustrates the structure of the TCPWM with HRPWM support.

***Note**: This document does not discuss the specific counter group configuration in detail. For example: if a second capture/compare feature is mentioned, refer Table 157 of main CPU TCPWM and Table 315 of PPCA TCPWM to know if these functions are available in TCPWM block.*

Each counter can have up to 254 input trigger signals and two constant input signals, '0' and '1'; all of them are synchronized with the CLK_PERI clock.

The TCPWM block has these interfaces:

- **Bus interface**: Connects the block to the CPU subsystem via AHB5 interface
- **I/O signal interface**: Consists of input triggers (such as reload, start, stop, count, capture0/1) and output signals (such as LINE_OUT, LINE_COMPL_OUT, TR_OUT0, and TR_OUT1). All the input signals are used to trigger an event within the counter, such as a reload trigger generating a reload event. The output signals are generated by internal events (underflow, overflow, and capture/compare, tc) and can be connected to other peripherals to trigger events

- **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 TCPWM registers summary chapter for more information on all registers required for this block.

33.2.1 Enabling and disabling counters in a 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 configuration registers.*

33.2.2 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[i] for clk_counter of each counter.

Each TCPWM counter can have its own clock. The only source for the clock is from the configurable peripheral clock dividers generated by the clocking system; see the Clocking system chapter for details. To select a clock divider for a particular counter inside a TCPWM, use the CLOCK_CTL register from the peripheral register space. In this section, the clock to the counter is called clk_counter. Event generation is performed on clk_counter. Another clock, system clock (clk_sys), is used for the pulse width of the output triggers. The system clock (clk_sys) is synchronous to clk_peri, but can be divided using CLOCK_CTL.

33.2.2.1 Clock pre-scaling

The clk_counter can be further divided inside each counter, with values of 1, 2, 4, 8, 16, 32, 64, and 128. This division is called pre-scaling. The pre-scaling is set in the PS_DIV [2:0] field of the PS register.

33.2.2.2 Counter operation clock

Event generation is performed on CLK_PERI (See Event edge detection). Other counter functionality is performed on an “active count” pre-scaled clock. “active count” indicates that the count event is active. If the PWM mode selects an input trigger as a count event, the “active count” pre-scaled counter clock is only present when the event is activated. The following figure illustrates the functionality.

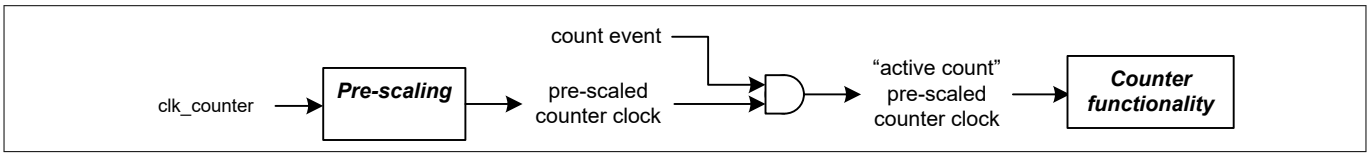


Figure 342 Generation of counter operating clock

All status or output change can only happen at the “active count” pre-scaled counter clock. In the other words, if the count event is inactive, counter, status, interrupt and all outputs will not change value. For example, in the case that the count event is pass through mode, if count event becomes low when the counter goes to “0” in down count mode, the terminal count (tc) event and underflow event will be generated at the next pre-scaled 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” pre-scaled counter clock.

33.2.3 Trigger inputs

Operation of each counter can be dynamically controlled by eight trigger input events: reload, start, stop, count, capture0, capture1, duty cycle and masking event. The specific function of these events is operation mode specific, but typically, they can be described as follows:

- A reload event initializes and starts the counter. A reload event should only be used when the counter is not running
- A start event starts the counter (the counter is not initialized). A start event should only be used when the counter is not running
- A stop event stops the counter
- A count event increments or decrements the counter
- A capture0 event copies COUNTER to CC0
- A capture1 event copies COUNTER to CC1 (only available when parameter GRP_CC1_PRESENT = 1)
- An external trigger event for duty cycle control
- An external trigger event for masking PWM output

Typical operation will use the reload event once to initialize and start the counter. The stop event is used 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.

In case 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.

Both external hardware (HW) input signals and software (SW) command can be used to generate input event for counter operation:

1. **External hardware input**: Configure input events edge detection mode and select one of the input triggers (tr_all_cnt_synced, tr_one_cnt_synced) which are synchronized from hardware input triggers, constant “0” or constant “1”. See Hardware trigger input for details
2. **Software command**: Program software (SW) trigger command register (TR_CMD) to generate input events. See Software trigger command for details

The following figure shows how counter input event is generated through hardware or software, the same logic is duplicated for six events of individual counter.

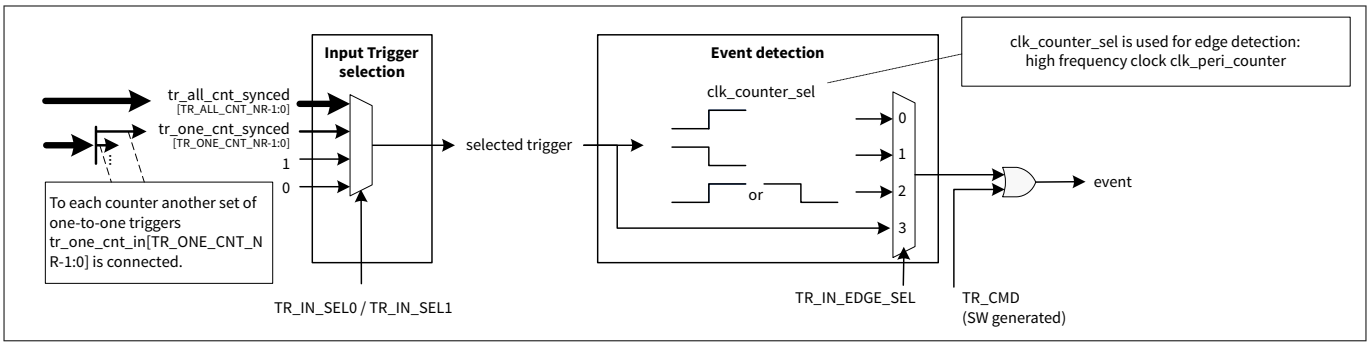


Figure 343 Counter input event generation

33.2.3.1 Hardware trigger input

Each counter can use up to 256 input triggers (Up to 254 external hardware input triggers are synchronized by 2DFF synchronize, constant “1” and constant “0”) to generate events to dynamically control counter operation.

***Note**: The number of external hardware input triggers specified by TR_ONE_CNT_NR and TR_ALL_GF_NR parameters of TCPWM block.*

To this end, each counter has an event generation component for all six input events, to proceed input trigger selection and edge detection. Please note that 2DFF synchronizer can be bypass through TR_ALL_SYNC_BYPASS register setting. This helps to reduce latency when the trigger source is running on the same clock root as TCPWM.

Input trigger selection

Two types of trigger signals are synchronized and can be used by the counters to generate events. The constant “0” and constant “1” can also be selected for events generation.

The constant input triggers are useful to generate constant counter events, for example, a timer that increments the counter on every counter clock cycle would select a constant ‘1’ input trigger as its count event. They are connected to counter trigger[0] and trigger[1].

- **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 or event processing unit). See the Trigger multiplexer (Trigger MUX) chapter or Event Processing Unit (EPU) chapter for more details
- **One-to-one triggers**: A separate set exists for each counter that is 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 #1 Counter #00

Use the input trigger selection registers TR_IN_SEL0 and TR_IN_SEL1 to configure which signals get routed to the Trigger_In for each TCPWM block. See Table 316 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 trigger signals to be the source for any of the following events:

- Capture 0 and Capture 1
- Count
- Reload
- Stop/Kill
- Start
- Duty cycle
- Masking

Table 316 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 input triggers as a capture0 trigger. In the PWM, PWM_DT modes, this trigger is used to switch the values of the compare and period registers with their buffer counterparts.
	COUNT_SEL	15:8	Selects one of the input triggers as a count trigger.
	RELOAD_SEL	23:16	Selects one of the input triggers as a reload trigger.
	STOP_SEL	31:24	Selects one of the input triggers as a stop trigger. In PWM, PWM_DT modes, this is the kill trigger.
TR_IN_SEL1	START_SEL	7:0	Selects one of the input triggers as a start trigger.

(table continues...)

Table 316 (continued) Multiplexer selection for input trigger events

Input trigger selection register	Bit Field	Bits	Description
	CAPTURE1_SEL	15:8	Selects one of the input triggers as a capture1 trigger.
	DC_SEL	23:16	Selects one of the input triggers as an external trigger for duty cycle control.
	MASK_SEL	31:24	Selects one of the input triggers as mask input signal.

As illustrated in Figure 343, each event can select one of the input triggers through register TR_IN_SEL0 and TR_IN_SEL1, with the following multiplex input order:

- **Constant '0'**
  Input trigger selection value is 0
- **Constant '1'**
  Input trigger selection value is 1
- **Specific one-to-one input triggers "tr_one_cnt_in"**
  Input trigger selection value is in range [2: (TR_ONE_CNT_NR + 1)]
- **General-purpose input triggers "tr_all_cnt_in"**
  Input trigger selection value is in range [(TR_ONE_CNT_NR + 2): (TR_ALL_CNT_NR) + (TR_ONE_CNT_NR + 1)]

***Note**: TR_ONE_CNT_NR: The number of input triggers per counter is only routed to one counter.*

***Note**: TR_ALL_CNT_NR: Number of input triggers routed to all counters.*

Event edge detection

The TR_IN_EDGE_SEL register specifies the event detection mechanism:

- A rising edge on the selected trigger generates an event
- A falling edge on the selected trigger generates an event
- A rising/falling edge on the selected trigger generates an event
- The selected trigger is treated as an event

Event detection is on the external event generation counter clock "clk_counter_sel". This clock is a gated version of "clk_peri". As a result, trigger frequencies should be equal or less than the clock frequency to ensure that they are successfully detected. In all modes, "clk_counter_sel" equals "clk_peri": event detection is on the peripheral clock.

Edge detection will remember the detected event till the next "active count" pre-scaled counter clock on which the event is used:

- Start and reload events are used on "active count" pre-scaled counter clocks
- Capture0 event is used on "active count" pre-scaled counter clocks or on "active count" pre-scaled counter clocks that have a tc (depend on function mode) event if it is used as switch event
- Stop events are used on "active count" pre-scaled counter clocks or on "active count" pre-scaled counter clocks that have a tc event (depend on the stop kill mode)
- Capture1 event (if GRP_CC1_PRESENT = "1") could be second capture event or kill/stop event, depends on different function mode

The following figure illustrates the timing on how input triggers are detected by the counter.

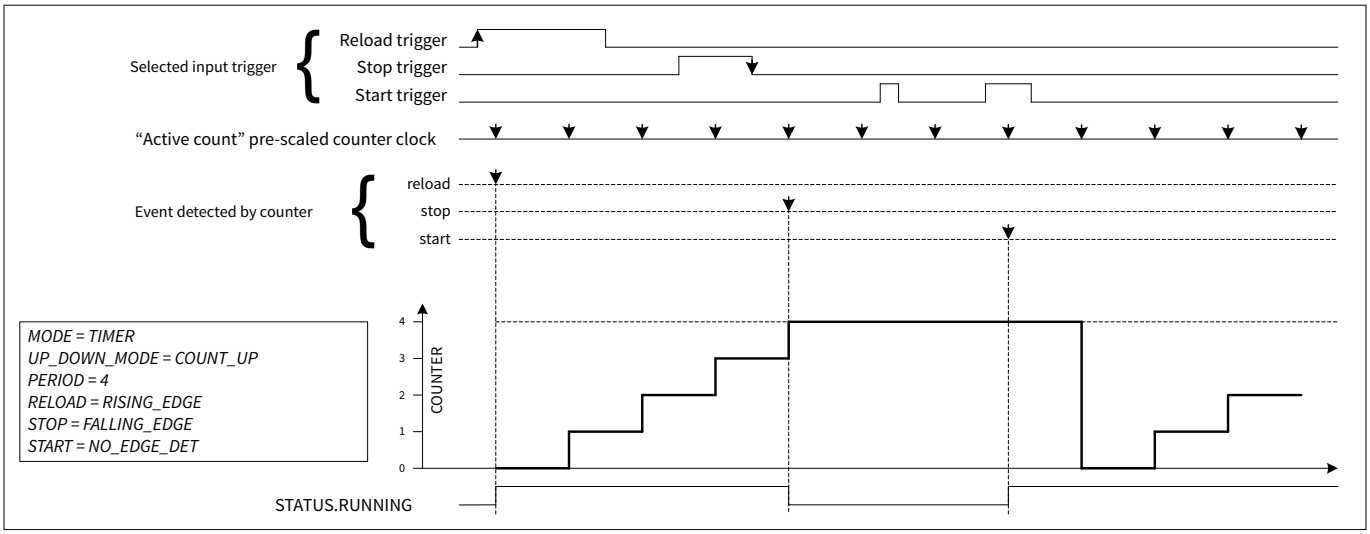


Figure 344 Input trigger detection by "active count" prescaled counter clock

***Note**: The event marked as an arrow in afterward figures depicts the event that has been detected by the counter.*

Two examples explain how edge detection event working on "active count" pre-scaled counter clocks:

1. 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 next pre-scaled counter clock after count event becomes active
2. In capture mode, if rising edge capture0 inputs while count event is inactive, CC0 and CC0_BUFF will get updated at next pre-scaled counter clock after count event becomes active

Typically, the count event is a constant '1' and pre-scaling is off. In this case, the "active count" pre-scaled 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" pre-scaled 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

Pass-through event will not be remembered by CLK_PERI, it will affect functionality as long as it lasts if it can be detected by counter operation clock as described in 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 event in PWM/PWM_DT mode, if it selects the constant "high" as the source.

33.2.3.2 Software trigger command

The previous sections described the hardware-based event generation. In addition, software-based event generation is supported: the reload, start, stop, capture0, capture1, duty cycle and masking events can be generated by writing to the TCPWM counter trigger command registers (TR_CMD). 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 PERI0_TR_CMD register in the PERI block or the configuration register of EPU block. See the Trigger multiplexer (Trigger MUX) section or PPCA Event Processing Unit (EPU) section for more details.

The following figure illustrates an example of how to start multiple counters with PERI0_TR_CMD of the PERI block synchronously.

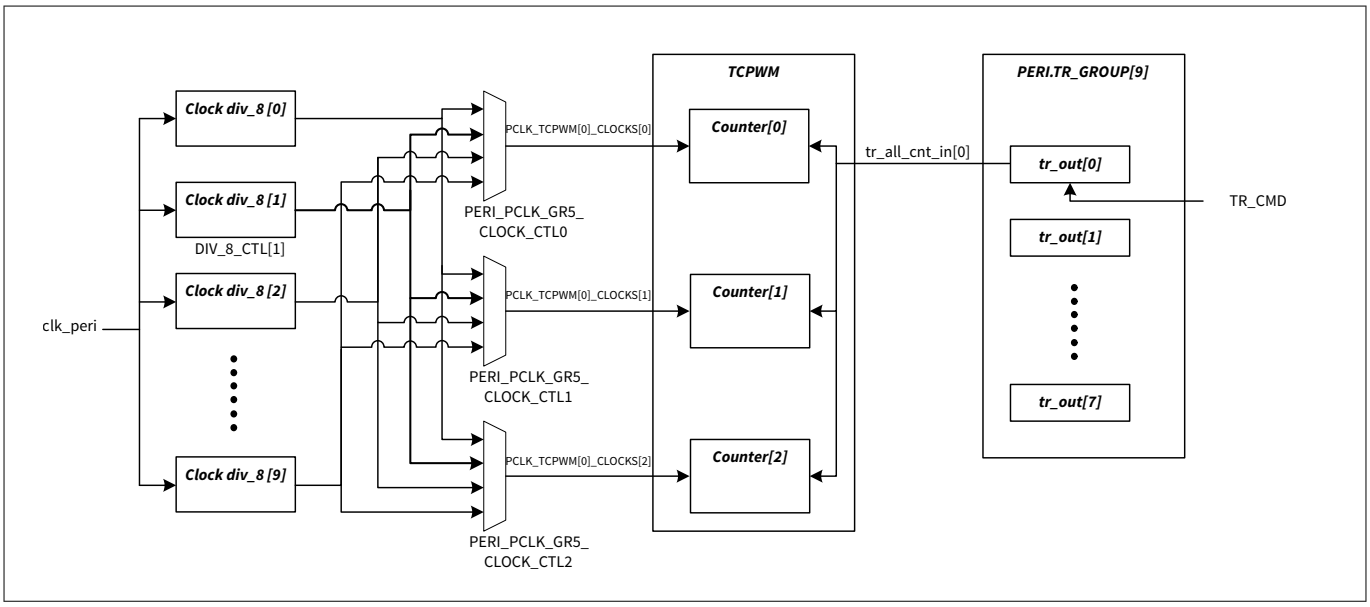


Figure 345 PERI TR_CMD synchronously starts counters

***Note**: Enable real synchronous operation of multiple counters, the pre-scaled clock should not be used (PS[2:0] = 0). The reason is that pre-scaled clock is free running after the counter is enabled, on the other words, different counters' pre-scaled counter clocks are not in sync because they are not enabled at the same time. While the trigger input is detected by a pre-scaled counter clock, the counter operation cannot be synchronized.*

The following example describes the required steps to start counters synchronously:

- Configure PERI_PCLK_GR5_CLOCK_CTL0/1/2 to select the 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 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 the start or reload event is detected
- Configure the three counters start event, all selecting tr_all_cnt_in[0]; this means that the three counters will run synchronously when tr_all_cnt_in[0] asserts
- Use TR_CMD to generate a trigger pulse on TR_GROUP[9].TR_OUT[0]. All three counters will run synchronously

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.

The following figure shows how software command bit works and how it is detected and cleared by the counter.

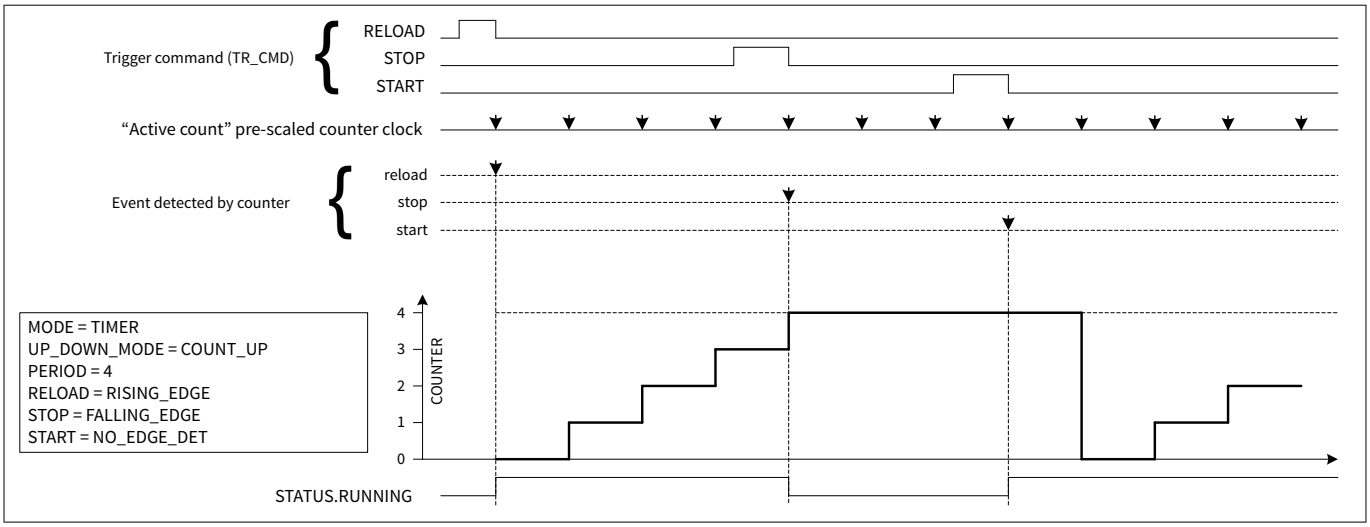


Figure 346 Software trigger command detection by active count prescaled counter clock

33.2.4 Internal events

33.2.4.1 Underflow event

An underflow event indicates that in down counting, the COUNTER equals zero, and changes to a different value. Reload also generates an underflow event in some specific mode. The following table summarizes the underflow generation of each function mode.

Table 317 Underflow generation

Mode	UP	DOWN	UPDN1	UPDN2
Timer	Counter is decremented and changes from a state in which COUNTER equals 0. Reload event in DOWN, UPDN1, and UPDN2 modes.			
Capture	Counter is decremented and changes from a state in which COUNTER equals 0. Reload event in DOWN, UPDN1, and UPDN2 modes.			
PWM	Counter is decremented and changes from a state in which COUNTER equals 0. Reload event in DOWN, UPDN1, and UPDN2 modes.			
PWM_DT	Counter is decremented and changes from a state in which COUNTER equals 0. Reload event in DOWN, UPDN1, and UPDN2 modes.			

33.2.4.2 Overflow event

An overflow event indicates that in up counting, the COUNTER equals PERIOD, and changes to a different value. Reload also generates an overflow event in some specific mode.

The following table summarizes the overflow generation of each function mode.

Table 318 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.			

(table continues...)

Table 318 (continued) Overflow generation

Mode	UP	DOWN	UPDN1	UPDN2
Capture	Counter is incrementing and changes from a state in which COUNTER equals PERIOD. Reload event in UP count mode.			
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.			

33.2.4.3 Terminal count event

A terminal count (TC) event is the logical OR of the underflow and overflow events. An exception is that a reload event generates an underflow or overflow, but not a TC event.

The following table summarizes the TC generation of each function mode.

Table 319 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
PWM	Overflow	Underflow	Underflow	Logic OR of overflow and underflow
PWM_DT	Overflow	Underflow	Underflow	Logic OR of overflow and underflow

33.2.4.4 cc0_match (cc1_match) event

A cc0_match event indicates that the COUNTER equals CC0. This event is either generated when the COUNTER is about to change to CC0, or when the 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 PWM mode chapter. In other specific operation modes, the event is used to indicate that the CC0/CC0_BUFF registers are updated. cc1_match is generated based on the state of the COUNTER and CC1; other behaviors are the same as cc0_match.

The following tables summarize the cc0/1_match generation of each function mode.

Table 320 cc0_match generation

Mode	UP	DOWN	UPDN1	UPDN2
Timer	Counter changes from a state in which COUNTER equals CC0			
Capture	Capture0 event			

(table continues...)

Table 320 (continued) cc0_match generation

Mode	UP	DOWN	UPDN1	UPDN2
PWM	Counter changes to a state in which COUNTER equals CC0 or COUNTER passed the CC0 value Left aligned case: CC0 <= COUNTER Right aligned case: CC0 >= COUNTER		COUNT_UPDN1/2: counter changes from a state in which COUNTER equals CC0. If the advanced motor control feature is present in a counter group, CC0_MATCH_DOWN_EN/ CC0_MATCH_UP_EN will enable/disable cc0_match generation	
PWM_DT	Counter changes to a state in which COUNTER equals CC0 or COUNTER passed the CC0 value Left aligned case: CC0 <= COUNTER Right aligned case: CC0 >= COUNTER		COUNT_UPDN1/2: counter changes from a state in which COUNTER equals CC0. If the advanced motor control feature is present in a counter group, CC0_MATCH_DOWN_EN/ CC0_MATCH_UP_EN will enable/disable cc0_match generation	

Table 321 cc1_match generation

Mode	UP	DOWN	UPDN1	UPDN2
Timer	Counter changes from a state in which COUNTER equals CC1			
Capture	Capture1 event			
PWM	Counter changes to a state in which COUNTER equals CC1 or COUNTER passed the CC1 value Left aligned case: CC1 <= COUNTER Right aligned case: CC1 >= COUNTER		COUNT_UPDN1/2: counter changes from a state in which COUNTER equals CC1. If the advanced motor control feature 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 or COUNTER passed the CC1 value Left aligned case: CC1 <= COUNTER Right aligned case: CC1 >= COUNTER		COUNT_UPDN1/2: counter changes from a state in which COUNTER equals CC1. If the advanced motor control feature is present in a counter group, CC1_MATCH_DOWN_EN/ CC1_MATCH_UP_EN will enable/disable cc1_match generation.	

33.2.5 Trigger outputs

Each counter has two trigger output signals (TR_OUT0 and TR_OUT1) and they can be routed through the trigger mux to other peripherals on the device. The bit field OUT0 in the TR_OUT_SEL register is used to select one of the internal events to generate output trigger 0 (TR_OUT0), and the bit field OUT1 selects one of the internal events to generate TR_OUT1. Also, it allows disabling of the output triggers.

The following are the events indicated by the counter trigger output signal:

- Overflow (OV): An overflow event indicates that in up counting mode, COUNTER equals the PERIOD register, and is changed to a different value

- Underflow (UN): An underflow event indicates that in down counting mode, COUNTER equals 0, and changes to a different value
- TC (Terminal count): 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 occurs:
  - Counter equals the compare value. This event is either generated when the match is about to occur (COUNTER does not equal CC0/1 and changes to CC0/1) or when the match is not about to occur (COUNTER equals CC0/1 and changes to a different value)
  - A capture event has occurred and the CC0 (CC1) and CC0_BUFF (CC1_BUFF) registers are updated

  ***Note**: CC1_MATCH event is only available when parameter GRP_CC1_PRESENT = 1*
- CC0_CC1_MATCH: This event is triggered when either CC0_MATCH or CC1_MATCH occurs
- 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, the output trigger bypasses the two-cycle pulses generation logic and directly outputs LINE_OUT.

The generated triggers have two main uses:

- Initiating a DMA data transfer. For example, in the 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)

33.2.6 PWM outputs

The PWM, PWM_DT 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”)

The following figure illustrates the process of “line_out” and “line_compl_out” generation.

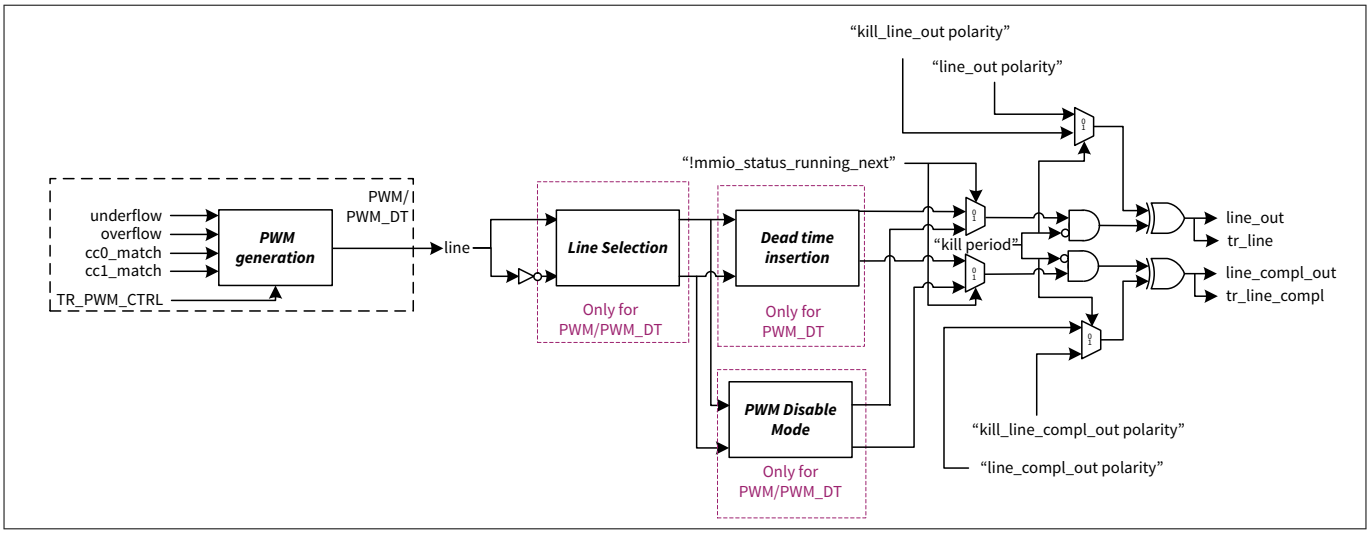


Figure 347 PWM output generation process in PWM/PWM_DT mode

Generally, "line_out" output reflects the state of PWM signal "line" and "line_compl_out" output reflects the inverted version of "line". The behavior of a line depends on different function modes. Furthermore, some other facts will impact "line_out" and "line_compl_out" per different function modes, they are "line selection", "dead time insertion", "kill function", "line polarity" and "kill line polarity".

***Note**: The tr_line is a replica of line_out and tr_line_compl is a replica of line_compl_out.*

- 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
  - The final step is inversion of PWM using quad_encoding_mode bits of CTRL register. Quad_encoding_mode[0] inverts line_out, while Quad_encoding_mode[1] inverts line_compl_out when counter is enabled in PWM/PWM_DT modes
- Line selection (Available when parameter GRP_SMC_PRESENT = 1)
  - "line_out" and "line_compl_out" can individually selects different output per settings of registers "LINE_SEL.OUT_SEL" and "LINE_SEL.COMPL_SEL". This functionality only works in PWM and PWM_DT mode
  - "line_out" and "line_compl_out" can individually selects "Low", "High", "Line", "Inverted line" and "HiZ", When selects "HiZ", "line_out_en" and "line_compl_out_en" will be low
- Dead time insertion
  - Dead time insertion functionality is mutual 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 PWM outputs "line_out" and "line_out_compl_out" while the TCPWM counter is disabled (CTRL.ENABLED=0) or stopped. The output signal of this selection can be further modified by the immediate kill logic and line_out polarity settings (CTRL.KILL_LINE_POLARITY/ CTRL.QUAD_ENCODING_MODE). There are three modes when enabled:
    - Retain

      When the counter is disabled or stopped upon a stop event the PWM outputs are retained (keep their previous levels). While the counter is disabled or stopped the PWM outputs can be changed via LINE_SEL (when parameter GRP_SMC_PRESENT = 1)
    - Low

When the counter is disabled or stopped upon a stop event the PWM output “line_out” is driven as a fixed ‘0’ and the PWM output “line_compl_out” is driven as a fixed ‘1’

- High

  When the counter is disabled or stopped upon a stop event the PWM output “line_out” is driven as a fixed ‘1’ and the PWM output “line_compl_out” is driven as a fixed ‘0’

- The behavior of line_out during disabled mode as combination of quad_encoding_mode bits and pwm_disable_mode bits is described in the following table

Table 322 line_out generation in disabled mode

PWM_DISABLE_MODE	line_out	line_compl_out
2’b00	High impedance	High impedance
2’b01	Retain	Retain
2’b10	quad_encoding_mode[0]	quad_encoding_mode[1]
2’b11	quad_encoding_mode[0]	quad_encoding_mode[1]

***Note**: During PWM disable mode 00, line_out_en and line_compl_out_en is set to zero and output of line_out and line_compl_out from GPIO is high impedance*

- Kill function
  - Kill function works in PWM and PWM_DT 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, the PWM output signals values are determined by their respective polarity settings
- Kill Polarity for “line_out” and “line_compl_out”
  - Kill Polarity inversion is used to determine the “line_out” and “line_compl_out” output signal values during kill event
  - CTRL.KILL_LINE_OUT_POLARITY is for “line_out” polarity and CTRL.KILL_LINE_COMPL_OUT_POLARITY is for “line_compl_out” polarity. The polarity is the applied on the kill behavior of the “line_out’ and “line_compl_out”
  - When the counter is not running ((temporarily) stopped or killed), the PWM output signals values are determined by their respective kill polarity settings
- Dead-time polarity for “line_out” and “line_compl_out”
  - Dead-time polarity depends on whether dead-time is being inserted because of kill (as in tc_sync_kill_dt or sync_kill_dt) or posedge of line outputs
  - For posedge of line_out and line_compl_out, the polarity is determined by quad_encoding_mode bits value
  - In contrast, dead-time because of kill does not the change the of line outputs (it holds the value of line out and line_compl_out before dead-time insertion started)

The following table summarizes which control signals determine the polarity of the PWM output signals:

Table 323 Control behavior of polarity of PWM output signal

	Counter not enable	Reset	Stopped	Killed	Dead-time insertion
PWM output signal	CTRL.QUADRATURE_ENCODING_MODE	CTRL.QUADRATURE_ENCODING_MODE	CTRL.KILL_LINE_POLARITY	CTRL.KILL_LINE_POLARITY	CTRL.QUADRATURE_ENCODING_MODE

Details of PWM “line” signal generation are been described in separate function mode sections below.

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 the counter is disabled.

33.2.7 Interrupts

The TCPWM block provides a dedicated interrupt output for each counter. Interrupts are counter mode-specific. Each counter has three interrupt causes:

- A cc0_match interrupt causes (INTR.CC0_MATCH). Hardware sets this interrupt cause to ‘1’ on a cc0_match internal event
- A cc1_match interrupt causes (INTR.CC1_MATCH). Hardware sets this interrupt cause to ‘1’ on a cc1_match internal event (present when GRP_CC1_PRESENT=1)
- A TC interrupt cause (INTR.TC). Hardware sets this interrupt cause to ‘1’ on a Terminal Count (TC) internal event. A TC is the logical OR of the overflow and underflow events

Four registers are used to handle interrupts in this block, as shown in the following table.

Table 324 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 the 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 the capture/compare1 (CC1) register value. Write '1' to clear this bit.

(table continues...)

Table 324 (continued) Interrupt register

Interrupt registers	Bits	Name	Description
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.

33.2.8 Debug mode

The TCPWM counters support debugging through CTRL.DBG_FREEZE_EN and CTRL.DBG_SUS_EN bits. 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 counter clock “clk_counter” when entering debug state while DBG_FREEZE_EN = ‘1’.

The table tabulates the behavior of the debug mode based on CTRL.DBG_FREEZE_EN and CTRL.DBG_SUS_EN bits.

Table 325 Debug mode without HRPWM

	DBG_FREEZE_EN	DBG_SUS_EN
Counter operation continues	0	0
Counter operation pauses immediately	1	0
Counter operation pauses after tc event and enter passive state. Passive state is controlled through CTRL.KILL_LINE_POLARITY	0	1
Counter operation pauses immediately and enters passive state. Passive state is controlled through CTRL.KILL_LINE_POLARITY	1	1

In a multicore environment ‘debug state’ means that at least one of the CPUs is in debug state. In cases where only 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 TCPWM has an input “tr_debug_freeze” which signalizes entering the debug state. It is connected via a PERI trigger multiplexer to the two cross-trigger interface outputs of the CPUSS. Depending on which CPU shall be debugged the debug state output trigger of that CPU can be selected as “tr_debug_freeze” input of the TCPWM. Since this signal is routed through the trigger multiplexers it needs to be synchronized (as all trigger signals are considered to be asynchronous due to potential long wires). For these counters which are controlled by the CPU to be debugged and which should be stopped when entering the debug mode DBG_FREEZE_EN needs to be set to ‘1’.

***Note**: The trigger input is not allowed to be to be assert when the counter is in debug state.*

***Note**: The freeze and suspend functions can be enabled only in the debug session is active; hence, check whether the debug session is active through the DEBUG_SESSION bit of the PWR_CTL register before enabling.*

33.2.9 Power modes

The TCPWM block works in ACTIVE, LPACTIVE, SLEEP, and LPSLEEP modes. The TCPWM block is connected to a digital core power supply. The retention registers will be still powered in DEEPSLEEP mode to keep state for the MMIO registers, which needs to retain the value.

In ACTIVE, LPACTIVE, SLEEP, and LPSLEEP power modes, clk_peri is provided. In DEEPSLEEP mode, there is no clk_peri running.

The vccact is cut off in HIBERNATE and DEEPSLEEP modes, but supplied in ACTIVE/LPACTIVE/SLEEP/LPSLEEP modes.

The vccret is supplied in DEEPSLEEP mode for retention registers and off in HIBERNATE mode.
There is no power switch integrated in the block.

Table 326 Power modes in TCPWM block

Power mode	Block status
ACTIVE	This block is fully operational in this mode with the clock running with power on.
LPACTIVE	LPACTIVE (LOWPOWERACTIVE) is similar to ACTIVE, with some tradeoffs applied to reduce current.
SLEEP	CPU is sleep in this mode but the function is still on. All counter clocks are ON.
LPSLEEP	LPSLEEP (LOWPOWERSLEEP) relates to LPACTIVE in the same way that SLEEP relates to ACTIVE.
DEEPSLEEP	In this mode, the vccact power to this block is off and no clk_peri is provided, hence, the logic is not functional. All retention registers will keep their state with power vccret.
HIBERNATE	In this mode, the entire block is powered off and no clk_peri is provided.

33.3 Counter function modes

Each counter supports multiple operation modes. At any time, a single counter is operating in one mode. Different counters can operate in different modes. The MODE [26:24] field of the counter control register (CTRL) configures the counter in the specific operational mode. The modes are described in the following table:

Table 327 Function 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.
PWM	100	Pulse width modulation (PWM) mode. HRPWM feature is supported in this mode through HRPWM_CTRL register.

(table continues...)

Table 327 (continued) Function mode configuration

Mode	MODE field [26:24]	Description
PWM_DT	101	Pulse width modulation with dead time insertion. HRPWM feature is supported in this mode through HRPWM_CTRL register.

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 the following table.

Table 328 Counting mode configuration

Counting modes	UP_DOWN_MODE[17:16]	Description
UP counting mode	00	Increments the counter until the period value is reached. A Terminal Count (TC) condition is generated when 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.

33.3.1 Timer mode

The timer mode can be used to measure how long an event takes or the time difference between two events. Timer functionality increments/decrements a counter between 0 and PERIOD. While counting, the counter value register COUNTER is compared with compare/capture register 0 (CC0). When the counter changes from a state in which COUNTER equals CC0, the cc0_match event is generated.

When parameter GRP_CC1_PRESENT = 1 and the counter is running, the counter value COUNTER is also compared with CC1. When COUNTER equals CC1, the cc1_match event is generated.

Timer functionality is typically used for one of the following:

- Timing a specific delay. In this case, the count event is a constant ‘1’
- Counting the occurrence of a specific event. In this case, the event should be connected as an input trigger and selected for the count event

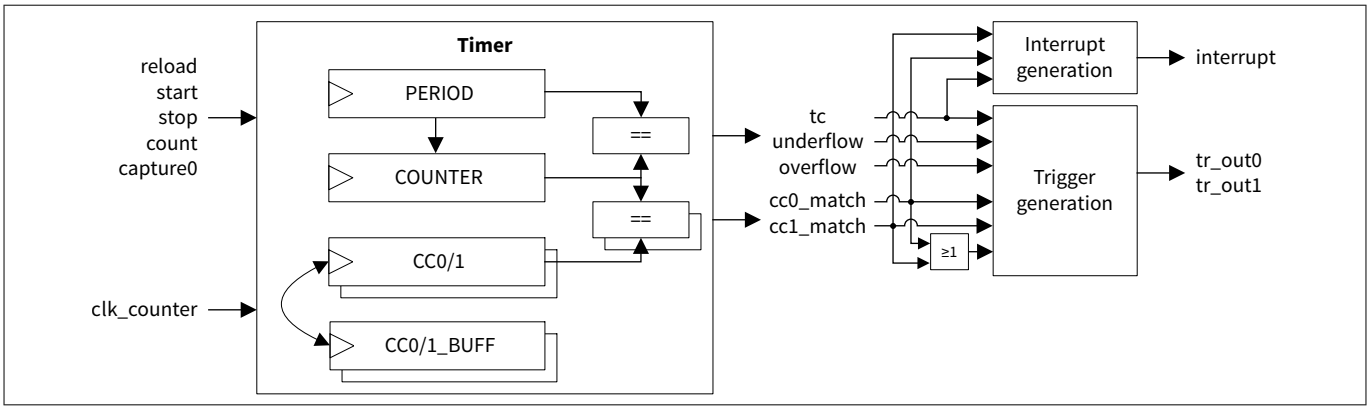


Figure 348 Timer mode function diagram

33.3.1.1 Overview

Table 329 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 the count direction is set to 'up' • COUNT_DOWN: The counter is set to PERIOD and the count direction is set to 'down' • COUNT_UPDN1/2: The counter is set to '1' and the 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 the 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 and can be used when the counter is running or not running.
Stop	Stops the counter.
Count	Count event increments/decrements the counter.

(table continues...)

Table 329 (continued) Timer mode trigger input description

Trigger inputs	Usage
Capture0	This event is used as an external up/down counting event. External direction control mode is enabled via QUAD_ENCODING_MODE[21:20] field of CTRL register: • EXT_DIR_FALLING: Counter changes current count direction based on falling edge of capture0 input • EXT_DIR_RISING: Counter changes current count direction based on rising edge of capture0 input • EXT_DIR_LEVEL: Counter count increases when capture0 level equal to high, counter count decrease when capture0 level equals too low
Capture1	Not used.

Incrementing and decrementing of the counter is controlled by the count event and the counter clock “clk_counter”. Typical operation will use a constant ‘1’ count event and “clk_counter” without pre-scaling. More advanced operations are possible. For example, the counter event configuration can decide to count the rising edges of a synchronized input trigger.

Table 330 Timer mode supported features

Supported features	Description
Clock pre-scaling	Pre-scales the counter clock “clk_counter”.
One shot	Counter is stopped by hardware on a tc event. In COUNT_UPDN2, the counter is stopped on tc event when underflow.
Auto reload CC	CC0 and CC0_BUFF are exchanged on a cc0_match event when CTRL.SWAP_ENABLE = “1” (when specified by CTRL.AUTO_RELOAD_CC, no input event is required). When parameter GRP_CC1_PRESENT = 1, CC1 and CC1_BUFF are exchanged on a cc1_match event when CTRL.SWAP_ENABLE = “1” (when specified by CTRL.AUTO_RELOAD_CC1, no input event is required). CC0_BUFF are is transferred to CC0 on a cc0_match event when CTRL.SWAP_ENABLE = “0” (when specified by CTRL.AUTO_RELOAD_CC0, no input event is required). When parameter GRP_CC1_PRESENT = 1, CC1_BUFF are is transferred to CC1 on a cc1_match event when CTRL.SWAP_ENABLE = “0” (when specified by CTRL.AUTO_RELOAD_CC1).

(table continues...)

Table 330 (continued) Timer mode supported features

Supported features	Description
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 331 Internal events of timer mode

Internal events	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. Only available when parameter GRP_CC1_PRESENT = 1.
Underflow (UN)	Counter is decremented and changes from a state in which COUNTER equals 0. Reload event generates 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 generates 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 generate tc.

Table 332 Line outputs of timer mode

PWM outputs	Description
LINE_OUT	Not used.
LINE_COMPL_OUT	Not used.

The triggers tr_out0 and tr_out1 are generated based on the internal events cc0_match, cc1_match, OR combination of cc0_match and cc1_match, underflow, overflow, and tc respectively (selection done by register TR_OUT_SEL).

***Note**: The timer functionality only uses the PERIOD register (and not PERIOD_BUFF). In the COUNT_UPDN2 mode, **software should only change PERIOD for a TC condition when COUNTER is "0"** (and not when COUNTER is PERIOD).*

33.3.1.2 Up/down count modes

Up/down count modes control the counting direction (increment or decrement) while the counter is running.

COUNT_UP mode

The following figure illustrates a timer in COUNT_UP mode. The counter is initialized (to 0) and started with a software-based reload event.

- 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 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

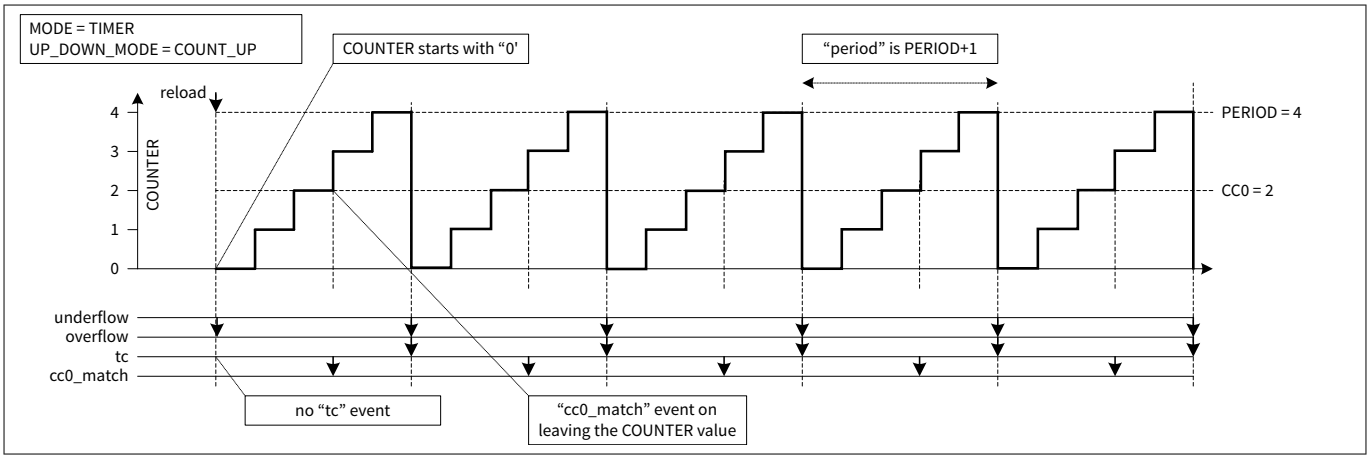


Figure 349 Timer in COUNT_UP mode

COUNT_DOWN mode

The following figure illustrates a timer in COUNT_DOWN mode. The counter is initialized (to PERIOD) and started with a software based reload event.

- 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 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

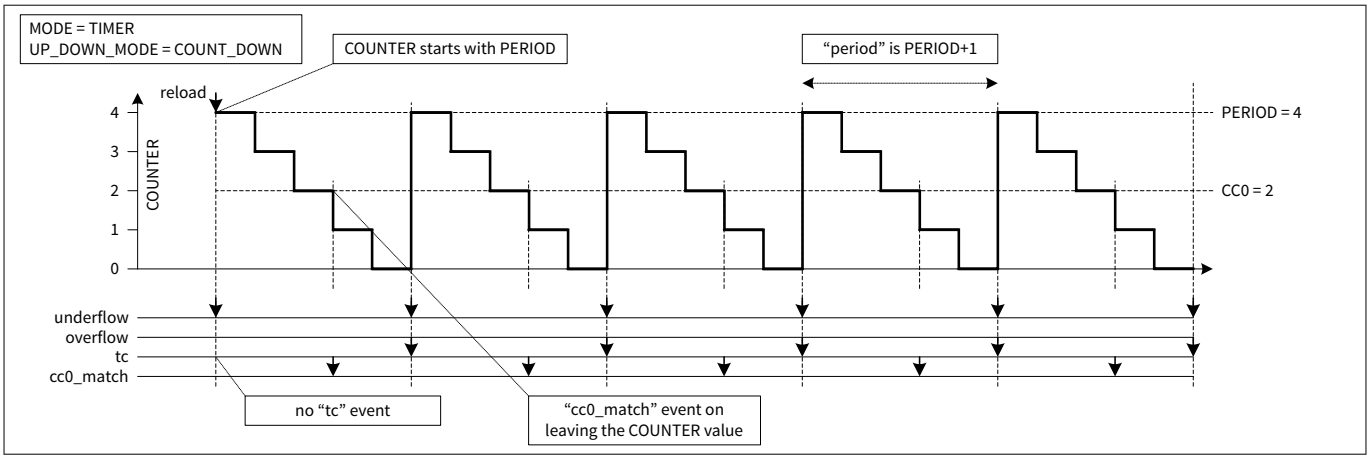


Figure 350 Timer in COUNT_DOWN mode

COUNT_UPDN1 mode

The following figure illustrates a timer in COUNT_UPDN1 mode. The counter is initialized (to 1) and started with a software-based reload event.

- 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 equals 2, a cc0_match event is generated
- PERIOD is 4, resulting in an effective repeating counter pattern of 2*4 = 8 counter clock periods

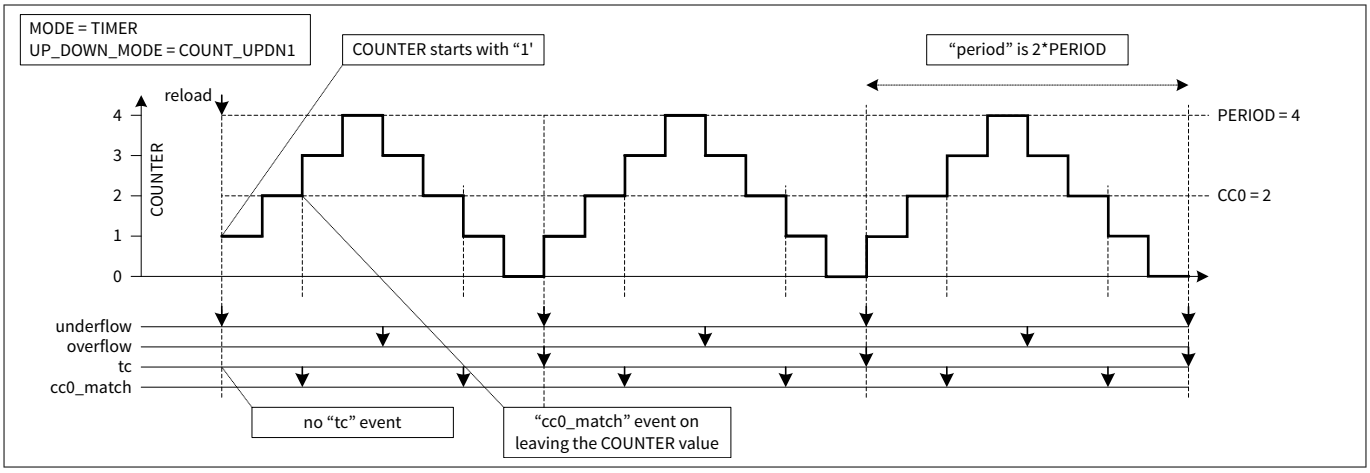


Figure 351 Timer in COUNT_UPDN1 mode

COUNT_UPDN2 has the same behavior with COUNT_UPDN1, except for the tc event which will generate with overflow event.

33.3.1.3 One shot mode

The following figure illustrates a timer in "one shot" operation mode.

Note:
- *The counter is stopped on a tc event*
- *In COUNT_UPDN2, counter is stopped on tc event when underflow*
- *COUNTER will stay at next value after tc. In following figures, counter stays at "0" after it stops running*

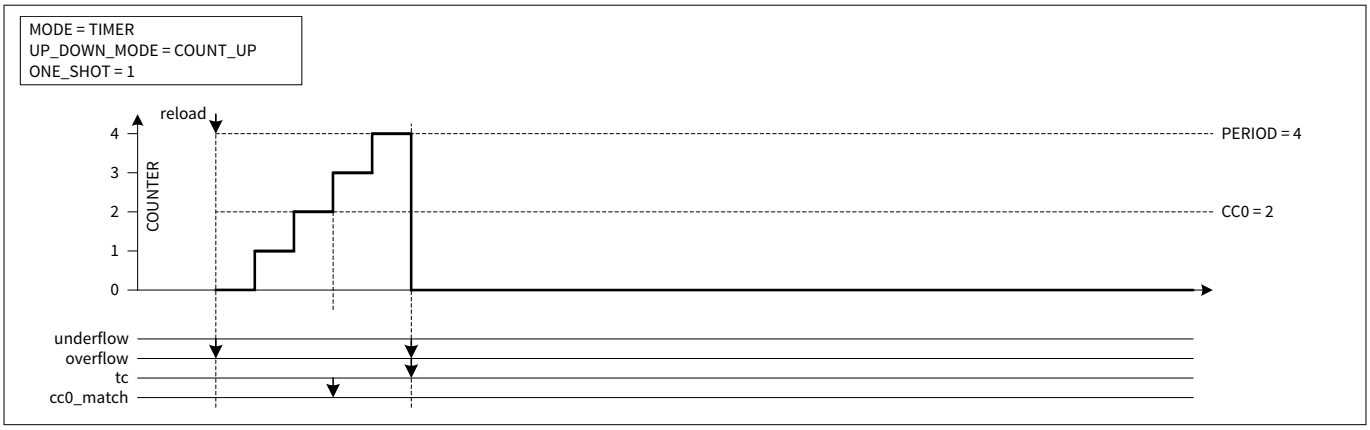


Figure 352 Timer in one-shot mode

33.3.1.4 Clock pre-scaling

The supported pre-scaler values are: “/1”, “/2”, “/4”, “/8”, “/16”, “/32”, “/64”, “/128”, The following figure illustrates clock pre-scaling equal “/2”.

***Note**: The counter is only incremented every other counter cycle.*

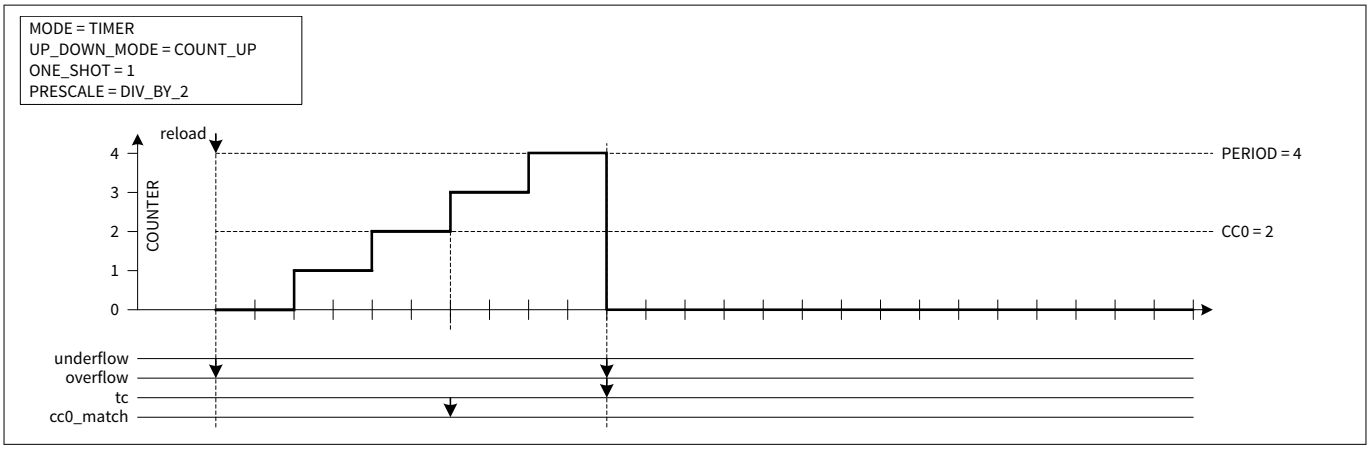


Figure 353 Timer clock pre-scaling

33.3.1.5 CC auto reload

Once the cc0_match event asserts, the CC0 and CC0_BUFF will be swapped by hardware if control bit CTRL.AUTO_RELOAD_CC0 = “1” and CTRL.SWAP_ENABLE = "1" (no input event is required to trigger the exchange). The following figure illustrates a timer that uses both CC0 and CC0_BUFF registers.

***Note**: CC0 and CC0_BUFF are exchanged on a cc0_match event.*

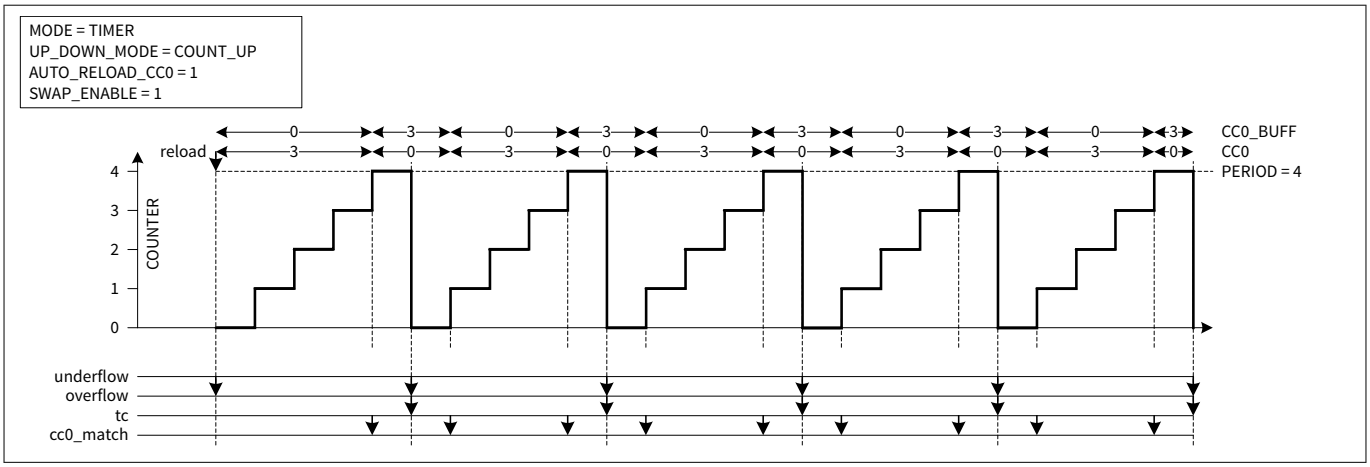


Figure 354 CC0/CC0_BUFF auto switch at cc0_match when SWAP_ENABLE = “1”

If CTRL.SWAP_ENABLE = “0”, CC0_BUFF value will be transferred to CC0 by hardware once the cc0_match event is asserted (no input event is required to trigger the exchange). The following figure illustrates a timer that uses both CC0 and CC0_BUFF registers.

***Note**: CC0_BUFF are is transferred to CC0 on a cc0_match event.*

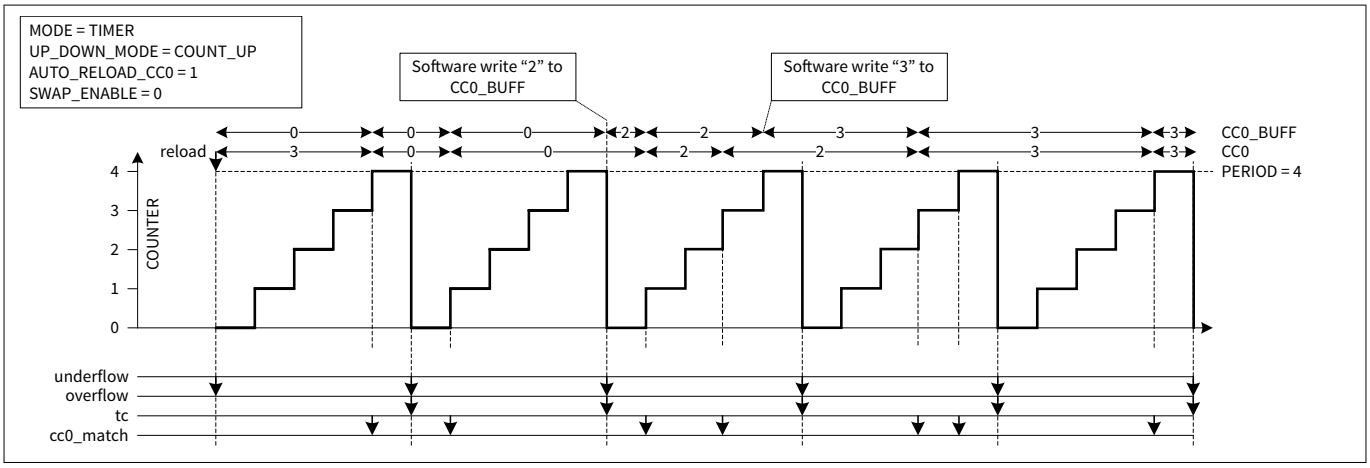


Figure 355 CC0_BUFF auto switch to CC0 at cc0_match with SWAP_ENABLE = “0”

The above description is applicable for CC1/CC1_BUFF when parameter GRP_CC1_PRESENT = “1”.

33.3.1.6 External input event

Timer mode supports reload/start/stop/count input event.

The following figure illustrates a counter that is initialized and started (reload event), stopped (stop event) and continued/started (start event).

***Note**: The counter does not change value when it is NOT running (STATUS.RUNNING).*

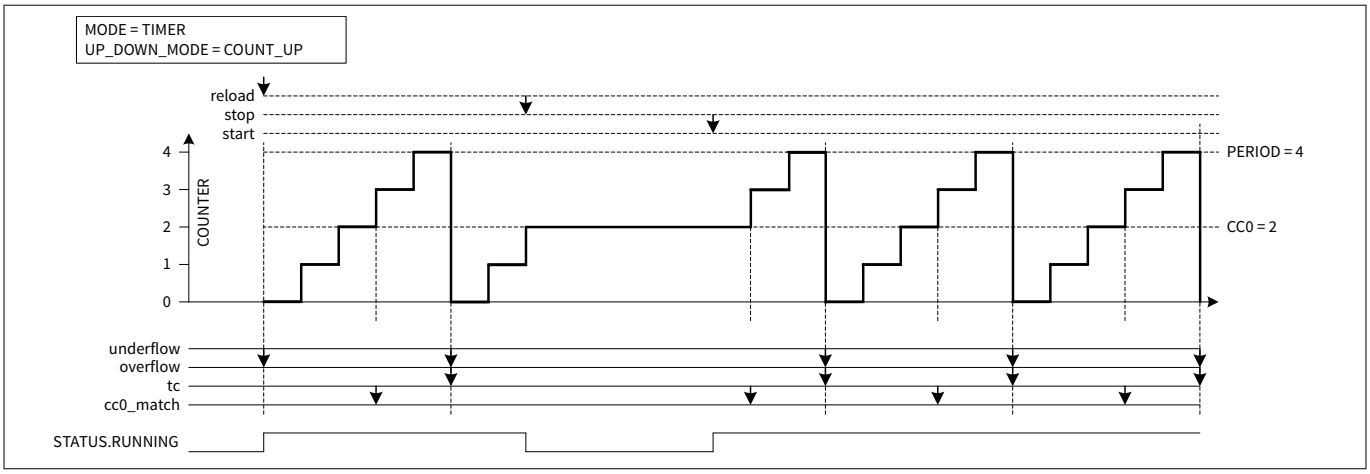


Figure 356 Counter start/stopped/continued

Reload can input when counter is running. The following description will clarify the behavior when reload coincides with tc event, cc0_match event or stop event.

- A stop event takes precedence over reload event
- Reload event generation takes precedence over “running counter” functionality (for underflow, overflow and tc event generation)
- Reload event determines:
- - underflow event generation
  - overflow event generation
  - tc event generation (always ‘0’)
- “Running counter” functionality determines:
  - underflow event generation
  - overflow event generation
  - tc event generation
  - cc0_match event generation
  - CC0/CC0_BUFF exchange

The following figure illustrates a count event that becomes low while counter is running, and a rising edge detection stop event asserts when count event is low (the case is count event is used as pass through mode). The COUNTER will stop run and hold value when count event is low. The stop event will not change running status to low until the count event becomes high (“active count” pre-scaled counter clock).

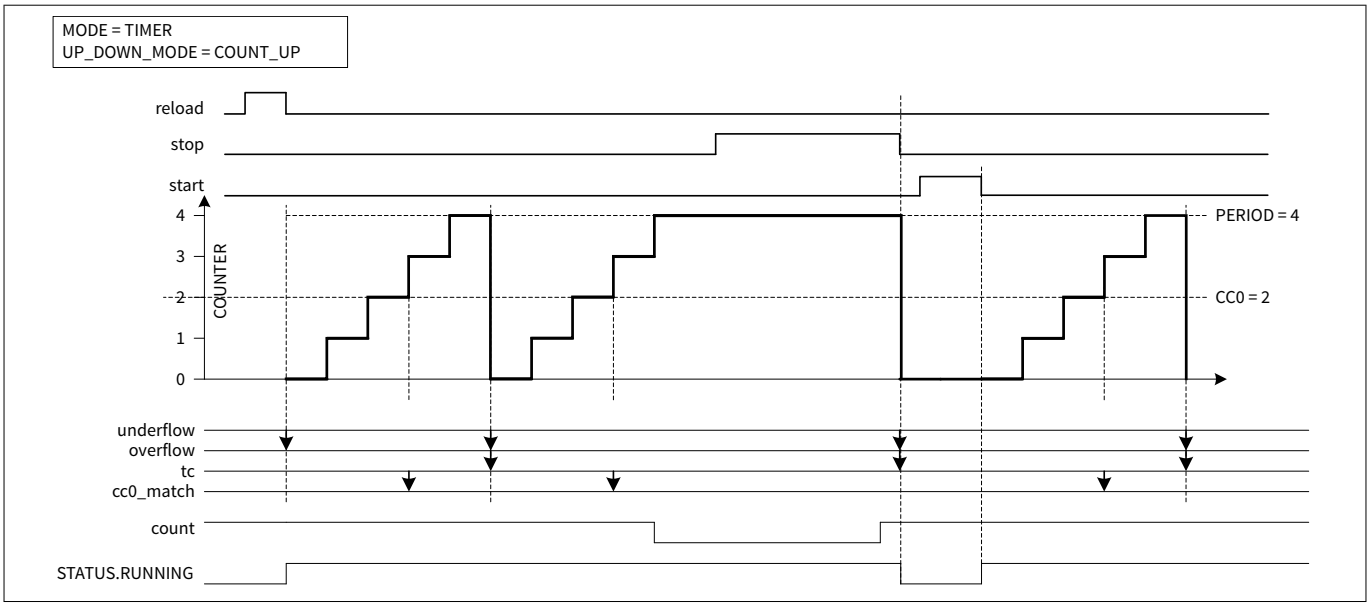


Figure 357 Function of count event

33.3.1.7 External direction control

Timer mode supports external direction control feature through the CTRL.QUAD_ENCODING_MODE bits. Timer mode uses the capture0 events to determine the up or down counting event. When CTRL.QUAD_ENDCODING_MODE bits is set to “0”, this feature is disable.

User is able to change the current count direction based on the falling/rising/level of the capture0 input. The following diagrams show the example how the counter changes current count direction based on falling edge/ rising edge/signal level in timer mode.

***Note**:*

- *User need to set TR_IN_EDGE_SEL.CAPTURE0_EDGE to “3” (NO_EDGE_DET) to use this feature*
- *This feature shall be used for COUNT_UP/COUNT_DOWN only in CTRL.UP_DOWN_MODE, unexpected behavior may be observed if COUNT_UPDN1/COUNT_UPDN2 is being set in CTRL.UP_DOWN_MODE*

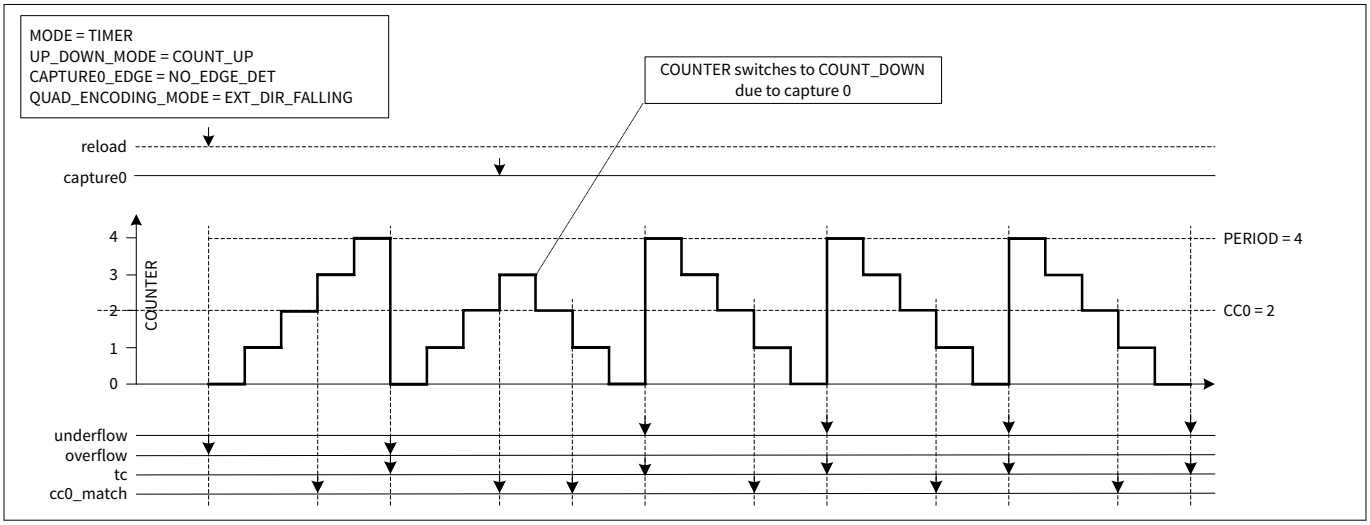


Figure 358 External direction control based on EXT_DIR_FALLING setting

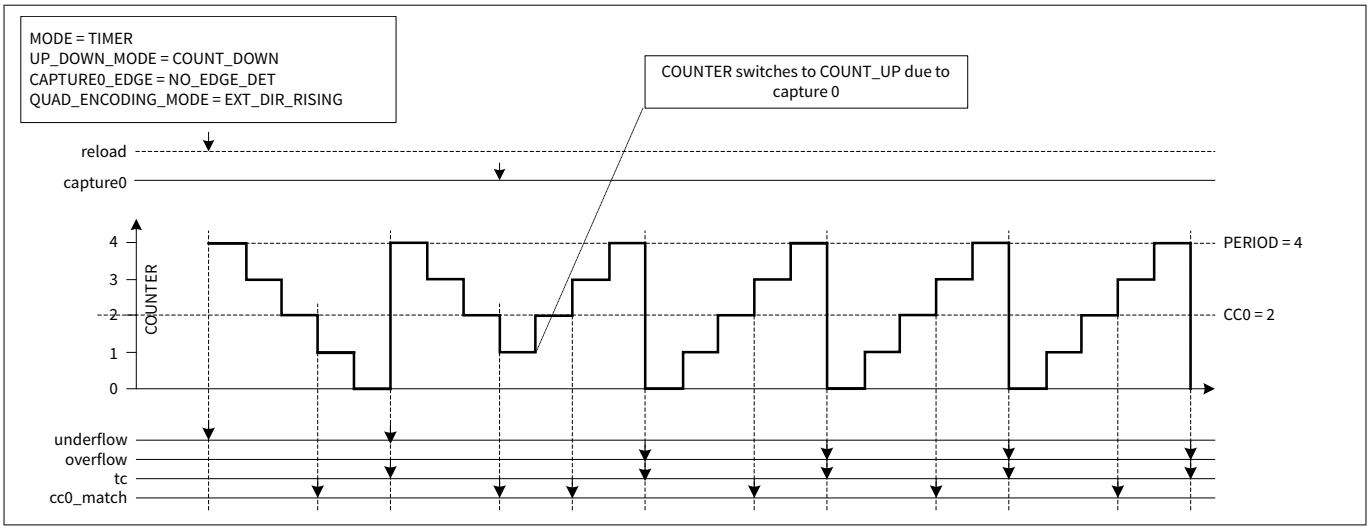


Figure 359 External direction control based on EXT_DIR_RISING setting

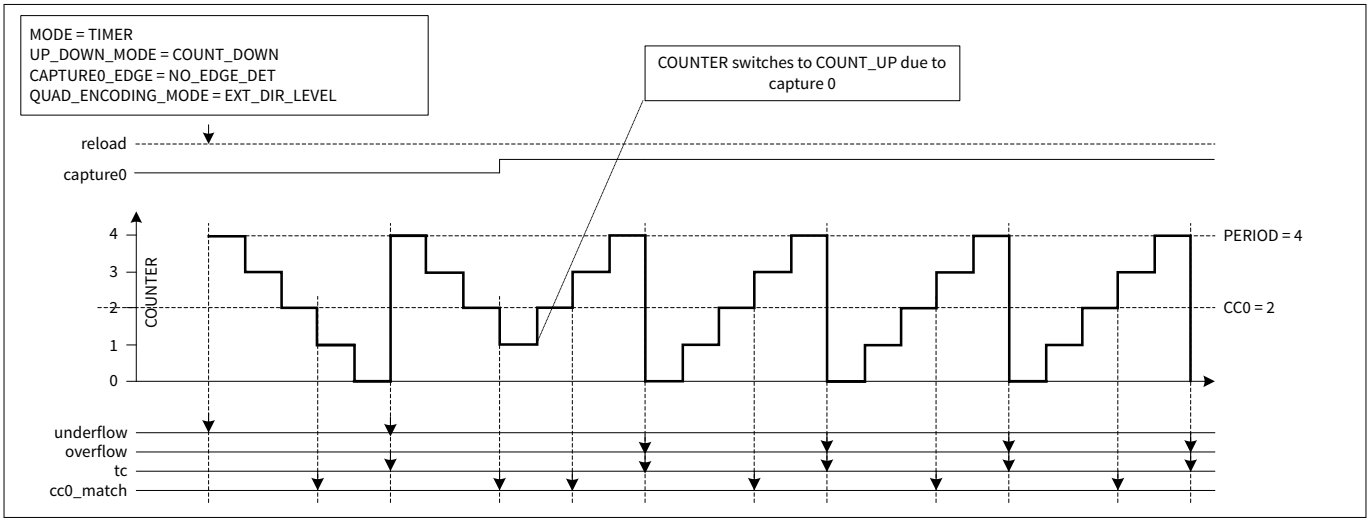


Figure 360 External direction control based on EXT_DIR_LEVEL setting

In addition, software capture trigger through TR_CMD.CAPTURE0 can be used trigger direction change in the counter but CTRL.QUAD_ENCODING_MODE shall be set to EXT_DIR_RISING only in Timer mode. hardware will clear TR_CMD.CAPTURE0 bit to 0 when direction change has taken effect.

33.3.1.8 Event generation on defined threshold

The timer mode can also be used to count the number of events using external count event. After reaching the defined number of events, the cc compare match indicates it using tr_out. The threshold value can be defined by cc match registers.

33.3.1.9 How to configure counter for timer mode

The steps to configure the counter for timer 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 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 the AUTO_RELOAD_CC0 field of the CTRL register, if required to switch values at every CC condition
- Set clock pre-scaling by writing to the PS_DIV [2:0] field of the PS 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, and Count)
- Set the TR_IN_EDGE_SEL register to select the edge of the trigger that causes the event (Reload, Start, Stop, Capture0, and Count)
- If required, set the interrupt upon TC or CC0_MATCH or CC1_MATCH condition
- Enable the counter by writing '1' to the ENABLED bit of the CTRL register. A reload trigger must be provided through the firmware(RELOAD bit in the PERI0_TR_CMD register) to start the counter if the hardware reload signal is not enabled

33.3.2 Capture mode

Capture functionality increments/decrements a counter between 0 and PERIOD. While counting, the capture0 event is observed. When the capture0 event is activated the counter value COUNTER is copied to CC0 (and CC0 is copied to CC0_BUFF). STATUS.CC0_READ_MISS bit will indicate whether data in the CC0 and CC0_BUFF are being been being overwritten before read by software. The STATUS.CC0_READ_MISS will be set to '1', when past capture event has been overwritten by new data from COUNTER before software read CC0 and CC0_BUFF. STATUS.CCO_READ_MISS will be clear to '0' once there is a read operation on either the CC0 or CC0_BUFF.

The capture1 event and STATUS.CC1_READ_MISS bit are present when parameter GRP_CC1_PRESENT = 1. If so, when the capture1 event is activated the counter value COUNTER is copied to CC1 (and CC1 is copied to CC1_BUFF).

Capture functionality can be used to measure the width of a pulse (connected as one of the input triggers and used as capture0 or capture1 event).

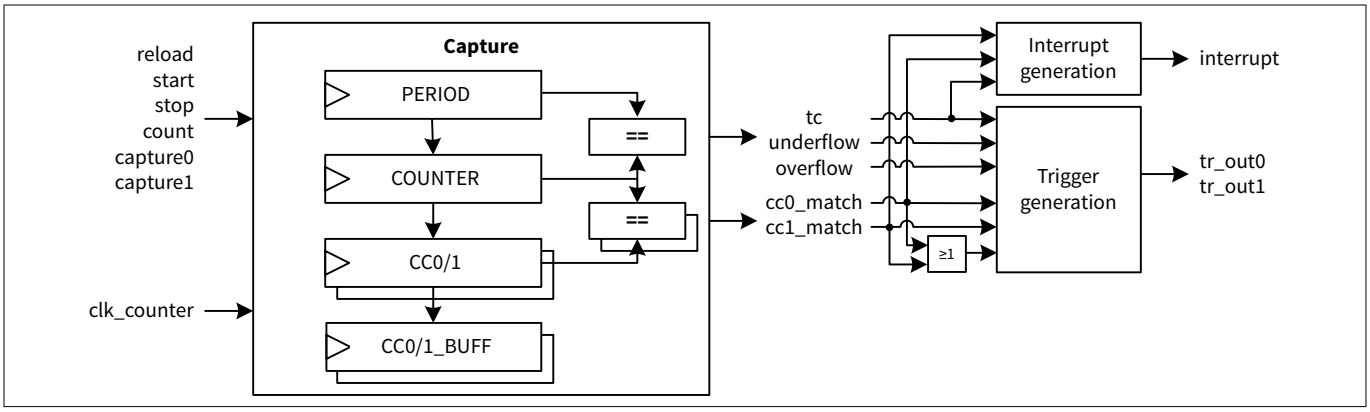


Figure 361 Capture mode function diagram

33.3.2.1 Overview

Table 333 Capture mode trigger input description

Generated events	Usage
Reload	Initializes and starts the counter. **Can be used while the counter is running or not running.** Initialization is dependent on UP_DOWN_MODE: • COUNT_UP: The counter is initialized to “0” • COUNT_DOWN: counter is initialized to PERIOD COUNT_UPDN1/2: The counter is initialized to “1”
Start	Starts the counter (the counter is NOT initialized by hardware). Can be used when the counter is running or not running. When the counter is running, the start event has no effect.
Stop	Stops the counter.
Count	Count event increments/decrements the counter per settings of up/down modes.
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. Only available when parameter GRP_CC1_PRESENT = 1.

Table 334 Supported features of capture

Supported features	Description
Clock pre-scaling	Pre-scales the counter clock “clk_counter”.
One shot	Counter is stopped by HW, on a tc event. In COUNT_UPDN2, the counter is stopped on tc event when underflow.
Up/down modes	Specified by UP_DOWN_MODE: • COUNT_UP: counter counts from 0 to PERIOD • COUNT_DOWN: counter counts from PERIOD to 0 COUNT_UPDN1/2: counter counts from 1 to PERIOD and back to 0.

Table 335 Internal events of capture

Internal events	Description
cc0_match	CC0 is copied to CC0_BUFF and the counter value is copied to CC0 (cc0_match equals capture0 event).

(table continues...)

Table 335 (continued) Internal events of capture

Internal events	Description
cc1_match	CC1 is copied to CC1_BUFF and the counter value is copied to CC1 (cc1_match equals capture1 event). Only available when parameter GRP_CC1_PRESENT = 1.
Underflow	Counter is decremented and changes from a state in which COUNTER equals “0”. Reload event generates underflow in COUNT_DOWN, COUNT_UPDN1 or COUNT_UPDN2 mode.
Overflow	Counter is incrementing and changes from a state in which COUNTER equals PERIOD. Reload event generates 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 generate tc.

Table 336 Line outputs of capture mode

PWM outputs	Description
line_out	Not used.
line_compl_out	Not used.

33.3.2.2 Capture COUNTER value into CC0/CC0_BUFF

When the capture0 event is detected, the COUNTER value will be copied into the CC0 register, meanwhile the value of the CC0 register will be copied into the CC0_BUFF register. This feature can be used to measure the width of a pulse (connected as one of the input triggers and used as capture0 event with rising/falling edge detection mode). The following Figure illustrates capture behavior in the COUNT_UP mode.

***Note**:*
- *Capture0 event detection uses rising edge detection. As a result, the capture0 event is remembered till the next “active count” pre-scaled counter clock*
- *When a capture0 event occurs, COUNTER is copied into CC0. CC0 is copied to CC0_BUFF*
- *A cc0_match event is generated when the counter value is captured*
- *CC0_READ_MISS will be set to ‘1’ if software never performs a read before the third capture event. It will be cleared to ‘0’ once software performs a read on either CC0 or CC0_BUFF*

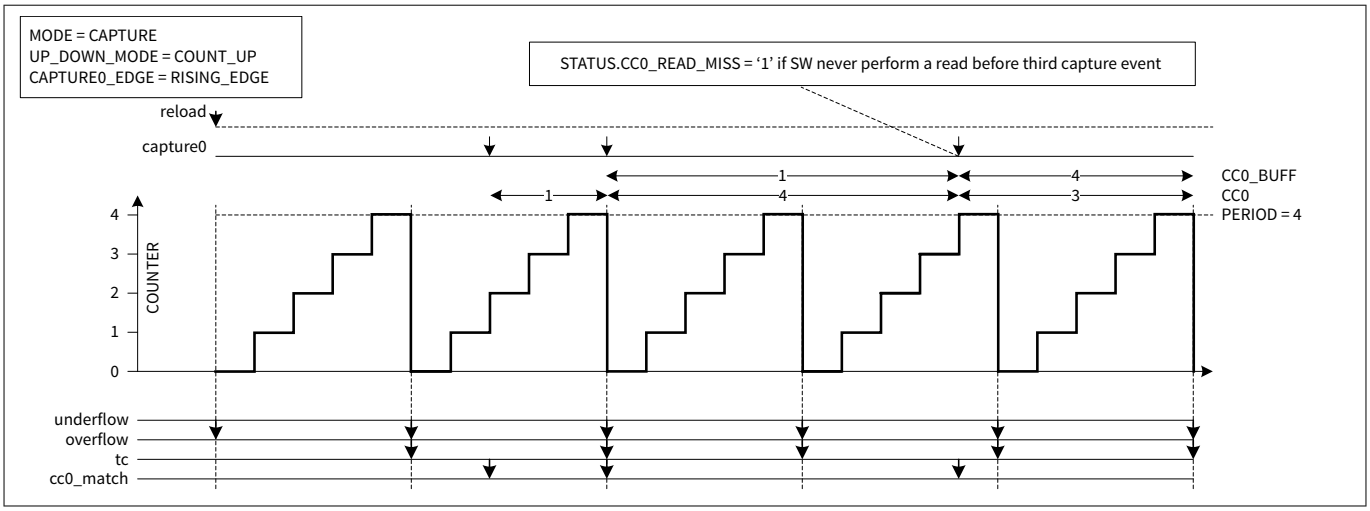


Figure 362 CC0/CC0_BUFF updated by capture0 event

When multiple capture events are detected before the next "active count" pre-scaled counter clock, capture events are treated as follows:

- In the rising edge or falling edge detection mode, multiple events are effectively reduced to a single event
- In the rising/falling edge detection mode, an even number of events is not detected and an odd number of events are reduced to a single event

The above behavior is illustrated by the following figures, in which a pre-scaler by a factor of 4 is used. The first figure uses rising edge detection. The second figure uses rising/falling edge detection.

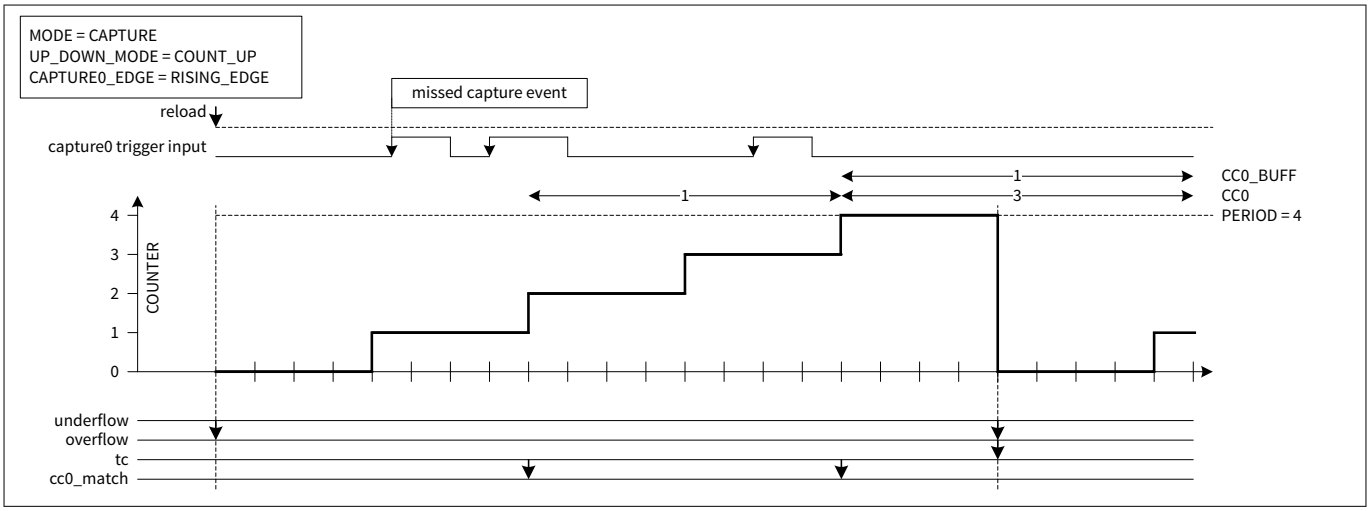


Figure 363 Multiple rising edge capture

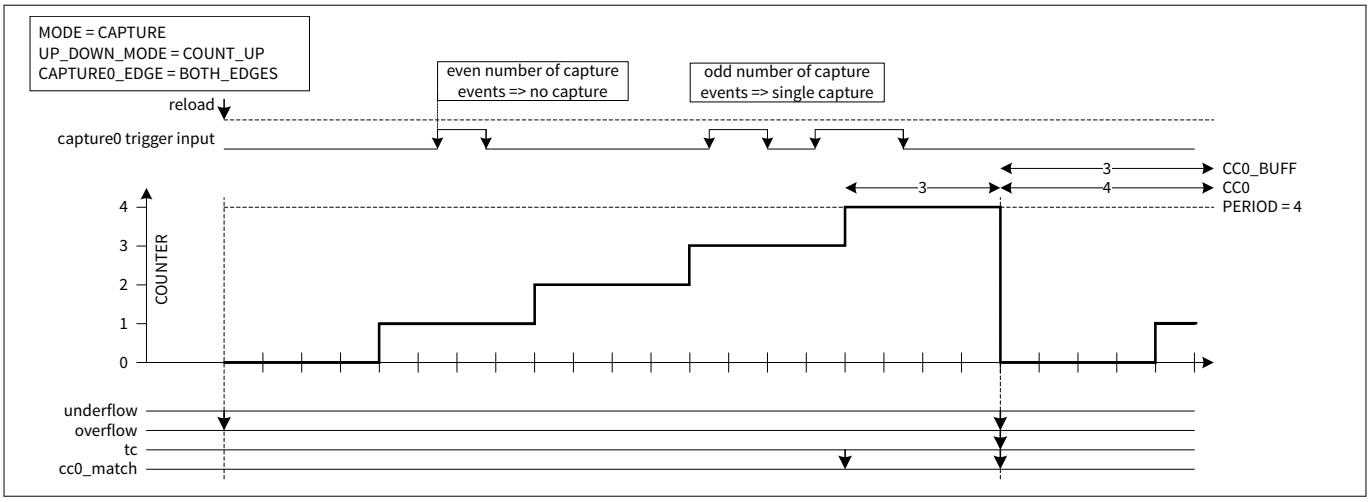


Figure 364 Multiple rising/falling both edge capture

When parameter GRP_CC1_PRESENT = 1, the capture1 function using registers CC1 and CC1_BUFF is the same as the capture0 function using registers CC0 and CC0_BUFF.

33.3.2.3 Up/down count modes

It is same as Up/down count modes.

33.3.2.4 One shot mode

It is same as One shot mode.

33.3.2.5 Clock pre-scaling

It is same as Clock pre-scaling.

33.3.2.6 How to configure counter for capture mode

The steps to configure the counter for Capture mode operation and the affected register bits are as follows:

- Disable the counter by writing '0' to the ENABLE bit of the CTRL register
- Select Capture mode by writing '010' to the MODE[26:24] field of the CTRL register
- Set the required 16-bit or 32-bit period in the PERIOD register
- Set clock pre-scaling by writing to the PS_DIV [2:0] field of the PS register
- Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of the CTRL register
- The 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
- 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 that causes the event (Reload, Start, Stop, Capture0/1, and Count)
- If required, set the interrupt upon TC or CC0_MATCH or CC1_MATCH condition
- Enable the counter by writing '1' to the ENABLED bit in the CTRL register. A reload trigger must be provided through firmware(RELOAD bit in PERI0_TR_CMD register) to start the counter if the hardware reload signal is not enabled

33.3.3 Pulse-width modulation (PWM) mode

PWM functionality increments/decrements a counter between 0 and PERIOD. When the counter is running, the counter value is compared with CC0. When COUNTER equals CC0, the cc0_match event is generated.

When parameter GRP_CC1_PRESENT = 1 and the counter is running, the counter value COUNTER is also compared with CC1. When COUNTER equals CC1, the 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.

When parameter GRP_CC1_PRESENT = 1, while the cc0_match event is used for PWM generation, the cc1_match event can be used for example to trigger an ADC conversion at an arbitrary time within the PWM period for PWM diagnosis. However, the cc1_match event can also be used for PWM generation. For example, asymmetric PWM signals can be generated using combined cc0_match and cc1_match in COUNT_UPDN1 mode.

When parameter GRP_AMC_PRESENT = 1, the PWM mode can be used for stepper motor control including micro stepping.

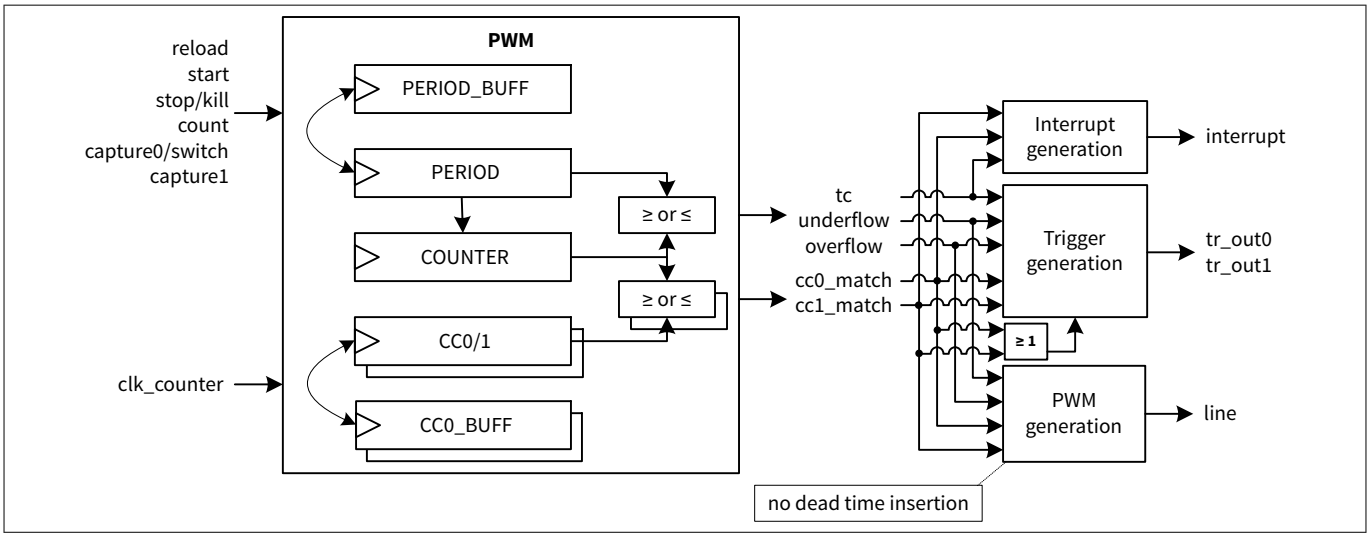


Figure 365 PWM mode function diagram

33.3.3.1 Overview

Table 337 PWM mode trigger input description

Generated events	Usage
reload	Initializes and starts the counter. **Should only be used when the counter is not running.** Initialization is dependent on UP_DOWN_MODE: • COUNT_UP: The counter is initialized to “0” • COUNT_DOWN: The counter is initialized to PERIOD • COUNT_UPDN1/2: The counter is initialized to “1”

(table continues...)

Table 337 (continued) PWM mode trigger input description

Generated events	Usage
start	Starts the counter (the counter is NOT initialized by hardware). Can be used when the counter is running or not running. When the counter is running, the start event has no effect.
Stop	Stops the counter. Different stop/kill modes exist.
Count	Count event increments/decrements the counter per settings of up/down modes.
Capture0	This event acts as a switch event. When this event is active, the CC0/CC0_BUFF, CC1/CC1_BUFF (when parameter GRP_CC1_PRESENT = 1), PERIOD/ PERIOD_BUFF and LINE_SEL/LINE_SEL_BUFF (when parameter GRP_SMC_PRESENT = 1) registers are exchanged on a tc event (when specified by CTRL.AUTO_RELOAD_CC0/1, CTRL.AUTO_RELOAD_PERIOD and CTRL.AUTO_RELOAD_LINE_SEL) when CTRL.SWAP_ENABLE = “1”. This event acts as a switch event. When this event is active, CC0_BUFF is transferred to CC0, CC1_BUFF is transferred to CC1 (when parameter GRP_CC1_PRESENT = 1), PERIOD_BUFF and LINE_SEL_BUFF registers are transferred to PERIOD and LINE_SEL on a tc event (when specified by CTRL.AUTO_RELOAD_CC0/1, CTRL.AUTO_RELOAD_PERIOD and CTRL.AUTO_RELOAD_LINE_SEL) when CTRL.SWAP_ENABLE = “0”. **A switch event requires rising, falling, or rising/ falling edge mode. Pass through mode is not supported, unless the selected event is a constant ‘0’ or ‘1’.** When a switch event is detected, it is kept pending till the next tc event. When a switch event is detected and the counter is not running, the event is cleared by hardware.

(table continues...)

Table 337 (continued) PWM mode trigger input description

Generated events	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. Only available when parameter GRP_CC1_PRESENT = 1. ***Note**: Having 2 stop/kill events for a PWM allows selecting for one stop/kill event a common trigger from a PERI trigger multiplexer (allowing synchronous stop/kill operation of multiple PWMs) while selecting for the other stop/kill event, for example, a dedicated ADC out-of-range trigger (for example, allowing real-time hardware stop of a PWM when current of PWM driven signal is out of range).*

Table 338 PWM mode supported features

Supported features	Description
Clock pre-scaling	Pre-scales the counter clock “clk_counter”.
One shot	Counter is stopped by HW, on a tc event. In COUNT_UPDN2, the counter is stopped on tc event when underflow.
Auto reload CC	CC0 and CC0_BUFF are exchanged on a switch event and tc event (when specified by CTRL.AUTO_RELOAD_CC0). When parameter GRP_CC1_PRESENT = 1, CC1 and CC1_BUFF are exchanged on a switch event and tc event (when specified by CTRL.AUTO_RELOAD_CC1).
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.*

(table continues...)

Table 338 (continued) PWM mode supported features

Supported features	Description
Auto reload LINE_SEL[1)]	LINE_SEL and LINE_SEL_BUFF are exchanged on a switch event and tc event (when specified by CTRL.AUTO_RELOAD_LINE_SEL) when CTRL.SWAP_ENABLE = “1”. LINE_SEL_BUFF is transferred to LINE_SEL on a switch event and tc event (when specified by CTRL.AUTO_RELOAD_LINE_SEL) when CTRL.SWAP_ENABLE = “0”.
Up/down modes	Specified by UP_DOWN_MODE: • COUNT_UP: counter counts from 0 to PERIOD • COUNT_DOWN: counter counts from PERIOD to 0 • COUNT_UPDN1/2: counter counts from 1 to PERIOD and back to 0
Kill modes	Specified by PWM_IMM_KILL, PWM_SYNC_KILL, PWM_STOP_ON_KILL, PWM_TC_SYN_KILL_DT and PWM_SYN_KILL_DT. See the sections below for further details.

1) When parameter GRP_SMC_PRESENT = 1.

***Note**: The PWM mode does not support dead time insertion. This functionality is supported by the separate PWM_DT mode.*

Table 339 Internal events of PWM mode

Internal events	Description
cc0_match	Specified by UP_DOWN_MODE: • COUNT_UP and COUNT_DOWN: counter changes to a state in which COUNTER equals CC0 • COUNT_UPDN1/2: counter changes from a state in which COUNTER equals CC0. If parameter GRP_AMC_PRESENT=1,CC0_MATCH_UP_EN/ CC0_MATCH_DOWN_EN will enable or disable cc0_match generation when count is in up/down counting CC0=0 and CC0=PERIOD will be the special case for 0/100% duty cycle PWM generation.

(table continues...)

Table 339 (continued) Internal events of PWM mode

Internal events	Description
cc1_match	Specified by UP_DOWN_MODE: • COUNT_UP and COUNT_DOWN: counter changes to a state in which COUNTER equals CC1 • COUNT_UPDN1/2: counter changes from a state in which COUNTER equals CC1. If parameter GRP_AMC_PRESENT=1, CC1_MATCH_UP_EN/ CC1_MATCH_DOWN_EN will enable or disable cc1_match generation when count is in up/down counting Only available when parameter GRP_CC1_PRESENT = 1.
Underflow	Counter is decremented and changes from a state in which COUNTER equals “0”. Reload event generates underflow in COUNT_DOWN, COUNT_UPDN1 or COUNT_UPDN2 mode.
Overflow	Counter is incrementing and changes from a state in which COUNTER equals PERIOD. Reload event generates 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 generate tc.

***Note**: The cc0_match event generating in COUNT_UP and COUNT_DOWN modes are different from the generation in other timer/capture modes. This is to ensure that 0% and 100% duty cycles can be generated.*

Table 340 Line outputs of PWM mode

PWM output	Description
line_out	PWM line output
line_compl_out	Complementary PWM line output.

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 when CTRL.SWAP_ENABLE = “1”. When CTRL.SWAP_ENABLE = “0”, the switch/capture event can be used to transfer the values of the buffered compare register to compare register. It also transfers the values of the buffered period register to the period register.

Following table summarizes the kill mode supported in PWM mode. Any settings that are not defined in the following table are not supported or not recommended.

Table 341 Kill modes of PWM

Kill mode	Settings	Kill behavior
No-IMM-Async	PWM_IMM_KILL = 0 PWM_SYNC_KILL = 0 PWM_TC_SYNC_KILL_DT = 0 PWM_SYNC_KILL_DT = 0 PWM_STOP_ON_KILL = 0 STOP_EDGE = NO_EDGE_DET	• PWM output gets suppressed at the next active count clock after kill input is active • PWM output suppress is removed at the next active count clock after kill input is inactive
IMM-Async (Asynchronous kill mode)	PWM_IMM_KILL = 1 PWM_SYNC_KILL = 0 PWM_TC_SYNC_KILL_DT = 0 PWM_SYNC_KILL_DT = 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 the next active count clock after kill input is inactive This mode keeps the counter running, only external input trigger can be selected as asynchronous kill.
No-IMM-Sync	PWM_IMM_KILL = 0 PWM_SYNC_KILL = 1 PWM_TC_SYNC_KILL_DT = 0 PWM_SYNC_KILL_DT = 0 PWM_STOP_ON_KILL = 0 STOP_EDGE = RISING	• PWM output gets is suppressed at the next active count clock after kill input is active • PWM output suppress is removed at the next tc event after kill input is inactive
IMM-Sync (Synchronous kill mode)	PWM_IMM_KILL = 1 PWM_SYNC_KILL = 1 PWM_TC_SYNC_KILL_DT = 0 PWM_SYNC_KILL_DT = 0 PWM_STOP_ON_KILL = 0 STOP_EDGE = RISING	• PWM output is suppressed immediately after kill input is active • PWM output suppress is removed at the next tc event after kill input is inactive This mode keeps the counter running, only external input trigger can be selected as synchronous kill.

(table continues...)

Table 341 (continued) Kill modes of PWM

Kill mode	Settings	Kill behavior
TC-Sync-DT (TC Synchronous DT kill mode)	PWM_IMM_KILL = 0 PWM_SYNC_KILL = 0 PWM_TC_SYNC_KILL_DT = 1 PWM_SYNC_KILL_DT = 0 PWM_STOP_ON_KILL = 0 STOP_EDGE = RISING	• PWM output gets suppressed at the next tc event after kill input is active. If deadtime is enabled, output gets suppressed after the tc event + deadtime • PWM output suppress is removed at the next tc event after kill input is inactive This mode keeps the counter running, only external input trigger can be selected as synchronous kill. This is only supported in PWM_DT mode.
IMM-Sync-DT (Synchronous DT kill mode)	PWM_IMM_KILL = 1 PWM_SYNC_KILL = 0 PWM_TC_SYNC_KILL_DT = 0 PWM_SYNC_KILL_DT = 1 PWM_STOP_ON_KILL = 0 STOP_EDGE = RISING	• PWM output line_out get suppressed immediately whereas line_compl_out get suppressed after dead time when kill input is active • PWM output suppress is removed at the next tc event after kill input is inactive This mode keeps the counter running, only external input trigger can be selected as synchronous kill. This is only supported in PWM_DT mode
No-IMM-Stop	PWM_IMM_KILL = 0 PWM_SYNC_KILL = Don’t care PWM_TC_SYNC_KILL_DT= Don’t care PWM_SYNC_KILL_DT = Don’t care PWM_STOP_ON_KILL = 1 STOP_EDGE = RISING_EDGE/ FALLING_EDGE/BOTH_EDGES	• PWM output get suppressed at the next active count clock after kill input is active • PWM output suppress is removed when counter restart after kill input is inactive

(table continues...)

Table 341 (continued) Kill modes of PWM

Kill mode	Settings	Kill behavior
IMM-Stop (Kill stop mode)	PWM_IMM_KILL = 1 PWM_SYNC_KILL = Don't care PWM_TC_SYNC_KILL_DT = Don't care PWM_SYNC_KILL_DT = 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 when counter restart after kill input is inactive This mode stops the counter on a stop/kill event. Reload or start event is required to start count running. Both software and external input trigger can be selected as stop kill.

Immediate kill mode (PWM_IMM_KILL = 1) is to suppress PWM output with maximum 3 clk_peri cycles delay after kill input is active.

***Note**: STATUS.KILL_STATUS shall be set immediately after kill input is active and shall be cleared after suppression is released. The suppression of the PWM output depends on the kill modes defined in the above table.*

***Note**: Whether the line_out waits for active_clock depends on whether the active_clock happens during event_stop. If active_clock happens when event_stop is high, then the line_out suppression is still next active_clock but if it does not then there is no waiting.*

33.3.3.2 Up/down count modes

Up/down count modes control the counting direction (increment or decrement) while the counter is running.

COUNT_UP mode

The following figure illustrates a PWM in COUNT_UP mode. The counter is initialized (to 0) and started with a software-based reload event.

- 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

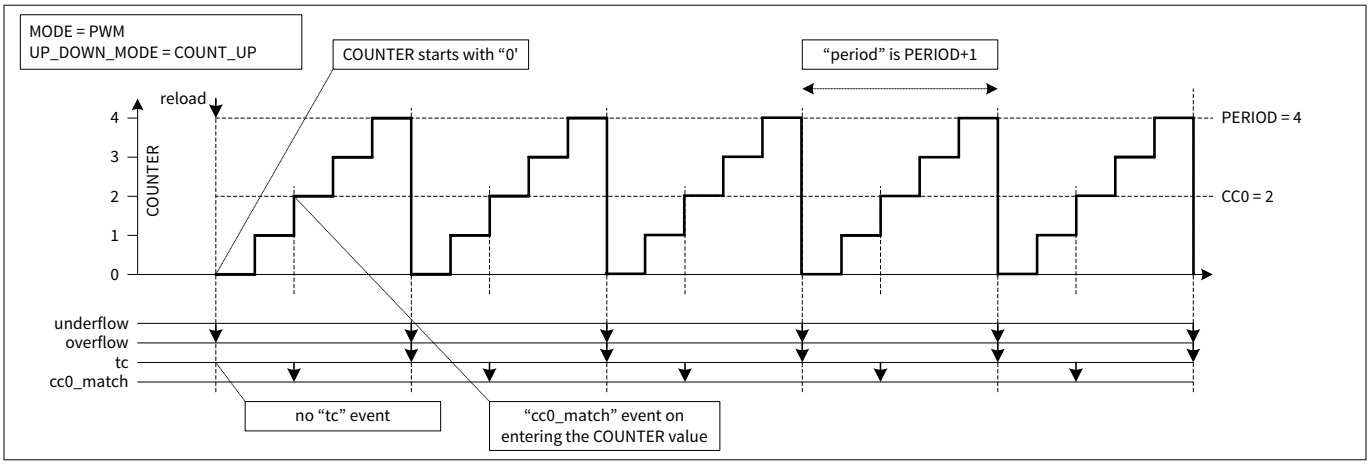


Figure 366 PWM in COUNT_UP mode

COUNT_DOWN mode

The following figure illustrates a PWM in COUNT_DOWN mode. The counter is initialized (to PERIOD) and started with a software-based reload event.

- 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 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

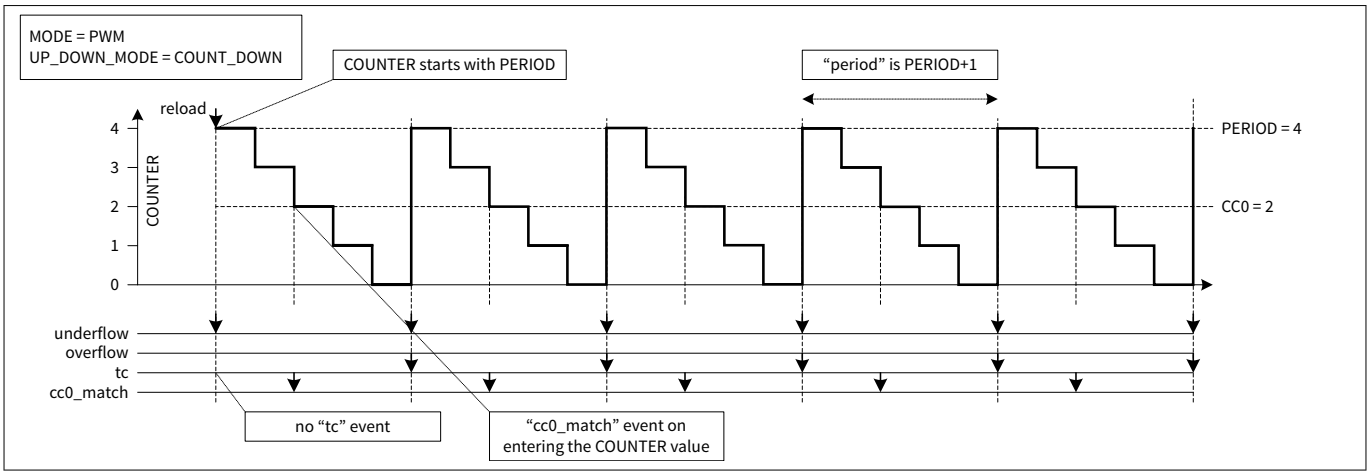


Figure 367 PWM in COUNT_DOWN mode

COUNT_UPDN1 mode

The following figure illustrates a PWM in COUNT_UPDN1 mode. The counter is initialized (to 1) and started with a software-based reload event.

- 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 equals 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

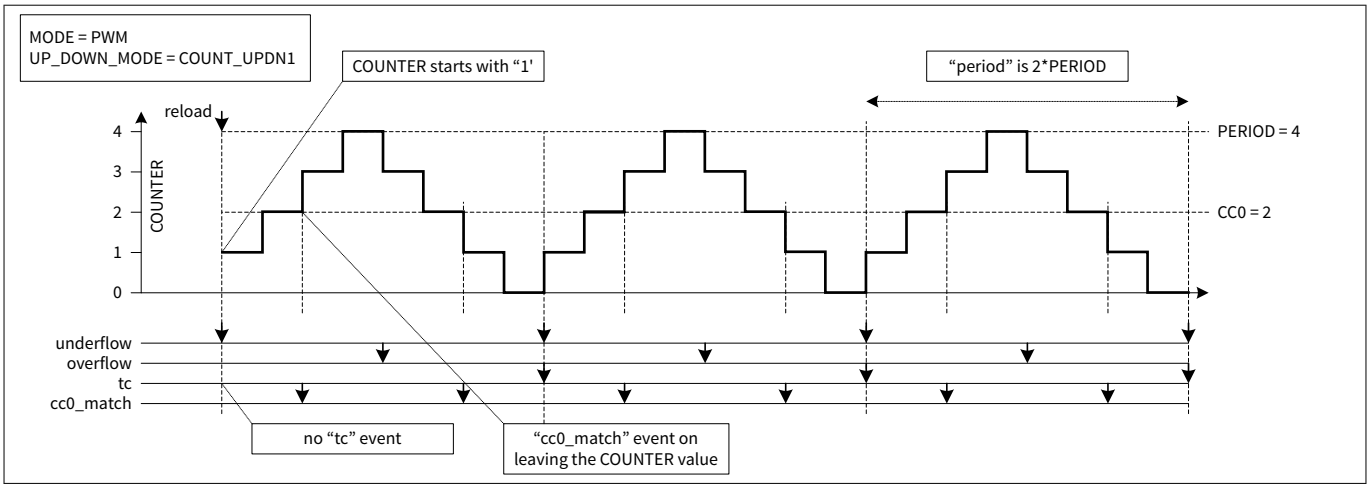


Figure 368 PWM in COUNT_UPDN1 mode

33.3.3.3 One shot mode

It is same as One shot mode.

33.3.3.4 Clock pre-scaling

It is same as Clock pre-scaling.

33.3.3.5 CC 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 CTRL.AUTO_RELOAD_CC0 is "1", CC0 and CC0_BUFF will exchange value
- At TC event, if switch event is active and CTRL.AUTO_RELOAD_PERIOD is "1", PERIOD and PERIOD_BUFF will exchange value
- When GRP_CC1_PRESENT=1, CC1 and CC1_BUFF will exchange value at TC event if switch event is active

The following figures illustrate that the period value is updated in COUNT_UP mode and COUNT_DOWN mode for SWAP_ENABLE="1" and SWAP_ENABLE="0".

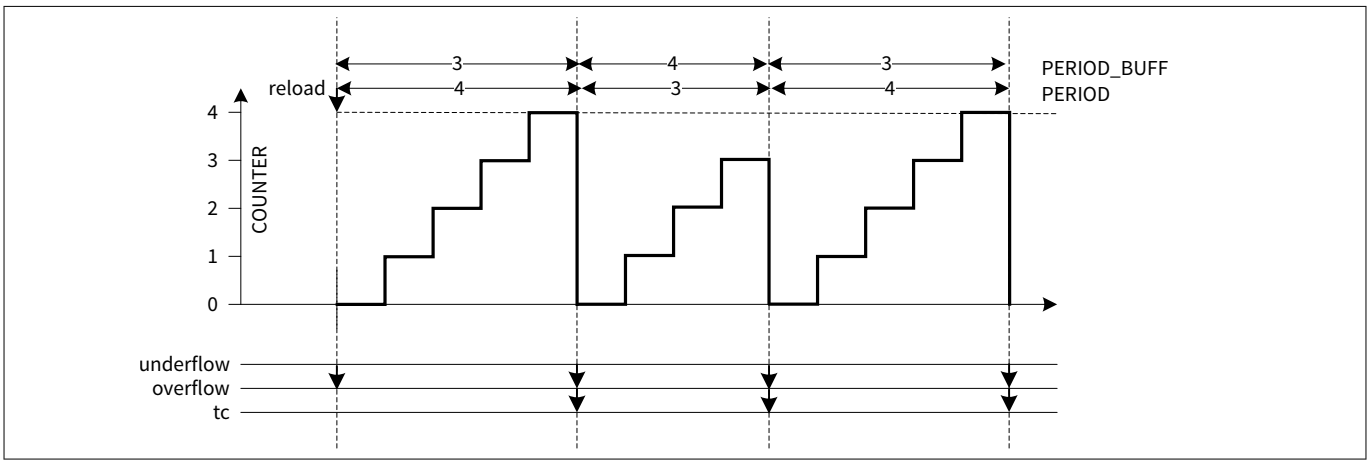


Figure 369 PERIOD/PERIOD_BUFF exchange in COUNT_UP mode (SWAP_ENABLE="1")

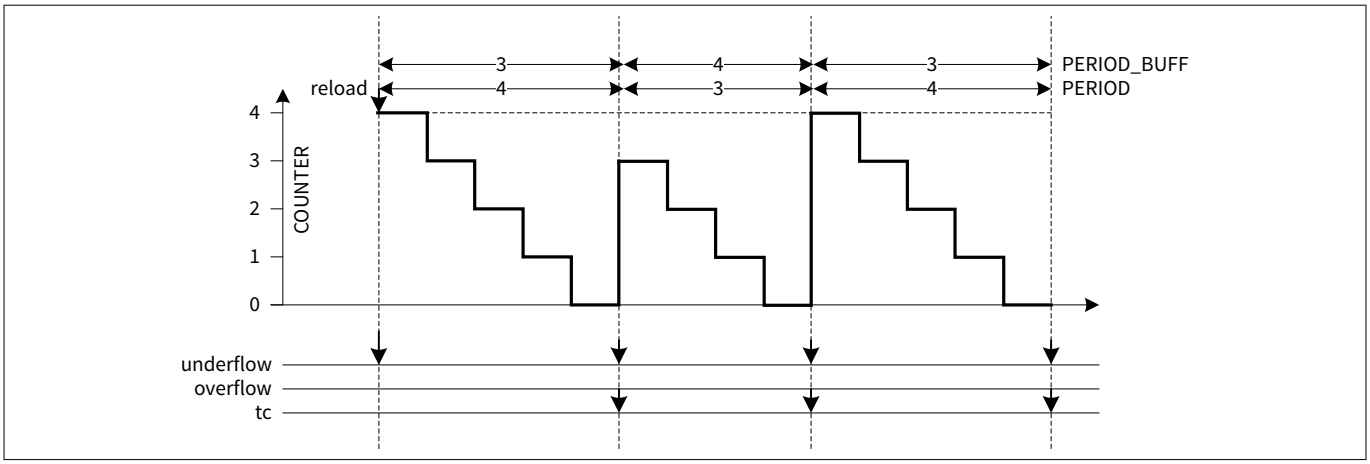


Figure 370 PERIOD/PERIOD_BUFF exchange in COUNT_DOWN mode (SWAP_ENABLE="1")

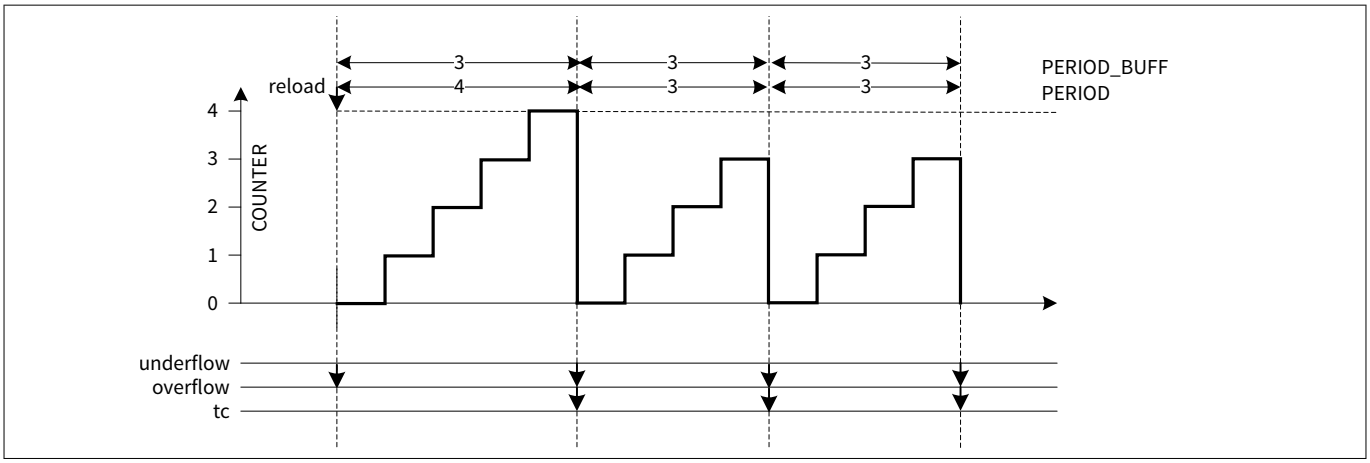


Figure 371 PERIOD/PERIOD_BUFF exchange in COUNT_UP mode (SWAP_ENABLE="0")

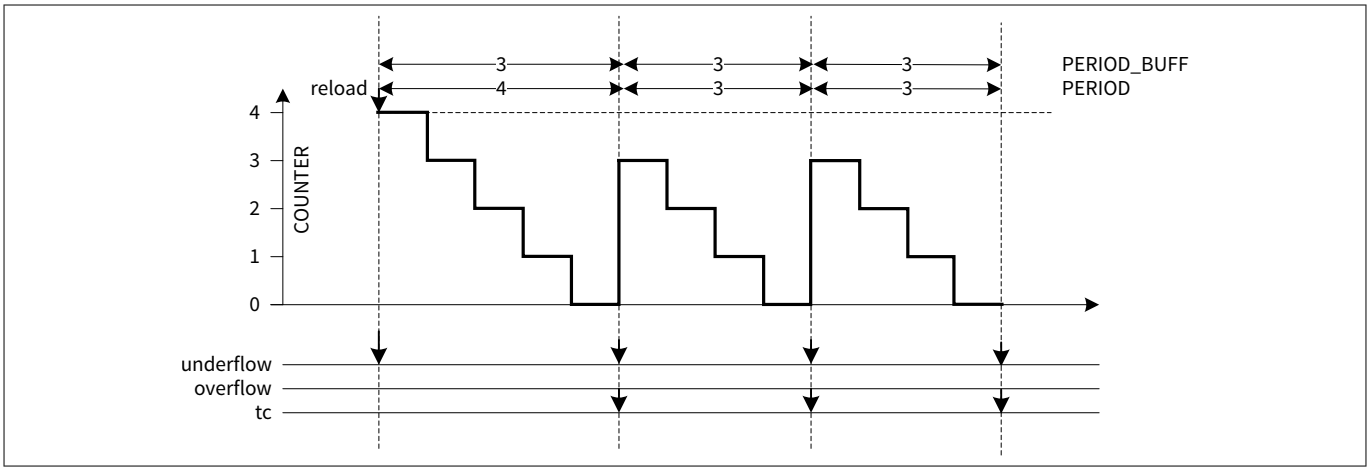


Figure 372 PERIOD/PERIOD_BUFF exchange in COUNT_DOWN mode (SWAP_ENABLE="0")

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 an MMIO register write after both the PERIOD_BUFF and CC0_BUFF registers are updated. The switch event is kept pending by the hardware till the next tc event occurs.

The previous paragraph addressed synchronized updates of the CC0/CC0_BUFF and PERIOD/PERIOD_BUFF registers of a single PWM by means of a software-generated switch event. During motor control, three PWMs work in unison and updates to all period and compare register pairs should be synchronized. All three PWMs have synchronized periods and as a result have synchronized tc events. The switch event for all three PWMs is generated by the same hardware trigger input to synchronously update CC0 at the same TC, after the PERIOD_BUFF and CC0_BUFF registers of all three PWMs are updated by SW.

33.3.3.6 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 internal event and can be configured in TR_PWM_CTRL register

The required settings for left align, right align, and center align PWM are:

- Left align: COUNT_UP mode with “overflow=SET, cc0_match=CLEAR”
- Right align: COUNT_DOWN mode with “cc0_match=SET, underflow=CLEAR”
- Center align: COUNT_UPDN1 mode with “overflow=SET, underflow=CLEAR, cc0_match=INVERT”

The expected pulse widths of PWM align modes are listed below:

- Left-align (COUNT_UP): pulse width = CC0

  To get 100% duty cycle, CC0 must be set a value larger than period. If period is $FFFF_H$ ($FFFFFFFF_H$ for 32-bit counter), 100% duty cycle cannot be achieved
- Right-align (COUNT_DOWN): pulse width = CC0+1

  To get 0% duty cycle, CC0 must be set a value larger than period. If period is $FFFF_H$ ($FFFFFFFF_H$ for 32 bit counter), 0% duty cycle cannot be achieved
- Center-align (COUNT_UPDN1): pulse width = 2*(PERIOD-CC0)

Left align PWM

The following figure illustrates a PWM in COUNT_UP mode with different CC0 values. The figure also illustrates how a left-aligned and and right-aligned PWM can be creating using the PWM in COUNT_UP mode.

Note:
- *CC0 is changed (to CC0_BUFF, which is not depicted) on a tc event*
- *Switch event is constant “1”*

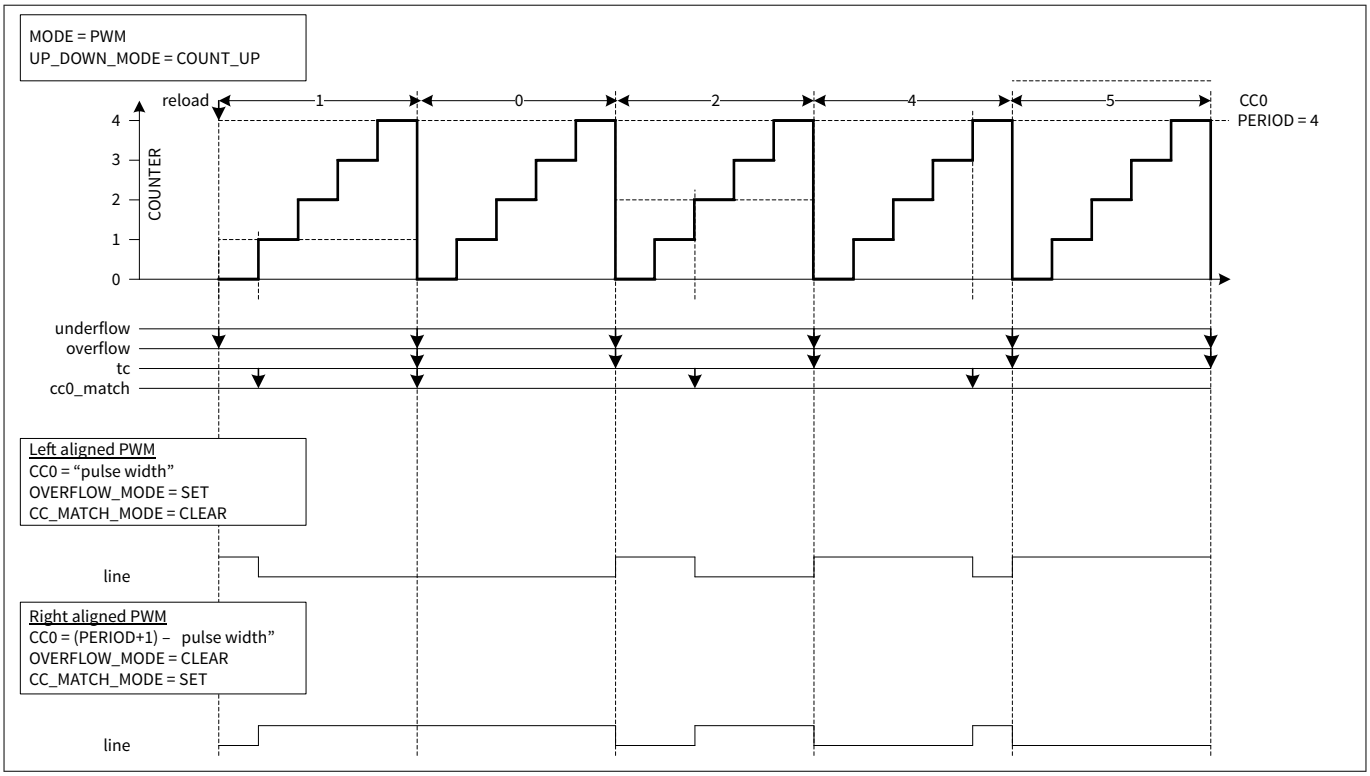


Figure 373 Left align PWM

Right align PWM

The following figure illustrates a PWM in COUNT_DOWN mode with different CC0 values. The figure also illustrates how a right-aligned PWM can be creating using the PWM in COUNT_DOWN mode.

***Note**:*
- *CC0 is changed (to CC0_BUFF, which is not depicted) on a tc event*
- *Switch event is constant "1"*

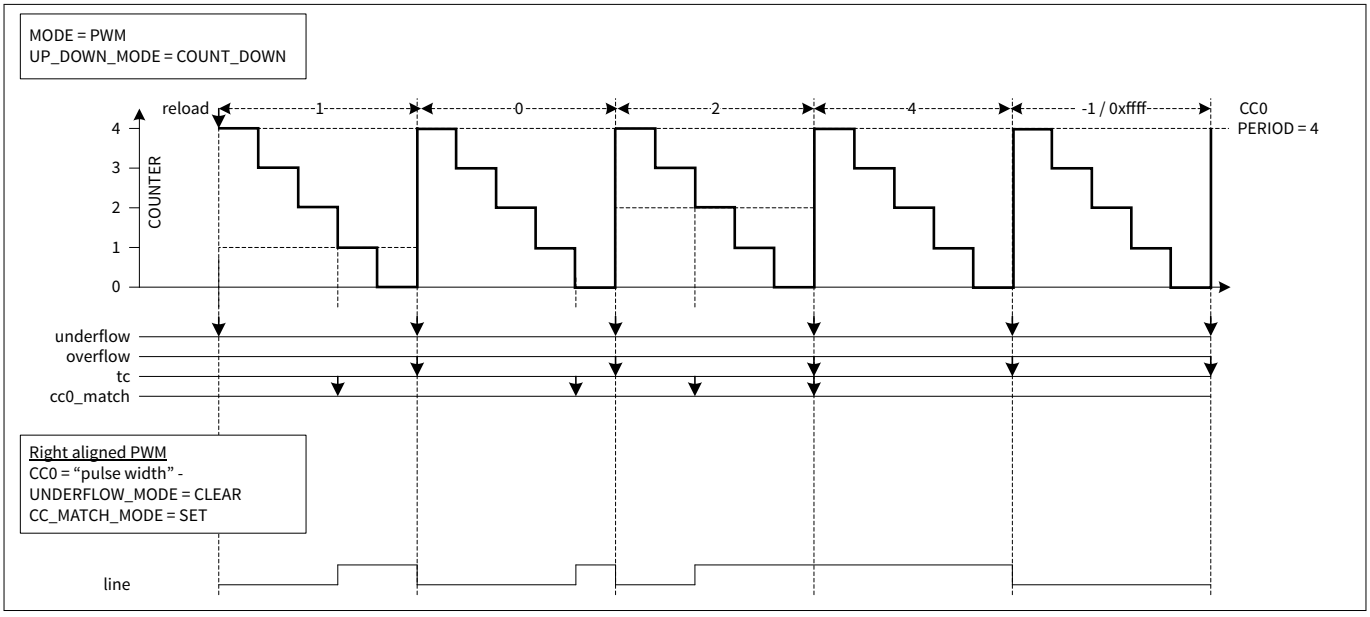


Figure 374 Right align PWM

Center align PWM

The following figure 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*
- *cc0_match will generate at the beginning of the count period at the same time the CC0 switches to "0". It's a special case that cc0_match generate without "COUNTER equals CC0" (CC0 changes to "0" at same time COUNTER changes to "1" from "0")*

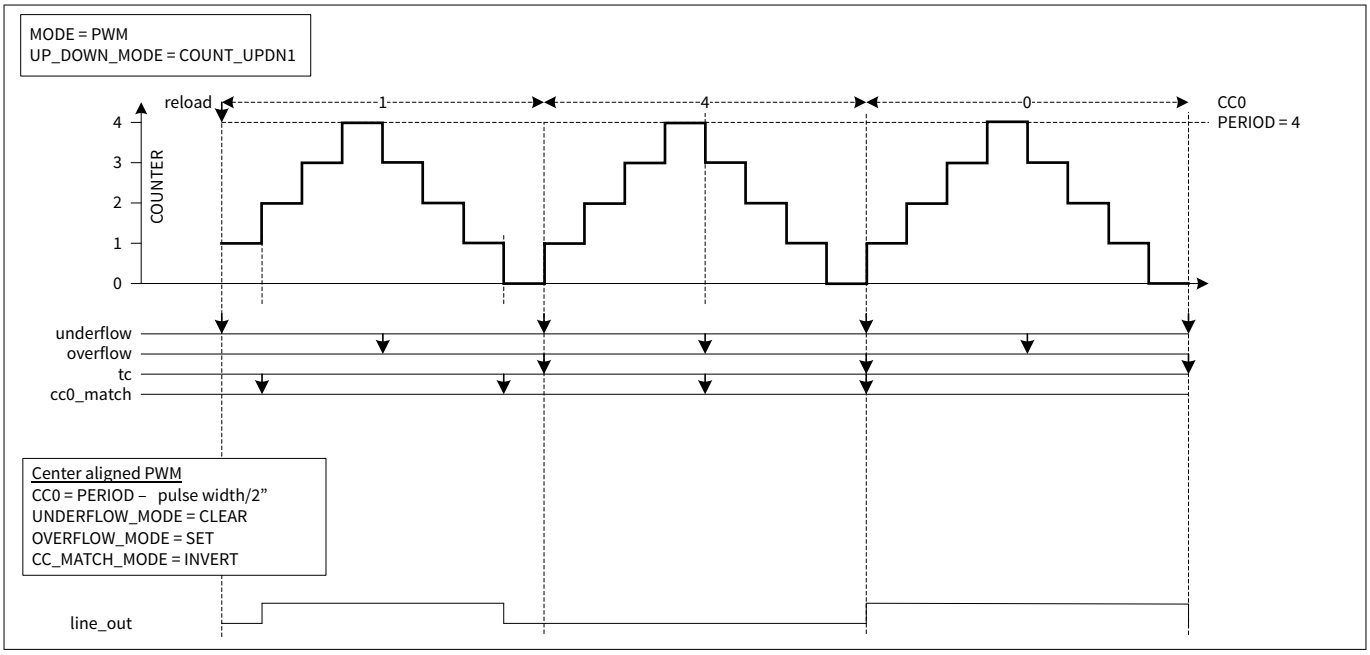


Figure 375 Center align PWM

The following figure illustrates 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.

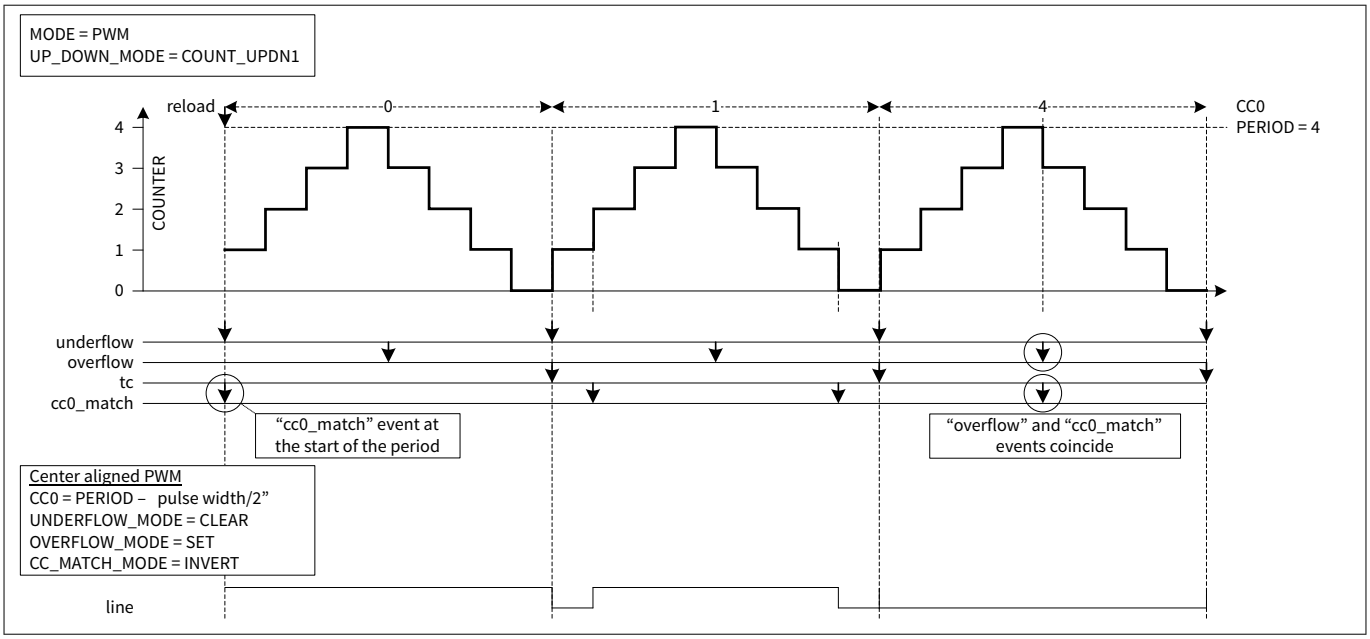


Figure 376 Center align PWM with CC0 = "0" after reload

33.3.3.7 Asymmetric PWM – Using a single compare function

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.

Following are the settings required for asymmetric PWM:

- Asymmetric PWM: COUNT_UPDN2 mode with "overflow=SET, underflow=CLEAR, cc0_match=INVERT"

The asymmetric PWM typically uses 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 that:

- Within a PWM period, 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, 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 377 illustrates how the COUNT_UPDN2 mode is used to generate an asymmetric PWM.

***Note**: 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.*

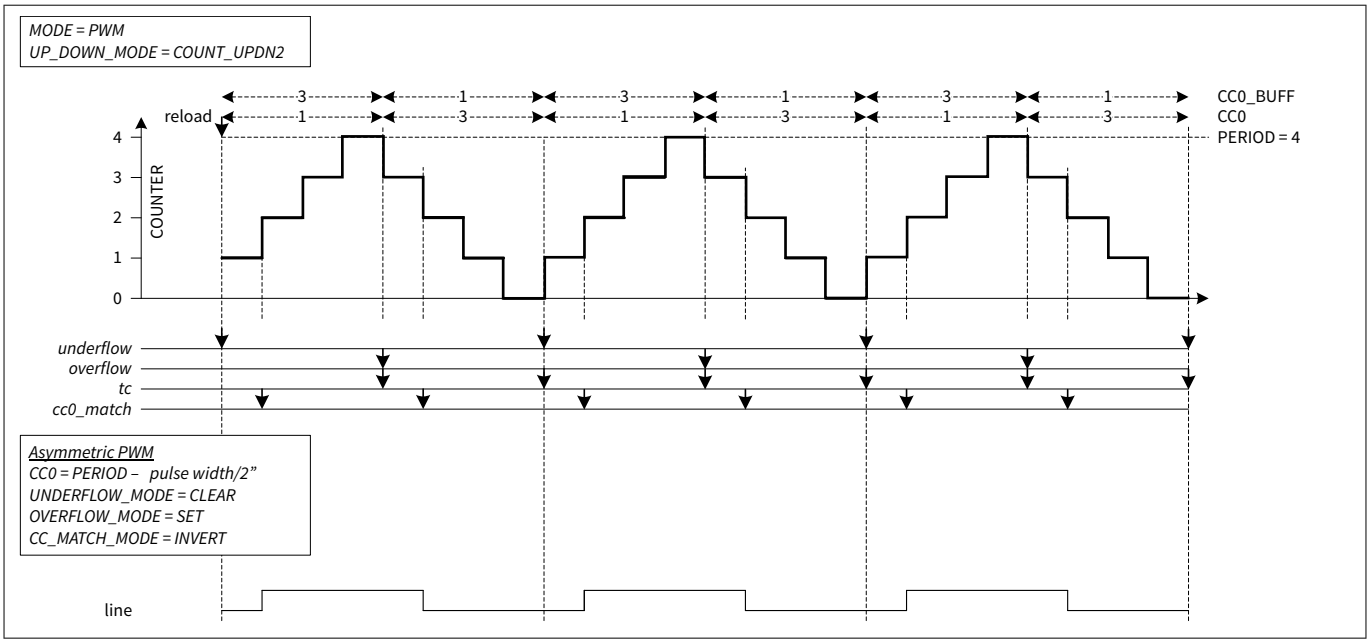


Figure 377 Asymmetric PWM

The previous waveform-illustrated functionality when the CC0 values are neither “0” not PERIOD. Corner case conditions in which the CC0 values equal “0” or PERIOD are illustrated below.

Note:
- *When up counting, when the CC0 value at the underflow event is 0, a cc0_match event is generated*
- *When down counting, when 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*
- *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)*

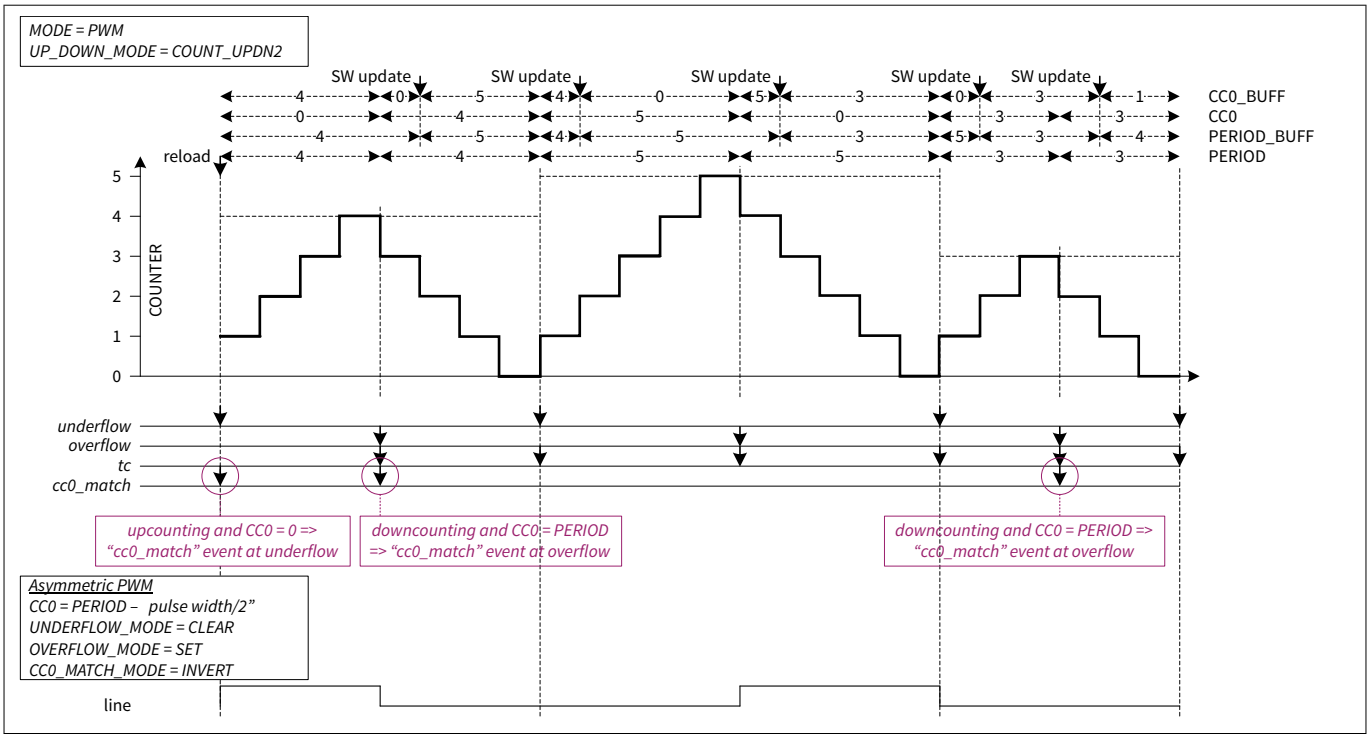


Figure 378 Asymmetric PWM with CC0=0 and CC0=PERIOD duty cycle

33.3.3.8 Asymmetric PWM – Using two compare functions

When parameter GRP_CC1_PRESENT = 1, the compare 1 function using registers CC1 and CC1_BUFF generating the cc1_match event is the same as the compare function using registers CC0 and CC0_BUFF generating the cc0_match event.

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 provide another way to generate an asymmetric PWM as shown in the following figure.

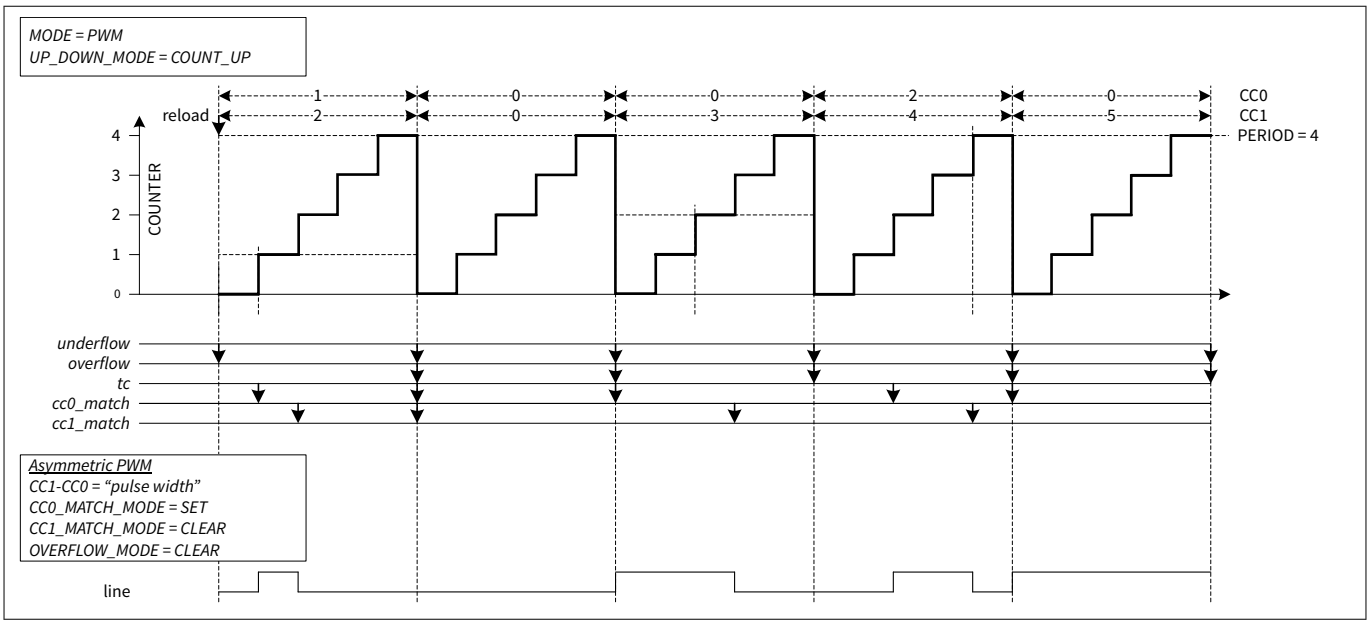


Figure 379 Asymmetric PWM by cc0_match and cc1_match in COUNT_UP mode

Such asymmetric PWM generation is more flexible than using only one compare function in the COUNT_UPDN2 mode. However, if another (3rd) compare function is needed, for example, to trigger an ADC, another synchronously running counter has to be used.

When parameter GRP_AMC_PRESENT = 1 (Advanced Motor Control feature supported), 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). That allows an 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 Figure 380, which shows one of three center aligned PWM phases for motor control during a time where the duty cycle value is increased from one period to the next (rising part of sign wave modulated onto the PWM signal).

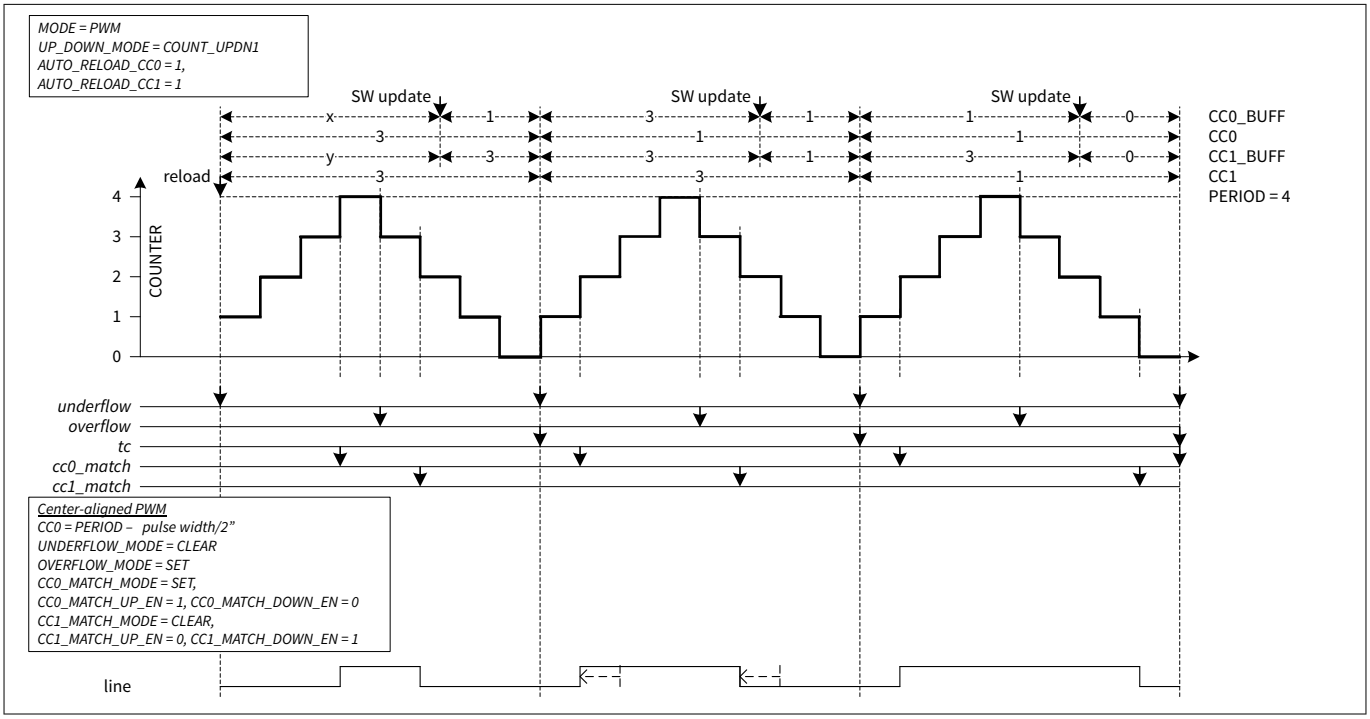


Figure 380 Asymmetric PWM by cc0_match and cc1_match in COUNT_UPDN1 mode

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 PMSM (permanent-magnet synchronous motor) when the current duty cycle values of the 3 phases do not allow that (too small window where 1 PWM channel is active, 2 others 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), that is, all updates can be done 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 ways to enable asymmetric PWM using the 2nd compare function:

- COUNT_UP/DOWN mode with "cc0_match =SET, cc1_match=CLEAR". Pulse width = CC1-CC0
- COUNT_UPDN1/2 mode with "overflow=SET, underflow=CLEAR, cc0_match=SET, cc1_match=CLEAR, CC0_MATCH_UP_EN=1, CC0_MATCH_DOWN_EN=0, CC1_MATCH_ UP_EN=0, CC1_MATCH_DOWN_EN=1". Pulse width = 2xPERIOD – CC1 – CC0

33.3.3.9 Kill mode

PWM mode has different stop/kill modes. The mode is specified by PWM_IMM_KILL, PWM_STOP_ON_KILL, PWM_SYNC_KILL, PWM_TC_SYNC_KILL_DT and PWM_SYNC_KILL_DT.

The configuration option PWM_IMM_KILL specifies when the PWM output signals are started to be suppressed.

- PWM_IMM_KILL is '1'. The PWM output signals LINE_OUT and LINE_COMPL_OUT are immediately suppressed when a kill event is detected
- PWM_IMM_KILL is '0'. The PWM output signals LINE_OUT and LINE_COMPL_OUT are suppressed synchronously with the next count clock after a kill event is detected

The following figures illustrate both configurations.

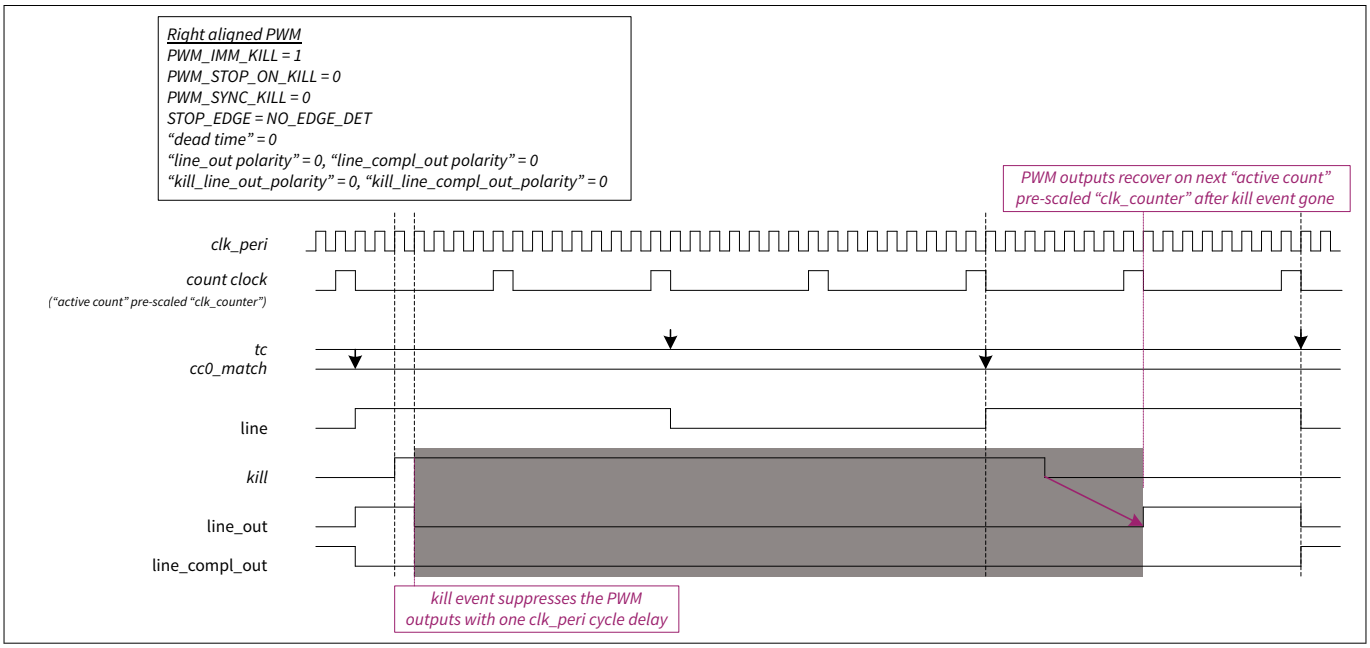


Figure 381 Kill suppresses line output immediately (PWM_IMM_KILL = 1)

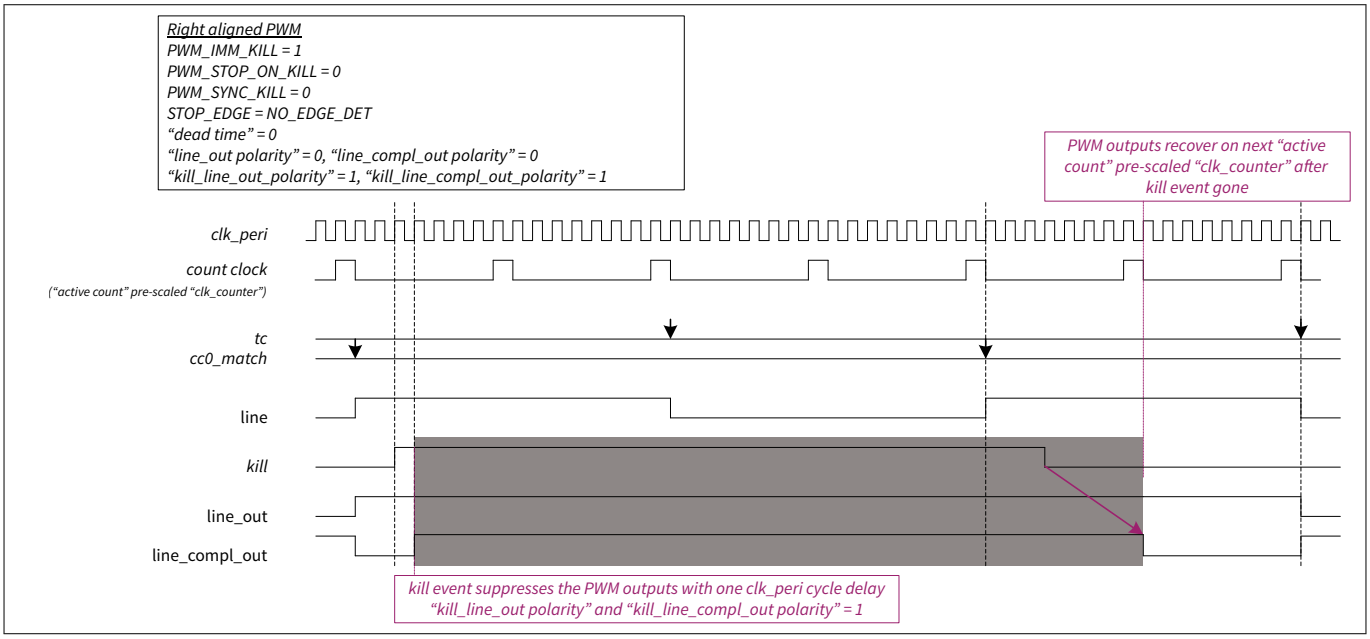


Figure 382 Kill suppresses line output immediately (PWM_IMM_KILL = 1) with "kill_line_out_polarity" and "kill_line_compl_out_polarity" = 1

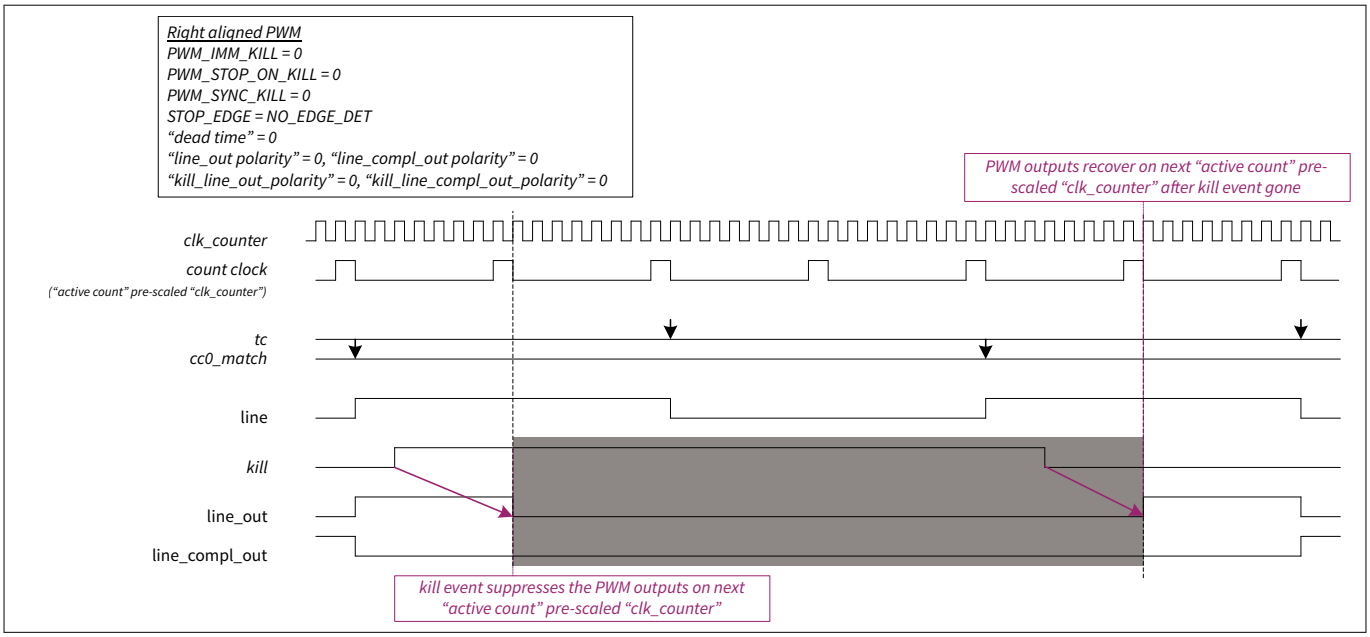


Figure 383 Kill suppresses line output by count clock (PWM_IMM_KILL = 0)

As shown in the above figures, kill event is an internal signal which is synchronized from external hardware asynchronous kill input (hardware kill input is not shown in the figure) with 2 clk_peri cycle delay. PWM output will be suppressed after one more clk_peri cycle.

The kill mode PWM_STOP_ON_KILL, PWM_SYNC_KILL, PWM_TC_SYNC_KILL_DT and PWM_SYNC_KILL_DT specify the functionality of kill input. Five modes are supported that mentioned in Table 341: Kill stop mode (IMM-Stop); Asynchronous kill mode (IMM-Async); Synchronous kill mode (IMM-Sync); TC Synchronous DT kill mode (TC-Sync-DT) and Synchronous DT kill mode (IMM-Sync-DT). The following figures illustrate the 5 modes, with “kill_line_out_polarity” and “kill_line_compl_out_polarity” set to 0 and 1.

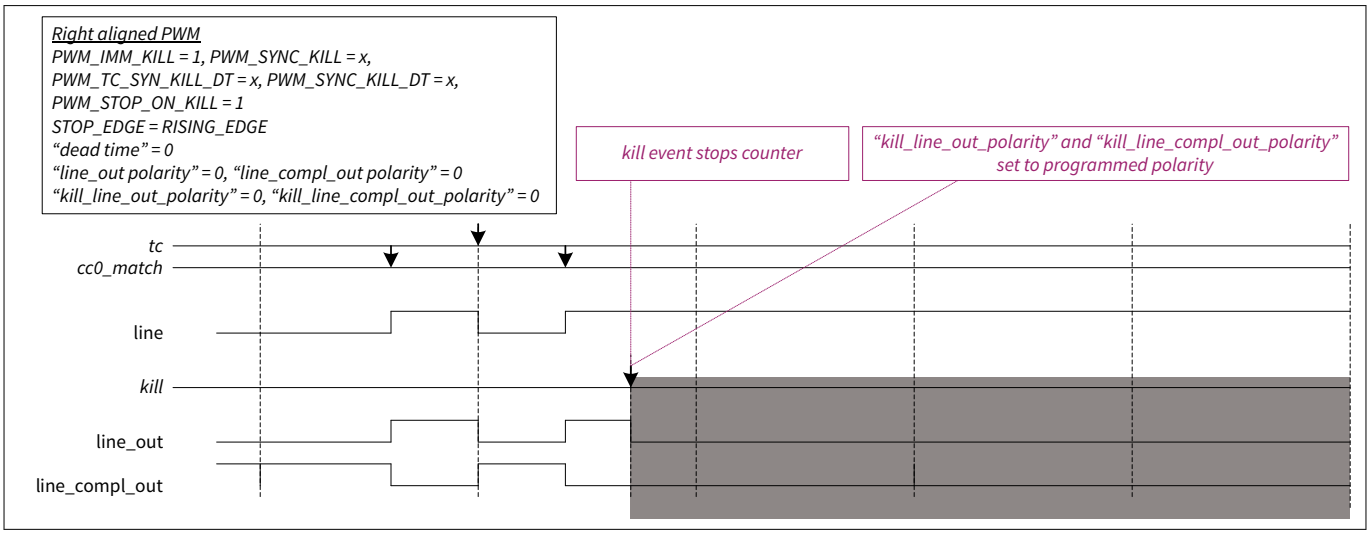


Figure 384 Stop on kill

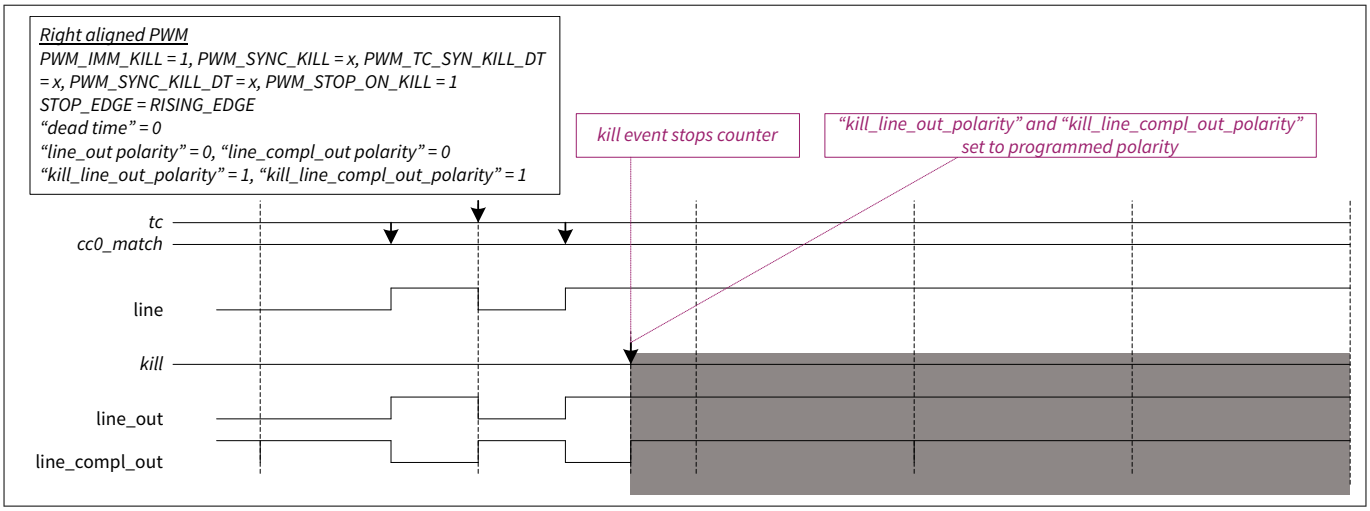


Figure 385 Stop on kill with “kill_line_out_polarity” and “kill_line_compl_out_polarity” = 1

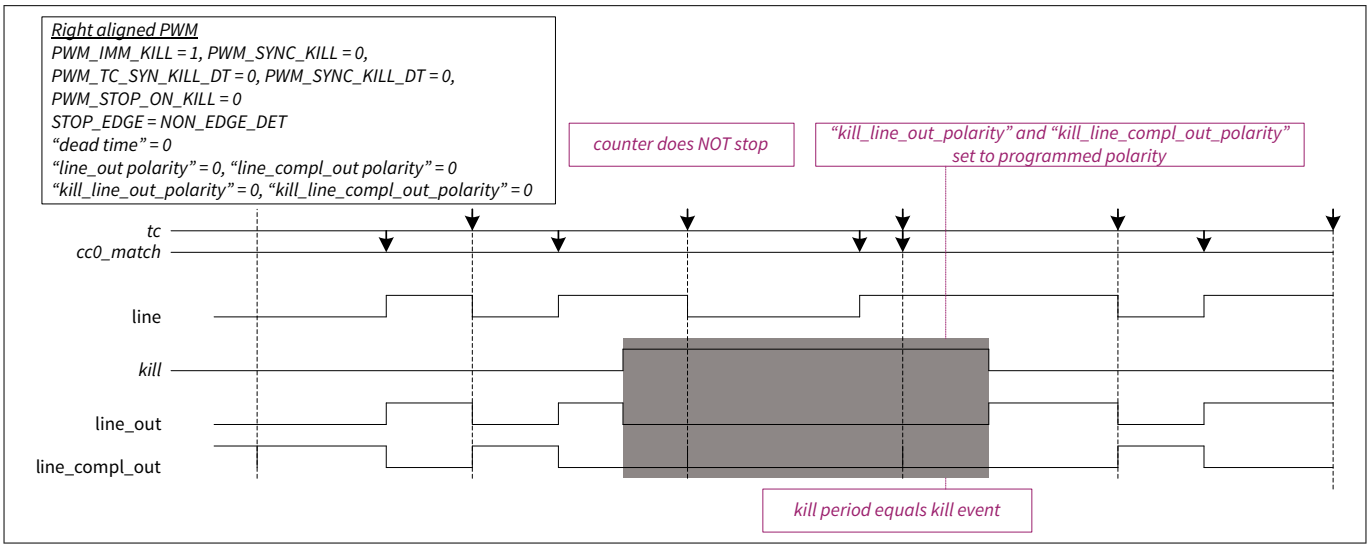


Figure 386 Asynchronous kill

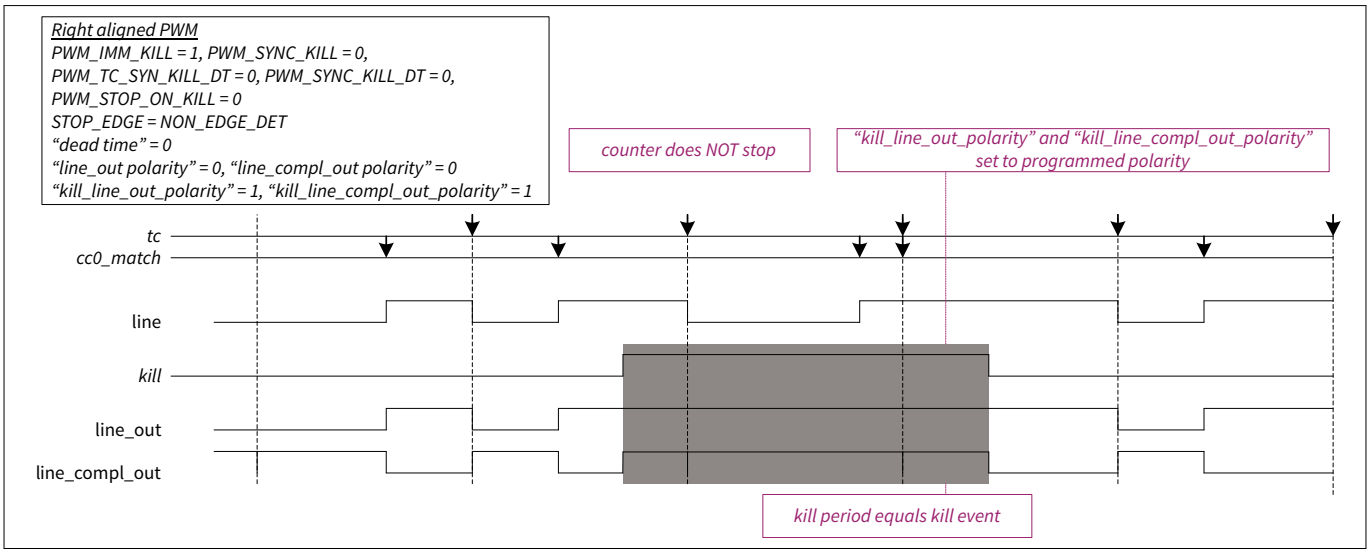


Figure 387 Asynchronous kill with “kill_line_out_polarity” and “kill_line_compl_out_polarity” = 1

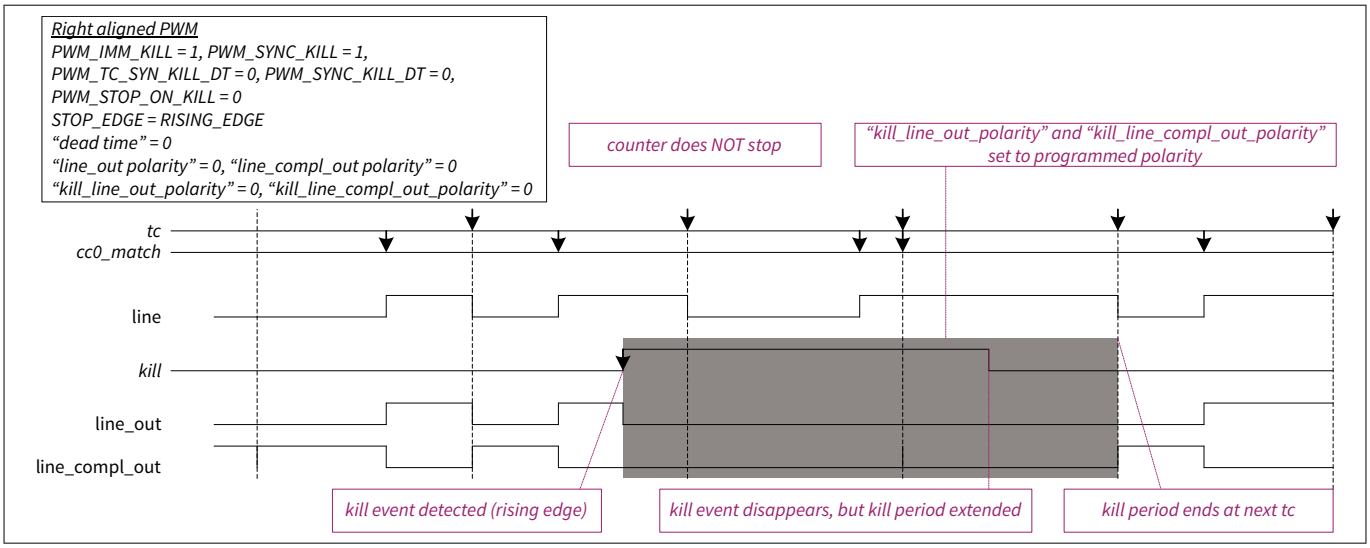


Figure 388 Synchronous kill

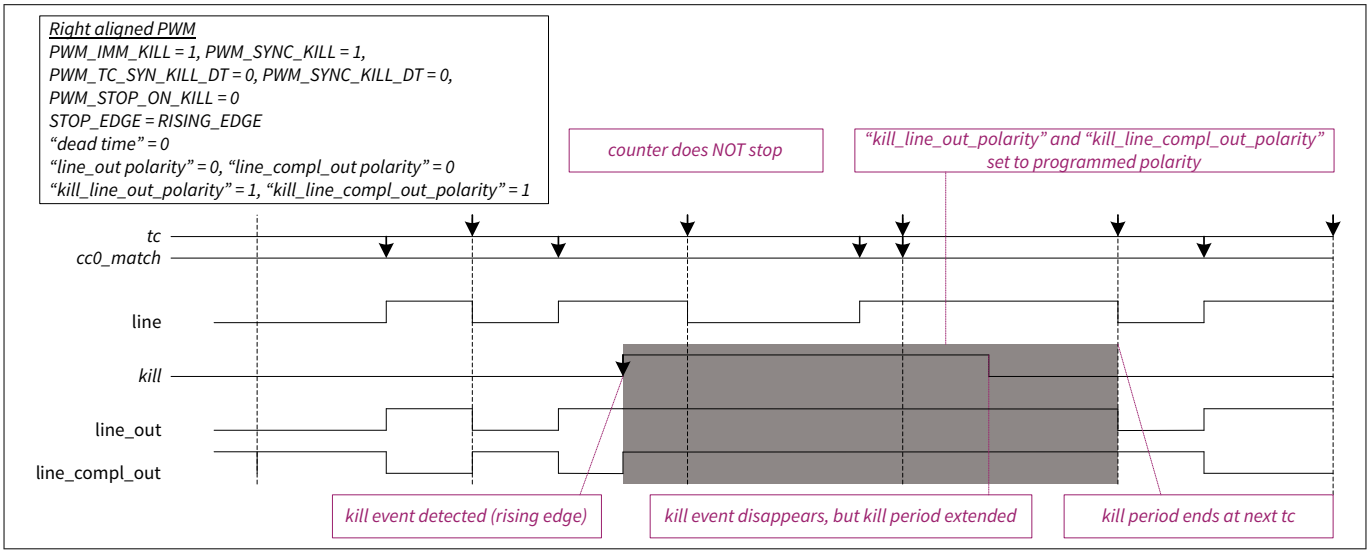


Figure 389 Synchronous kill with “kill_line_out_polarity” and “kill_line_compl_out_polarity” = 1

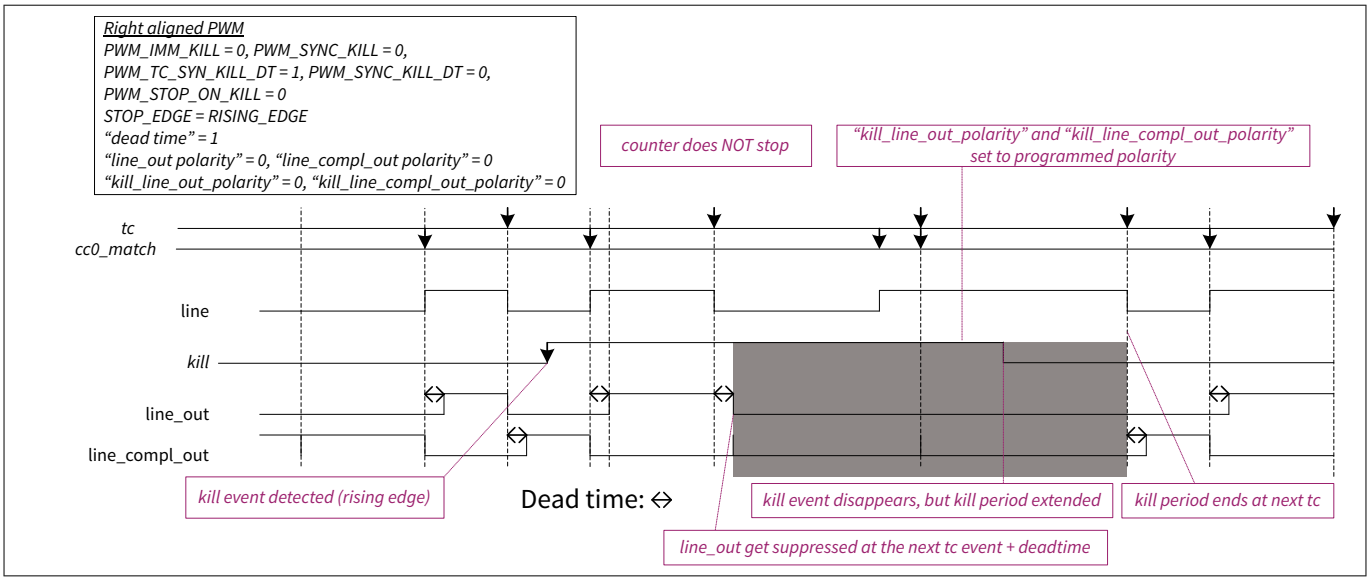


Figure 390 TC synchronous DT kill

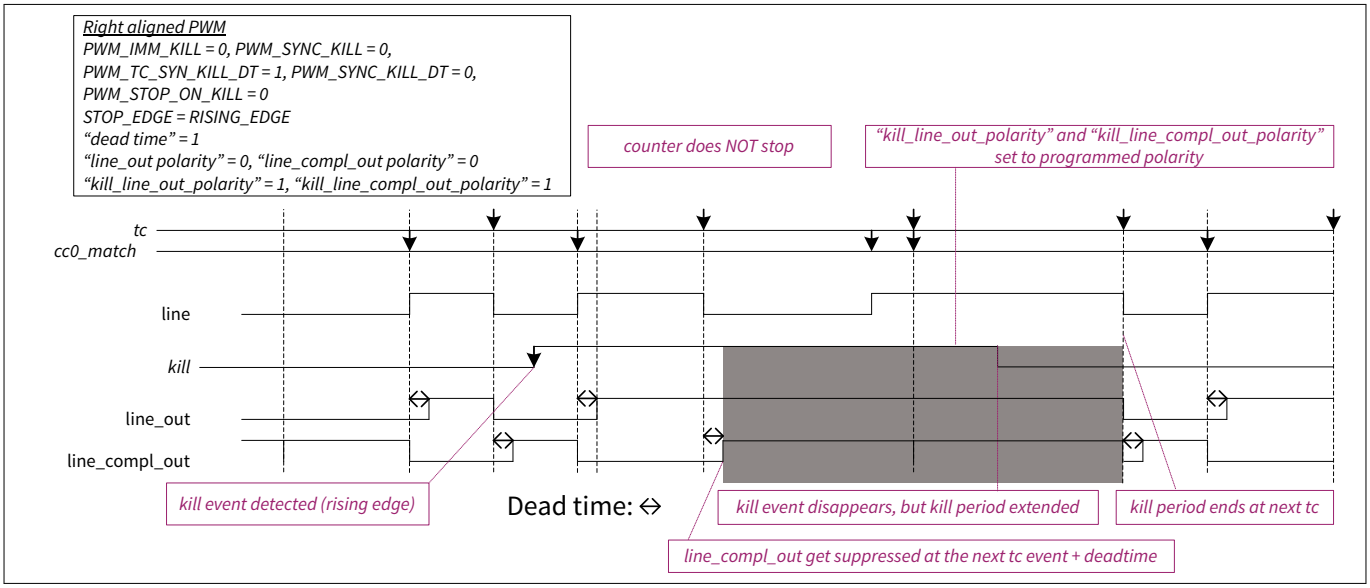


Figure 391 TC synchronous DT kill with “kill_line_out_polarity” and “kill_line_compl_out_polarity” = 1

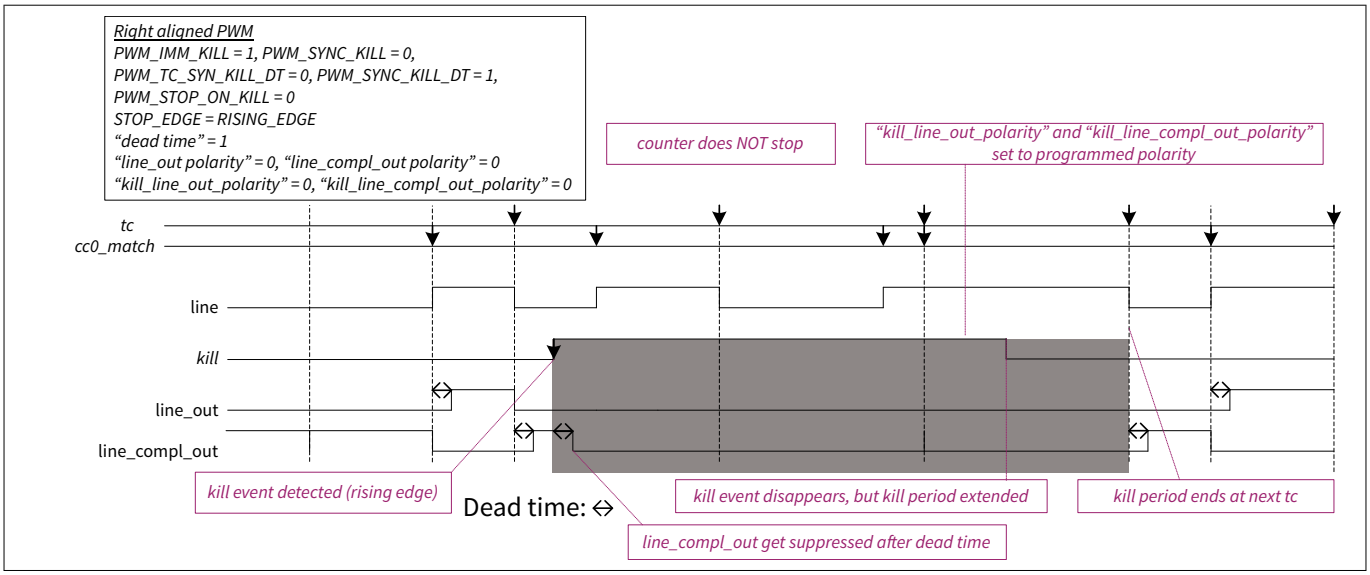


Figure 392 Synchronous DT kill

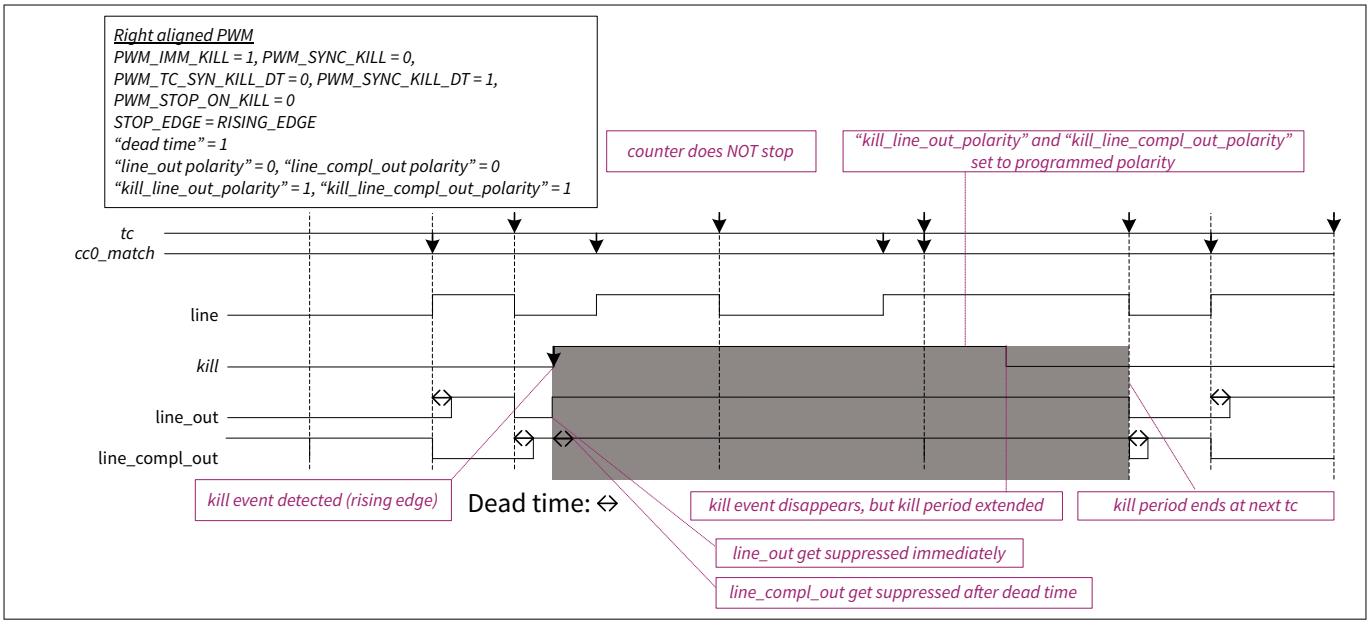


Figure 393 Synchronous DT kill with “kill_line_out_polarity” and “kill_line_compl_out_polarity” = 1

Different from asynchronous kill, synchronous kill will stop suppressing line output on tc event after kill is gone.

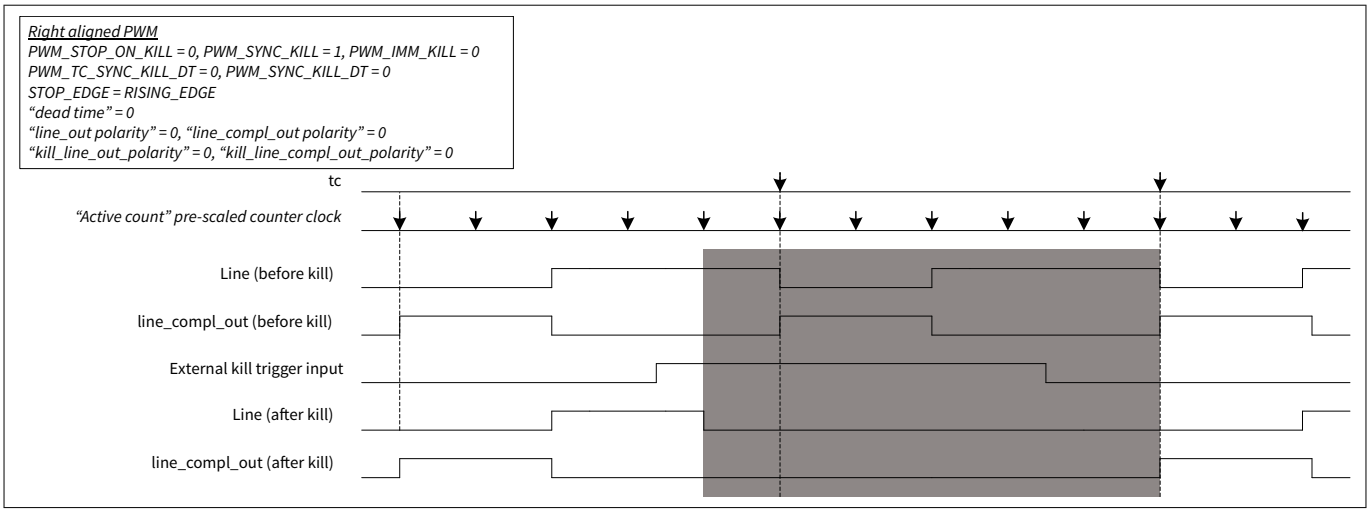


Figure 394 Synchronous kill timing (PWM_IMM_KILL = 0)

Whatever the value of PWM_IMM_KILL is, PWM recovers on the next "active count" pre-scaled counter clock (PWM_SYNC_KILL=0) or next TC (PWM_SYNC_KILL=1) after the kill event is inactive. An exception is that when PWM_SYNC_KILL= "0" and PWM_IMM_KILL= "1", the pulse width of kill event is too narrow to be recognized by the "active count" pre-scaled counter clock. The PWM output recovers immediately as illustrated in the following figure.

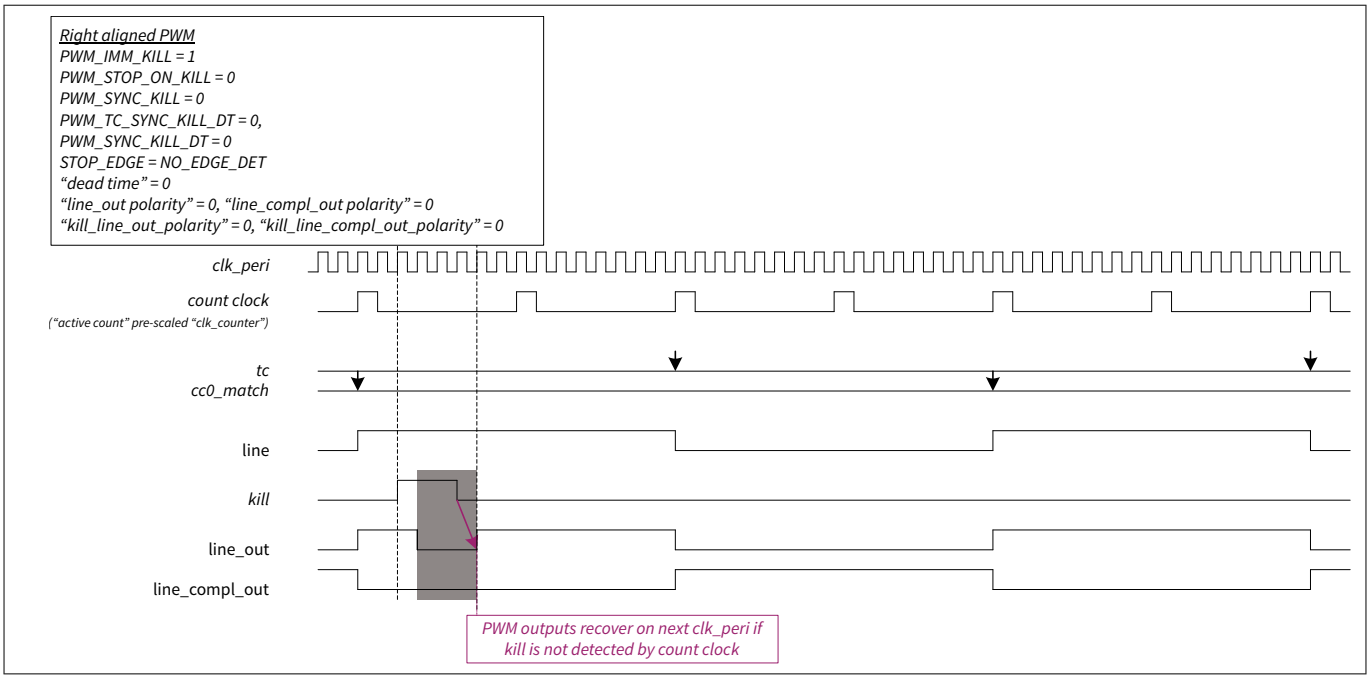


Figure 395 PWM outputs recover on the next clk_peri if kill is not detected by count clock

When GRP_CC1_PRESENT = 1, the capture1 event also works as a kill input, just like the stop event. Both events are OR combined and share the same kill mode settings. Having two stop/kill events for a PWM allows selecting for one stop/kill event a common trigger from a PERI trigger multiplexer (allowing synchronous stop/kill operation of multiple PWMs) while selecting for the other stop/kill event, for example, a dedicated ADC out-of-range trigger (for example, allowing real-time hardware stop of a PWM when current of PWM driven signal is out of range).

33.3.3.10 PWM output selection for stepper motor control

In PWM mode, the TCPWM counters with parameter GRP_SMC_PRESENT = 1 can be used for Stepper Motor Control 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 (register LINE_SEL, fields OUT_SEL and COMPL_OUT_SEL):

- constant low (‘0’)
- constant high (‘1’)
- PWM signal “line”
- Inverted PWM signal “line”
- ‘Z’ (high impedance)

***Note**: This feature generates digital signals for stepper motor control. These signals cannot directly drive a stepper motor, as they lack sufficient current strength.*

Constant low or high values are used for full stepping while a PWM signal is used for micro stepping. With the option “Z” the TCPWM output signal “line_out_en” or “line_compl_out_en” can be set to ‘0’ which causes the connected IO cell not to drive the output. This allows it to measure the induced voltage of a currently not driven coil via ADC, which is used for zero point detection.

***Note**: For full-stepping, set the TCPWM counter to PWM mode. However, a PWM generation is not needed, that is, the compare function resources (CC0/CC1) are free to be used for other purposes (for example, trigger generation).*

For each coil of a stepper motor, a separate TCPWM counter is used. These counters are running 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 for updating the CC0/CC1 or PERIOD registers. Therefore, a LINE_SEL_BUFF register is used which is exchanged with the LINE_SEL register on a TC event with an actively pending switch event (when specified by CTRL.AUTO_RELOAD_LINE_SEL). This is shown in the following figure of an example driving the two coils of a stepper motor with two TCPWM counters when CTRL.SWAP_ENABLE = “1”.

It illustrates three full steps (180°, 270°, 0°(=360°) electrical angles), followed by three micro steps using a PWM (~30°, ~60°, ~120° electrical angles). It is just an illustration example showing one step per PWM counter PERIOD. In reality, the steps for micro stepping are significantly slower compared to the PWM period and therefore, the PWM duty cycles (CC0) and/or output select registers (LINE_SEL) are stable over many PWM periods before they get changed by a switch event.

The time base for the steps can be realized by 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 that trigger the CPU to update PWM duty cycles and/or output select registers and to finally generate the switch event using the PERI0_TR_CMD register in the MXPERI block.

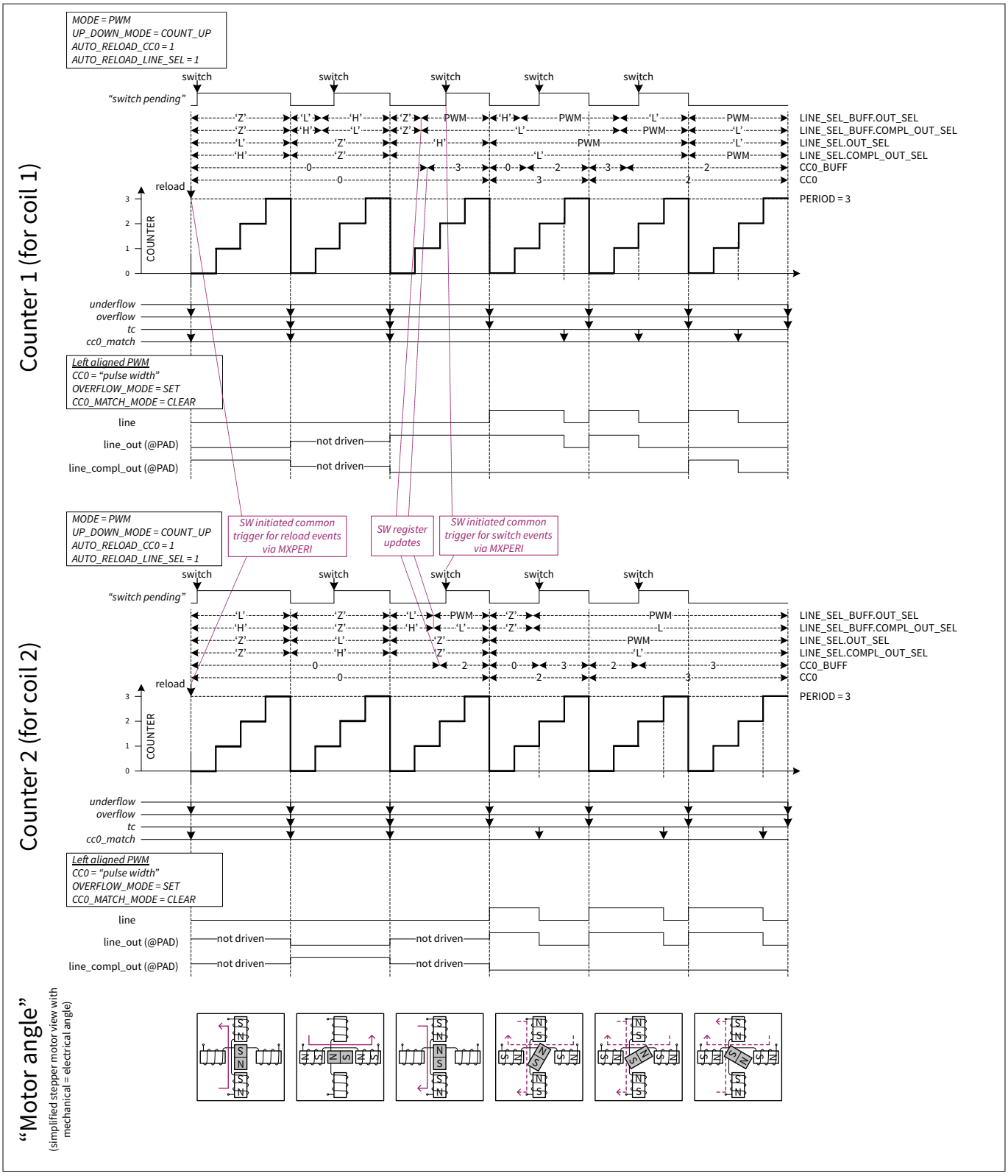


Figure 396 PWM output for stepper motor

The following figure shows an example of driving the two coils of a stepper motor with two TCPWM counters when CTRL.SWAP_ENABLE = "0". It illustrates how the software can update the register for the stepper motor control.

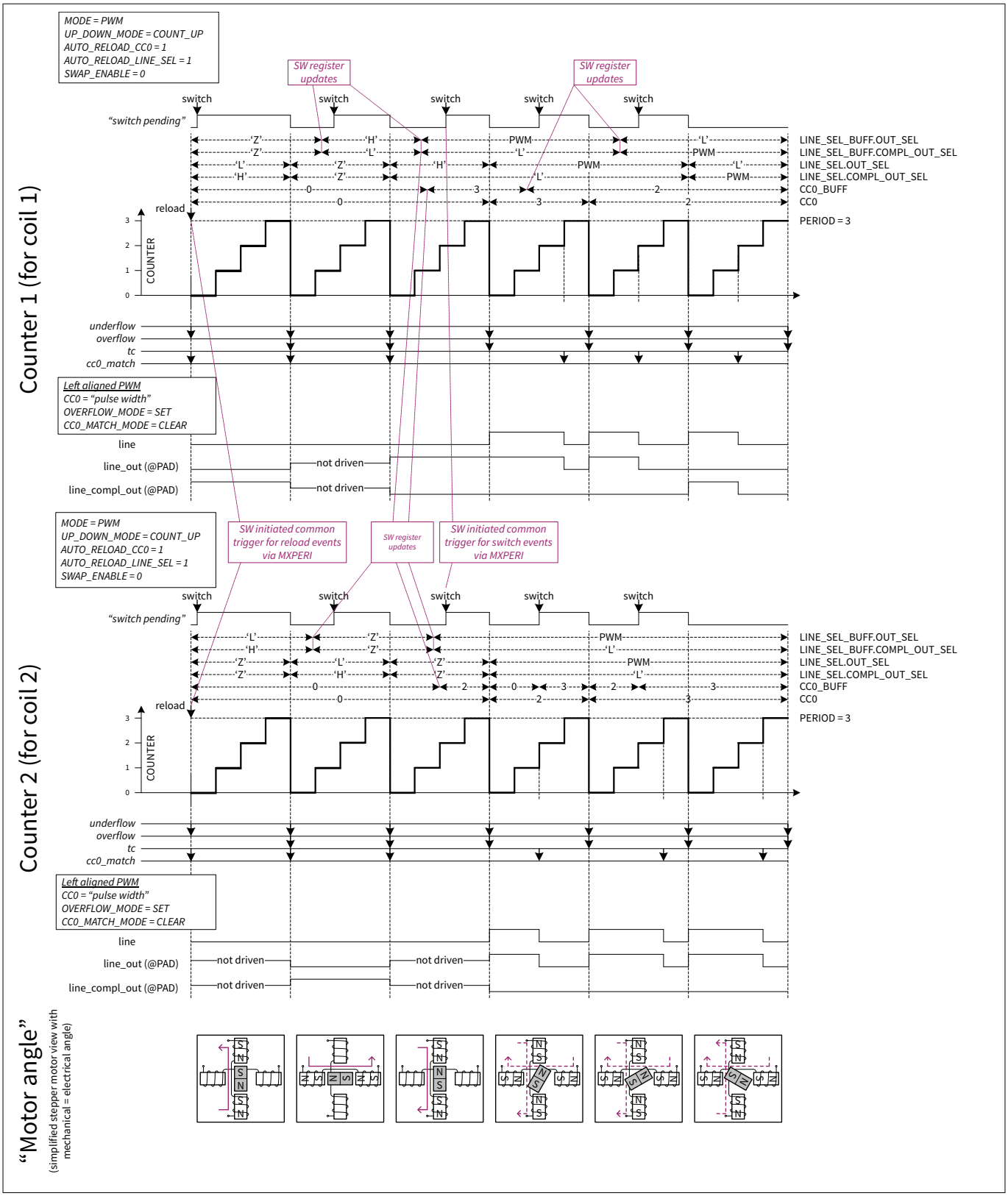


Figure 397 PWM output for stepper motor (SWAP_ENABLE=0 case)

33.3.3.11 Shifting PWM output using counter value

To shift to the PWM output of different counters with respect to each other is possible using different initial values of the counter during start. The start value determines the shift of PWM output with respect to other

counters. In the following figure, three counters: Counter 2 and Counter 3 are shifted by X and Y with respect to Counter 1 respectively using some non-zero initial counter value.

***Note**: If put cc value that has already passed than cc match would be generated immediately due to the immediate compare match. If it is not required, then the value of cc must be put to be something greater than the period so that no cc match is generated while the correct value can be put in cc buffer registers so that normal cc matches are generated after shadow transfer.*

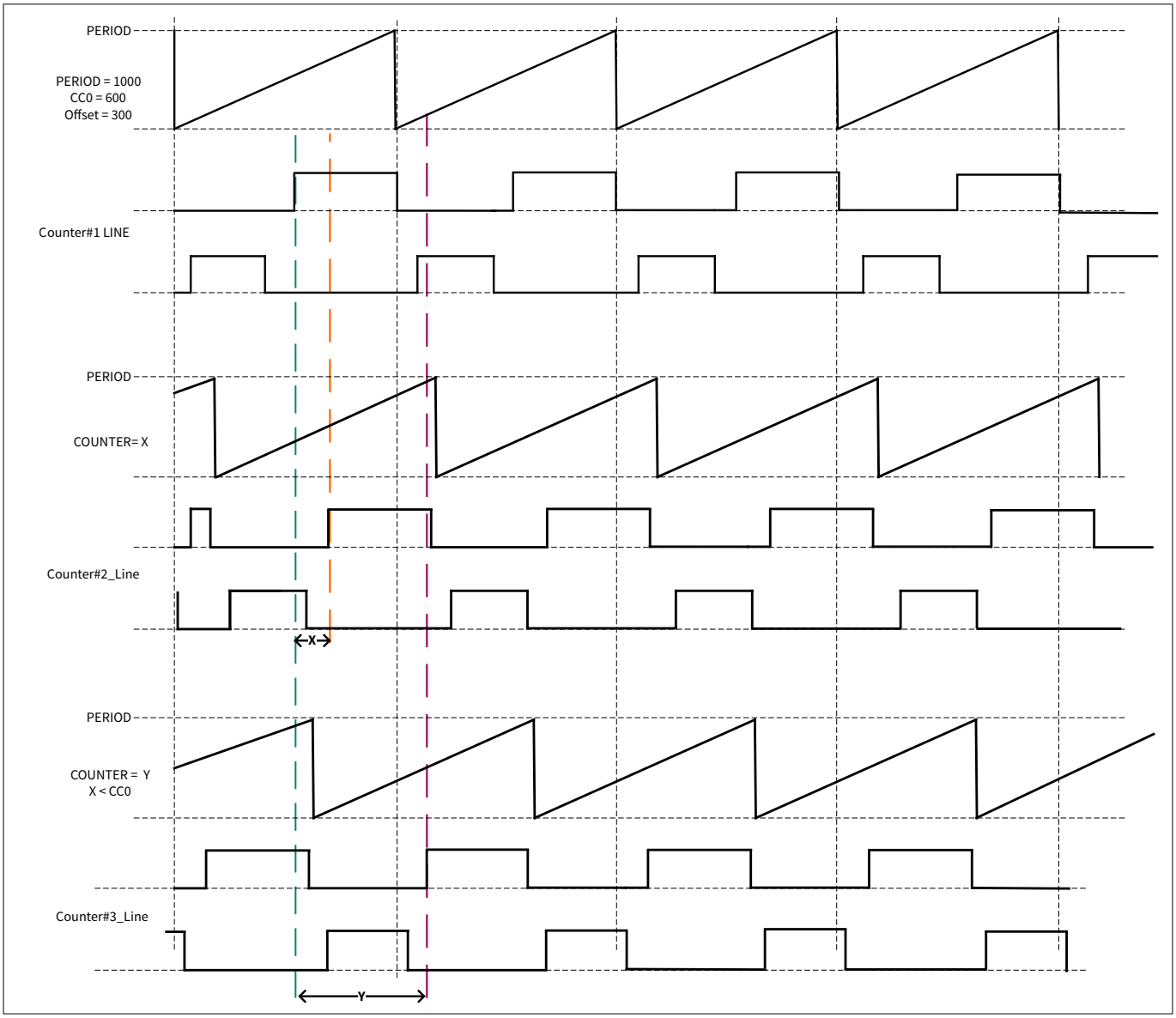


Figure 398 PWM output shift using counter offset value

33.3.3.12 External event based mode

This mode supports an external trigger-based duty cycle control. In this mode, the duty cycle shall not be given by CC0 or CC1 match. This mode supports to the serve the need of special handling in LLC topologies where the external trigger controls the duty and the period of the HRPWM signal. Period is twice the duty. Dead time insertion shall work as expected.

The external trigger signal can be a hardware-generated signal or generated by firmware. A hardware-based duty generation should be maintained, linked to the slope of the counter in dual slope (UPDN1/2) mode with or

without HRPWM enabled, while the output period is defined by the defined PERIOD Register value or an External Trigger event. Duty cycle of output is based on the positive slope of the counter in UPDN1/2 mode.

This mode supports with the following settings:

- PWM and PWM_DT mode
- Dual Slope Mode (UPDN1/2 mode)
- With or without HRPWM mode enabled
- Duty information is always dependent on period irrespective of whether period is generated by external event or by PERIOD (PI/PF) register values. CC0 and CC1 Registers are not used in this case

Enable the event-based mode by setting the DC_EN bit of CTRL2 register. Set the DC_SEL bits of the TR_IN_SEL1 register to select the external trigger event signal. In case no trigger comes, period and duty are given by programmed value in PERIOD and CC0/CC1 values (Time-based mode).

In UPDN2 mode, tc event is also generated along with an external event during overflow. However, shadow transfer only happens at underflow event for both UPDN1 and UPDN2 modes.

The firmware handling is similar to the other modes already defined in this register. Debug Freeze (CTRL Register Bit[30]) DBG_FREEZE_EN" has higher priority than the External Trigger Event (in EVENT BASED MODE).

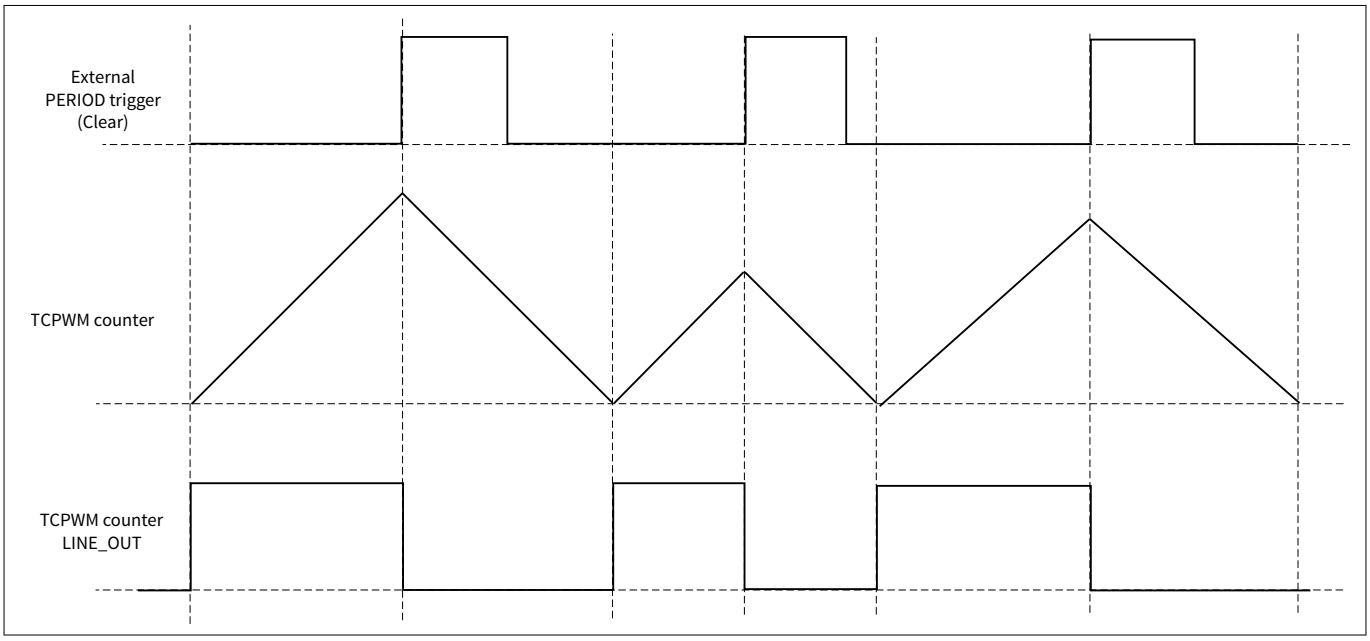


Figure 399 External trigger mode-based duty cycle and period control

Time-based mode

In dual slope mode (UPDN1/2 mode) the high-resolution period is defined by the programmed PERIOD register settings for PI (Period integer) and PF (Period fractional).

External event-based mode

In the external event-based mode, the period restart happens based on the asynchronous event/trigger input.

The external trigger event is only effective during the rising slope of the counter, it is ignored during the down slope of the counter.

In case no trigger comes, period and duty are given by programmed value in PERIOD and CC0/CC1 values.

At the occurrence of the event, the fractional period computation update shall be frozen.

PWM output modes

The typical PWM output modes are listed in the below table:

Table 342 PWM output modes

PWM output mode	UP_DOWN_MODE	TR_PWM_CTL	Other setting	"line" Pulse width
Right align	COUNT_DOWN	UNDERFLOW=CLEAR CC0_MATCH=SET	N/A	CC0+1
Left align	COUNT_UP	OVERFLOW=SET CC0_MATCH=CLEAR	N/A	CC0
Center align	COUNT_UPDN1	OVERFLOW=SET UNDERFLOW=CLEAR CC0_MATCH=INVERT		2 x (PERIOD – CC0)
Time-based mode Center align without CC0 match	COUNT_UPDN1/2	OVERFLOW=CLEAR/SET UNDERFLOW=SET/CLEAR	CC0_MATCH_UP_EN = 0 CC0_MATCH_DOWN_EN = 0 CC1_MATCH_UP_EN = 0 CC1_MATCH_DOWN_EN = 0	PERIOD
External event mode	COUNT_UPDN1/2	OVERFLOW=CLEAR/SET UNDERFLOW=SET/CLEAR	DC_EN=1 CC0_MATCH_UP_EN = 0 CC0_MATCH_DOWN_EN = 0 CC1_MATCH_UP_EN = 0 CC1_MATCH_DOWN_EN = 0	Defined by external event or PERIOD
Asymmetric with cc0	COUNT_UPDN2	OVERFLOW=SET UNDERFLOW=CLEAR CC0_MATCH=INVERT	N/A	2 x PERIOD – CC0 – CC0_BUFF
Asymmetric with cc0 & cc1	COUNT_DOWN COUNT_UP	CC0_MATCH=SET CC1_MATCH=CLEAR OVERFLOW = CLEAR (UP) UNDERFLOW = CLEAR (DOWN)	N/A	CC1-CC0
Center align asymmetric with cc0 & cc1	COUNT_UPDN1 COUNT_UPDN2	OVERFLOW=SET UNDERFLOW=CLEAR CC0_MATCH=SET CC1_MATCH=CLEAR	CC0_MATCH_UP_EN = 1 CC0_MATCH_DOWN_EN = 0 CC1_MATCH_UP_EN = 0 CC1_MATCH_DOWN_EN = 1	2 x PERIOD – CC0 – CC1 (No swap at overflow in UPDN2)

HRPWM generation for time-based and event-based mode

The following timing diagram shows examples of HRPWM timings with timing defined in time-based mode (PERIOD register) and event-based mode (External event-defined period).

- 1st HRPWM period: Time based with carry on PF (CY_PF = 1), one cycle added
- 2nd HRPWM period: External event defines HRPWM Period, no PF update
- 3rd HRPWM period: External event defines HRPWM Period, no PF update
- 4th HRPWM period: Time based with carry on PF (CY_PF = 1), one cycle added on count up, one cycle added on count Down (CY_PF = 1)
- 5th HRPWM period: Time based with carry on PF (CY_PF = 1), one cycle added

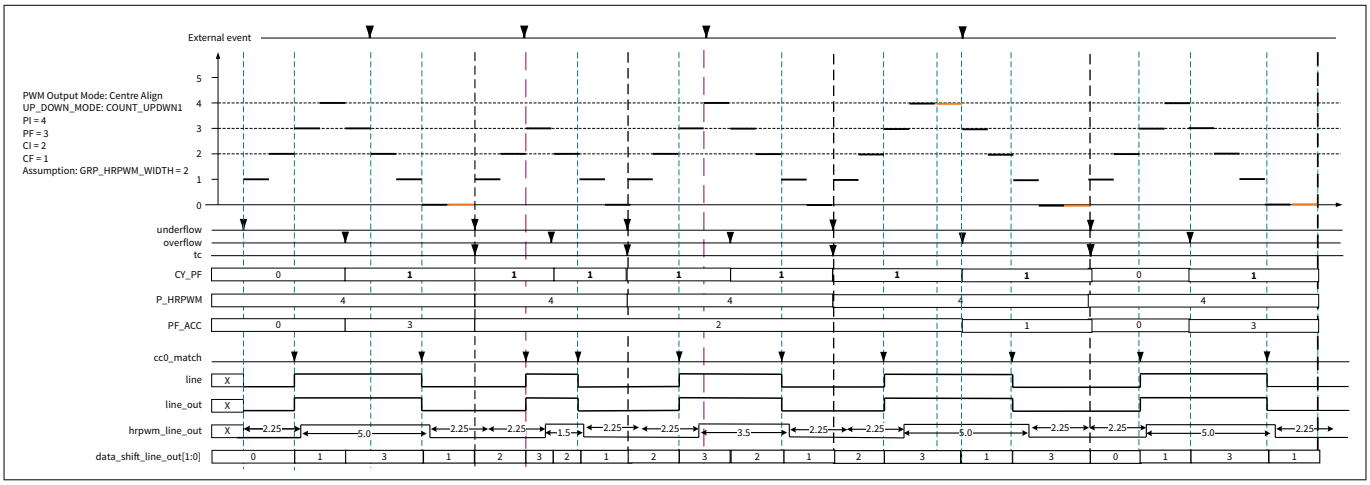


Figure 400 HRPWM generation for time-based and event-based mode

33.3.3.13 Masking of the PWM output

For power conversion topology, controlled by non-complimentary PWM outputs signals from two counters, it is required to avoid overlapping PWM signals (outputs = “1”) to avoid short circuit current flow.

Therefore a specific masking feature is controlled during run-time by SW/FW implemented to mask the PWM outputs. The mask feature can be enabled/disabled anytime while the counter is enabled or is running. The main functionality is to enable masking of the “line_out” signals in correlation with the mask input signals. Set the MASK_EN bit of CTL2 register to 1 to enable external mask mode.

The mask input signals are routed from external components (for example: EPU) to the TCPWM trigger inputs. The mask signals are connected to input bits of the existing trigger inputs. Selects one of the inputs triggers as mask input signal by configuring MASK_SEL bit-filed of TR_IN_SEL1 register. Configure the MASK_EDGE bits of the TR_IN_EDGE_SEL register to select the edge of the input trigger signal. Controls the masking functionality.

***Note**: In order not to delay the masking signal, the trigger signal used for masking should not use the edge detection it should use the trigger signal level as it is (High means Masking) by configuring MASK_EDGE bits of TR_IN_EDGE_SEL register to NO_EDGE_DET.*

The Register TR_OUT_SEL Bits [3:0] define the selection of the Trigger Output signals. The signals for the masking feature “LINE_OUT_INTERIM” and “LINE_COMPL_OUT_INTERIM” have been added to the selection. Those signals indicate the values of the line_out signals on the Trigger and can be used for masking on the corresponding counter used for a specific power conversion topology.

The following figure shows the diagram of masking feature.

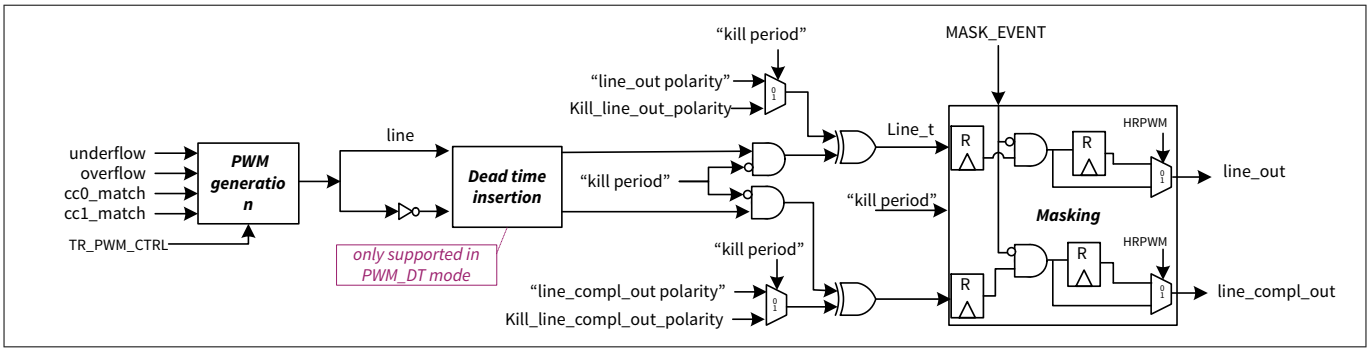


Figure 401 Masking feature diagram

- Only supports PWM and PWM_DT mode for all count modes including COUNT_UP, COUNT_DOWM, COUNT_UPDN1/2
- Mask functionality is applied on signals "line_out" and "line_compl_out"
- The masking feature introduces a delay that depends on the state of HRPWM. When HRPWM is enabled, the masking feature adds a one-cycle delay to both line_out and line_compl_out. This ensures that the outputs of the counters are synchronized to external mask. In contrast, when HRPWM is disabled, two cycle delay is added (an extra register to prevent glitches). This is register is not needed in HRPWM because the analog interpolator includes a flip-flop (FF), which removes glitches, making the extra register unnecessary
- Masking also supported with HRPWM enabled or disabled
- Not possible to get two line_out from two counters at the same time when masking is enabled
- Immediate KILL have higher priority than MASK feature on line_out/line_compl_out. Other kill modes would be masked normally
- When counters are operating at same frequency, only one clock cycle shall be spent to resynchronize an input trigger. The Trigger inputs will be used as Mask inputs and they have the option to use 1 clock synchronization or 2 Clock synchronization (TR_ONE_SYNC_BYPASS Bits [7:0] SYNC_BYPASS). The software should handle this accordingly
- After input mask is disabled, it takes one extra for the line_out to become high again. This delay has been added as a safety feature and particularly useful for compensating HRPWM applications (as line_out can be shifted)

The mask_event is applied directly when HRPWM is enabled, otherwise the mask_event goes through a register, which results in one cycle delay on when masking_event (tr_mask) is applied as shown in figure above.

If masking is disabled, then Masking logic bypass the masking (line_out = line t).

The following timing diagram shows the resulting timing of the line_out signals when the masking event is applied.

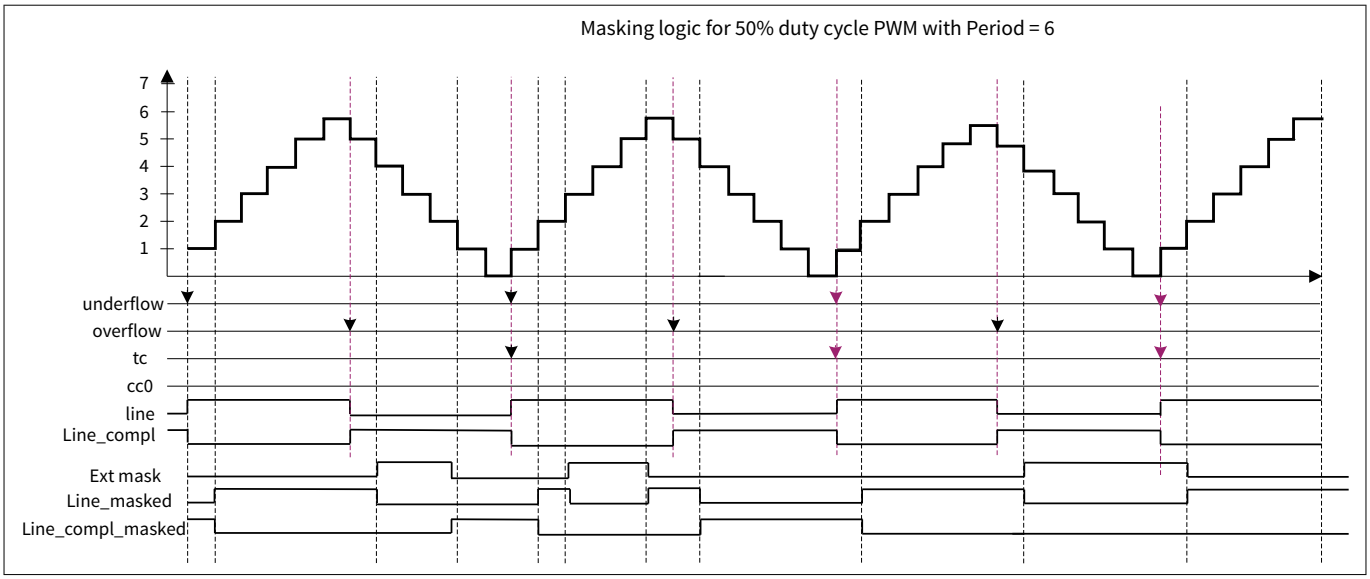


Figure 402 Timing-diagram masking of line_out

Decoupling of line_out using masking

line_outs from two different counters can be decoupled if tr_out0_sel is configured as 7 and tr_out is connected to tr_in for both counters. In this case, tr_out0 acts as an external mask for the other counter. This decoupling ensures that if one counter's line_out is already high, the other line_out will not become high under any condition. A particularly important scenario is when both line_outs have the same rising edge (as shown in the second pulse of the figure below). In this situation, both counters are set to low to avoid any overlap.

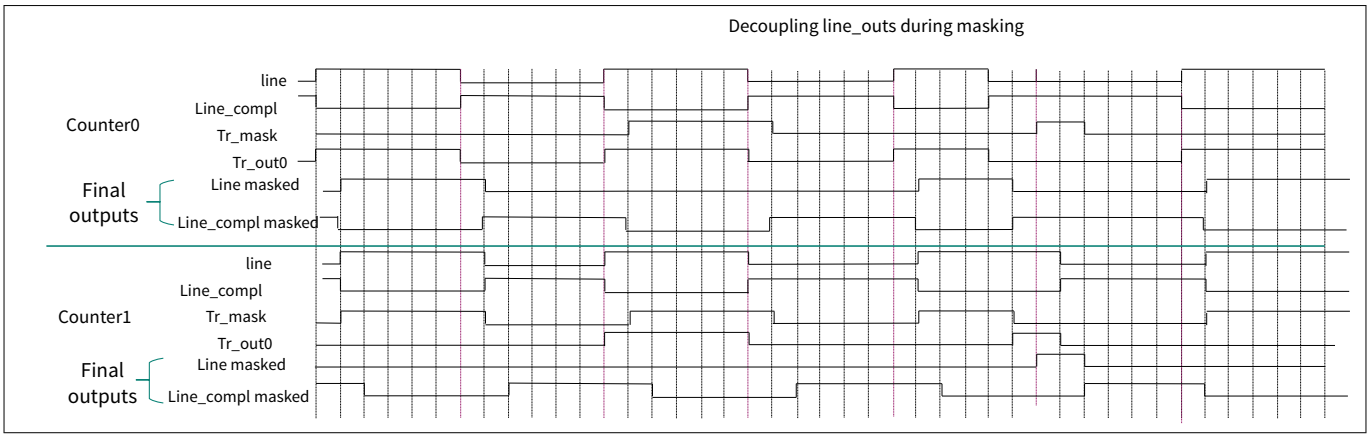


Figure 403 Decoupling line_out of two different counters (connected through tr_out0)

The following figure shows the state diagram for two counter line_outs (A and B), which are masked together to produce masked line_outs (A_m and B_m):

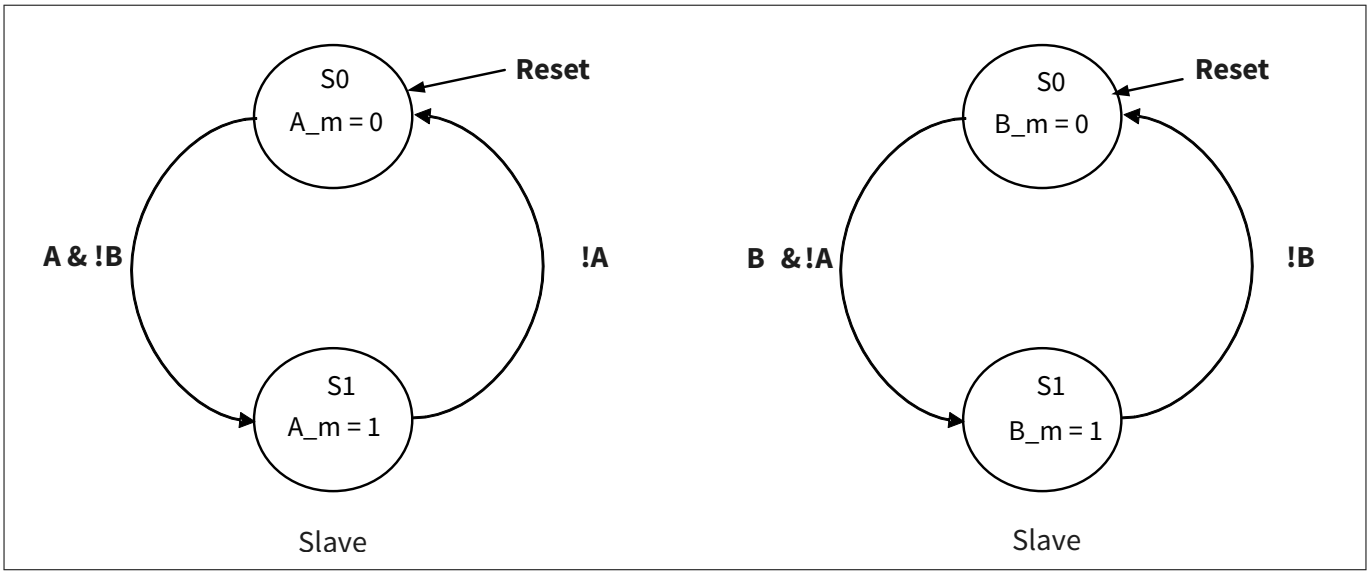


Figure 404 State diagram for masking of two counters

33.3.3.14 How to configure 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 the required 16-bit/32-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/32-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 the ENABLED bit of the CTRL register. A reload trigger must be provided through firmware (RELOAD bit in the PERI0_TR_CMD register) to start the counter if the hardware reload signal is not enabled

33.3.4 PWM with dead time (PWM-DT) mode

The PWM_DT functionality is the same as the PWM functionality, except for the following differences:

- PWM_DT supports dead time insertion. PWM does NOT support dead time insertion

Dead time insertion is a step that operates on a preliminary PWM output signal “line”.

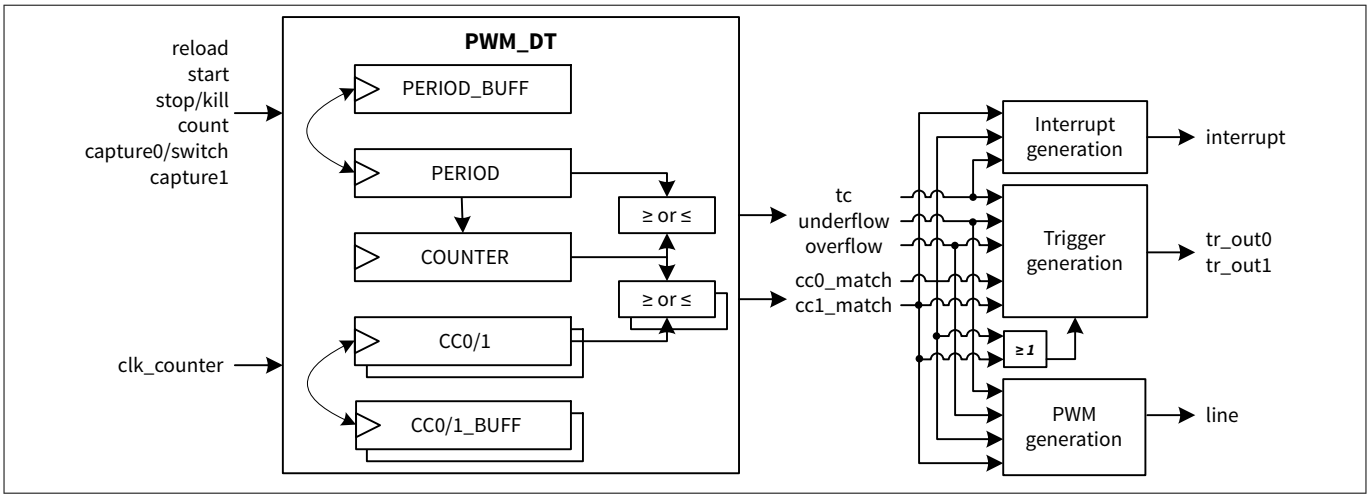


Figure 405 PWM with dead time functionality

33.3.4.1 Functionality details

The dead time generation depends on whether an advanced motor control (AMC) feature (GRP_AMC_PRESENT) is supported by the counter group.

- When the counter group does not support the advanced motor control feature: GRP_AMC_PRESENT = 0
  - The dead time is an 8-bit value (max. dead time is 255 counter cycles)
  - The same dead time is used before activating both PWM output signals “line_out” and “line_compl_out”. It is specified by the 8-bit register field DT_LINE_OUT_L
- When the counter group supports the advanced motor control feature: GRP_AMC_PRESENT = 1
  - The dead time is a 16-bit value (max. dead time is 65535 counter cycles)
  - Different dead times can be used before activating PWM output signals “line_out” and “line_compl_out”. The dead time for “line_out” is specified by the 16-bit register field DT_LINE_OUT (consists of DT_LINE_OUT_H and DT_LINE_OUT_L). The dead time for “line_compl_out” is specified by the 16-bit register field DT_LINE_COMPL_OUT

***Note**: To prevent undefined behavior, ensure that the dead time insertion (dead time value) does not exceed the period value.*

The following figure illustrates dead time insertion for different dead times and different output signal polarity settings when GRP_AMC_PRESENT = 0.

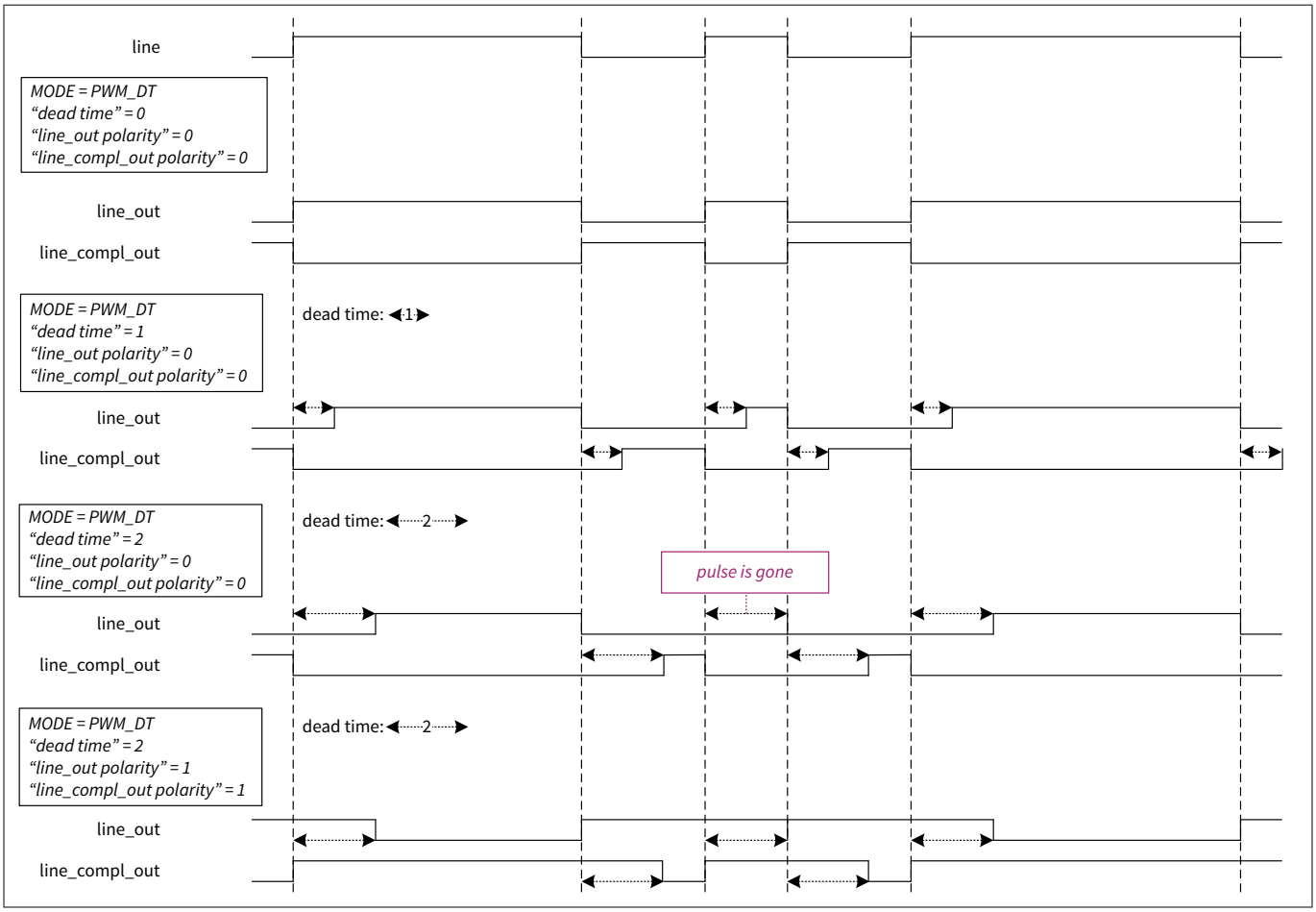


Figure 406 “line_out” and “line_compl_out” with unique dead time

The following figure illustrates how the polarity settings and stop/kill functionality combined control the PWM output signals “line_out” and “line_compl_out”.

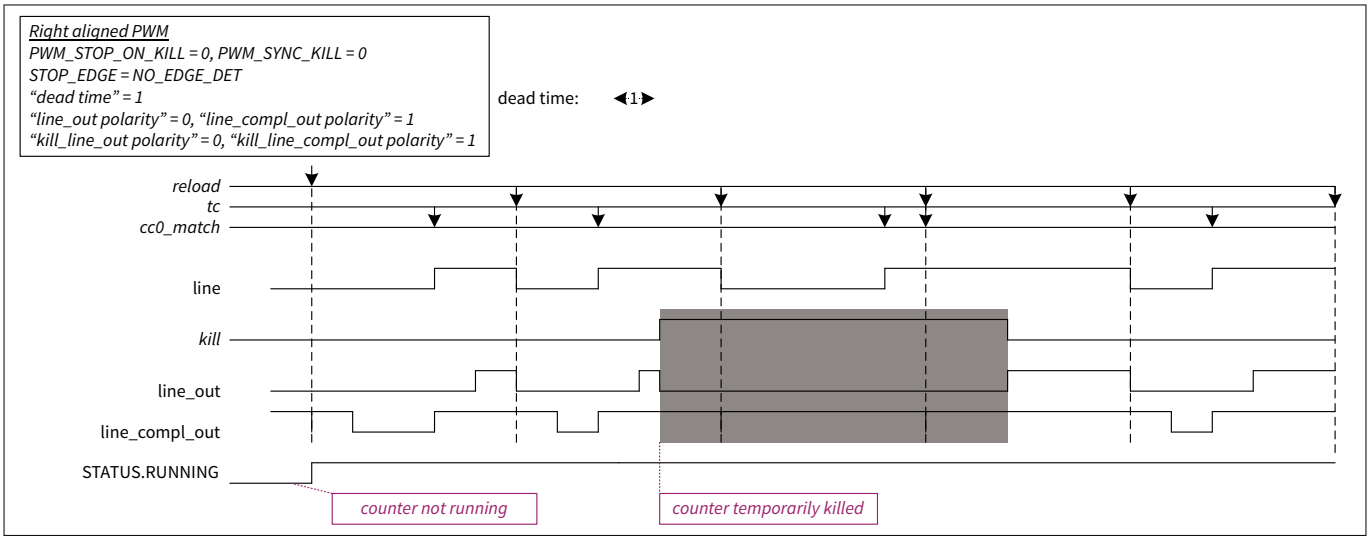


Figure 407 Dead time, kill input, and polarity function

The following figure shows an example with different dead times for the PWM output signals “line_out” and “line_compl_out” when parameter GRP_AMC_PRESENT = 1.

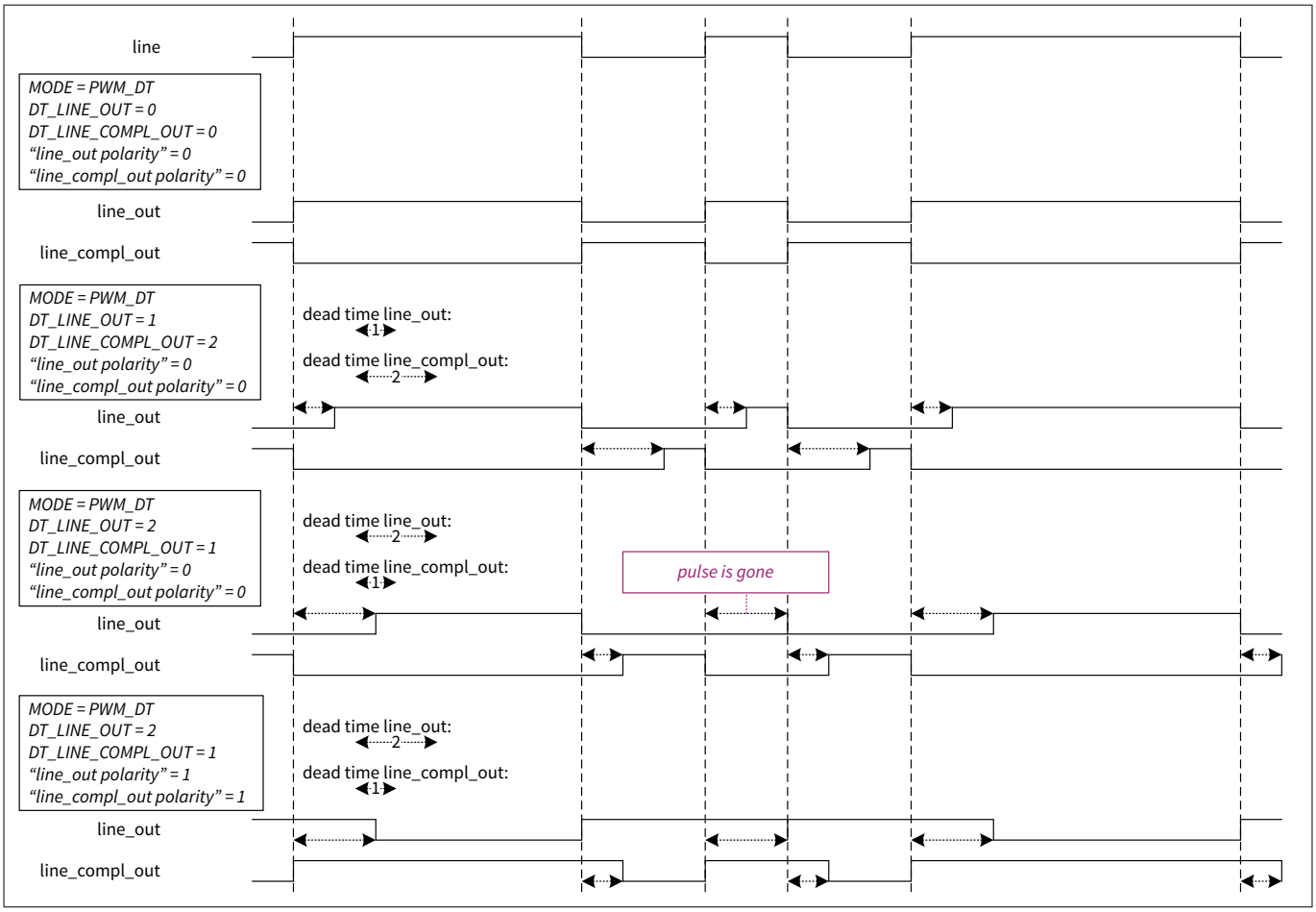


Figure 408 “line_out” and “line_compl_out” with different dead time

The following figures show an example driving the two coils of a stepper motor with two TCPWM counters when CTRL.SWAP_ENABLE = “0”. It illustrates how the software can update the register for the stepper motor control to incorporate dead times for the PWM output signals “line_out” and “line_compl_out”. Software can update the dead time in the DT_BUFF register and it will transfer to the DT register at the tc event. For COUNT_UPDN2 mode, the shadow transfer happens at the period match where TC and underflow coincides.

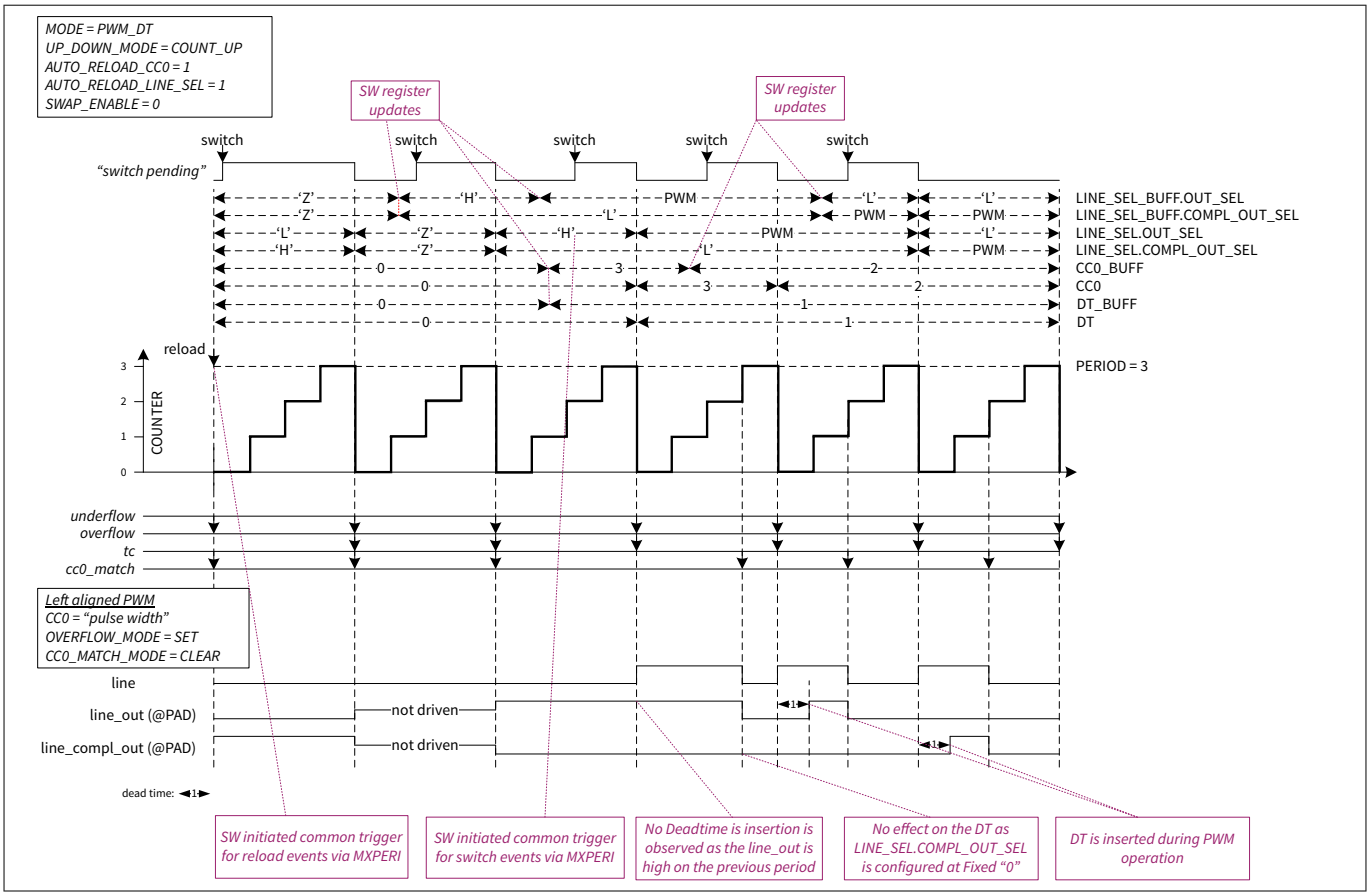


Figure 409 PWM_DT output case 1 scenario (SWAP_ENABLE=0 case)

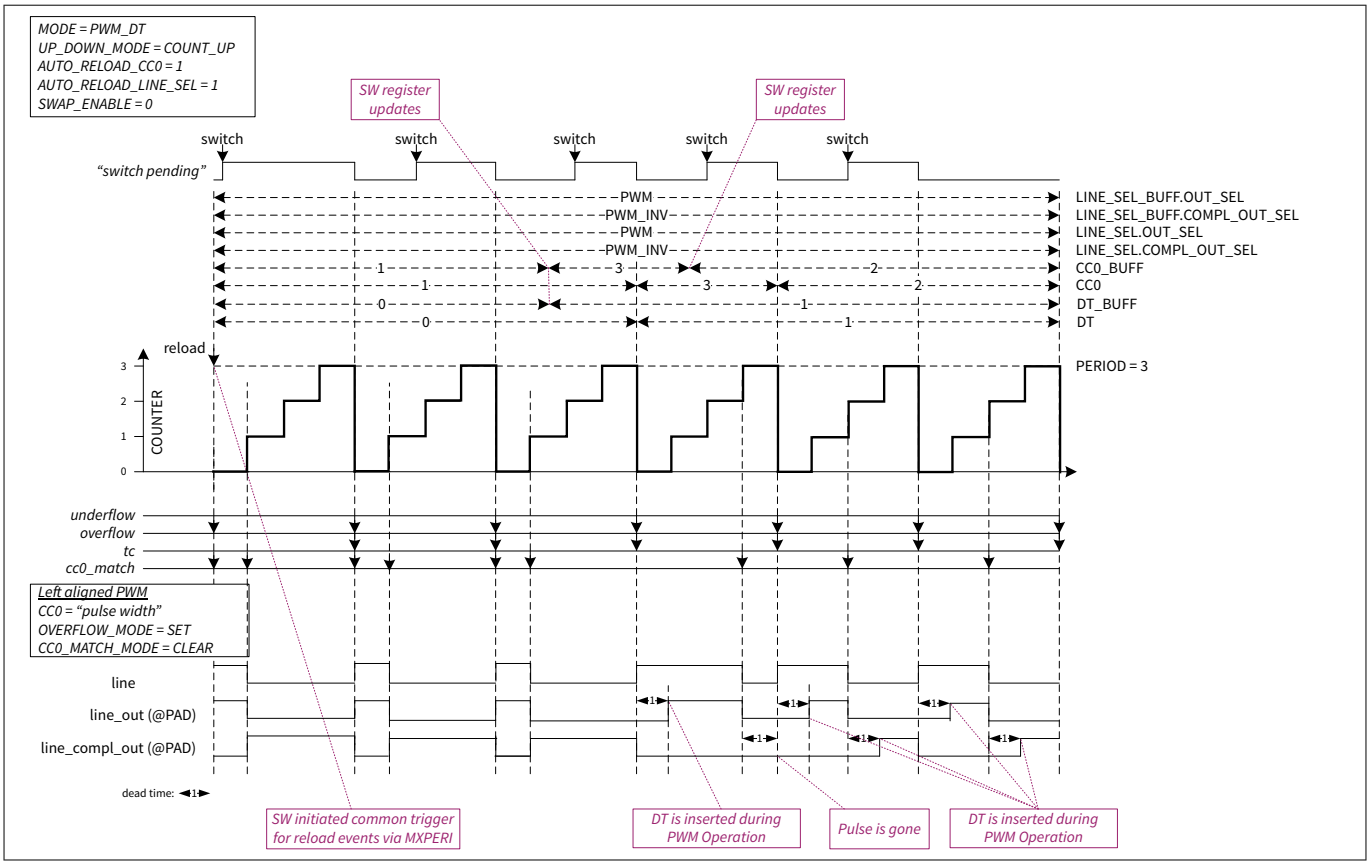


Figure 410 PWM_DT output case 2 scenario (SWAP_ENABLE=0 case)

33.3.4.2 How to configure counter for PWM_DT mode

The steps to configure the counter for PWM with Dead Time 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 with Dead Time mode by writing '101' to the MODE[26:24] field of the CTRL register
- Set the required dead time by writing to the DT_LINE_OUT_L [7:0]/DT_LINE_OUT_H [15:8], and DT_LINE_COMPL_OUT[31:16] fields of the DT register
- Set the required 16-bit/32-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/32-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, PWM_SYNC_KILL, PWM_TC_SYNC_KILL and PWM_SYNC_KILL_DT 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 the ENABLED bit of the CTRL register. A reload trigger must be provided through firmware (RELOAD bit in the PERI0_TR_CMD register) to start the counter if the hardware reload signal is not enabled

33.4 High-resolution PWM (HRPWM)

This feature is available when counter group parameter GRP_HRPWM_PRESENT=1.

33.4.1 Overview

The high-resolution PWM (HRPWM) generation feature is supported in PWM and PWM_DT mode only. The HRPWM enhancement allows higher resolution control for power conversion application.

33.4.1.1 Features

- Support HRPWM in PWM and PWM_DT mode only
- Support HRPWM feature in 32-bit counter
- Support period, duty cycle, and dead time with integer and factional features
  - Existing `PERIOD`, `PERIOD_BUFF`, `CC0`, `CC0_BUFF`, `CC1`, `CC1_BUFF`, `DT`, and `DT_BUFF` registers will be reused
  - PERIOD, CC0, CC1, DT registers are used for integer and fractional feature purposes
  - `PERIOD_BUFF`, `CC0_BUFF`, `CC1_BUFF`, `DT_BUFF` registers are used for shadow update purpose
- Support the fractional bits for delay adjustment. The number of fractional bits are defined by the parameter GRP_HRPWM_WIDTH
- Support the following PWM output modes: “right align, left align, center align, asymmetric with cc0, asymmetric with cc0 & cc1 and center align asymmetric with cc0 & cc1” for HRPWM feature
- When parameter `GRP_DATA_IN_CC_PRESENT`=1 and `HRPWM_CTRL.DATA_IN_CC0_EN='1'/ HRPWM_CTRL.DATA_IN_CC1_EN='1'`
  - `CC0/CC0_BUF` and `CC1/CC1_BUFF` registers cannot be updated over the AHB5 interface
  - `CC0_BUFF` and `CC1_BUFF` register are disabled
  - CC0 register is updated over the `DATA_IN_CC0` interface when there is valid write enable signal, `DATA_IN_CC0_WRITE_EN = 1`. The data update in CC0 register is done immediately, it does not need to wait for next `tc` or `cc0_match` event
  - CC1 register is updated over the `DATA_IN_CC1` interface when there is valid write enable signal, `DATA_IN_CC1_WRITE_EN = 1`. The data update in CC1 register is done immediately, it does not need to wait for next `tc` or `cc1_match` event
  - The following figure illustrates the data transfer from `data_in_cc0` to `cc0` register based on `HRPWM_CTRL.HRPWM_EN` and `data_in_cc0_write_en` signals. Please note that control is being transferred to AHB5 interface when `HRPWM_CTRL.DATA_IN_CC0_EN` is set to ‘0’. This means that data will not be transfer from `data_in_cc0` to `cc0` register even through `data_in_cc0_write_en` is set to ‘1’. This is behavior is same for cc1 register when `HRPWM_CTRL.DATA_IN_CC0_EN` is set to ‘0’

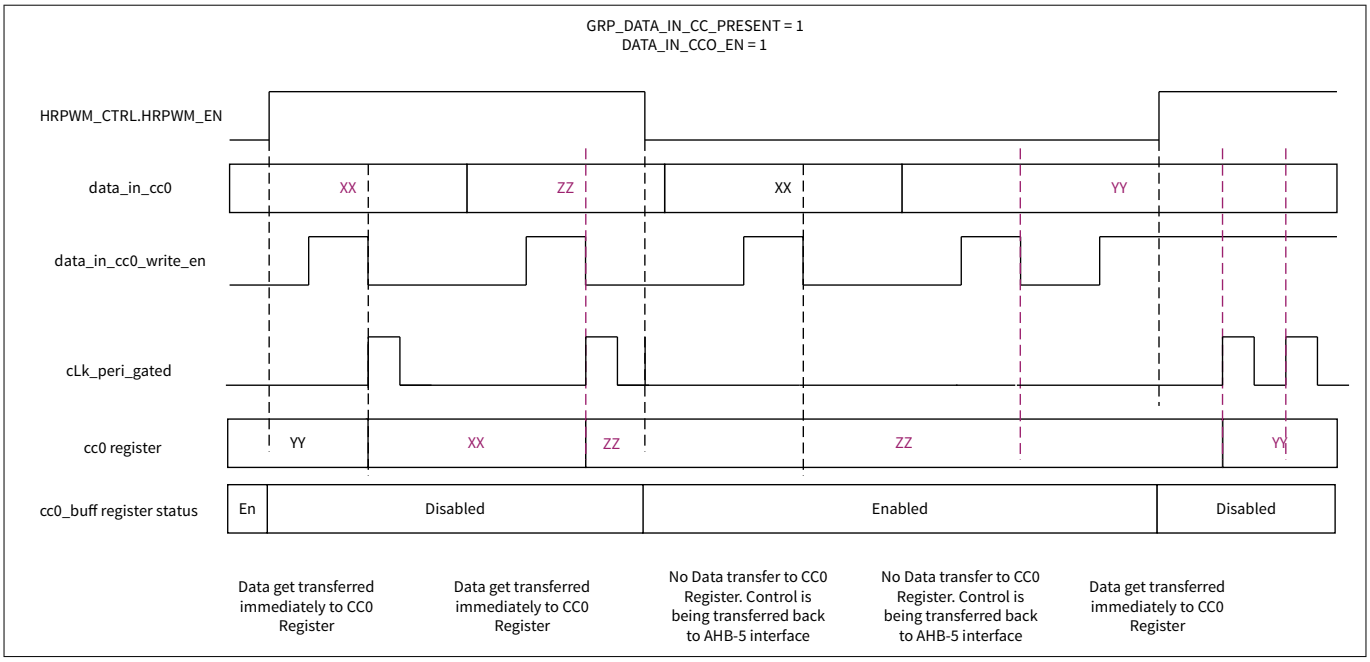


Figure 411 Data transfer from `data_in_cc0` to `cc0` under parallel data path support

***Notes**:*
For each TCPWM counter with HRPWM functionality; the correct order to disable the counter is:

1. *Disable the HRPWM function by setting `HRPWM_EN` ='0'*
2. *Disable the counter by setting ENABLED ='0'*

***Note**: Make sure PERIOD – CC0 >=3 when the HRPWM feature is enabled. otherwise, if the PERIOD - CC0 <= 2, no CC0_match event will get generated when HRPWM is enabled under center aligned PWM mode and UPDWN is enabled, The PWM waveform is abnormally output.*

33.4.1.2 Block diagram

The Figure 412 shows an example of the connection of the HRPWM for a counter group. It shows how the connection is connected to the delay element (analog interpolator) in the TCPWM.

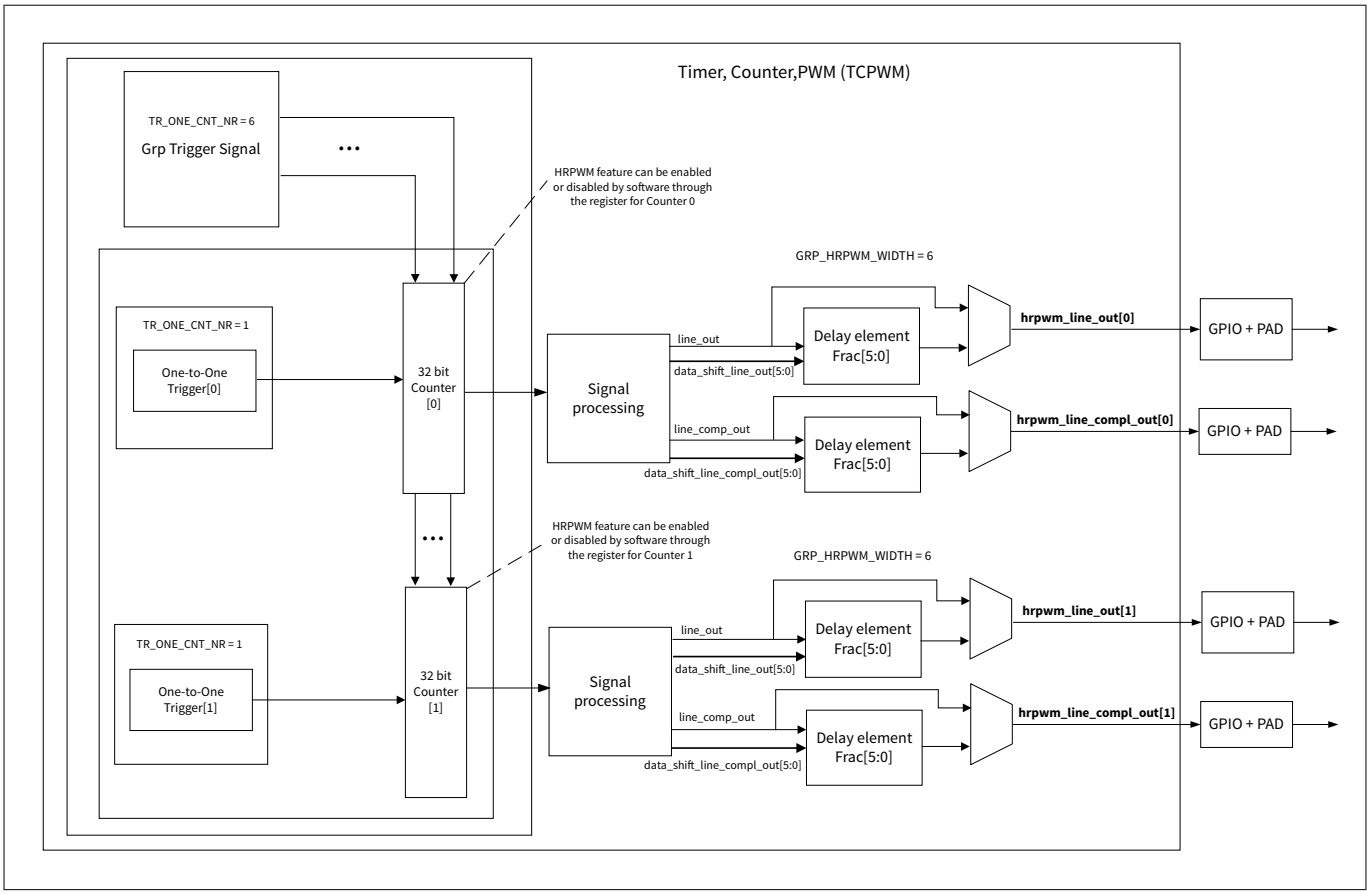


Figure 412 HRPWM block diagram

33.4.1.3 Clocks

Clocking has the following restrictions:

- When using the HRPWM feature with TCPWM, ensure that the clk_peri clock is not divided, as division is not allowed in this configuration
- To ensure proper operation, set the PS.PS_DIV to '0' when the HRPWM feature is enabled
- Configure the TCPWM to generate a clk_out signal for the delay element, which is used to generate micro ticks for shifting the edges of the PWM signal. Ensure that the clk_out signal has the same frequency as the counter clock

33.4.2 PWM

In the PWM mode, when the HRPWM feature is enabled by configuring register HRPWM_CTRL.HRPWM_EN to '1', the following are supported:

- The LSBs are used for fractional or micro ticks for delaying the line_out and line_compl_out outputs for the registers PERIOD/PERIOD_BUFF, CC0/CC0_BUFF, and CC1/CC1_BUFF and DT/DT_BUFF.

  Number of LSB bits used for fractional control is been controlled by the parameter GRP_HRPWM_WIDTH[5]. The remaining bits given by [GRP_CNT_WIDTH-1:GRP_HRPWM_WIDTH] are used by the counter.

  The following figure illustrated the integer and fractional partition of the register:

5 See Table 157 to check the number of micro ticks bits supported of counter group.

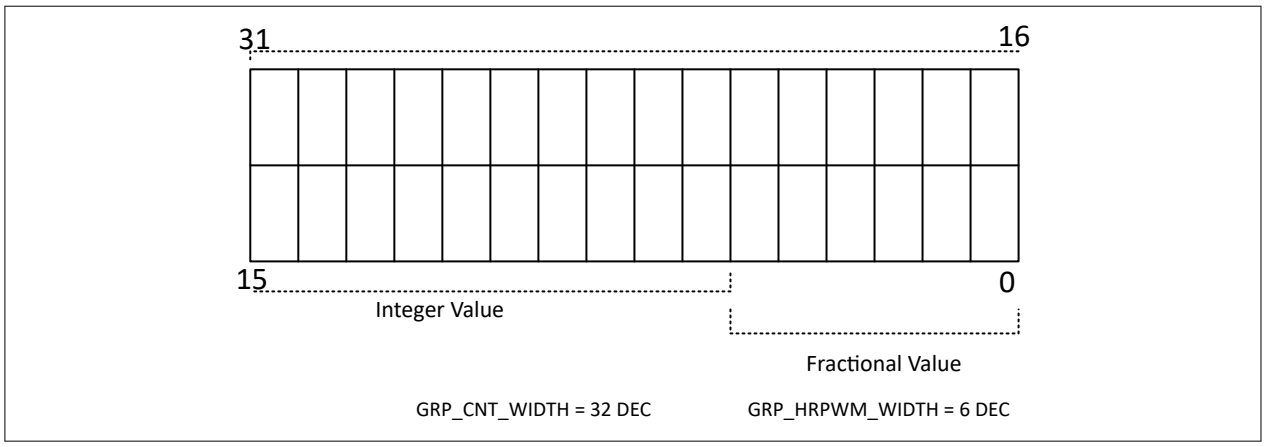


Figure 413 Example of register layout

- The maximum value for the integer part support will be $2^{[GRP_CNT_WIDTH-GRP_HRPWM_WIDTH]}-1$ if HRPWM_CTRL.HRPWM_EN is set to 1
- The maximum value for the fractional part support will be $2^{[GRP_HRPWM_WIDTH]}-1$ if HRPWM_CTRL.HRPWM_EN is set to 1
- If HRPWM_CTRL.HRPWM_EN is set to 0, the maximum value will be $2^{[GRP_CNT_WIDTH]}-1$

- The following figure illustrates how to program the integer and fractional value into the PERIOD register. Software will write 2374 DEC into the PERIOD register to set the 37 DEC for the integer and 6 DEC for the fractional value

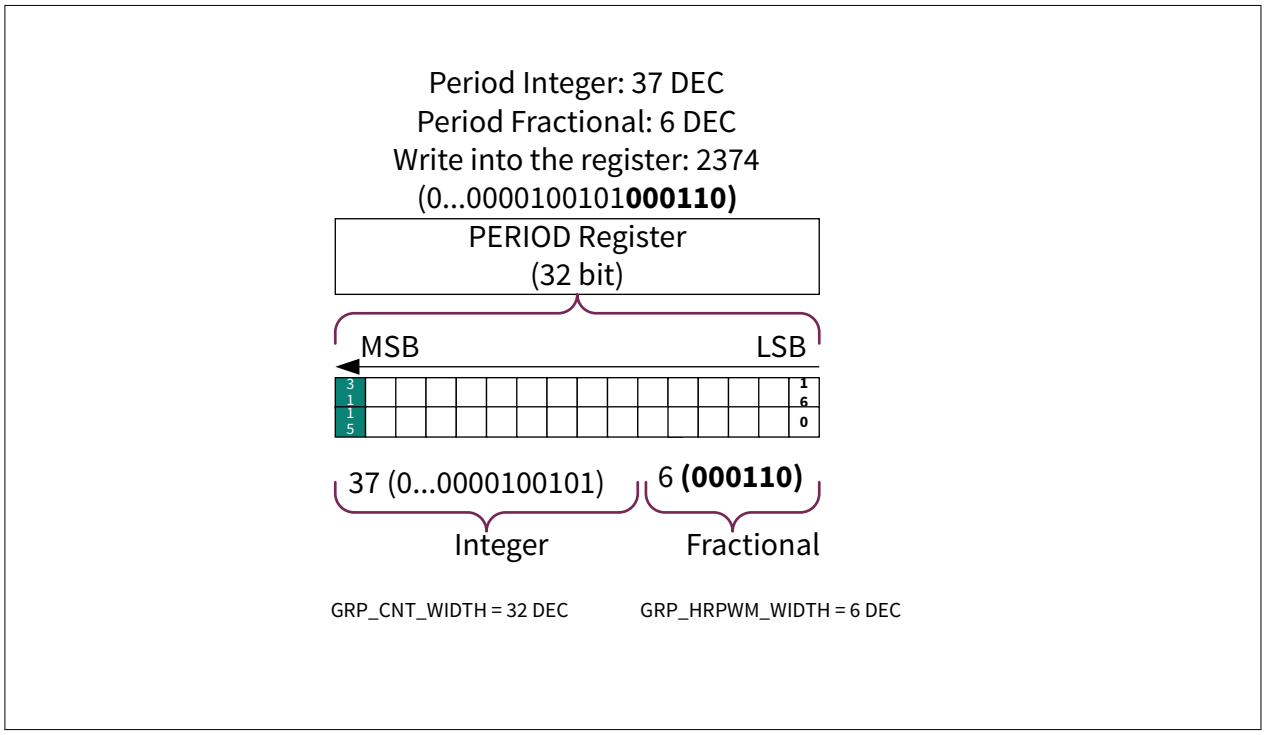


Figure 414 Example of programming integer and fractional values into the register

- The fractional value programmed in the registers is used to generate data outputs:
  - data_shift_line_out[GRP_HRPWM_WIDTH-1:0] – This is used by the delay element to shift the rising and falling edge of line_out by equivalent micro ticks. TCPWM generates a value in synchronous to the

tc and cc0/cc1 match event for the rising and falling edge of line_out to control the micro ticks the corresponding edge must be delayed

- data_shift_line_compl_out[GRP_HRPWM_WIDTH-1:0] – This is used by the delay element to shift the rising and falling edge of line_compl_out by equivalent micro ticks. TCPWM generates a value in synchronous to the tc and cc0/cc1 match event for the rising and falling edge of line_compl_out to control the micro ticks the corresponding edge must be delayed

***Note**: For each TCPWM counter with HRPWM functionality, there are two delay elements: one for delaying the rising and falling edges of line_out and another one for delaying the rising and falling edge of the line_compl_out*

33.4.2.1 Computation concept

Figure 415 and Figure 416 shows an example of how data_shift_line_out and data_shift_line_compl_out are calculated in PWM mode and sent out with line_out and line_compl_out to the delay element.

***Note**: A fractional delay is inserted by the delay element, resulting in hrpwm_line_out and hrpwm_line_compl_out out of the delay element.*

***Note**: The hrpwm_* signals shown below are just for clarification and not actual outputs from TCPWM. hrpwm_line_out and hrpwm_line_compl_out are signal names used to differentiate between TCPWM output and delay element output. In the final, there will be two output signals with the name line_out and line_compl_out.*

When the PERIOD register contains a fractional value >0, it is added into PF_ACC. When the PF_ACC value overflows and the carry is added into the integer value (PI) of the PERIOD register and the carry is cleared at the next TC event.

PF_ACC is also added fractional value configured into the CC0/CC1 register. When the added value CC0_ACC or CC1_ACC overflows as well, the carry is added to the corresponding CC0/CC1 register integer value for the interval of one TC event and then the carry is cleared.

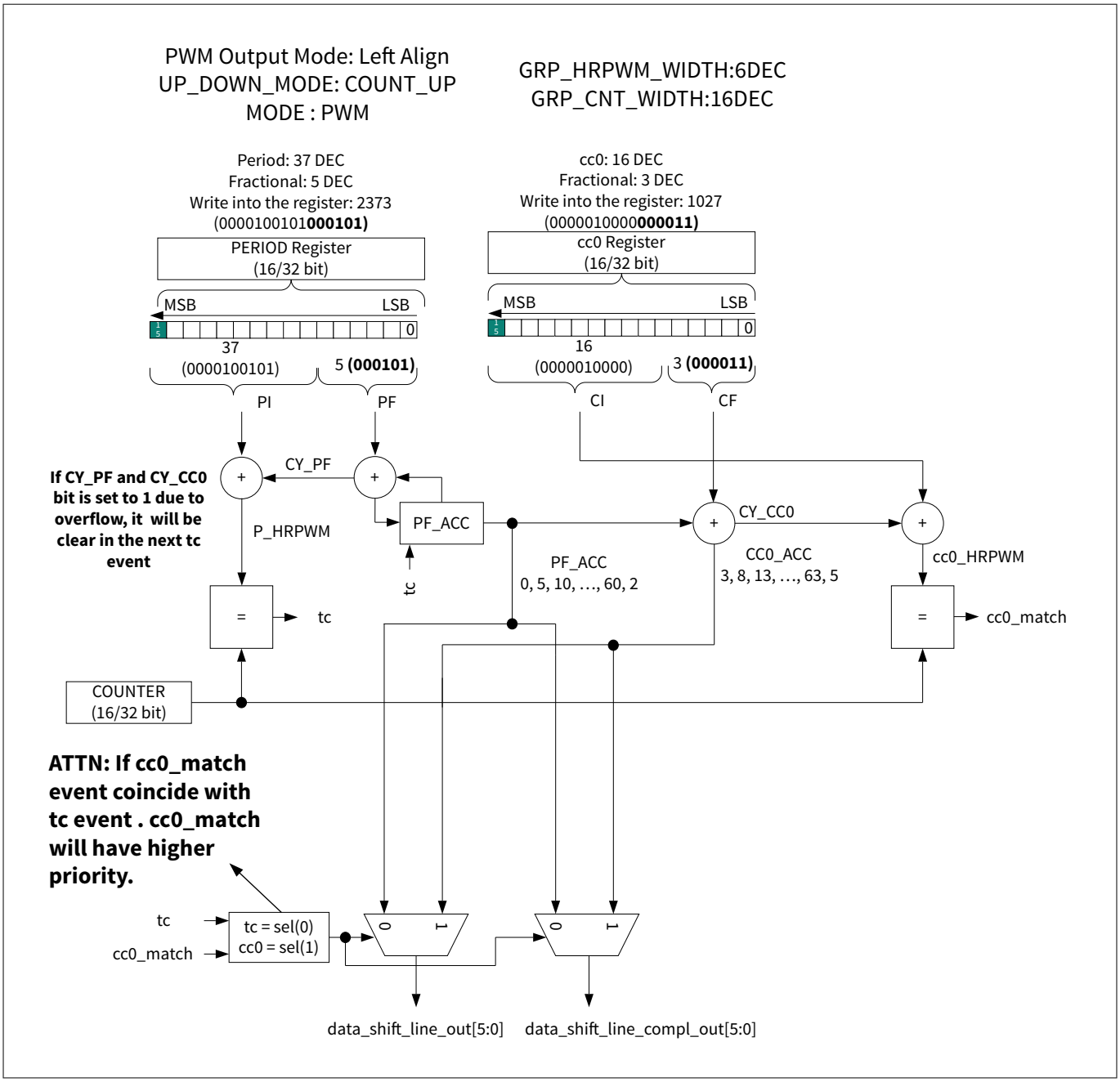


Figure 415 Calculation of data_shift_line_out and data_shift_line_compl_out in PWM mode

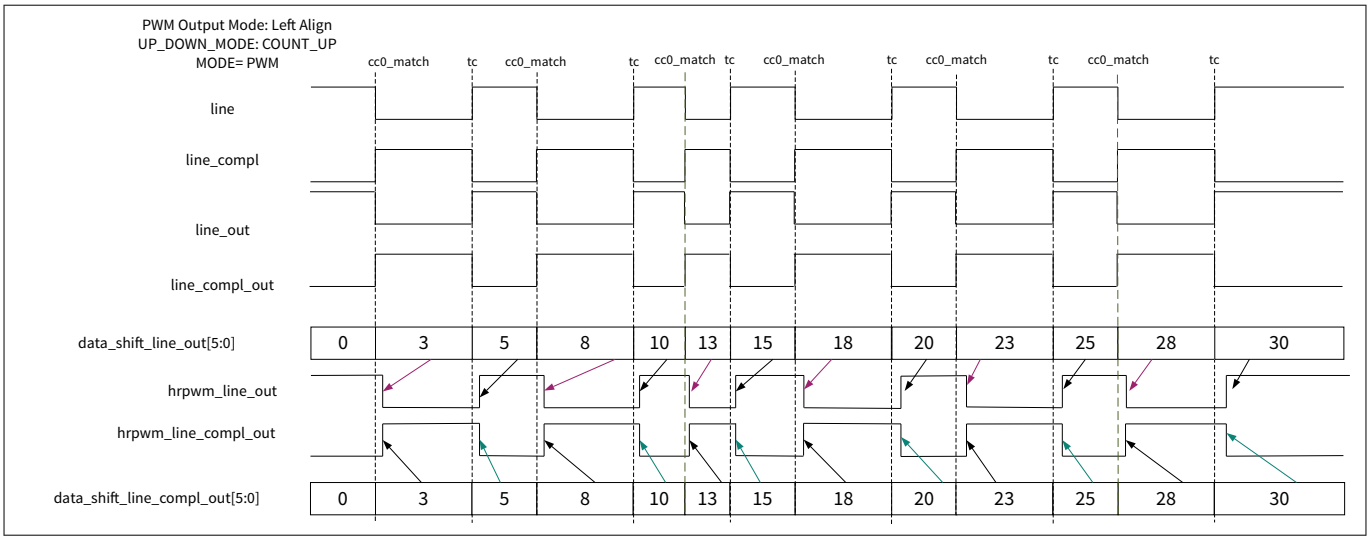


Figure 416 Example of TCPWM outputs in PWM mode

33.4.2.2 PWM output mode

The following output modes are supported in the HRPWM.

Table 343 PWM output mode in HRPWM

PWM output mode	UP_DOWN_MODE	TR_PWM_CRL	Other settings	"line" pulse width	Period
Right align	COUNT_DOWN	UNDERFLOW = CLEAR CC0_MATCH = SET	N/A	CC0+1	PERIOD + 1
Left align	COUNT_UP	OVERFLOW = SET CC0_MATCH = CLEAR	N/A	CC0	PERIOD + 1
Center align	COUNT_UPDN1	OVERFLOW = SET UNDERFLOW = CLEAR CC0_MATCH = INVERT	N/A	2xPERIOD – 2xCC0-1 (when CF=non-zero) 2xPERIOD – 2xCCO (when CF=zero)	PERIOD * 2
Center align asymmetrically with cc0	COUNT_UPDN2	OVERFLOW = SET UNDERFLOW = CLEAR CC0_MATCH = INVERT	N/A	2xPERIOD – CC0 – CC0_BUFF-1 (when CF=non-zero) 2xPERIOD – CC0 – CC0_BUFF (when CF=zero)	PERIOD * 2

(table continues...)

Table 343 (continued) PWM output mode in HRPWM

PWM output mode	UP_DOWN_MODE	TR_PWM_CRL	Other settings	"line" pulse width	Period
Asymmetric with cc0 & cc1	COUNT_DOWN COUNT_UP	CC0_MATCH = SET CC1_MATCH = CLEAR OVERFLOW = CLEAR (UP) UNDERFLOW = CLEAR (DOWN)	N/A	CC1-CC0	PERIOD + 1
Center align asymmetric with cc0 & cc1	COUNT_UPDN1 COUNT_UPDN2	OVERFLOW = SET UNDERFLOW = CLEAR CC0_MATCH = SET CC1_MATCH = CLEAR	CC0_MATCH_UP_EN = 1, CC0_MATCH_DOWN_EN = 0, CC1_MATCH_UP_EN = 0, CC1_MATCH_DOWN_EN = 1,	2xPERIOD – CC0 – CC1 – 1 (when CF=non-zero) 2xPERIOD – CC0 – CC1 (when CF=zero) (No swap at overflow in UPDN2)	PERIOD * 2

33.4.2.3 PWM output mode - Left align

The following diagrams illustrate a few examples with different CF setup of PWM output mode configured in left align with HRPWM feature enabled.

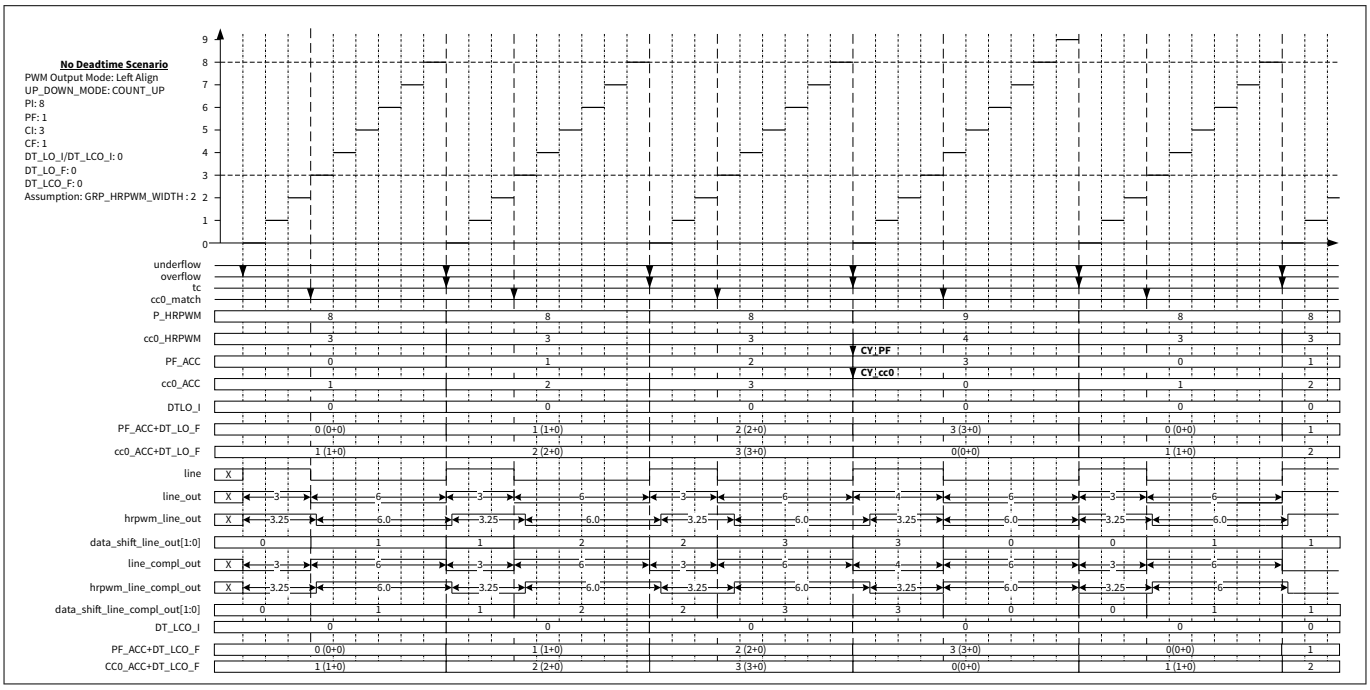


Figure 417 Example of left align in PWM mode with HRPWM feature enabled - Scenario 1 (CF=1)

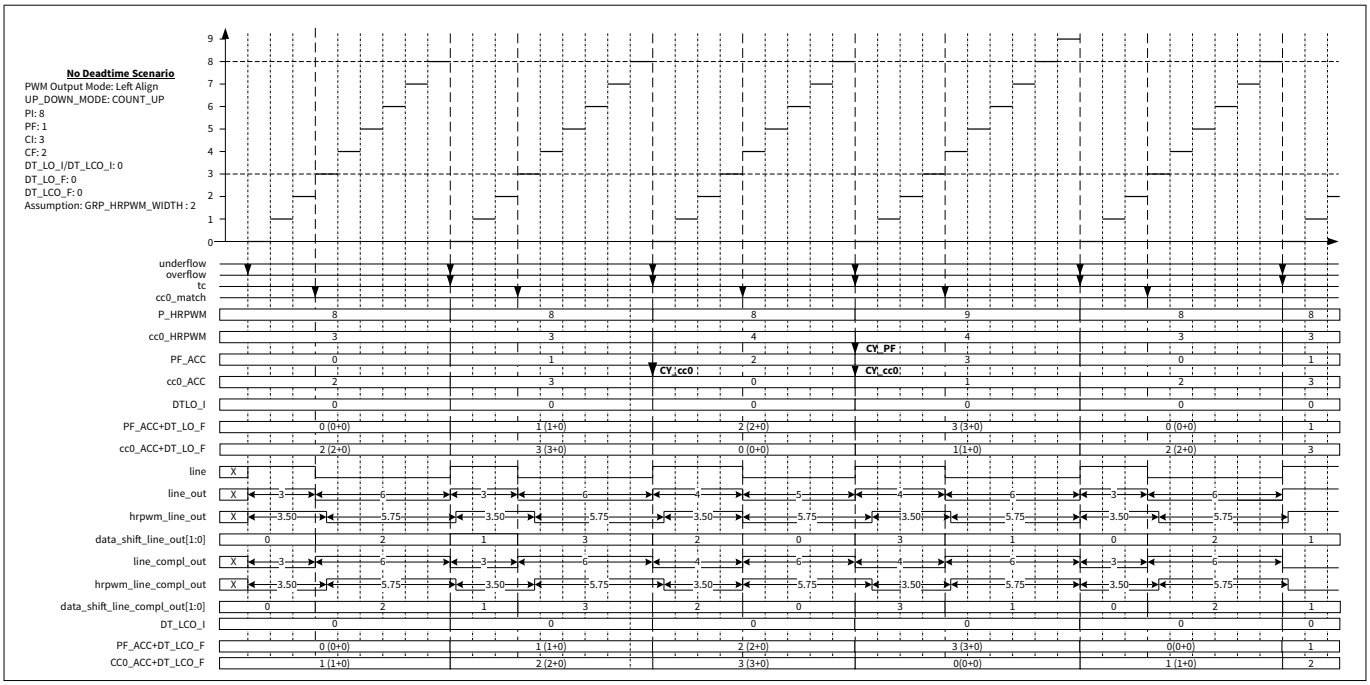


Figure 418 Example of left align in PWM mode with HRPWM feature enabled - Scenario 2 (CF=2)

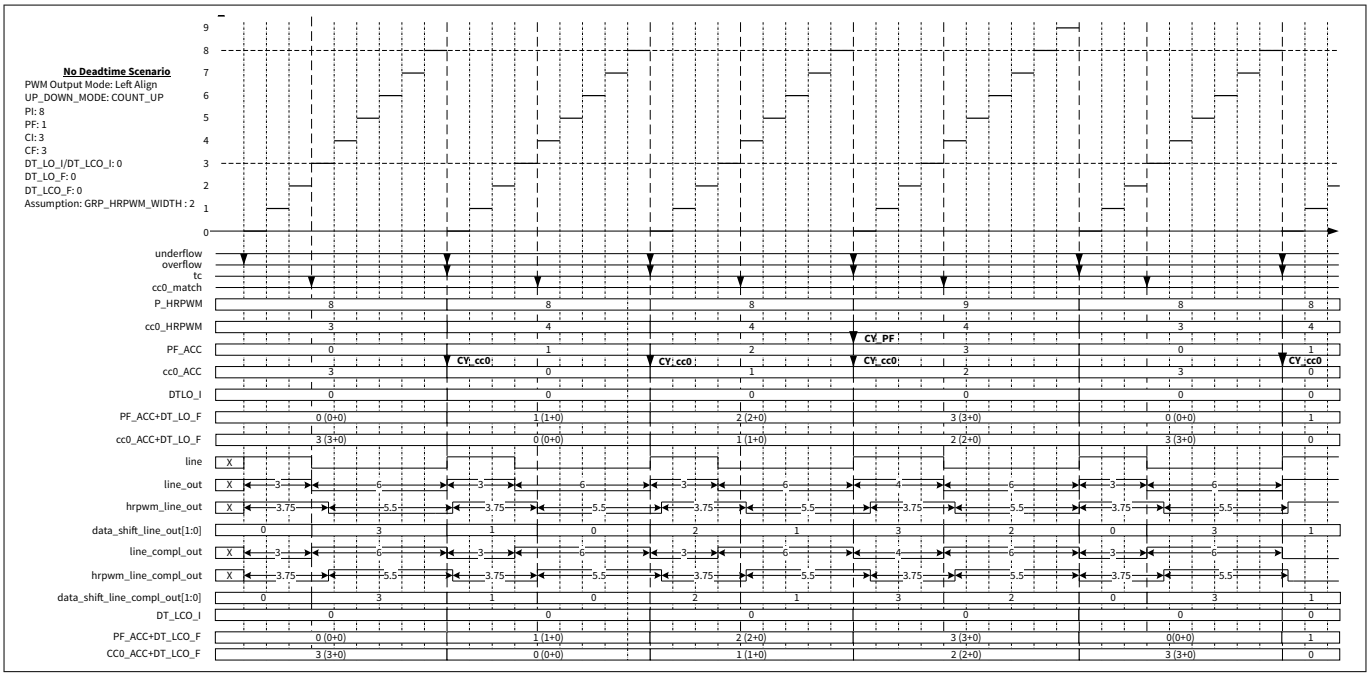


Figure 419 Example of left align in PWM mode with HRPWM feature enabled - Scenario 3 (CF=3)

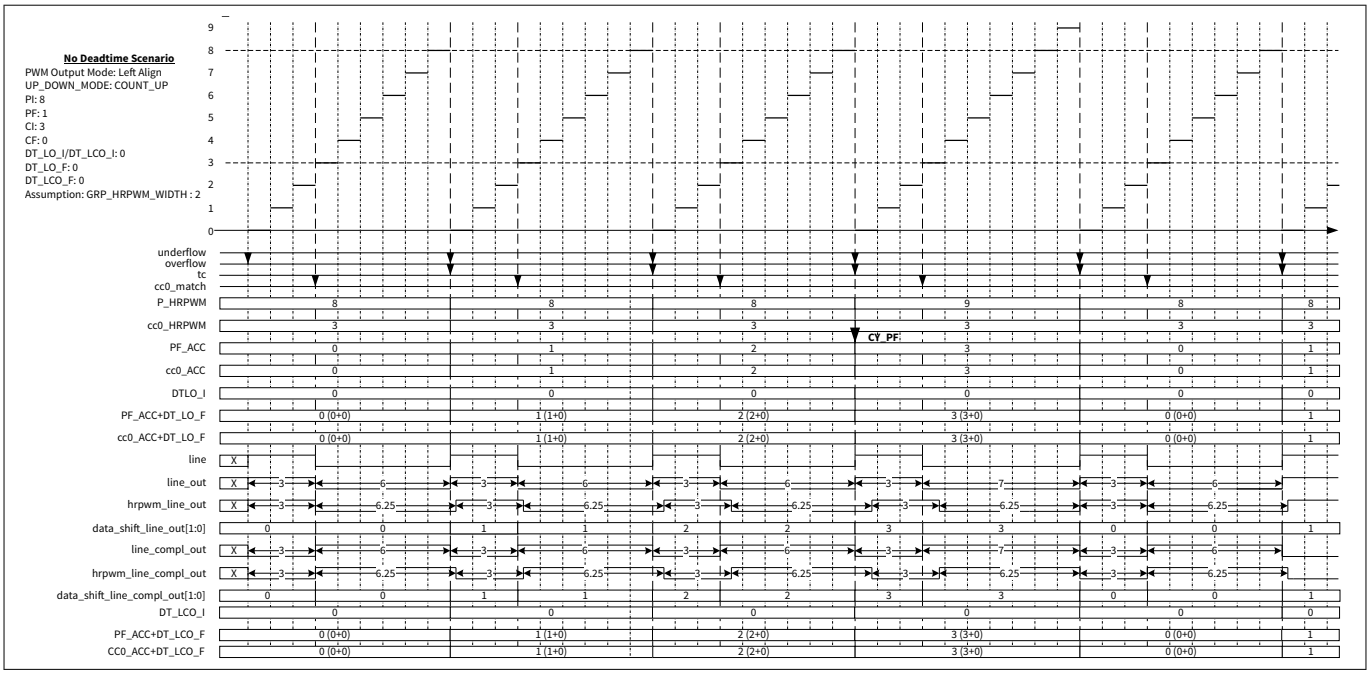


Figure 420 Example of left align in PWM mode with HRPWM feature enabled - Scenario 4 (CF=0)

33.4.2.4 PWM output mode - Right align

For the right align case, cc0_ACC and cc0_HRPWM computation are as followed:

- signed_cc0_ACC = PF_ACC + PF – CF, if signed_cc0_ACC is negative, cc0_ACC will be two complements
- msb_cc0_acc will be set to 1 when signed_cc0_ACC value is greater equal than 2^GRP_HRPWM_WIDTH
- sn_cc0_acc will be set to 1 when signed_cc0_ACC is negative
- cc0_hrpwm = CI + CY_PF + sn_cc0_acc – signed_cy_cc0, where signed_cy_cc0 = msb_cc0_acc XOR sn_cc0_acc

The following table shows a few examples of the computation for GRP_HRPWM_WIDTH set to 2:

Table 344 Examples of computation for right align PWM output mode

PF_ACC	PF	CF	CI	signed_cc0_ACC (PF_ACC+PF-CF)	msb_cc0_acc	sn_cc0_acc	signed_cy_cc0	cc0_hrpwm
3	3	1	5	5 (101B)	1	0	1	4 (CY_PF=0)
1	1	3	5	-1 (111B)	1	1	0	6 (CY_PF=0)
1	1	2	5	0 (000B)	0	0	0	5 (CY_PF=0)

The following diagram illustrates a few examples with different CF setup of PWM output mode configured in right align with HRPWM feature enabled.

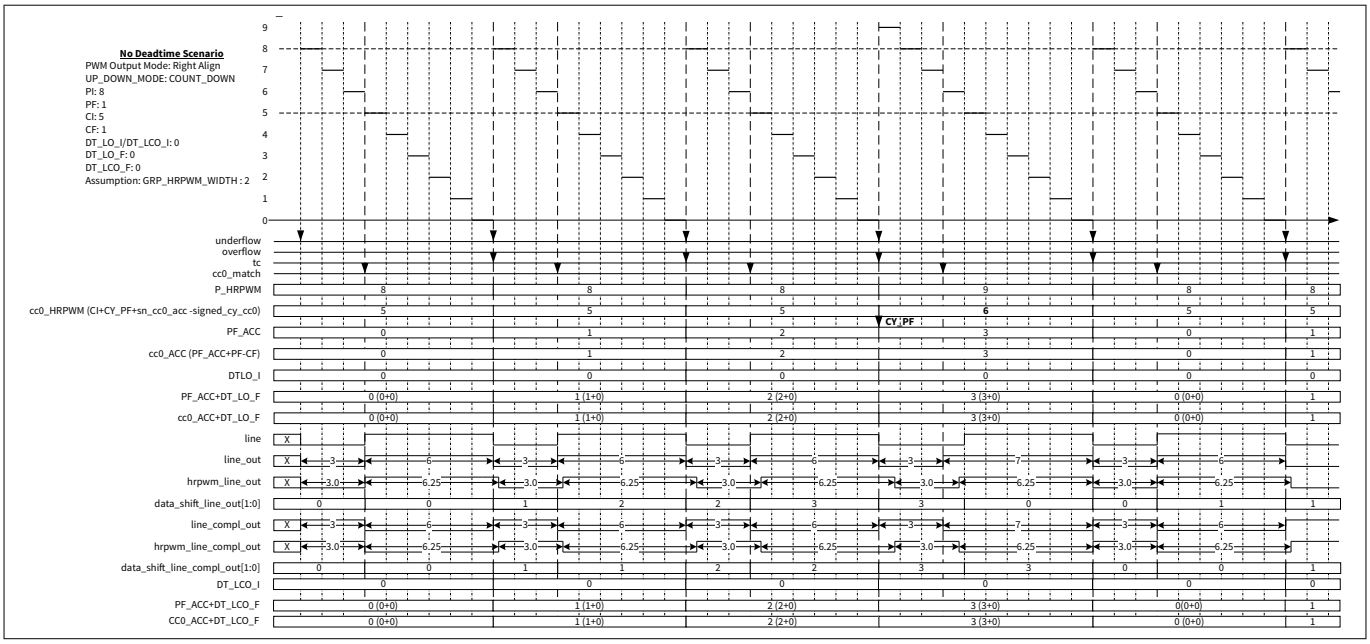


Figure 421 Example of right align in PWM mode with HRPWM feature enabled – Scenario 1 (CF=1)

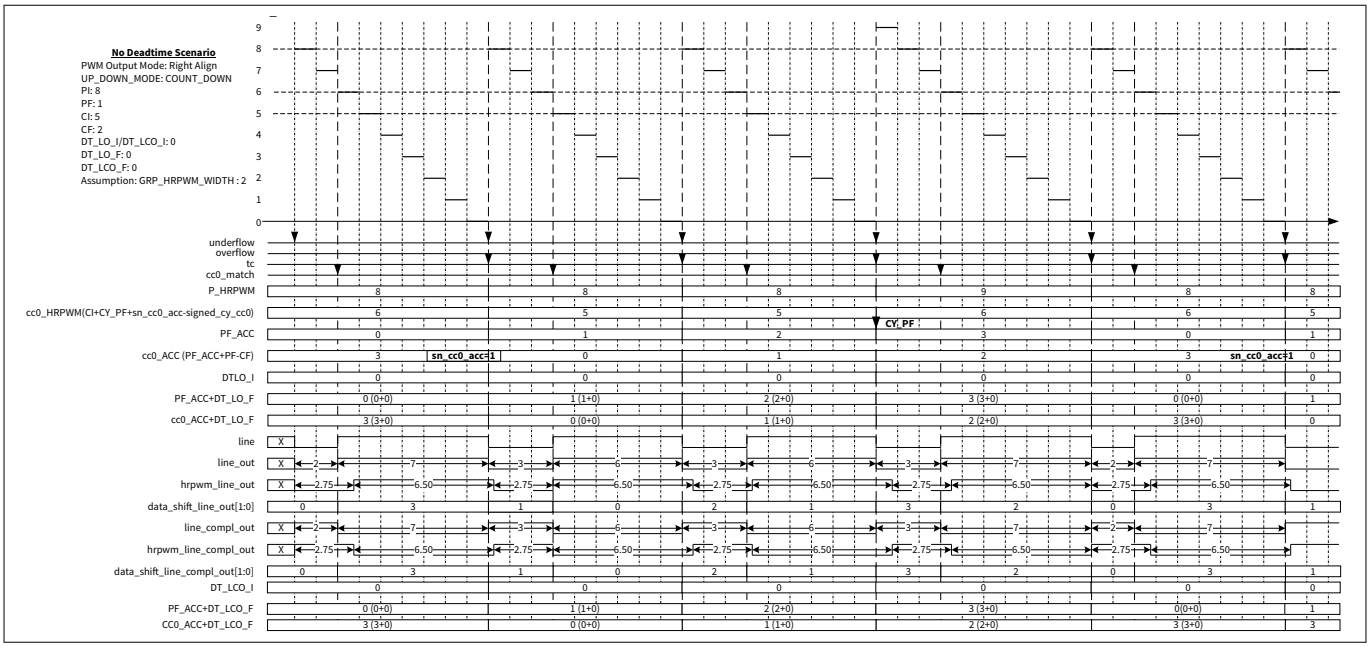


Figure 422 Example of right align in PWM mode with HRPWM feature enabled – Scenario 2 (CF=2)

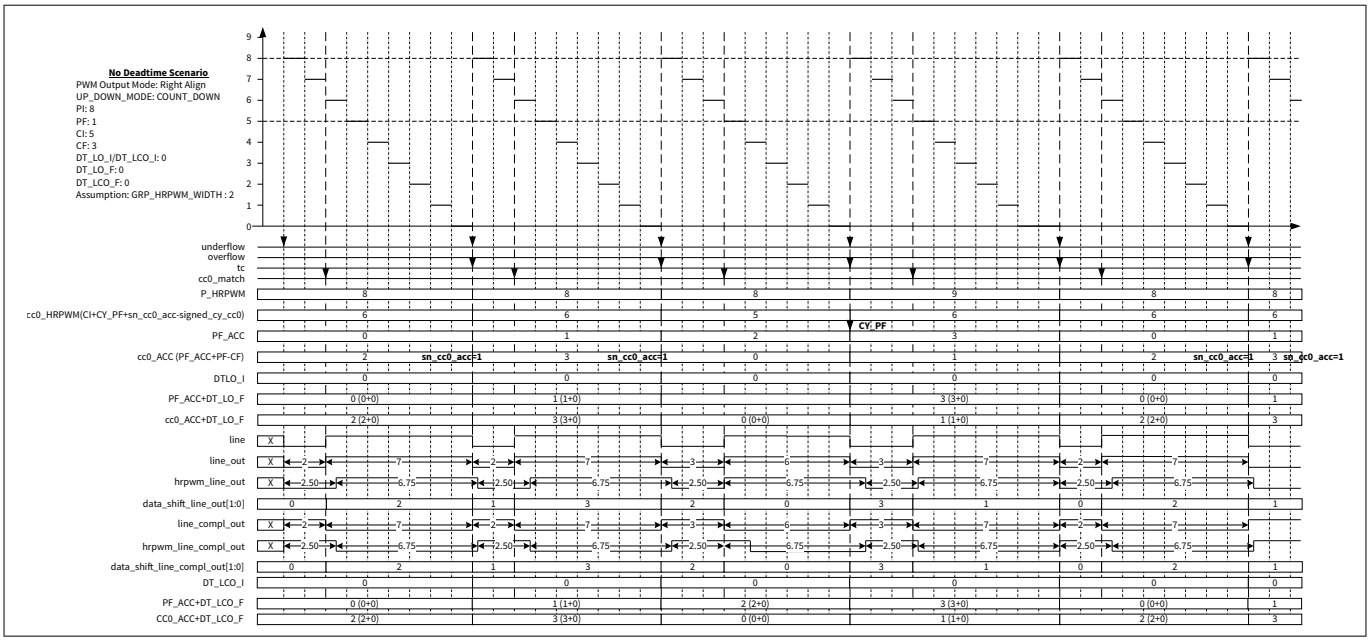


Figure 423 Example of right align in PWM mode with HRPWM feature enabled – Scenario 3 (CF=3)

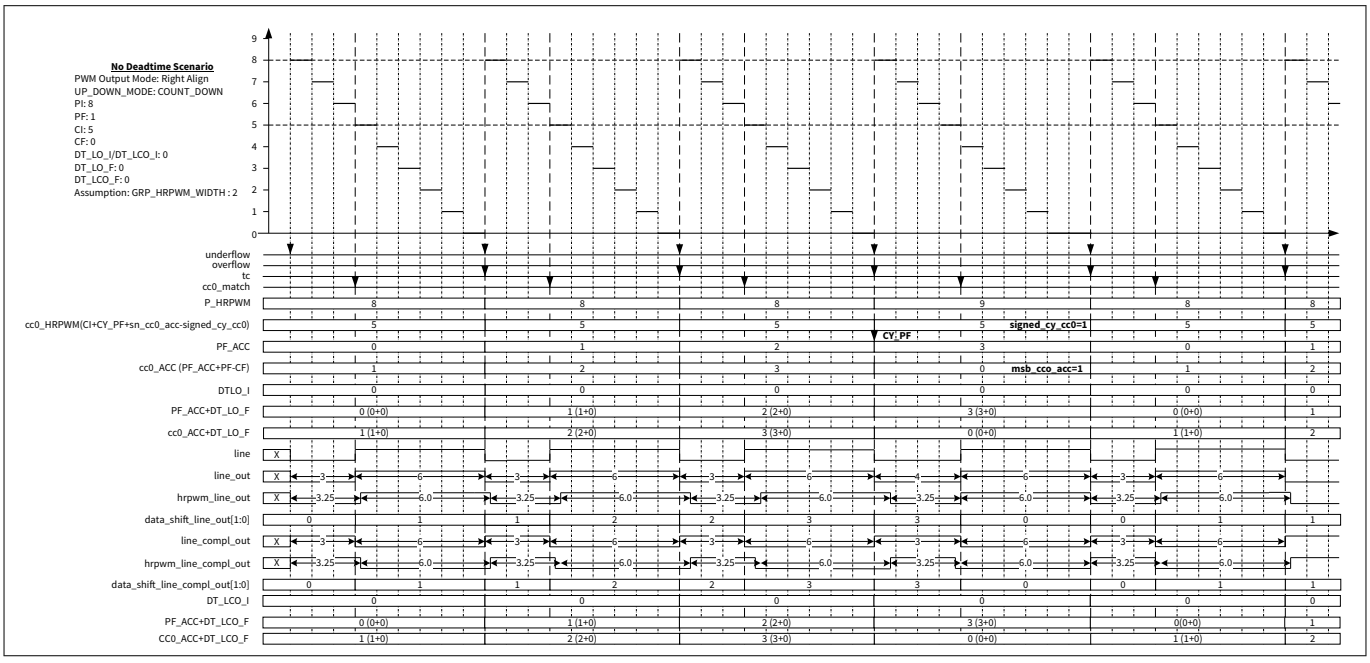


Figure 424 Example of right align in PWM mode with HRPWM feature enabled – Scenario 4 (CF=0)

33.4.2.5 PWM output mode – Asymmetric with cc0 & cc1

The following diagram illustrates an example of a PWM output mode configured in asymmetrical with cc0 & cc1 with HRPWM feature enabled.

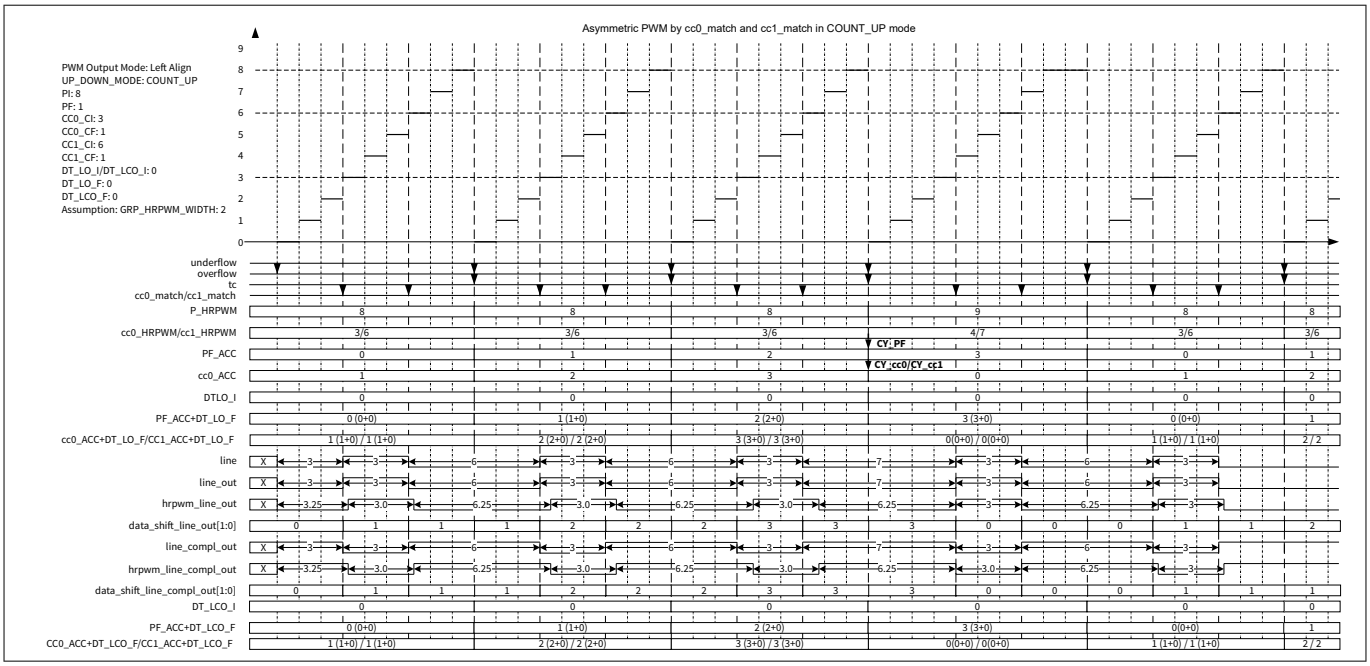


Figure 425 Example of asymmetric with cc0 & cc1 in PWM mode with HRPWM feature enabled

33.4.2.6 PWM output mode – Center align

The following diagram illustrates the expectation of HRPWM for the center align case. It shows the handling of the dead time and its fraction component in addition to the overall fractional shift value that will be forwarded to the delay elements.

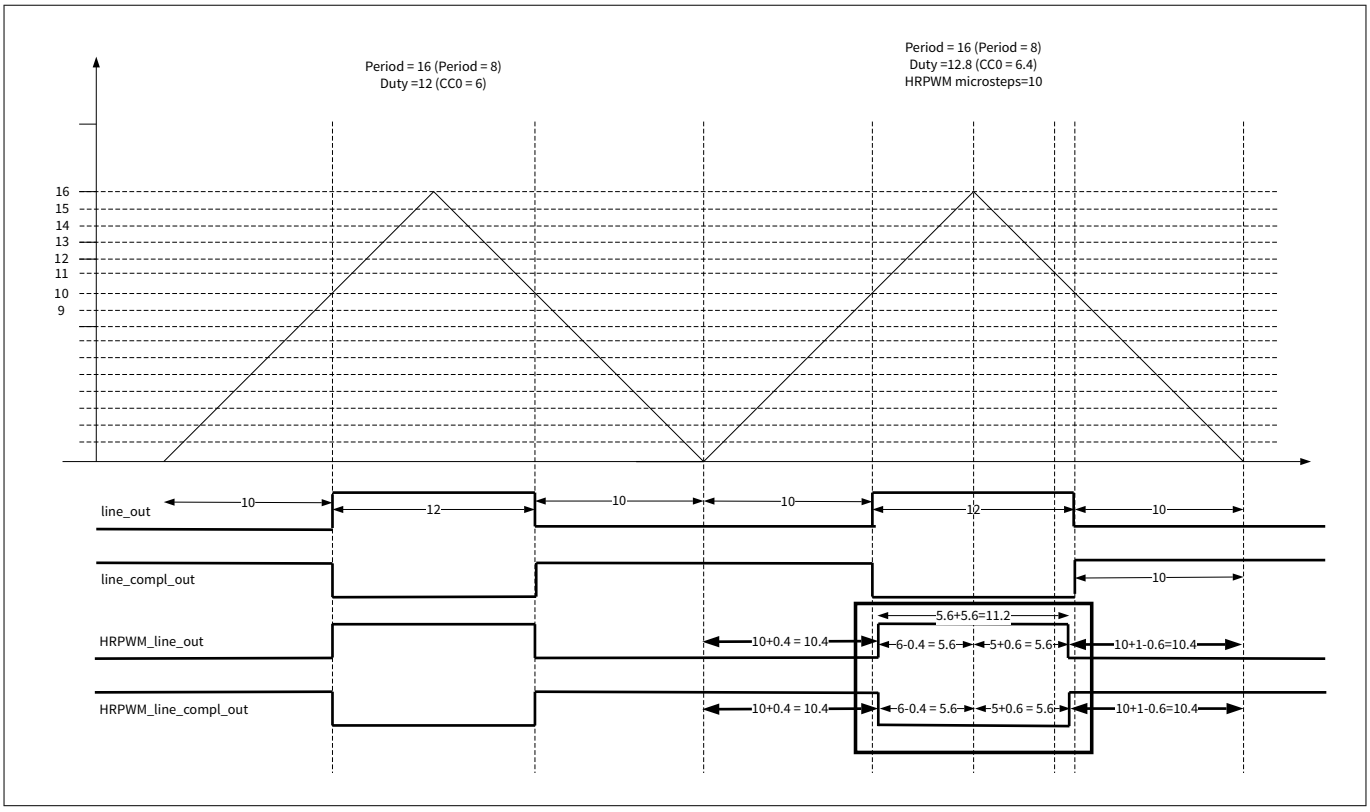


Figure 426 Center align case HRPWM expectation

COUNT_UPDN1 and COUNT_UPDN2 handling (center-align and asymmetric-align) is implemented similarly to the handling of COUNT_UP (left-align) and COUNT_DOWN (right-align) with minor differences, described in the following table.

Table 345 Carry calculation in HRPWM

	COUNT_UP	COUNT_DOWN	COUNT_UPDN1/2
PF_ACC increments	At overflow	At underflow	At underflow and overflow Do not increment if external event was detected in up slope.
data_shift at overflow/ underflow	Equal to PF_ACC	Equal to PF_ACC	Equal to PF_ACC
CY_PF	PF_ACC+PF≥2^GRP_HRPWM_WIDTH	PF_ACC+PF≥2^GRP_HRPWM_WIDTH	PF_ACC + PF ≥ 2^GRP_HRPWM_WIDTH
Handling of CY_PF	P_HRPWM = PI Extend counter value (when COUNTER= PI and CY_PF = 1)	P_HRPWM = PI Extend counter value (when COUNTER= 0 and CY_PF = 1)	Extend counter value (when COUNTER=PI, down=0 and CY_PF = 1) Extend counter value (when COUNTER=0, down=1 and CY_PF = 1) Ignore CY_PF if external event was detected in up slope
CC0_ACC (CC1_ACC)	PF_ACC + CF	PF_ACC + PF - CF	PF_ACC + CF (when down=0) PF_ACC + PF - CF (when down=1) PF_ACC - CF (when down=1 and external event was detected in up slope)
CY_CC0 (CY_CC1)	PF_ACC + CF ≥ 2^GRP_HRPWM_WIDTH	msb_cc0_acc XOR sn_cc0_acc	PF_ACC + CF ≥ 2^GRP_HRPWM_WIDTH (when down=0) msb_cc0_acc XOR sn_Cc0_acc (when down=1)
CC0_hrpwm (CC1_hrpwm)	CI+CY_CC0	CI + sn_cc0_acc - CY_CC0	CI + CY_CC0 (when down=0) CI + sn_cc0_acc - CY_CC0 (when down=1)
CC0_match (CC1_match)	COUNTER changes to CC0_hrpwm	COUNTER changes to CC0_hrpwm	COUNTER changes from CC0_hrpwm
data_shift at cc0_match (cc1_match)	CC0_ACC[GRP_HRPWM_WIDTH-1:0]	CC0_ACC[GRP_HRPWM_WIDTH-1:0]	CC0_ACC[GRP_HRPWM_WIDTH-1:0]

The following diagrams illustrate a few examples with different CF setup of PWM output mode configured in center align with HRPWM feature enabled. Please be aware that for the center align case, the value of cc0 changes to CI+1 upon an overflow event when CF is non-zero. The duty cycle duration will be equal for each period cycle when PF is set to zero.

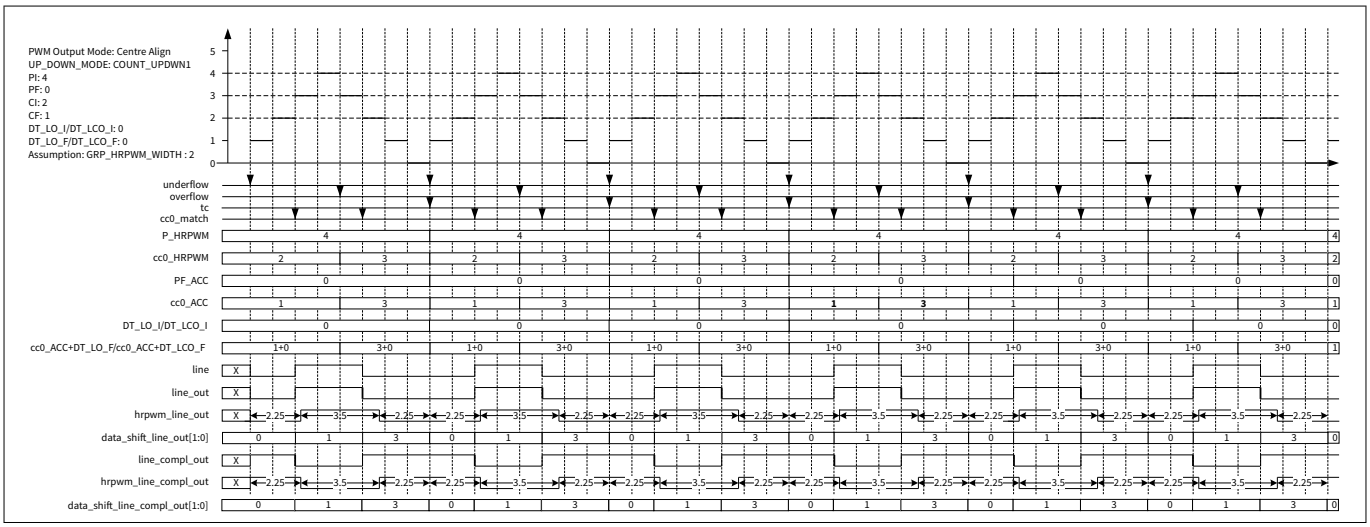


Figure 427 **Example of center align in PWM mode with HRPWM feature enabled - Scenario 1 (PI=4, PF=0, CI=2, CF=1)**

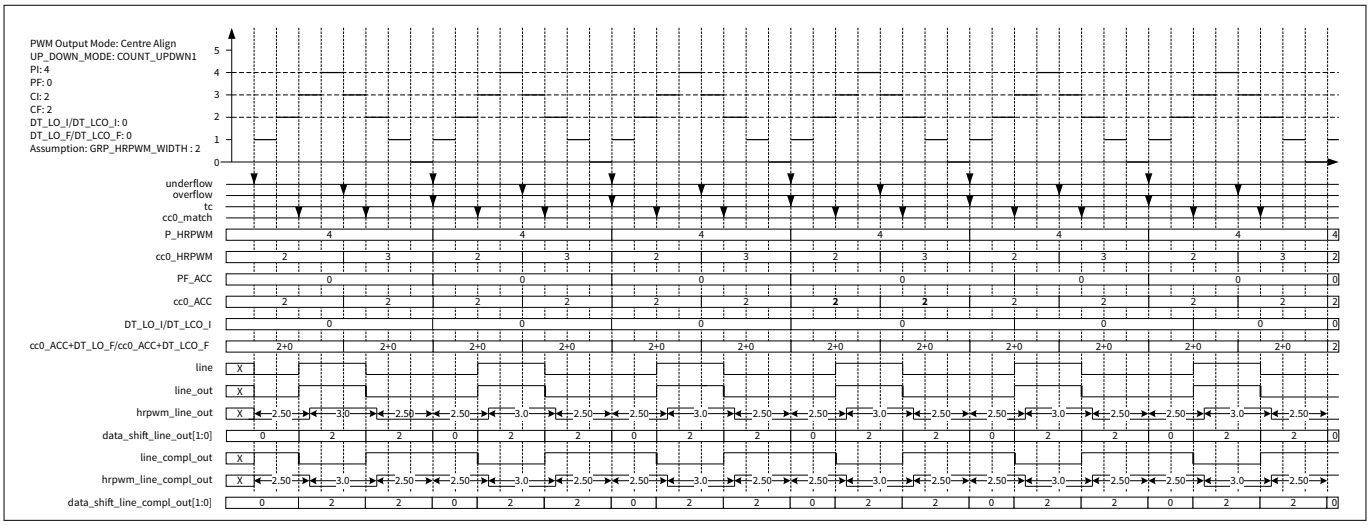


Figure 428 **Example of center align in PWM mode with HRPWM feature enabled - Scenario 2 (PI=4, PF=0, CI=2, CF=2)**

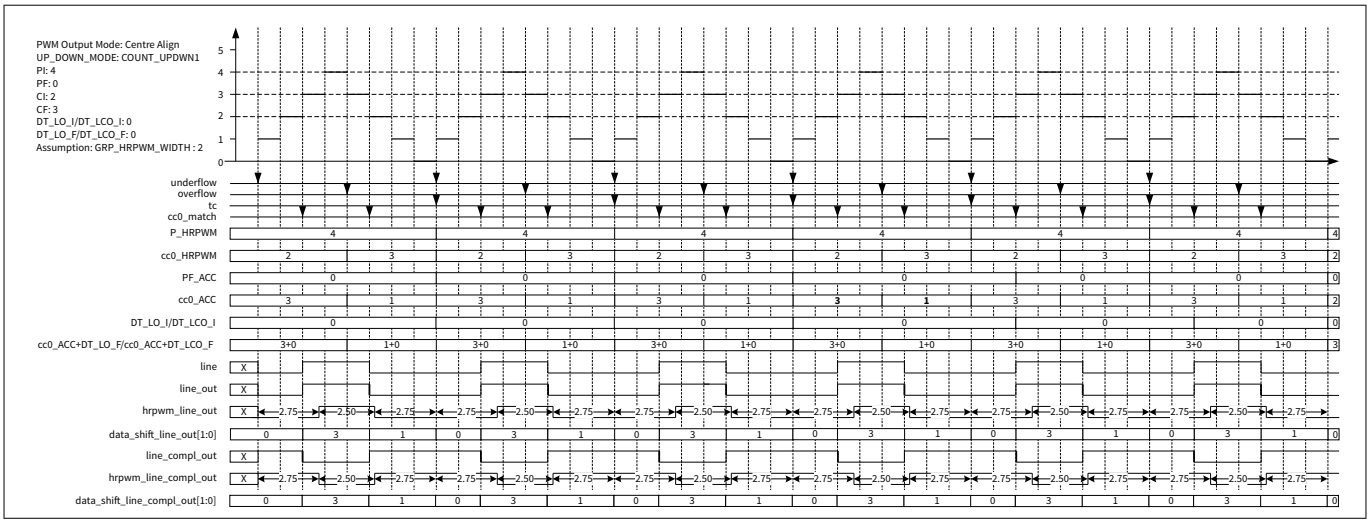


Figure 429 **Example of center align in PWM mode with HRPWM feature enabled - – Scenario 3 (PI=4, PF=0, CI=2, CF=3)**

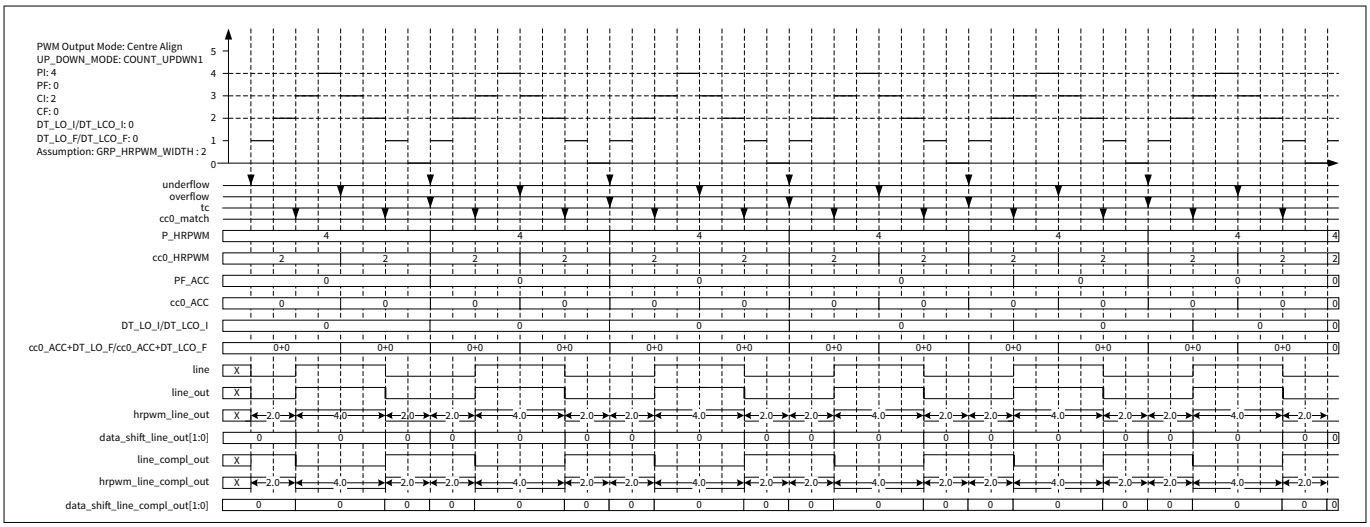


Figure 430 **Example of center align in PWM mode with HRPWM feature enabled - Scenario 4 (PI=4, PF=0, CI=2, CF=0)**

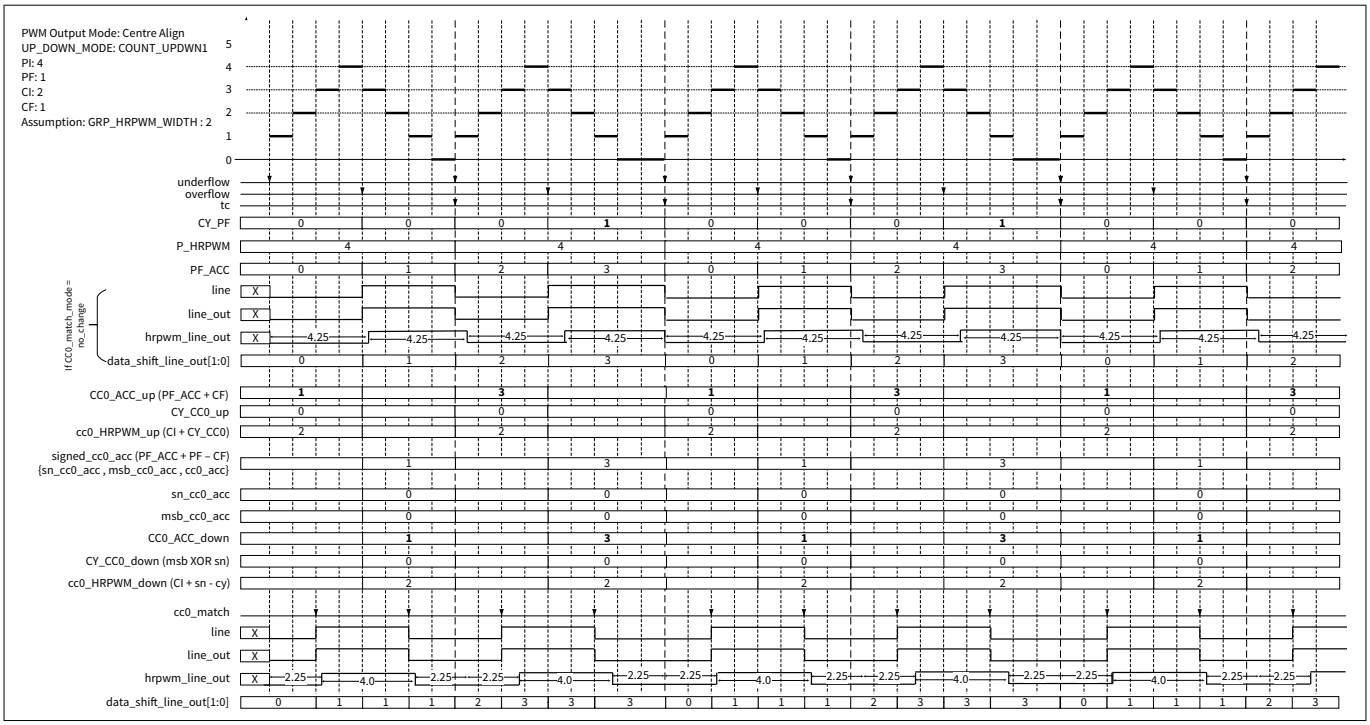


Figure 431 **Example of center align in PWM mode with HRPWM feature enabled - Scenario 5 (PI=4, PF=1, CI=2, CF=1)**

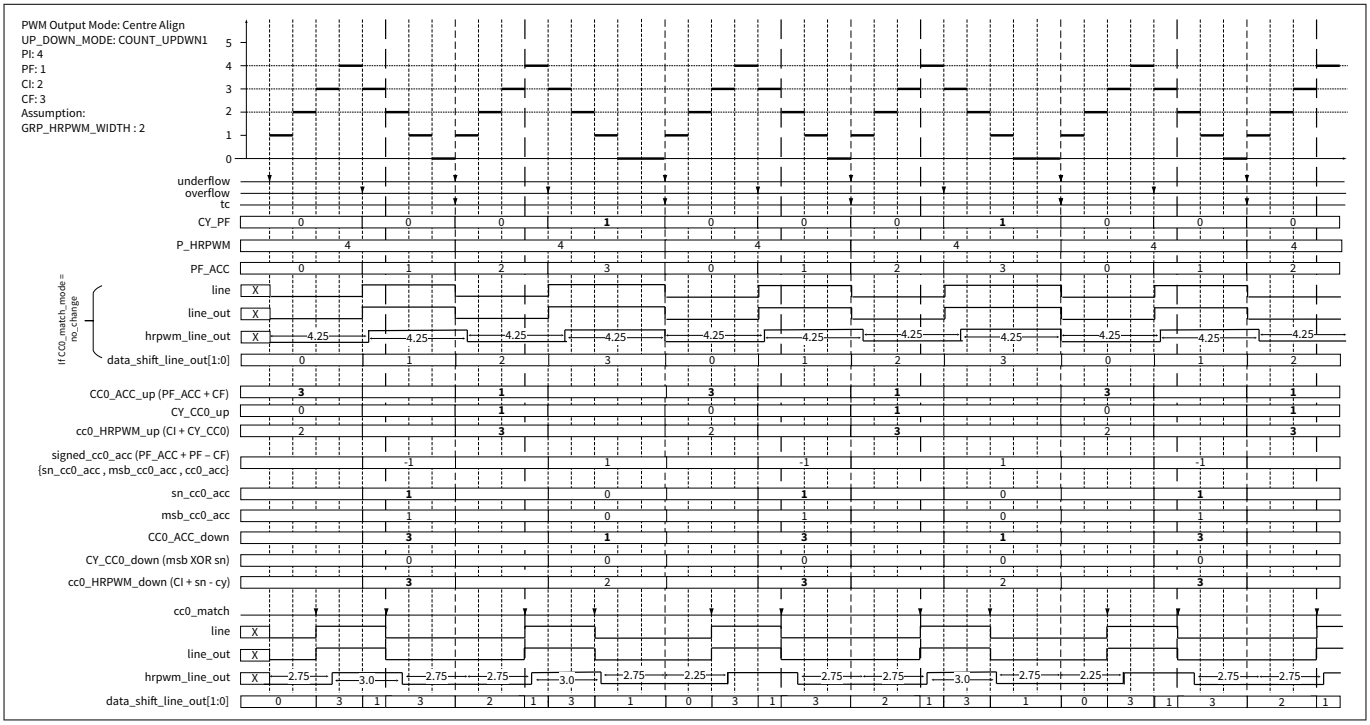


Figure 432 **Example of center align in PWM mode with HRPWM feature enabled - Scenario 6 (PI=4, PF=1, CI=2, CF=3)**

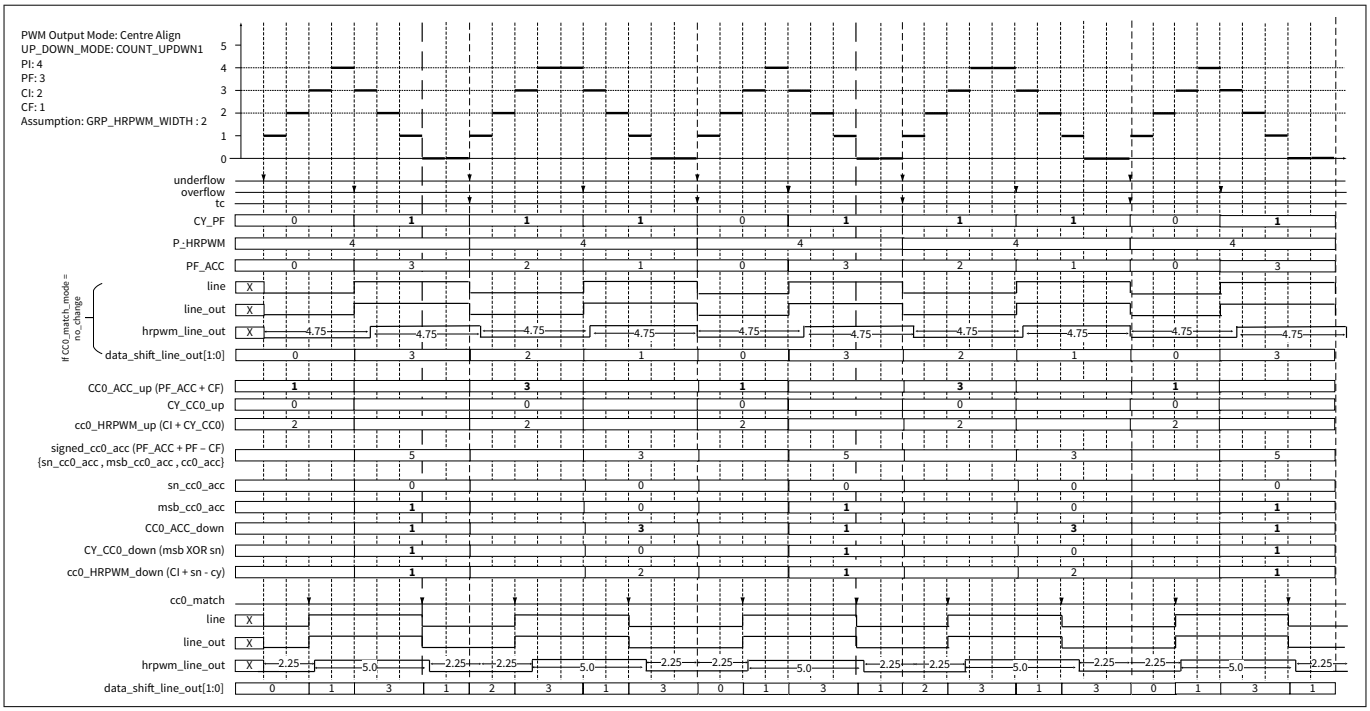


Figure 433 **Example of center align in PWM mode with HRPWM feature enabled - Scenario 7 (PI=4, PF=3, CI=2, CF=1)**

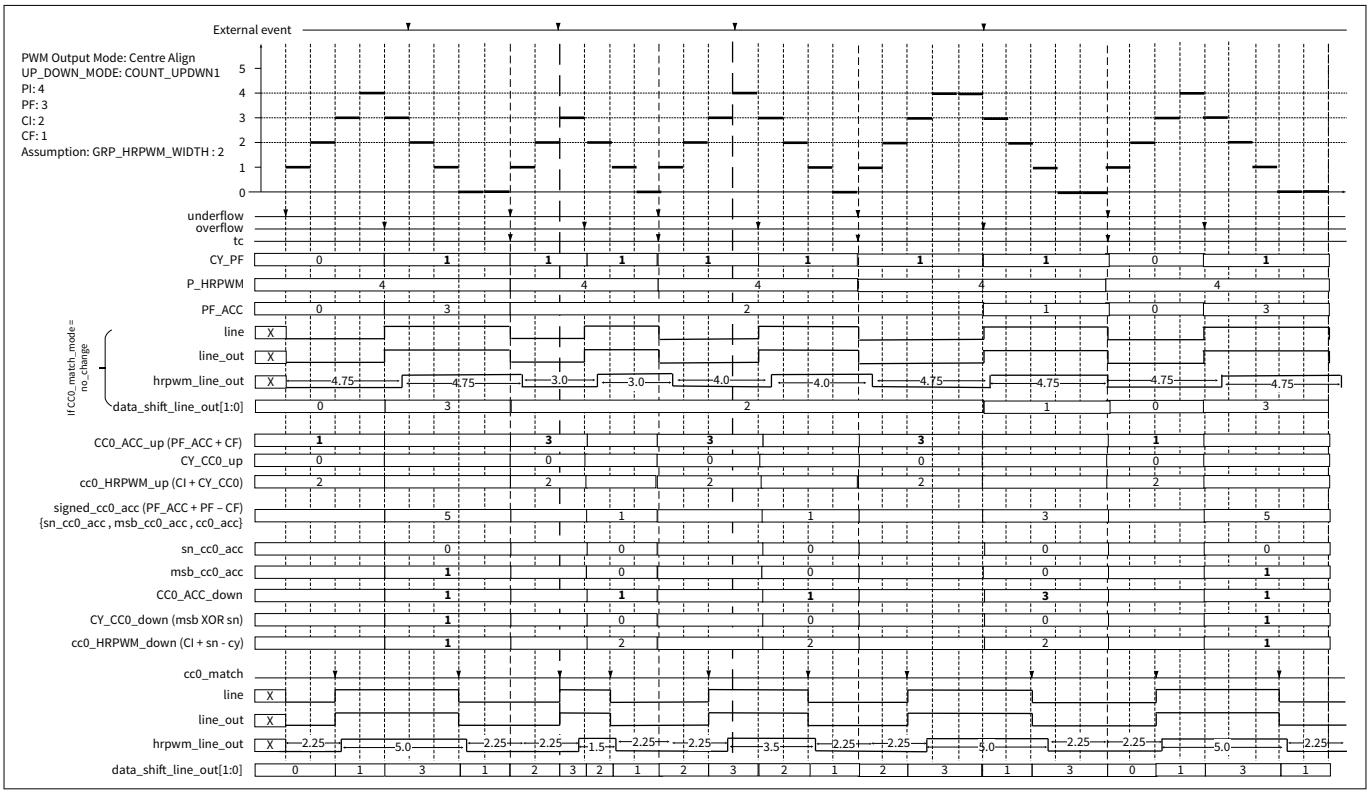


Figure 434 **Example of center align in PWM mode with HRPWM feature enabled and external event triggering overflow - Scenario 8 (PI=4, PF=3, CI=2, CF=1)**

33.4.2.7 PWM output mode – Center align asymmetrical with cc0

The following diagram illustrates an example of a PWM output mode configured in center align asymmetrical with cc0 with HRPWM feature enabled.

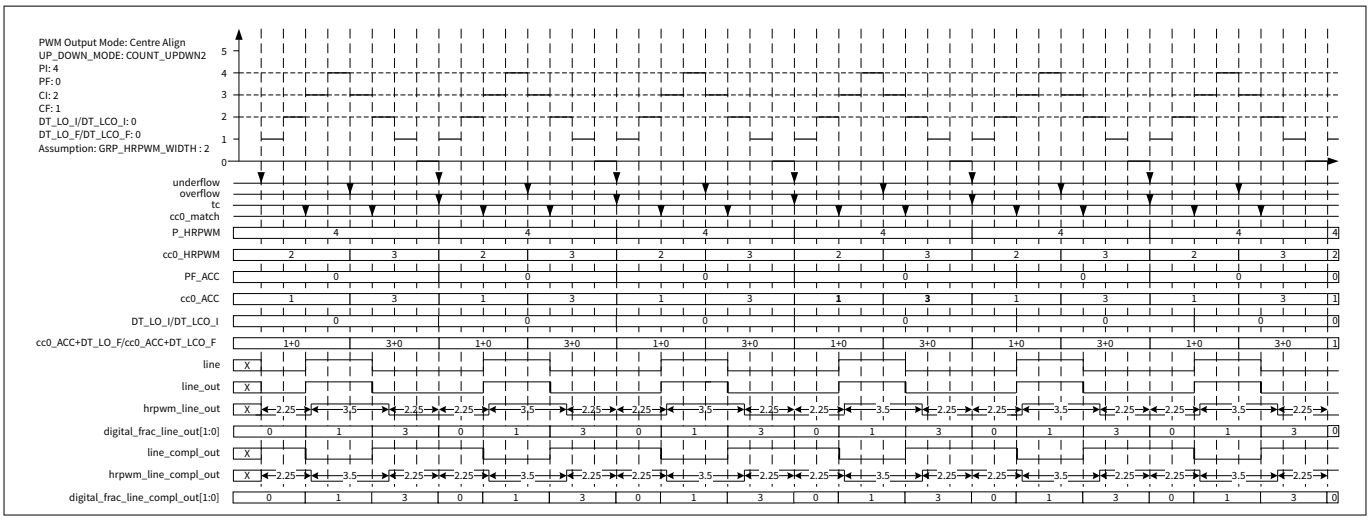


Figure 435 Example of center align asymmetric with cc0 in PWM mode with HRPWM feature enabled

33.4.2.8 PWM output mode – Center align asymmetric with cc0 and cc1

The following diagram illustrates an example of a PWM output mode configured in center align asymmetrical with cc0 and cc1 with HRPWM feature enabled.

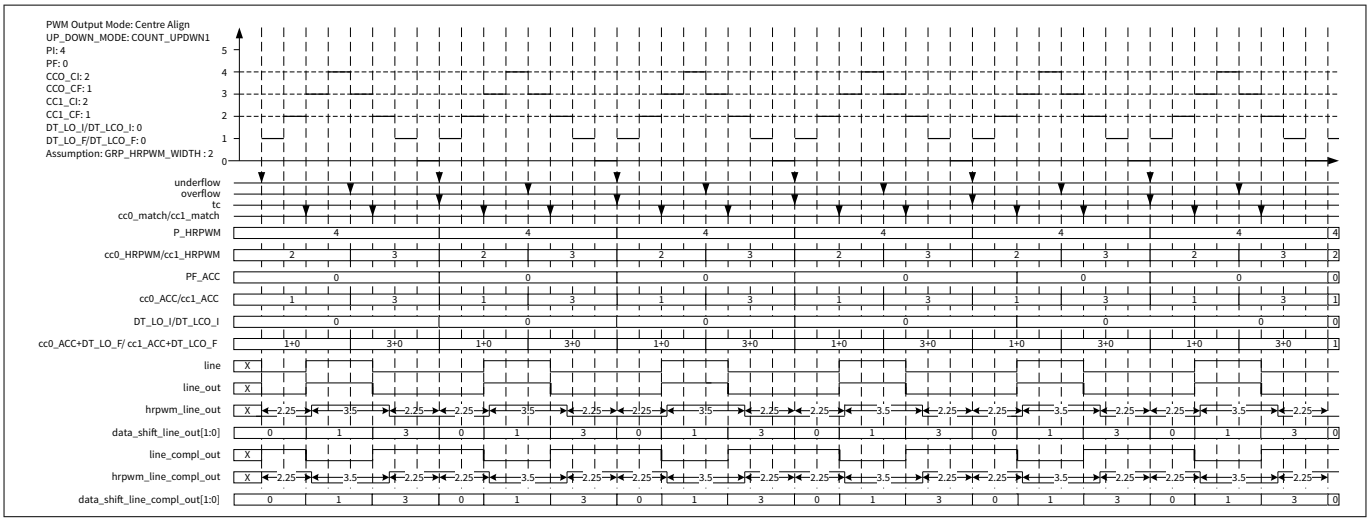


Figure 436 Example of center align asymmetric with cc0 and cc1 in PWM mode with HRPWM feature enabled

33.4.3 PWM_DT

In the PWM_DT mode, when the HRPWM feature is enabled by configuring register HRPWM_CTRL.HRPWM_EN to '1', the following are supported:

- The LSBs are used for fractional or micro ticks for delaying the line_out and line_compl_out outputs for the registers PERIOD/PERIOD_BUFF, CC0/CC0_BUFF, and CC1/CC1_BUFF and DT/DT_BUFF. Number of LSB bits

used for fractional control is been controlled by the parameter GRP_HRPWM_WIDTH[6]. The remaining bits given by [GRP_CNT_WIDTH-1:GRP_HRPWM_WIDTH] are used by the counter

- The LSBs are used for fractional or micro ticks for delaying the line_out and line_compl_out outputs in deadtime insertion for the registers DT/DT_BUFF. Number of LSB bits used for fractional control is been controlled by the parameter GRP_HRPWM_WIDTH. The remaining bits given by [GRP_CNT_WIDTH-1:GRP_HRPWM_WIDTH] are used by the counter

The same PWM output mode that is defined in PWM mode for the HRPWM feature is supported in PWM_DT Mode. Please refer to PWM output mode chapter for more information.

33.4.3.1 Computation concept

The following figures show an example of how data_shift_line_out and data_shift_line_compl_out are calculated in PWM_DT mode and sent out with line_out and line_compl_out to the delay element.

When the DT register contains a fractional value > 0 for line_out, it is added into PF_ACC. Similarly, the DT fractional value for line_compl_out is added to CC0_ACC. In case the final value overflows, the carry is added to the corresponding DT integer value for one TC event period.

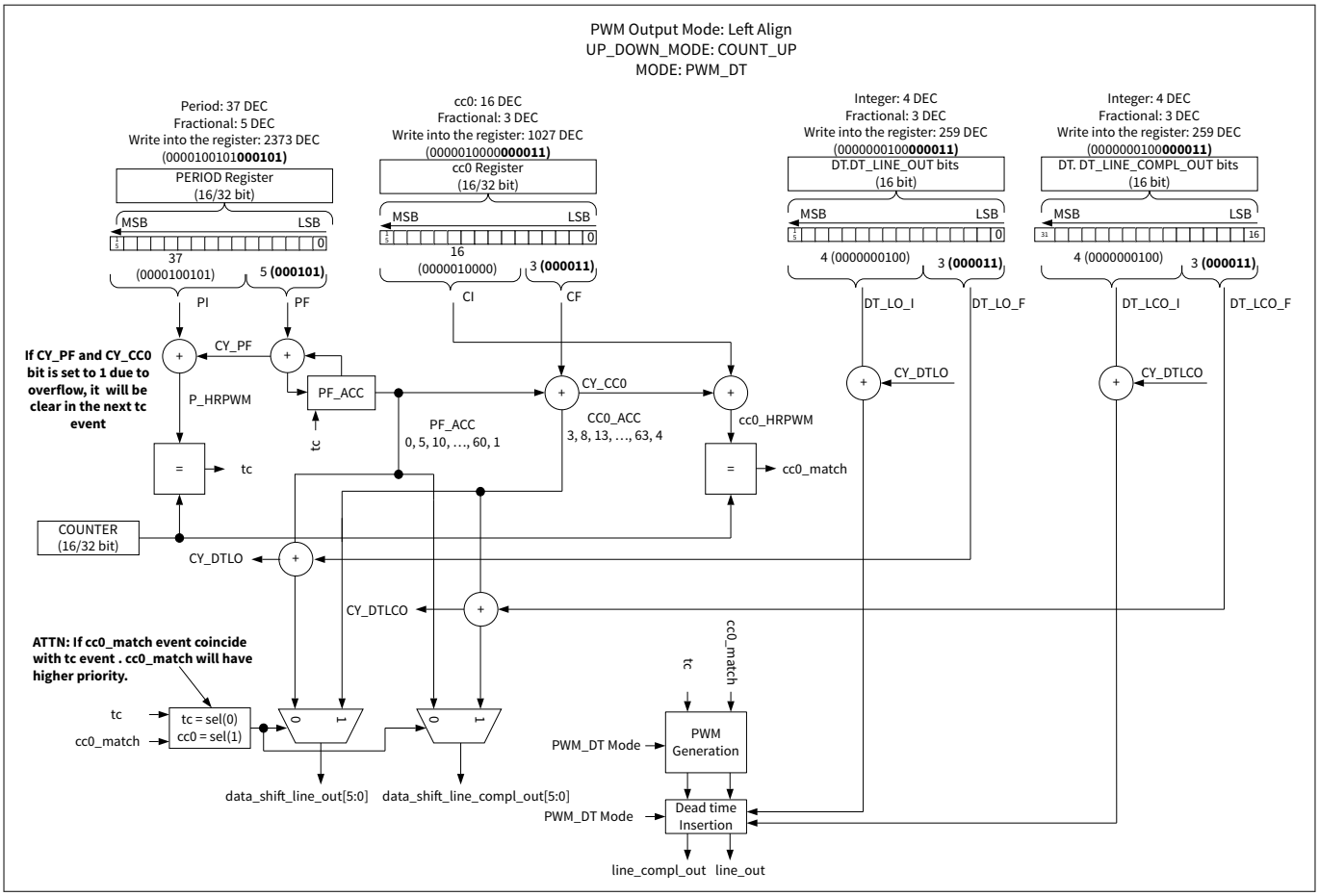


Figure 437 Calculation of data_shift_line_out and data_shift_line_compl_out in PWM_DT mode

6 See Table 157 to check the number of micro ticks bits supported of counter group.

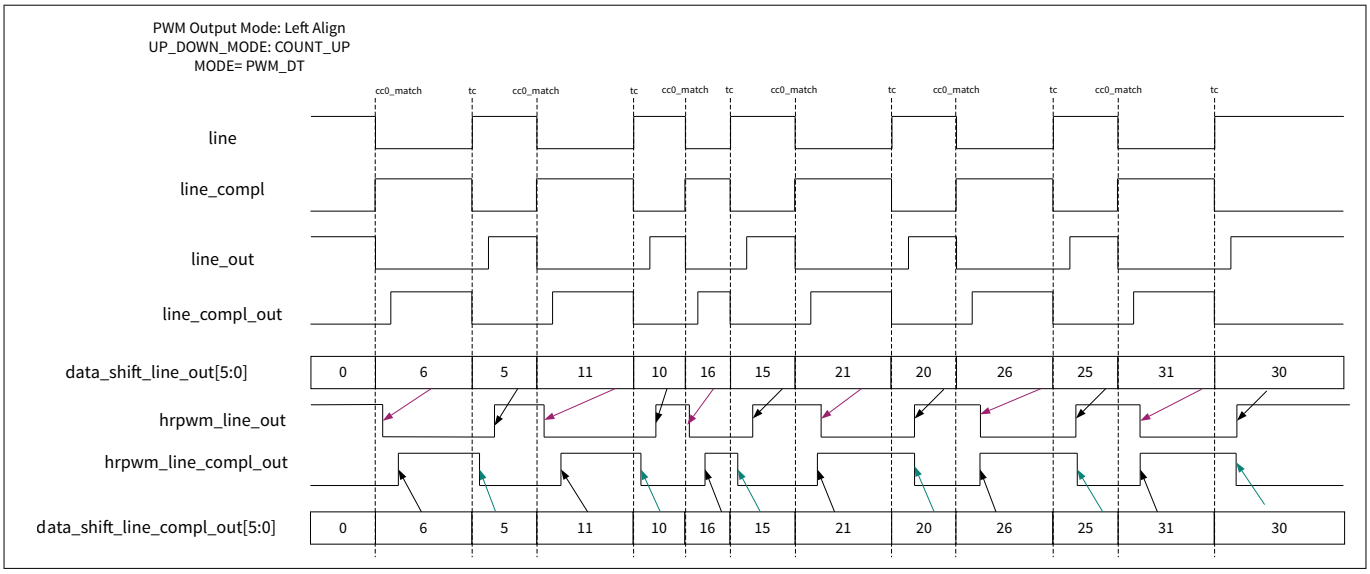


Figure 438 Example of TCPWM outputs in PWM_DT mode

33.4.3.2 PWM_DT output mode – left align

The following diagram illustrates an example of PWM output mode configured in left align with HRPWM feature enabled under PWM_DT mode. Please take note that the duty cycle duration will be equal for each period cycle. Carry event for dead time will occurred due to DT_LCO_F + CC0_ACC and DT_LO_F + PF_ACC event.

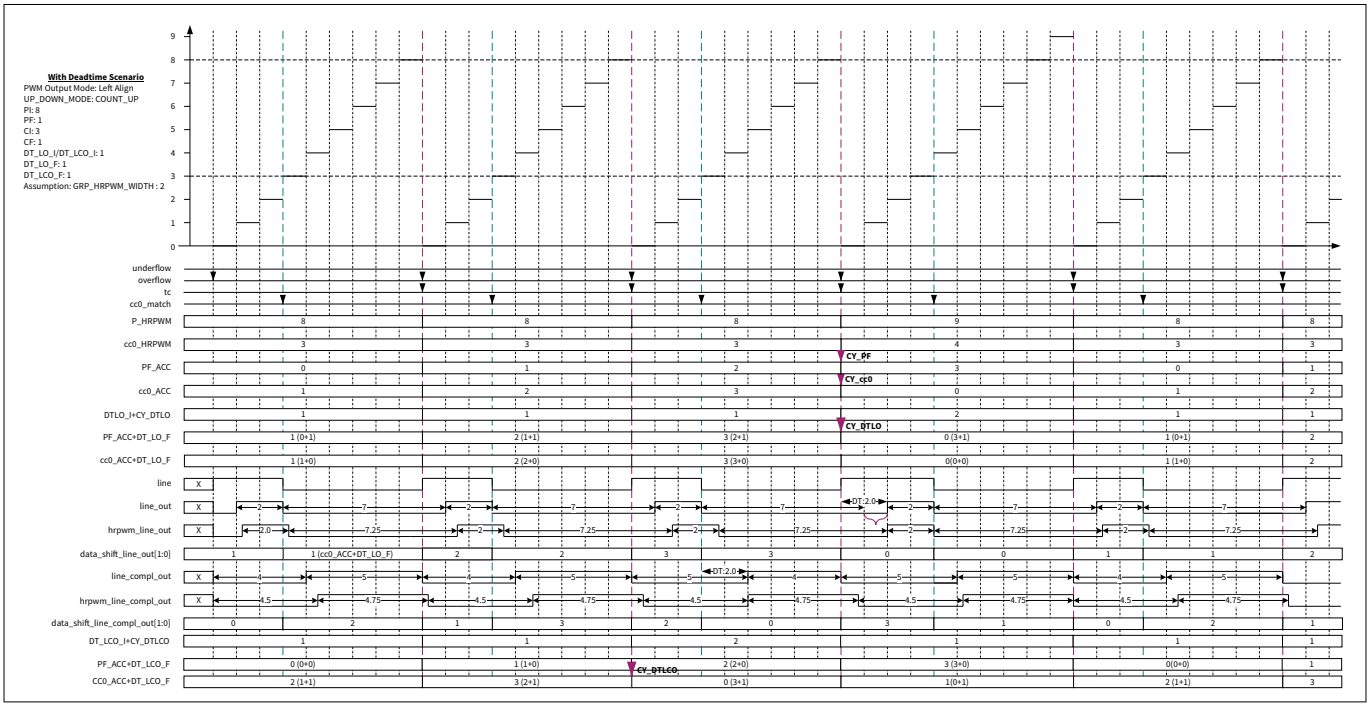


Figure 439 Example of left align in PWM_DT mode with HRPWM feature enabled

33.4.3.3 PWM_DT output mode – right align

The following diagram illustrates an example of PWM output mode configured in right align with HRPWM feature enabled under PWM_DT mode. Please take note that the duty cycle duration will be equal for each period cycle.

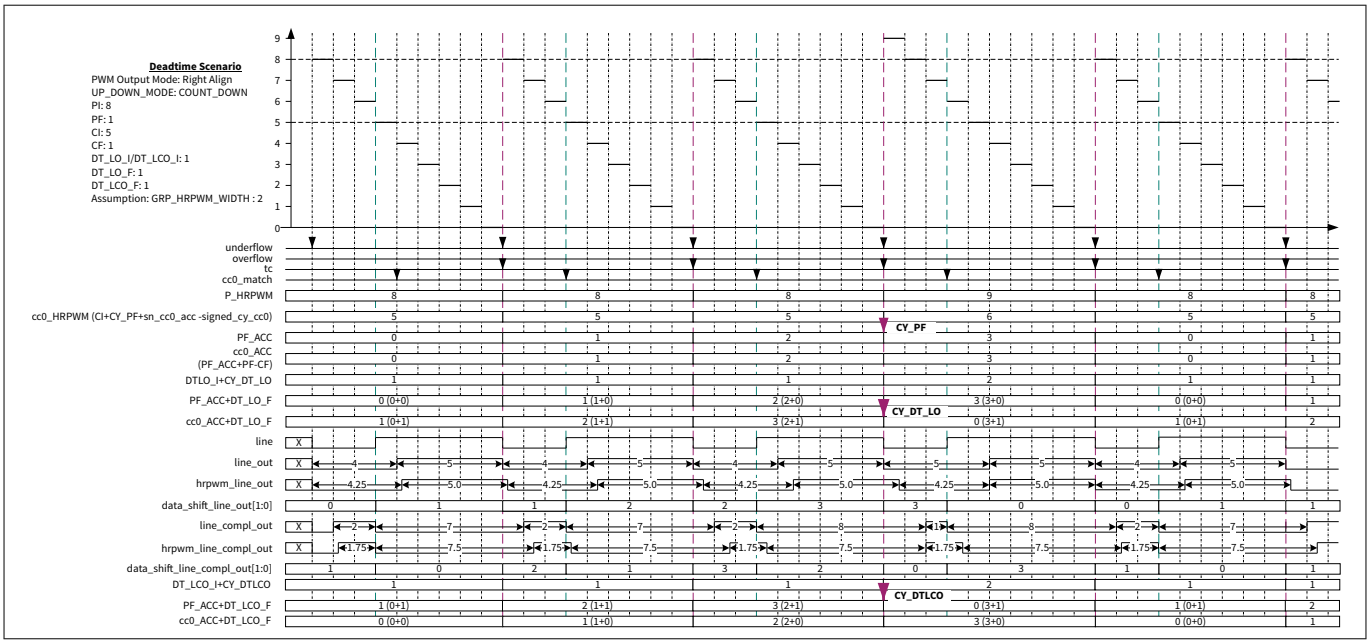


Figure 440 Example of right align in PWM_DT with HRPWM feature enabled

33.4.3.4 PWM_DT output mode – center align

The following diagram illustrates the expectation of HRPWM for the center align case. It shows the handling of the dead time and its fraction component in addition to the overall fractional shift value that will be forwarded to the delay elements.

- Edge at #1(Pointed by blue dot), adds the fractional part of the DEADTIME_LINE_OUT to the overall fractional
- Edge at #2(Pointed by blue dot), shall not add the fractional part of the DEADTIME_LINE_OUT to the overall fractional
- Edge at #3(Pointed by blue dot), shall not add the fractional part of the DEADTIME_LINE_COMPL_OUT to the overall fractional
- Edge at #4(Pointed by blue dot), add fractional part of the DEADTIME_LINE_ COMPL_OUT to the overall fractional

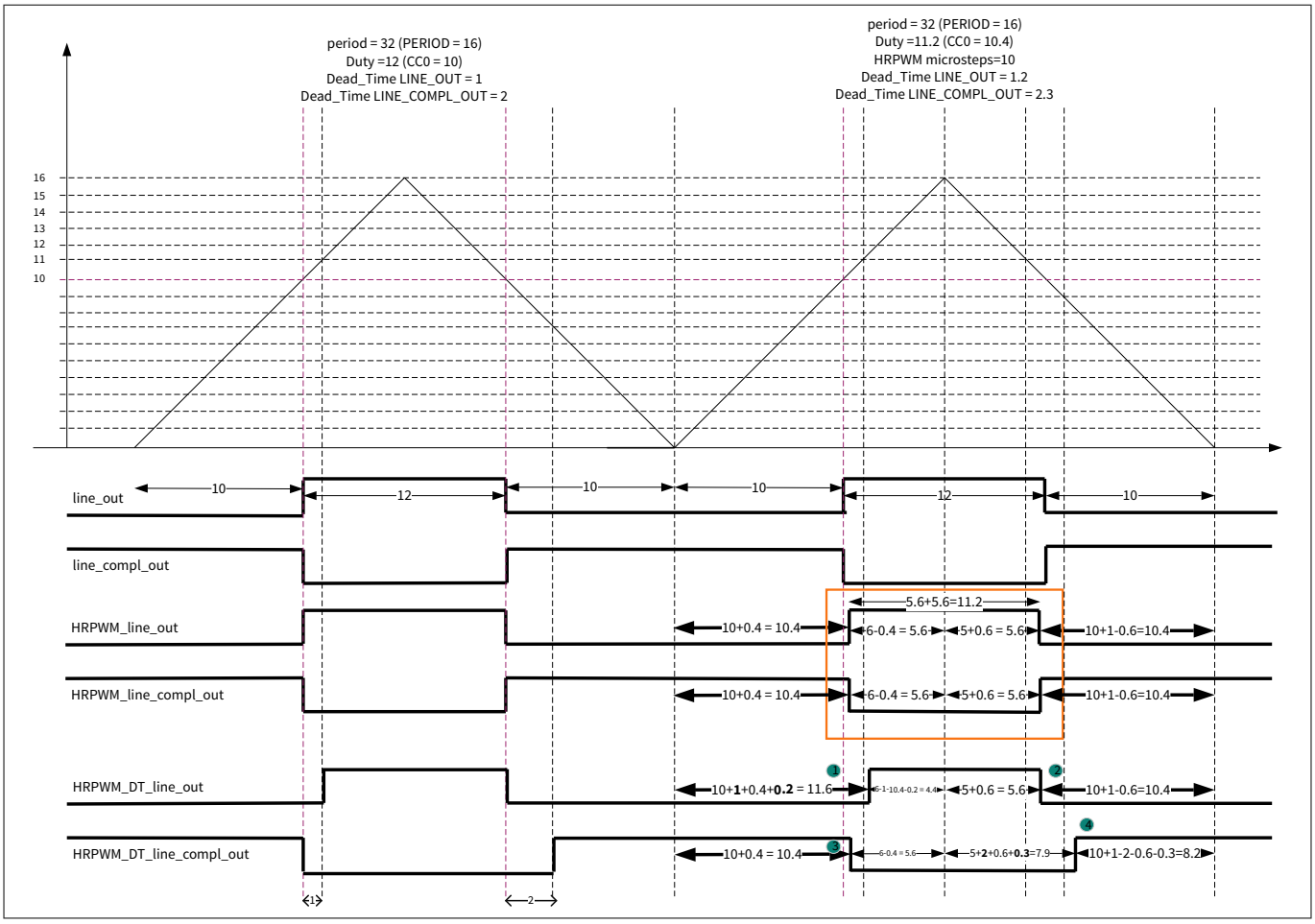


Figure 441 Center align case HRPWM expectation for PWM_DT mode

The following diagram illustrates an example of a PWM output mode configured in center align with HRPWM feature enabled under PWM_DT mode. The duty cycle duration will be equal for each period cycle when PF is set to zero. Carry event for dead time will occurred due to cc0_ACC + DT_LCO_F.

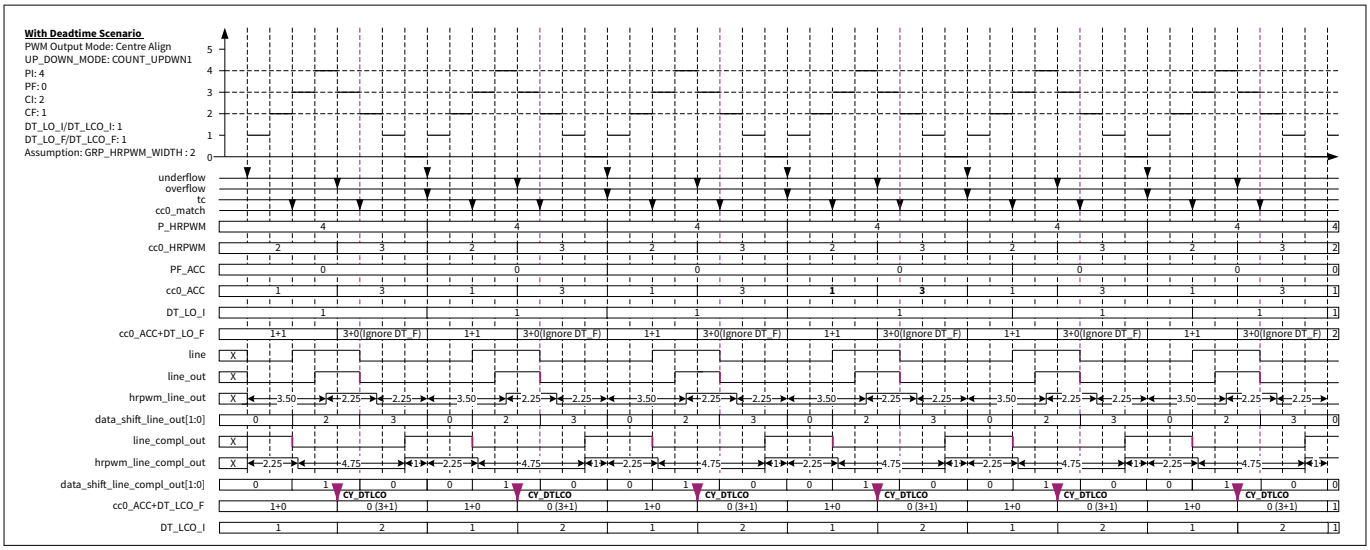


Figure 442 Example of center align in PWM_DT mode with HRPWM feature enabled

33.5 Motion interface (MOTIF)

This feature is available when counter group parameter GRP_MOTIF_NR >= 1.

33.5.1 Overview

The MOTIF module comprises three major sub units:

- Quadrature decoder unit
- Hall sensor control unit
- Multi-channel mode unit

The quadrature decoder unit is used for position control linked with a rotary incremental encoder. The hall sensor control unit is used for direct control of brushless DC motors. The multi-channel mode unit is used with the hall sensor mode to output the wanted motor control pattern but also can be used in a stand-alone manner to perform a simple multi-channel control of several control units.

33.5.1.1 Features

General features

- Quadrature decoder
  - Interface for position measurement
  - Interface for motor revolution measurement
  - Interface for velocity measurement
  - Interrupt sources for phase error, motor revolution, direction change and error on phase decoding
- Hall sensor
  - Simple build-in mode for brushless DC motor control
  - Shadow register for the multi-channel pattern
  - Complete synchronization with the PWM signals and the multi-channel pattern update
  - Interrupt sources for correct hall event detection, wrong hall event detection
- Multi-channel mode
  - Simple usage with hall sensor mode
  - Stand-alone multi-channel mode
  - Shadow register for the multi-channel pattern

33.5.1.2 Block diagram

The MOTIF module can operate in four different modes:

- Quadrature decoder
- Hall sensor
- Stand-alone multi-channel
- Quadrature mode with stand-alone multi-channel

To complete the control/measurement loop of all these modes, link the MOTIF with TCPWM counter(s). Each MOTIF module connects to a dedicated channel counter group. Each MOTIF module contains several inputs and outputs (including service requests) as shown in Figure 443 can be mapped to available module functions, depending on the configuration that was selected: quadrature decoder, hall sensor, or stand-alone multi-channel. The module also has two dedicated service request lines.

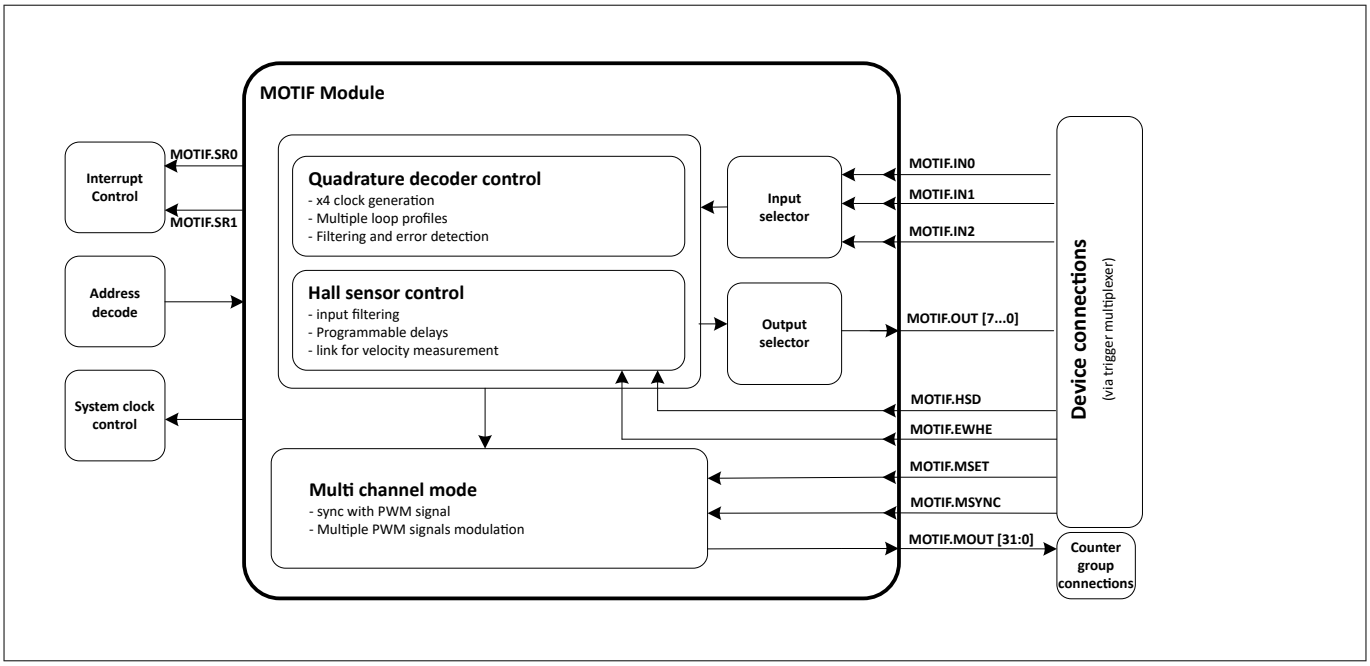


Figure 443 MOTIF block diagram

33.5.2 MOTIF functional description

The following sections describe the complete set of functions and usability of the MOTIF block. In each figure several registers may be depicted to indicate controllability or configurability. These registers follow the description given in Figure 444.

***Note**: The indexing in a register can be done via the non-capital y, x, or n.*

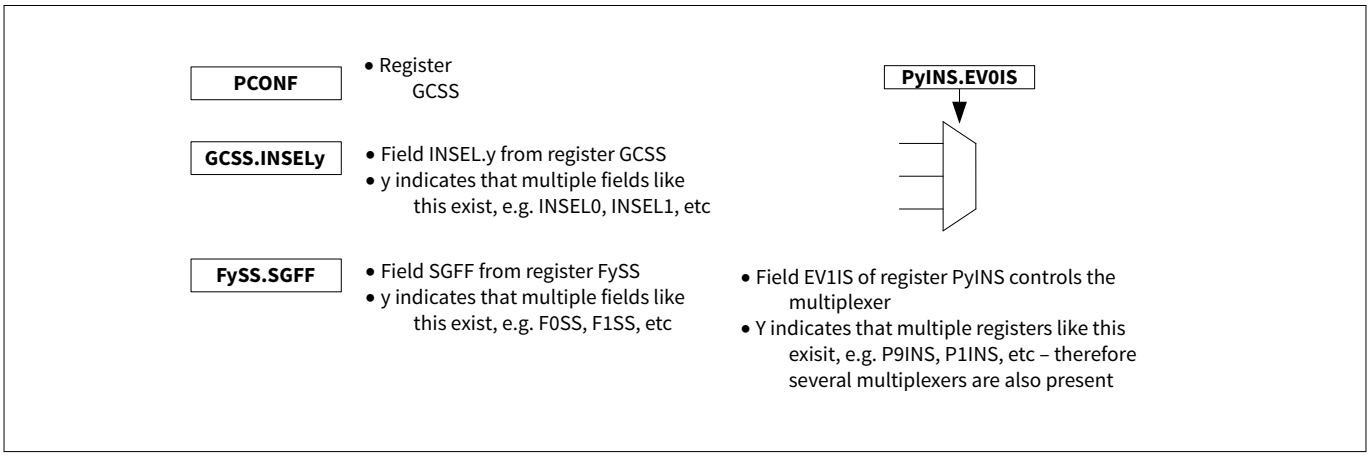


Figure 444 Register description in figures (example)

The MOTIF module contains a function selector unit that is used in parallel by both quadrature decoder and hall sensor control units. This block selects which input signals should be decoded for each control unit. On the output selector, one can select which functions need to be forwarded to the outputs, depending on the operation mode of the module. The MOTIF module also contains a subset of inputs that are only used in hall sensor/multi-channel mode. These inputs are connected to TCPWM counter(s) via trigger multiplexer and are used to control the delays between pattern sampling, multi-channel pattern update, and synchronization, and so on.

The multi-channel unit contains 32 dedicated outputs that contain the current multi-channel pattern, which are connected to the respective TCPWM group. Each MOTIF connects to a dedicated channel group counter. Up to 8x32 MOTIF can be present, a maximum of 32 MOTIF per COUNTER GROUP. The MOTIF control unit contains a dedicated run bit that can be set/clear by software. It can also generate a sync start signal that can be connected to the TCPWM counter(s).

In the quadrature decoder mode, it is possible to define which of the signals is the leading phase and also the active level for each one. The quadrature decoder mode can also receive the clock and direction signals directly from an external source that can be used with simple encoders. The multi-channel offers a shadow register for the multi-channel pattern enabling this way an update on the fly of these parameters. A write access always addresses the shadow register while a read always returns the actual value of the multi-channel pattern.

33.5.2.1 MOTIF MMIO registers

A set of MOTIF specific MMIO registers support MOTIF functionality:

MOTIF Kernel Registers

- PCONF Global control register
- PSUS Suspend Configuration
- PRUNS MOTIF run bit set
- PRUN MOTIF run bit status
- MIDR Module Identification register

Hall Sensor Mode Registers

- HIST Hall Inputs Sample Trigger
- HMEC Hall Mode Extra Config
- HALP Hall Current and Expected patterns
- HALPS Hall Current and Expected shadow patterns
- HOSC Hall Sensor Output Config
- CLUT Hall Mode LUT Config
- SLUT Hall Mode LUT status

Multi-Channel Mode Registers

- MCM Multi-Channel Mode Pattern
- MCSM Multi-Channel Mode shadow Pattern
- MCMS Multi-Channel Mode Control set
- MCMC Multi-Channel Mode Control clear
- MCMF Multi-Channel Mode flag status
- MCPF Multi-Channel Pattern Fault
- MOSC Multi-Channel Output Config
- MCMEC Multi-Channel Extra Config
- MCSM1 Multi-Channel Mode shadow pattern LUT1
- MCSM2 Multi-Channel Mode shadow pattern LUT2
- MCSM3 Multi-Channel Mode shadow pattern LUT3
- MCSM4 Multi-Channel Mode shadow pattern LUT4
- MCSM5 Multi-Channel Mode shadow pattern LUT5

Quadrature Decoder Mode Registers

- QDC Quadrature Decoder Configuration
- QOSC Quadrature Output Config

Interrupt Registers

- PFLG MOTIF interrupt status
- PFLGE MOTIF interrupt enable
- SPFLG Interrupt set register
- RPFLG Interrupt clear register
- PDBG MOTIF Debug Register
- PLP0S MOTIF Low Pass 0 Status
- PLP1S MOTIF Low Pass 1 Status
- PLP2S MOTIF Low Pass 2 Status

33.5.2.2 MOTIF Pin mapping

The mapping of MOTIF global inputs and outputs is depicted in the following table, please take note that the formula in the table is based on “GRP_INDEX” and “GPR_MOTIF_INDEX”. Group[GRP_INDEX] motif[GRP_MOTIF_INDEX] where GRP_INDEX ranges from 0 to 7 and GRP_MOTIF_INDEX ranges from 0 to 31.

Table 346 MOTIF pin description

Pin (global)	I/O	Map to	Hall Sensor mode	Quadrature mode	Multi-channel Mode (stand-alone)
tr_motif_in[(GRP_INDEX*256+0) + GRP_MOTIF_INDEX*8]	I	MOTIFx.IN0	Hall Input 1	Encoder Phase A or clock	Not used
tr_motif_in[(GRP_INDEX*256+1) + GRP_MOTIF_INDEX*8]	I	MOTIFx.IN1	Hall Input 2	Encoder Phase B or clock	Not used
tr_motif_in[(GRP_INDEX*256+2) + GRP_MOTIF_INDEX*8]	I	MOTIFx.IN2	Hall Input 3	Index/Zero marker	Not used
tr_motif_in[(GRP_INDEX*256+3) + GRP_MOTIF_INDEX*8]	I	MOTIFx.HSD	Hall pattern sample delay	Not used	Not used
tr_motif_in[(GRP_INDEX*256+4) + GRP_MOTIF_INDEX*8]	I	MOTIFx.EWHE	Fault event input	Not used	Fault event input
tr_motif_in[(GRP_INDEX*256+5) + GRP_MOTIF_INDEX*8]	I	MOTIFx.MSET	Multi-channel next pattern update	Not used	Multi-channel next pattern update
tr_motif_in[(GRP_INDEX*256+6) + GRP_MOTIF_INDEX*8]	I	MOTIFx.MSYNC	Multi-channel pattern update synchronization	Not used	Multi-channel pattern update synchronization
tr_motif_in[(GRP_INDEX*256+7) + GRP_MOTIF_INDEX*8]	I	MOTIFx.SPARE	Not used	Not used	Not used
tr_motif_out[(GRP_INDEX*256+6) + GRP_MOTIF_INDEX*8: (GRP_INDEX*256+0) + GRP_MOTIF_INDEX*8]	O	MOTIFx.OUT[6:0]	Programmable outputs	Programmable outputs	Programmable outputs
tr_motif_out[(GRP_INDEX*256+7) + GRP_MOTIF_INDEX*8]	O	MOTIFx.OUT[7]	Not used	Not used	Not used

(table continues...)

Table 346 (continued) MOTIF pin description

Pin (global)	I/O	Map to	Hall Sensor mode	Quadrature mode	Multi-channel Mode (stand-alone)
motif_interrupt [(GRP_INDEX*64+0) + GRP_MOTIF_INDEX*2	O	MOTIFx.SR0	Service request line 0	Service request line 0	Service request line 0
motif_interrupt [(GRP_INDEX*64+1) + GRP_MOTIF_INDEX*2	O	MOTIFx.SR1	Service request line 1	Service request line 1	Service request line 1

33.5.2.3 MOTIF clocks

The TCPWM receives a single clock “clk_peri”. Furthermore, it receives a system clock enable signal “clock_sys_en” to generate internal “clk_root”, if all counters and MOTIF modules are disabled via CTRL.ENABLE, PCONF.ENABLE then “clk_root” still gated.

Inside each MOTIF block, two clocks are derived from “clk_root”, these are “clk_ahb_motif” and “clk_motif”. “clk_ahb_motif” is used for MOTIF MMIO registers whereas “clk_motif” is used for MOTIF operational logic and input event synchronization logic.

MOTIF block receives a clock enable signal “clock_motif_en[i]” which can be used to gate all clocks inside a particular MOTIF instance.

The mapping of clock_motif_en[] to motif[] is shown below:

clock_motif_en [0] → group[0] motif [0]

clock_motif_en [1] → group[0] motif [1]

…

clock_motif_en [30] → group[0] motif [30]

clock_motif_en [31] → group[0] motif [31]

…

clock_motif_en [254] → group[7] motif [30]

clock_motif_en [255] → group[7] motif [31]

When MOTIF module is disabled via PCONF.ENABLE=0 register the “clk_motif” is gated whereas “clk_ahb_motif” is active, all non-retention flip flops inside MMIO register and Operational logic are held in reset whereas the retention register can still be accessed by SW.

When debug mode is active, “clk_ahb_motif” is still active, whereas “clk_motif” is gated. Refer section MOTIF debug behavior.

Event generation is performed on clk_motif.

All status or output change can only happen on clk_motif.

The TCPWM receives a single clock “clk_peri”. Furthermore, it receives a system clock enable signal “clock_sys_en” and a MOTIF clock enable signal “clock_motif_en[i]” for each MOTIF block (up to 256).

The “clock_sys_en” signal is used to derive clock “clk_root”.

The “clock_ahb_en” signal is generated by the AHB5 interface and used to derive “clk_ahb”.

The “clock_motif_en[i]” signal is used to derive MOTIF clock “clk_motif[i]”. “clock_motif_en[i]” is provided by the PERI/PCLK component. Note that the PERI/PCLK component allows for multiple clock gate enable signals to be connected to the same divider. This allows for synchronous operation of multiple MOTIF blocks.

MOTIF MMIO registers are accessible by both the hardware (“clk_motif[i]” clock domain) and the AHB5 interface (“clk_ahb” clock domain). These registers use clock “clk_ahb_motif[i]”, which is a gated version of the input clock “clk_peri”, using clock enable signals “clk_ahb_en” and “clock_motif_en[i]”.

***Note**: In application case, MOTIF is always running at the higher frequency.*

33.5.2.4 MOTIF reset

There are two global resets inputs to TCPWM, rst_sys_act_n and rst_sys_dpslp_n. The rst_sys_act_n is ANDed with PCONF.ENABLE register to produce the internal MOTIF reset signal called “rst_motif_act_n”.

MOTIF operational logic and non-retention MMIO registers can be reset using rst_motif_act_n. Synchronizer for tr_motif_in[] is also reset via “rst_motif_act_n” signal.

All retention MMIO MOTIF registers will be reset by rst_sys_dpslp_n.

33.5.2.5 Input trigger mapping

The mapping of input trigger “tr_motif_in[]” to motif[] is shown below (example of GRP_NR = 2, GRP_MOTIF_NR = [0,0,0,0,0,0,1,1]

tr_motif_in(7:0) → group[0] motif[0]

tr_motif_in(255:8) → unused (removed by synthesis)

tr_motif_in(263:256) → group[1] motif[0]

tr_motif_in(511:264) → unused (removed by synthesis)

tr_motif_in(8*256-1:512) → unused (removed by synthesis)

33.5.2.6 Output trigger mapping

The mapping of input trigger “tr_motif_out[]” to motif[] is shown below (example of GRP_NR = 2, GRP_MOTIF_NR = [0,0,0,0,0,0,1,1]

tr_motif_out(7:0) → group[0] motif[0]

tr_motif_out(255:8) → unused (removed by synthesis)

tr_motif_out(263:256) → group[1] motif[0]

tr_motif_out(511:264) → unused (removed by synthesis)

tr_motif_out(8*256-1:512) → unused (removed by synthesis)

33.5.2.7 Function selector connection scheme

The Function selector maps the input function signals to the selected operating mode, whether Quadrature or Hall Sensor Mode.

It is also possible to perform a low pass filtering on the MI0, MI1 and MI2 inputs, the field PCONF.LPC controls the low pass filters cut frequency. The low pass filter is counter based with a configurable depth, power of 2. The counter increments from zero to depth when an edge is detected at the input. The input has to be the same while the counter is counting up to the desired depth for the output to be updated. If the input changes before counter reaches depth, the counter is reset and the output filter is not changed. The low pass filter is active when MOTIF is stopped, this enables Software to trigger a Hall sample if required. If MOTIF is disabled, the Low Pass Filter logic is in reset. The Low pass filter has a valid output which gets asserted after the first correct sample. The low pass filter runs on clock_motif and is reset with rst_motif_act_n signal.

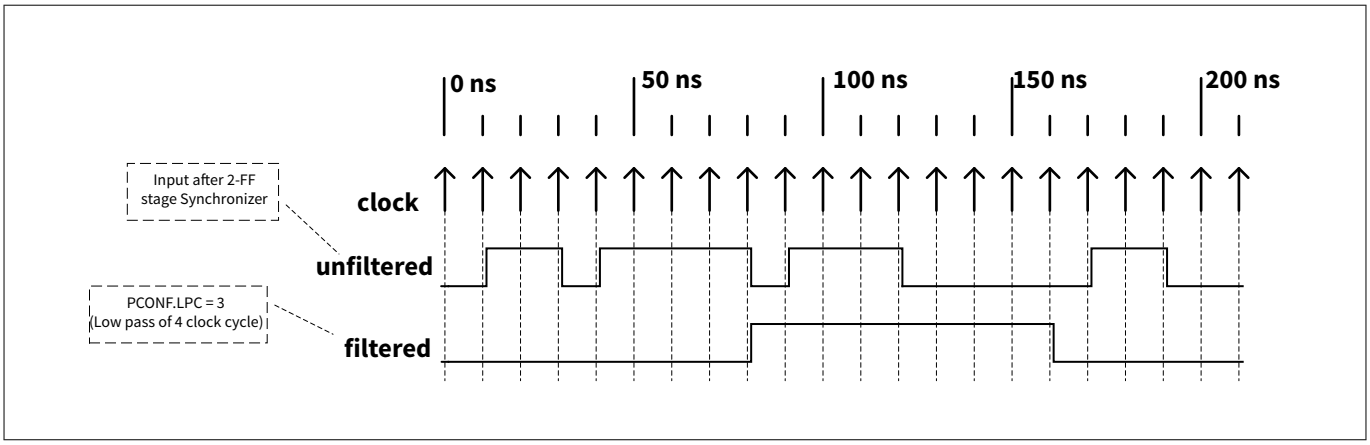


Figure 445 MOTIF Low pass filter

The Hall Sensor Mode is the default function, PCONF.FSEL = 00B. In this mode, the Function selector maps the MI0 to the Hall Input 1, the MI1 to the Hall input 2 and the MI2 to the Hall input 3. When the Quadrature Decoder mode is selected, PCONF.FSEL = 01B, MI0, MI1, and MI2 can be mapped to the following functions: Phase A or Clock, Phase B or Direction or Index signal coming from the rotary motor encoder - this is programmable on via the QDC register. It is also possible to enable in parallel the Quadrature Decoder and stand-alone Multi-Channel mode, by setting PCONF.FSEL = 11B (Hall Sensor Mode and Quadrature Mode can never be used in parallel). The Output Selector allows a very flexible mapping of the events generated by the three available modes, into a specific module output pin.

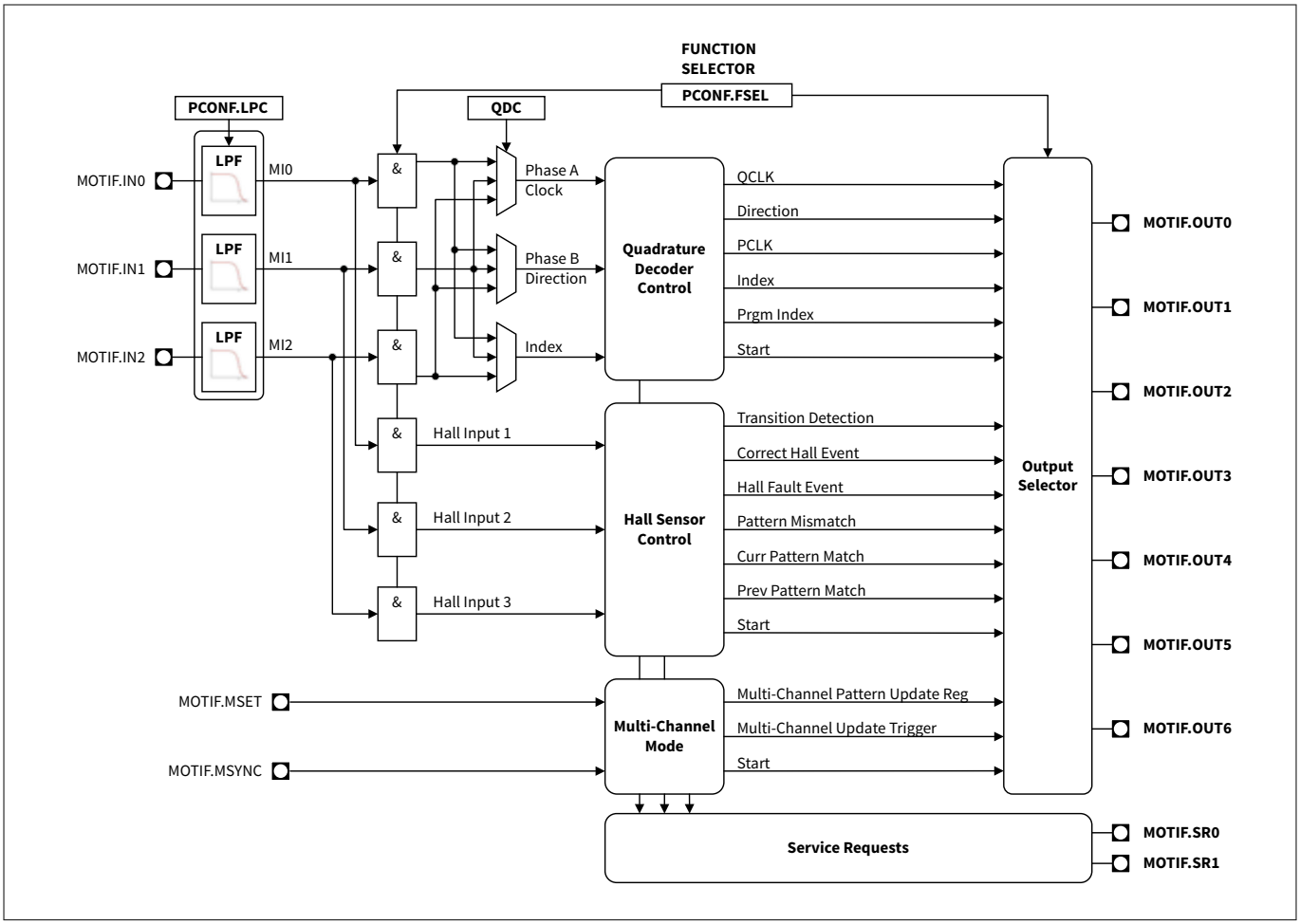


Figure 446 MOTIF connection scheme

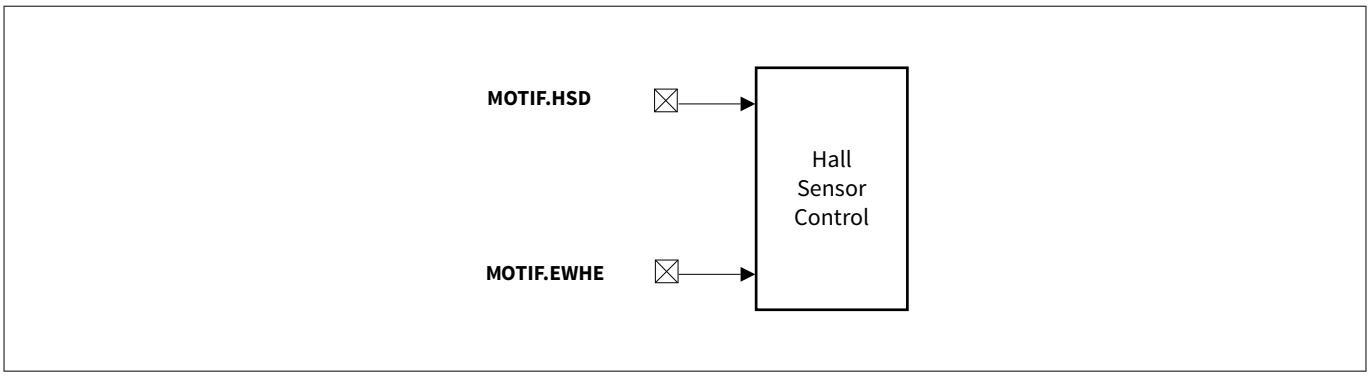


Figure 447 MOTIF additional connections

33.5.3 Hall sensor control

33.5.3.1 Hall Sensor standard operation

The Hall sensor mode is divided into three major loops:

- Detection of an update in the hall inputs
- Delay between the detection
- Sampling of the hall inputs for comparison against the expected hall pattern and the update of the multi-channel pattern

The hall inputs are directly connected to an edge detection circuitry and with any modification in any of these three inputs, an event is generated-transition detection, see Figure 448. This event can be connected to one TCPWM counter via a trigger multiplexer that is controlling the delay between the edge detection and the next step on the hall sensor mode, the sampling of the hall inputs. The signal used to trigger the sample of the Hall inputs can be active at the rising or falling edge (PCONF.SPES). The application software can also issue/request a sample of the hall inputs, by writing 1_B into the HIST.HIS field (normally used for start-up).

When the sampling trigger is sensed, the hall inputs are stored and compared against the current pattern, HALP.HCP, the expected hall pattern, HALP.HEP, and the previous pattern HALP.HPP (the previous pattern comparison can be disabled).

The edge detection circuit generates an enable signal for sampling the hall inputs, PIFHSP, and the sample logic generates a pulse every time that new values are captured, PIFHRDY, that is used inside the pattern compares logic.

The external sample trigger can also be bypass by setting the HMEC.HDBP bit field to 1_B. This will trigger an immediate sample of the hall inputs after the filtering stage if a hall transition is detected. The hall inputs are also sampled as soon as the MOTIF start bit sees a rising edge. The software has to make sure to start MOTIF after LPF output is valid.

When the sampled value matches the expected hall pattern, an event/pulse is generated - the correct hall event - to indicate that the sampled pattern matched the expected one. At the same time, the values programmed into the shadow registers are loaded. The HALP.HCP[LSB] is linked to the hall input 1, and the HALP.HCP[MSB] is linked to the hall input 3 (the same is applicable for the HALP.HEP register).

The update of the fields HALP.HCP, HALP.HEP, and HALP.HPP can come from software by writing 1_B to MCMS.STHR.

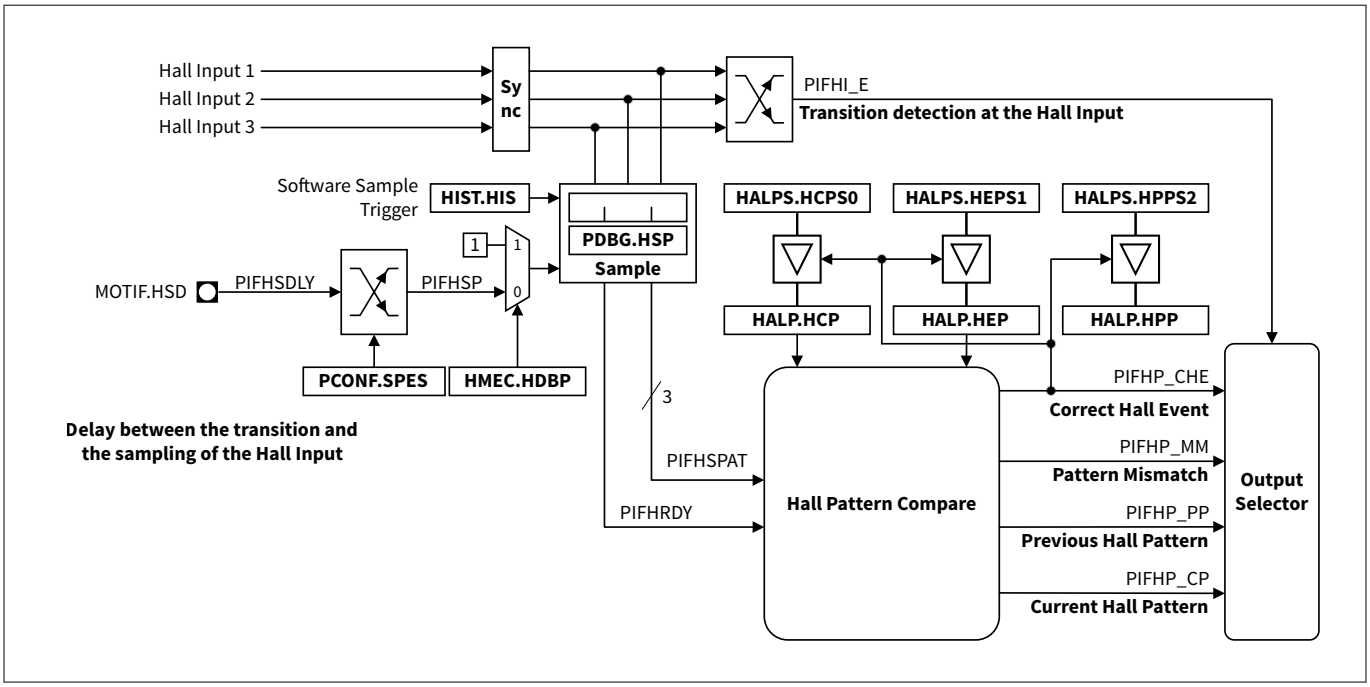


Figure 448 Hall sensor control diagram

When the sampled value matches the current hall pattern, a line glitch is deemed to have occurred and no action is taken. Besides the comparison against the current hall pattern it is also possible to use a third comparison rule - the previous hall pattern. This rule can be enabled by setting the HMEC.HPPE bit field to 1_B. With this comparison, a wrong hall event is not generated if the sampled pattern matches the value programmed in the HALP.HPP. This is important when the motor is in a stand-still position, which may impose that with too less torque the previous hall pattern is sensed at the MOTIF inputs.

When the sampled value does not match any of the values (the current, expected, and previous pattern), a major error is deemed to have occurred and a wrong hall event signal is generated. Every time that a sampled pattern leads to a wrong hall event or when it matches the current pattern or previous pattern (this one can be enabled or disabled) a pattern mismatch event is generated and can be forward to one of the module output pins (via the output selector), see Figure 449

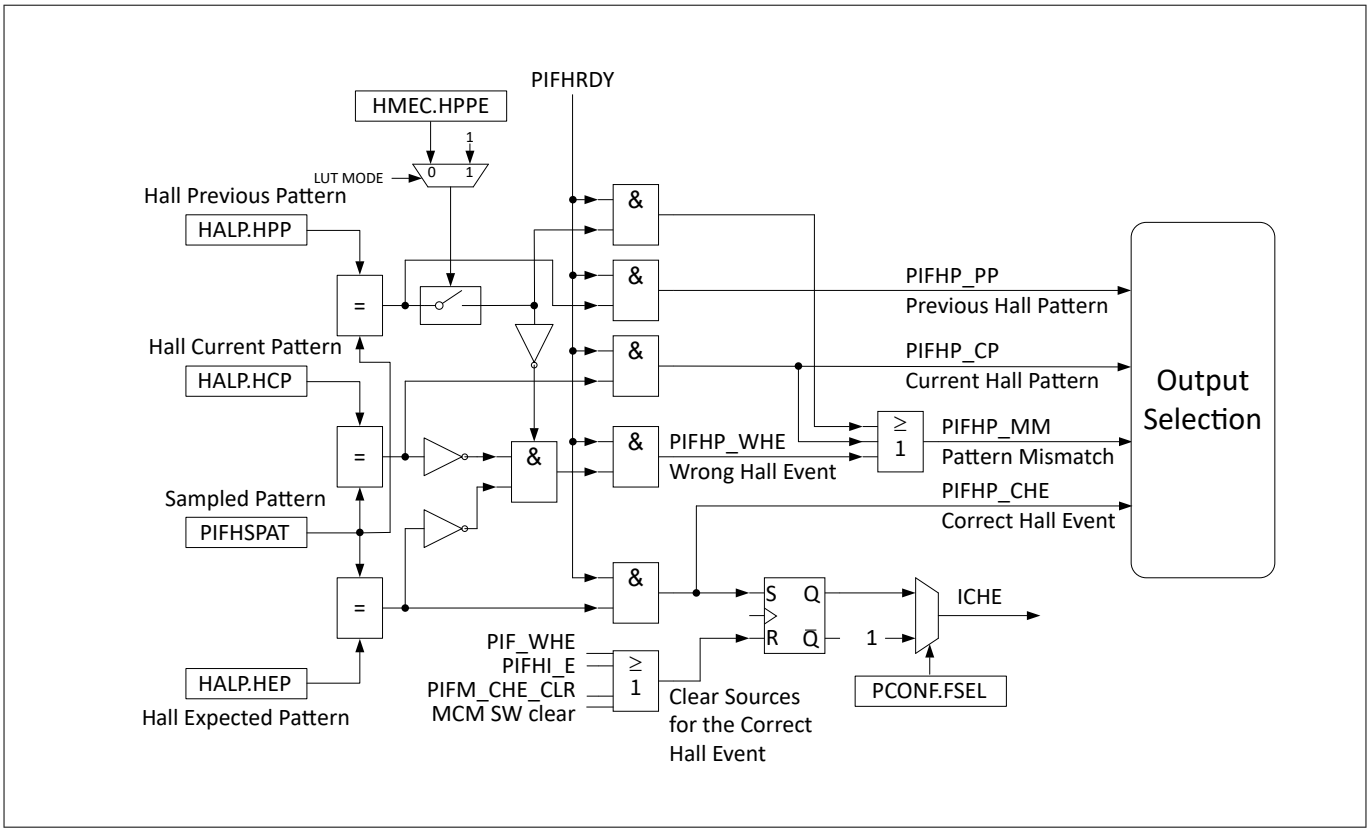


Figure 449 Hall sensor compare logic

A fault hall event can also be decoded by the MOTIF module. This fault hall event is a selectable combination of the wrong hall event coming from the hall compare logic and a programmable module input pin, see Figure 450.

This fault hall event can be forwarded to one of the MOTIFx.OUTy module pins via the output selector (for example, to perform a forced stop operation in the PWM units) and also can be used to clear the multi-channel pattern, see Figure 451.

When a fault event is detected in the hall sensor mode, the hardware shall stop MOTIF as a safety mechanism. This safety mechanism feature is only applicable in LUT Mode. The software needs to act and reconfigure and start MOTIF again after a fault event is detected.

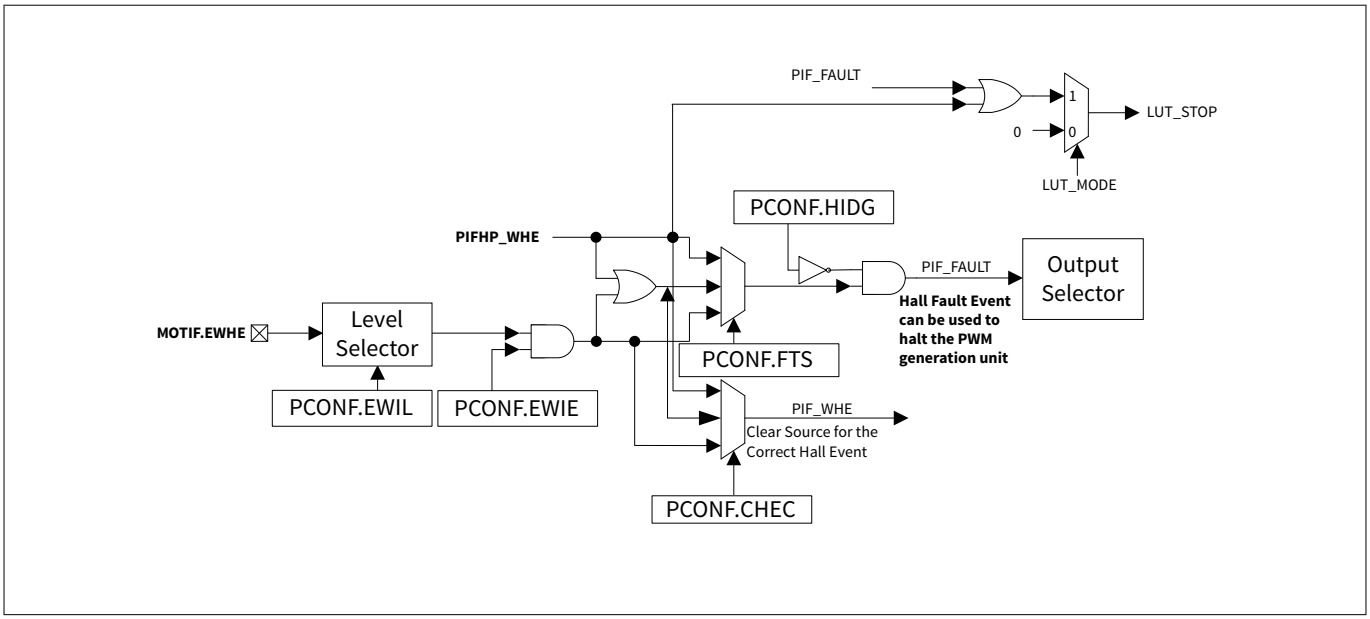


Figure 450 Hall fault event/Idle logic

After the correct hall event is detected, a delay can be generated between this detection and the update of the multi-channel pattern. In the Figure 451, it is demonstrated the control logic of the multi-channel mode.

The delay for the update of the multi-channel pattern can be controlled directly by a TCPWM counter. The trigger that indicates that the delay is finished, is mapped to input MOTIFx.MSET. One can also select the active edge for the trigger via PCONF.MSES. This delay insertion can be bypass by setting the MCMEC.MEBP bitfield to 1_B.

The PCONF.MCUE field selects the source that enables an update of the multi-channel pattern. When set to 1_B , the multi-channel pattern can only be updated after the software has written a 1_B into the MCMS.MNPS field.

After the update delay (if enabled), the multi-channel pattern still needs to be synchronized with the PWM signal. For this, the input MOTIFx.MSYNC is used. One can also select the active edge for the trigger via PCONF.MSYES. When the configured edge is detected in this signal, then the new multi pattern is applied at the MOTIFx.MOUT[31:0] outputs, with the MCM.MCMP[31] linked to the MOTIFx.MOUT[31] and MCM.MCMP[0] to MOTIFx.MOUT[0]. This synchronization stage can also be bypass by setting the MCMEC.MSBP to 1_B.

A multi-channel pattern update request trigger is connected to the MCMF.MSS register field and it can be forwarded to a module pin via the output selector. This register field (MSS) is enabling the multi-channel pattern update (that is done upon receiving the sync signal, PIFMSYNC). The multi-channel pattern is used with a TCWPM counter to perform a synchronous update of the multi-channel PWM output. The update of the MCM.MCMP can be triggered by the software by writing 1_B to MCMS.STMR.

It is also possible to use the hall fault event to clear/load the multi-channel pattern with a pre-configured value - MCPF.MCFV. This mechanism can be used whenever a wrong operation is detected and a specific safe pattern needs to be applied to the PWM timers that are controlling the output switches.

The 32-bit multi-channel pattern outputs can affect the PWM output of the associated counter in two different ways:

- AND gating the PWM - forcing the PWM to passive
- OR gating the PWM - forcing the PWM to active

Each counter will then receive four signals driven by the multi-channel pattern outputs to perform modulation in line_out and line_comp_out. One signal dictates if an OR or an AND is needed, and another signal is used as the second input for the function.

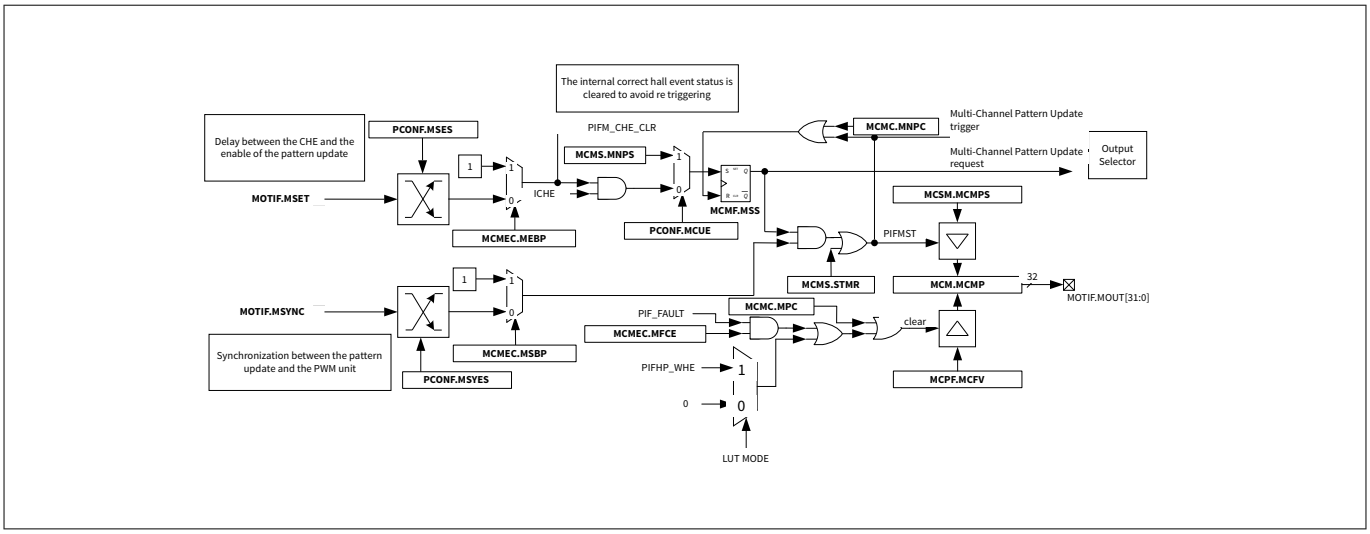


Figure 451 Multi-channel mode diagram

Figure 452 depicts an example of a state diagram using the Hall Sensor Mode for a Brushless DC Motor Control Application. Notice that in this example several states can be bypassed/not used (for example, is possible to avoid the PWM Sync stage by setting the MCMEC.MSBP bitfield to 1B), therefore one should consider that this example contains a super set of the Hall Sensor Mode Control functions.

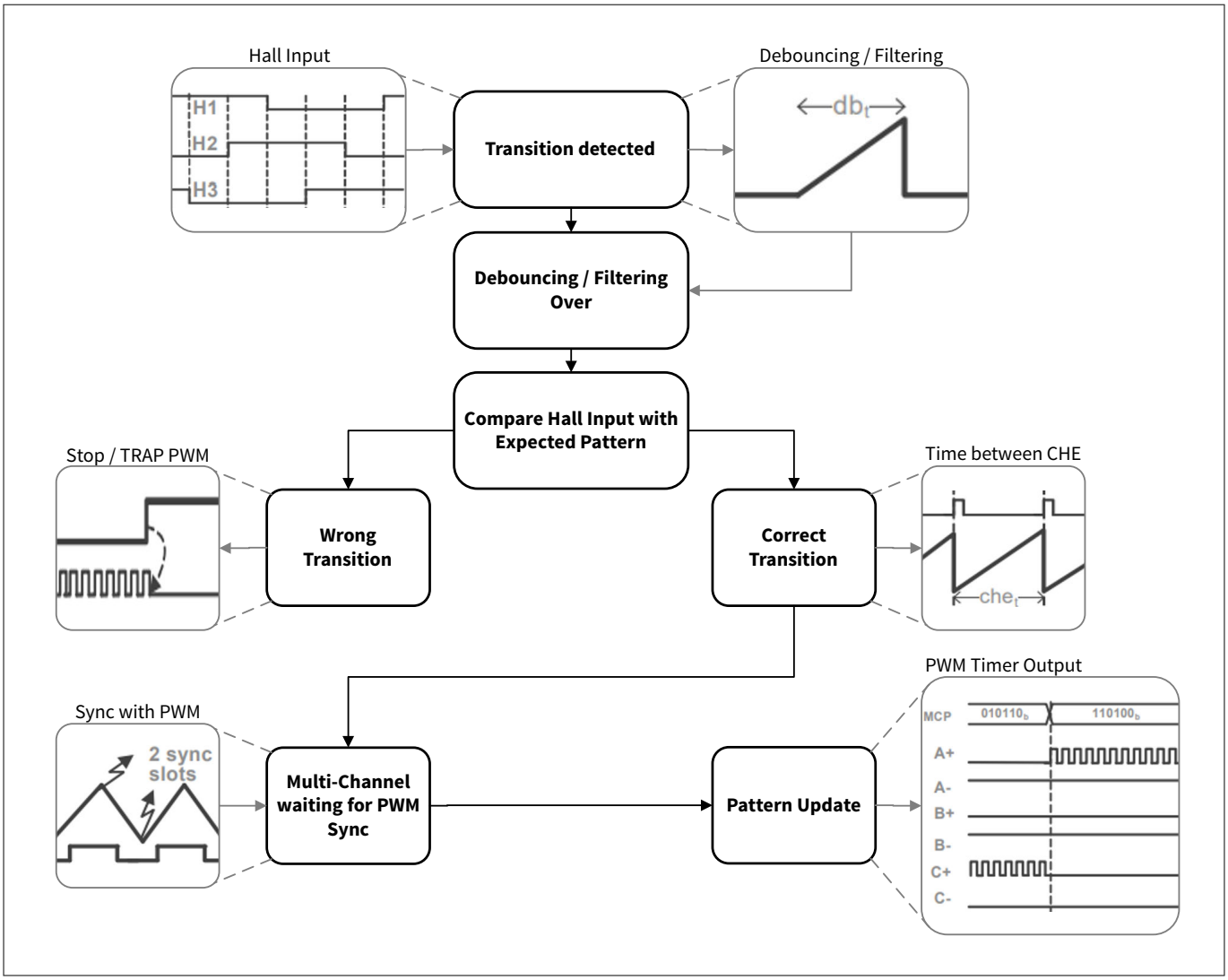


Figure 452 State diagram example for BLDC

Figure 453 shows all the previous described steps in the hall sensor mode description within the timer domain: every time that a transition on a hall input is detected, the signal - transition detection - is asserted. This signal is used to start the timing delay between the transition detection and the sampling of the hall inputs.

In this scenario, the status output (TR_OUT0 in Figure 453) of a TCPWM counter was used to control the timing 1 (delay between the transition and the sampling of the hall inputs). With the rising edge of the TR_OUT0 signal, the hall inputs are sampled and if they match the expected pattern, the signal - the correct hall event - is asserted.

A period match event output (TR_OUT1 signal) mapped to the same TCPWM counter is used to control the delay between a correct hall event and the update of the multi-channel pattern. This service request is asserted when a period match is detected.

Another TCPWM counter is used to measure the time between each correct hall event. This timer cell is represented by the timing 2 in the Figure 453 (the CR symbolizes the capture register of the counter block).

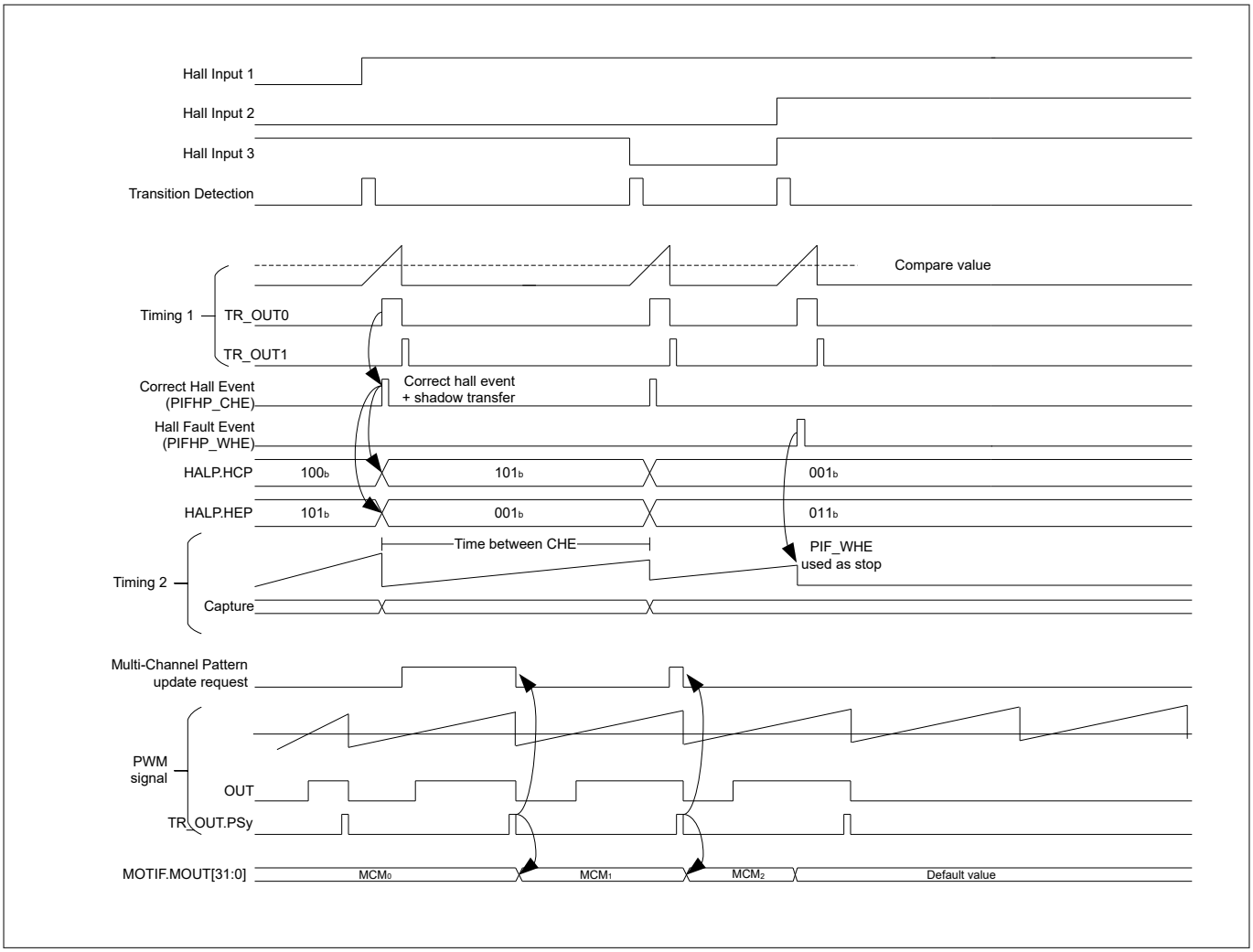


Figure 453 Hall sensor timing diagram

33.5.3.2 Hall Sensor operation with LUT

The hall sensor can operate in look-up table (LUT) mode. This mode aims to reduce CPU load for typical motor control phase commutation in a semi-autonomous way. The LUT mode is a fully hardware-based feedback control loop assisted by software. As long as the motor runs in the same direction, the hardware takes full control on multi-channel pattern output, MCM.MCMP. The Output pattern will be taken from a pre-loaded LUT by software. When the motor changes direction, MOTIF is stopped and the software needs to repopulate the LUT content and start MOTIF again.

Each MOTIF module contains two LUTs. Hall LUT (HLUT, also called “input LUT”) and Modulation LUT (MLUT, also called “output LUT). HLUT contains a hall input pattern, which gets loaded to HALP registers. MLUT contains an output PWM modulation pattern which gets loaded to the MCM.MCMP register.

Each MOTIF has a LUT control logic, which is used to assist the LUT operation. The following registers are associated with this mode:

- LUT control configuration

CLUT.LUTEN -> enable/disable LUT mode

- LUT status and debug register

SLUT.PTR -> internal pointer value status

- MLUT (output LUT)

MCSM, MCSM1, MCSM2, MCSM3, MCSM4, MCSM5

- HLUT (input LUT)

HALPS.HCPS0[2:0], HALPS.HEPS1[2:0], HALPS.HPPS2[2:0], HALPS.HLUT3[2:0], HALPS.HLUT4[2:0], HALPS.HLUT5[2:0]

The following figure shows a basic diagram of the main logic involved during LUT operation mode. If LUT mode is enabled, LUT control logic decides which value from HLUT and MLUT gets loaded to HALP and MCM registers respectively. If LUT mode is disabled the loading of HALP and MCM registers is done from a fixed register location as described in Hall Sensor standard operation section.

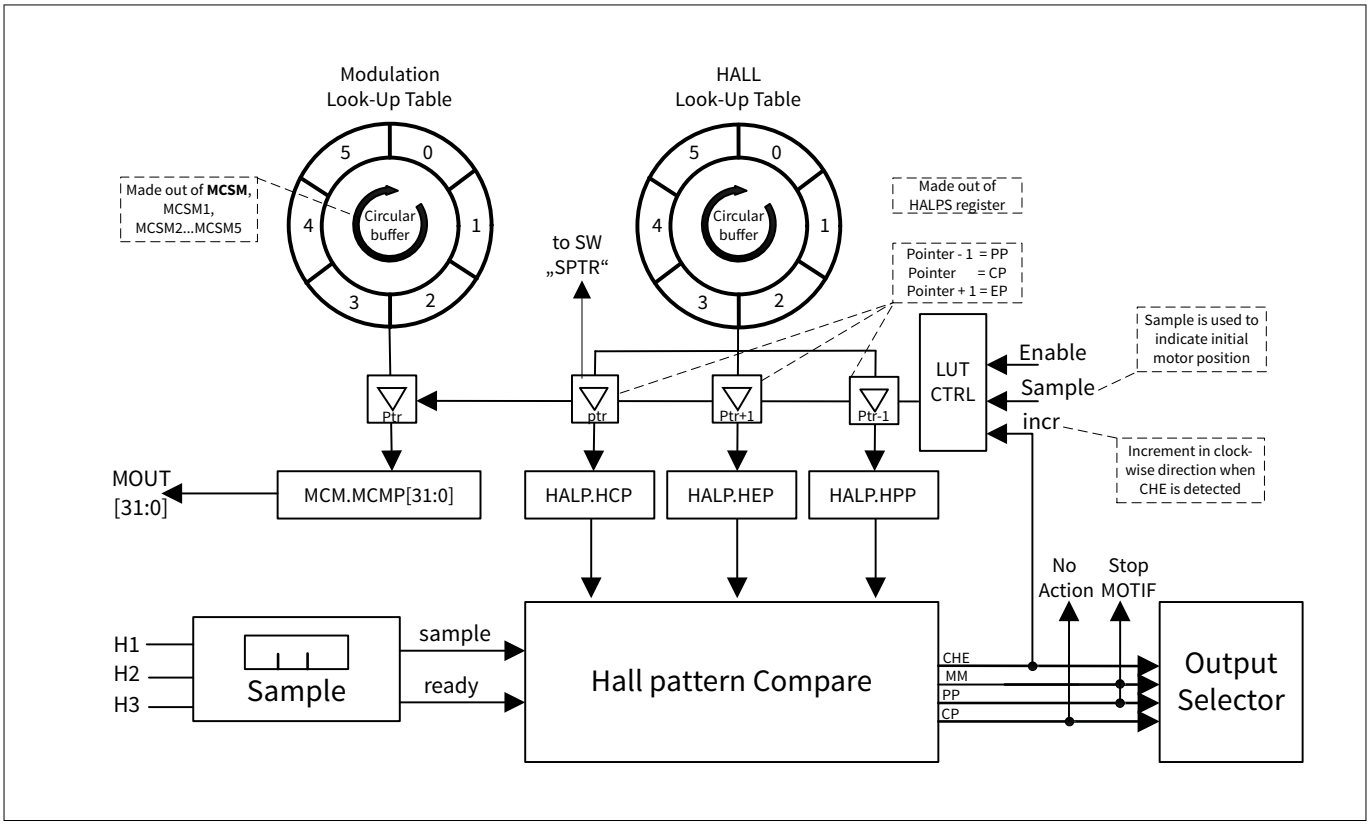


Figure 454 Hall sensor mode with LUT support

The LUT operation is a hardware-based loop, however it requires software assistance in some conditions as described below.

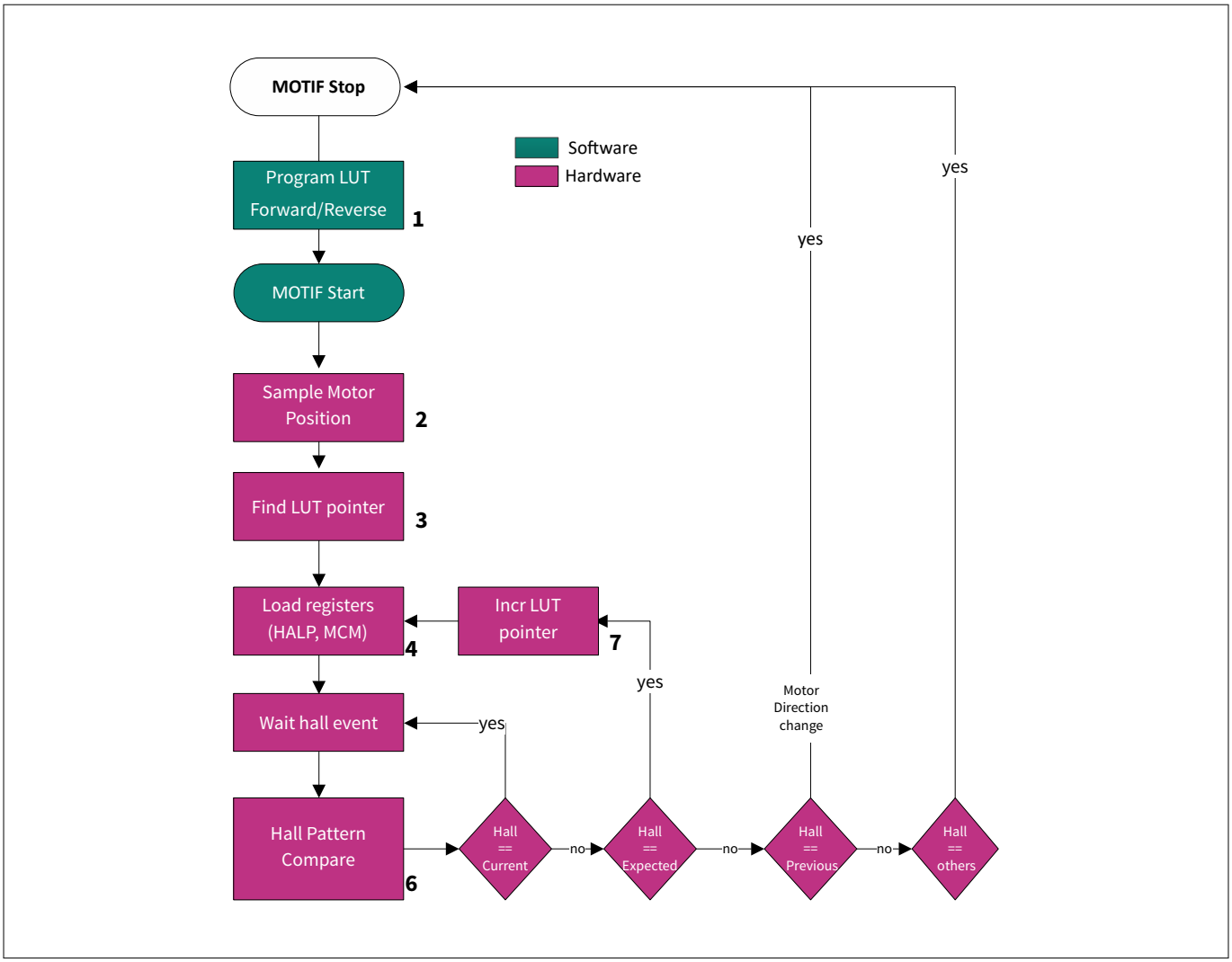


Figure 455 LUT mode - Sequence of events

STEP 1: Program LUT

Software writes HLUT and MLUT when MOTIF is stopped. Writing into both LUT while MOTIF is started might lead to incorrect behavior. The LUT is loaded with a forward or reverse pattern depending on motor desired direction. The pattern is stored in LUT independently from motor position. The hardware is responsible to find out the current motor position and to load the right hall and modulation pattern.

HLUT and MLUT share the same hardware pointer for data extraction, therefore software needs to take care to keep them aligned with respect to LUT address. A wrong programming might lead to incorrect behavior. For example, if the current motor position matches HLUT(x), the output modulation needs to be located in MLUT(x), the same index. Software must take care of keeping them aligned.

The LUT works as a circular buffer, going only in a clockwise direction over 6 addresses, 0 to 5. HLUT cannot have repeated values. All six values are expected to be unique. If the current pattern is stored in HLUT(x) then the expected pattern must be located in HLUT(x+1) and the previous pattern must be located in HLUT(x-1). The same applies for MLUT.

STEP 2: Motor position sample

The sample of HALL inputs is done automatically by hardware when the MOTIF module is started and HALL mode is enabled, this is done to obtain current motor position. Enabling LUT mode after MOTIF is started might lead to incorrect behavior. It shall be prevented by software. The motor position sample is available 2 clock cycles after the MOTIF module is started and with LUT mode enabled.

STEP 3: Find LUT pointer

After the hall sample, the hardware compares the current motor position PDBG.HSP (available from STEP 2) against all six HLUT values in a parallel way. The LUT pointer corresponds to the address which content matches the current motor position. Only one compare out of six is expected to be valid. This is because all HLUT values are expected to be unique.

If all six HALL LUT values differ from motor current position, a pattern mismatch event is generated. Hardware will stop MOTIF. The pattern mismatch event can be used by software to load a modulation fault safe pattern. An interrupt can be generated. The LUT address pointer “PTR”, which corresponds to the current pattern, is updated and can be accessible via software through the SLUT.PTR register. Valid values are 0, 1, 2, 3, 4, and 5. Invalid values 6, 7. This operation is executed only once at the starting phase after MOTIF is enabled (transit from stop to start). The LUT pointer will not be incremented at this stage as the purpose is to search and align the index of the LUT.

STEP 4: Load Registers (HALP, MCM).

Automatic shadow transfer takes place to load HALP and MCM registers with LUT pointer index. After the correct hall event is detected, a delay can be generated between this detection and the update of the multi-channel pattern. In the Figure 451, it is demonstrated the control logic of the multi-channel mode.

The delay for the update of the multi-channel pattern can be controlled directly by a TCPWM counter. The trigger that indicates that the delay is finished is mapped to input MOTIFx.MSET. One can also select the active edge for the trigger via PCONF.MSES. This delay insertion can be bypass by setting the MCMEC.MEBP bitfield to 1_B.

The PCONF.MCUE field selects the source that enables an update of the multi-channel pattern. When set to 1B, the multi-channel pattern can only be updated after the software has written a 1B into the MCMS.MNPS field.

After the update delay (if enabled), the multi-channel pattern still needs to be synchronized with the PWM signal. For this, the input MOTIFx.MSYNC is used. One can also select the active edge for the trigger via PCONF.MSYES. When the configured edge is detected in this signal, then the new multi-pattern is applied at the MOTIFx.MOUT[31:0] outputs from the output look-up table. This synchronization stage can also be bypass by setting the MCMEC.MSBP to 1_B.

STEP 5: Wait hall event

When the sampling trigger is sensed, the hall inputs are stored and compared against the current pattern, HALP.HCP, the expected hall pattern, HALP.HEP, and the previous pattern HALP.HPP.

Step 6: Hall pattern compares

If hall input == expected pattern:

- MCUE = 0 needs to be set by software
- MOTIF runs freely without software requiring to update LUT
- CHE is generated, next values from LUT are loaded to HALP, MCM
- Output modulation MCM.MCMP = MCSM
- Go to step 7

If hall input == current pattern:

- MOTIF runs freely without software requiring to update LUT
- No action is taken (line glitch deemed to have occurred)
- Output modulation MCM.MCMP = keep value
- Back to step 5

If hall input == previous pattern:

- MOTIF is stopped by hardware
- PHP is generated to software to indicate motor direction change
- Output modulation MCM.MCMP = keep value
- Back to step 0 (stop MOTIF)

If hall input != current, expected, previous patterns

- MOTIF is stopped by hardware
- WHE is generated to software
- Output modulation MCM.MCMP = MCPF.MCFV (safe pattern)
- Back to step 0 (stop MOTIF)

STEP 7: Increment LUT pointer

If the correct hall event is decoded, LUT pointers are incremented by 1. PTR always points to the current hall pattern.

- PTR = Current pattern
- PTR + 1 = Expected pattern
- PTR - 1 = Previous pattern

The LUT address pointer “PTR”, which corresponds to the current pattern, is updated and can be accessible via software through the SLUT.PTR register. Valid values are 0, 1, 2, 3, 4, and 5. Invalid values 6, 7. This operation is performed every time a correct hall event is detected.

The following table summarizes the following registers that have an impact or relationship with LUT Mode:

Table 347 Register relationship for LUT operation

Register	Description	LUT mode usage
PCONF.FTS	LUT mode can use a fault hall event signal for the operation	Can be used in LUT mode
PCONF.CHEC	LUT mode can use correct hall event clear selection	Can be used in LUT mode
PCONF.HIDG	LUT mode will only use the WHE signal for the operation when PCONF.HIDG = 1 LUT mode will use WHE and PIF_FAULT for the operation when PCONF.HIDG = 0	Can be used in LUT mode
PCONF.EWIE	External wrong hall event enable (Used with PCONF.FTS) 0 - External wrong hall event emulation signal, MOTIFx.EWHE, is disabled 1 - External wrong hall event emulation signal, MOTIFx.EWHE, is enabled.	Can be used in LUT mode

(table continues...)

Table 347 (continued) Register relationship for LUT operation

Register	Description	LUT mode usage
PCONF.MCUE	The PCONF.MCUE field selects the source that enables an update of the multi-channel pattern. When set to 1B, the multi-channel pattern can only be updated after the software has written a 1B into the MCMS.MNPS field. Multi-channel pattern software update enable 0 - Multi-channel pattern update is controlled via hardware 1 - Multi-channel pattern update is controlled via software	Can be used in LUT mode
HMEC.HPPE	Will not have an impact on the LUT mode, the hardware will use WHE signal for the operation	Cannot be used in LUT Mode
MCMEC.MNPS	Multi-channel pattern can only be updated after the software has written a 1 into the MCMS.MNPS field provided PCONF.MCUE is set to 1.	Can be used in LUT Mode

33.5.4 Quadrature decoder control

The quadrature decoder mode is selected by setting PCONF.FSEL = 01_B or PCONF.FSEL = 11_B (in this case the multi-channel mode is also enabled).

Inside the quadrature decoder mode, two different subsets are available:

- Standard quadrature mode
- Direction count mode

The standard mode is used when the external rotary encoder provides two-phase signals and additionally and index/marker signal that is generated once per shaft revolution. The direction count mode is used when the external encoder only provides a clock and a direction signal.

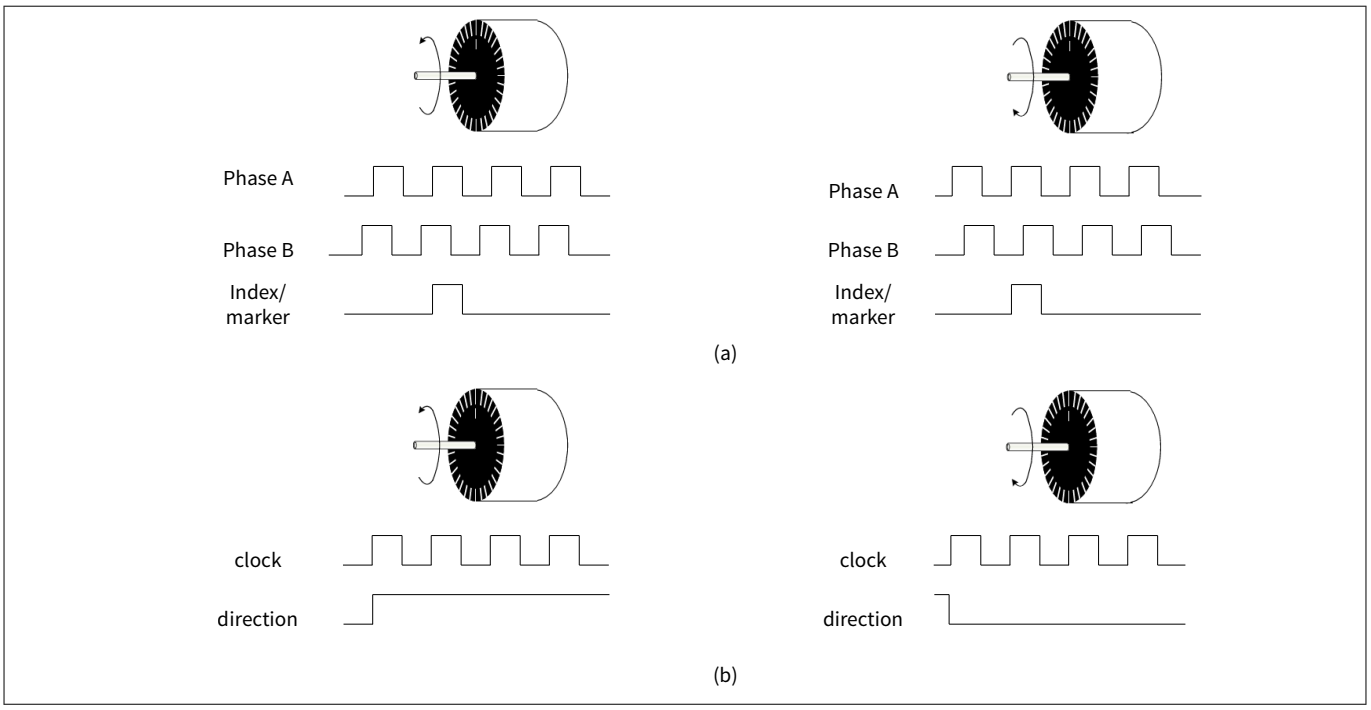


Figure 456 Rotary encoder types - a) standard two phase plus index signal; b) clock plus direction

Standard quadrature mode

The quadrature decoder unit offers a very flexible phaseA/phaseB configuration stage. Normally for a clockwise motor shaft rotation, phase A should precede phase B but nevertheless, you can configure the leading phase as well as the specific active state for each signal, see Figure 457

The selection of the phase A, phase B and Index signals are done via the QDC.PACS, QDC.PBDS, and QDC.IDS control fields. There are two major blocks that build the quadrature decoder control unit: the block that decodes the quadrature clock and motor shaft direction and the block that handles the index (motor revolution) control.

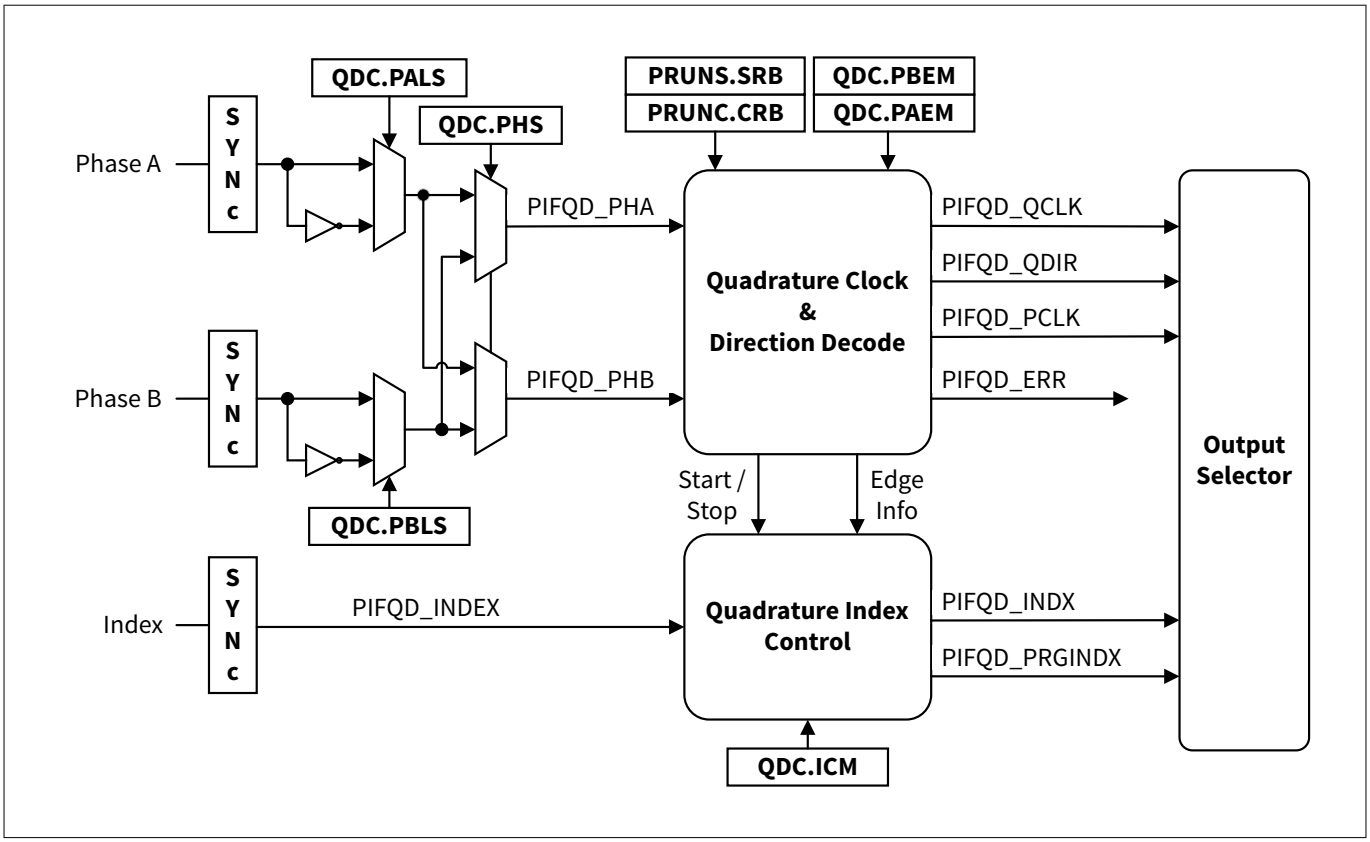


Figure 457 Quadrature decoder control overview

The quadrature clock can be used for position measurement and it can be forwarded to any of the MOTIFx.OUTy outputs. This clock can be decoded from every edge of the phase signals and therefore there can be 4 clock pulses per phase period. The edges (from the Phase A and Phase B signals) that are used to decode the quadrature clock signal can be configured by the application software.

The following figures show several examples for the clock decoding from the phase A and phase B signals. The configuration of which edges, rising, falling, both or none, is done via the QDC.PAEM and QDC.PBEM (for phase A and phase B accordingly). Please take note that when QDC.PHS is set to '1', QDC.PAEM is used to configure Phase B and QDC.PBEM is used to configure phase A.

The following diagram shows the X1 clock generation behavior.

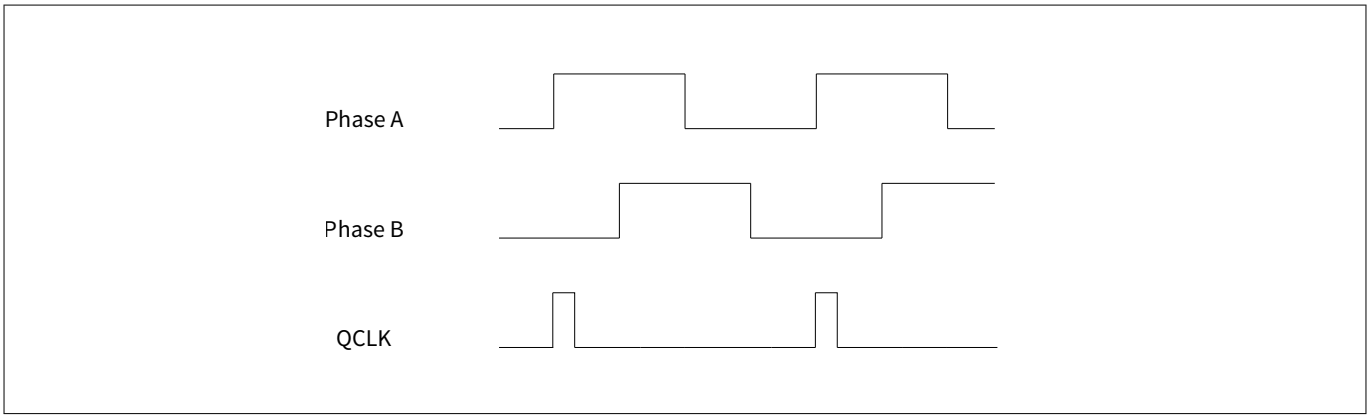


Figure 458 X1 clock generation PAEM=RISE/PBEM=NONE

The following diagram shows the X2 clock generation behavior.

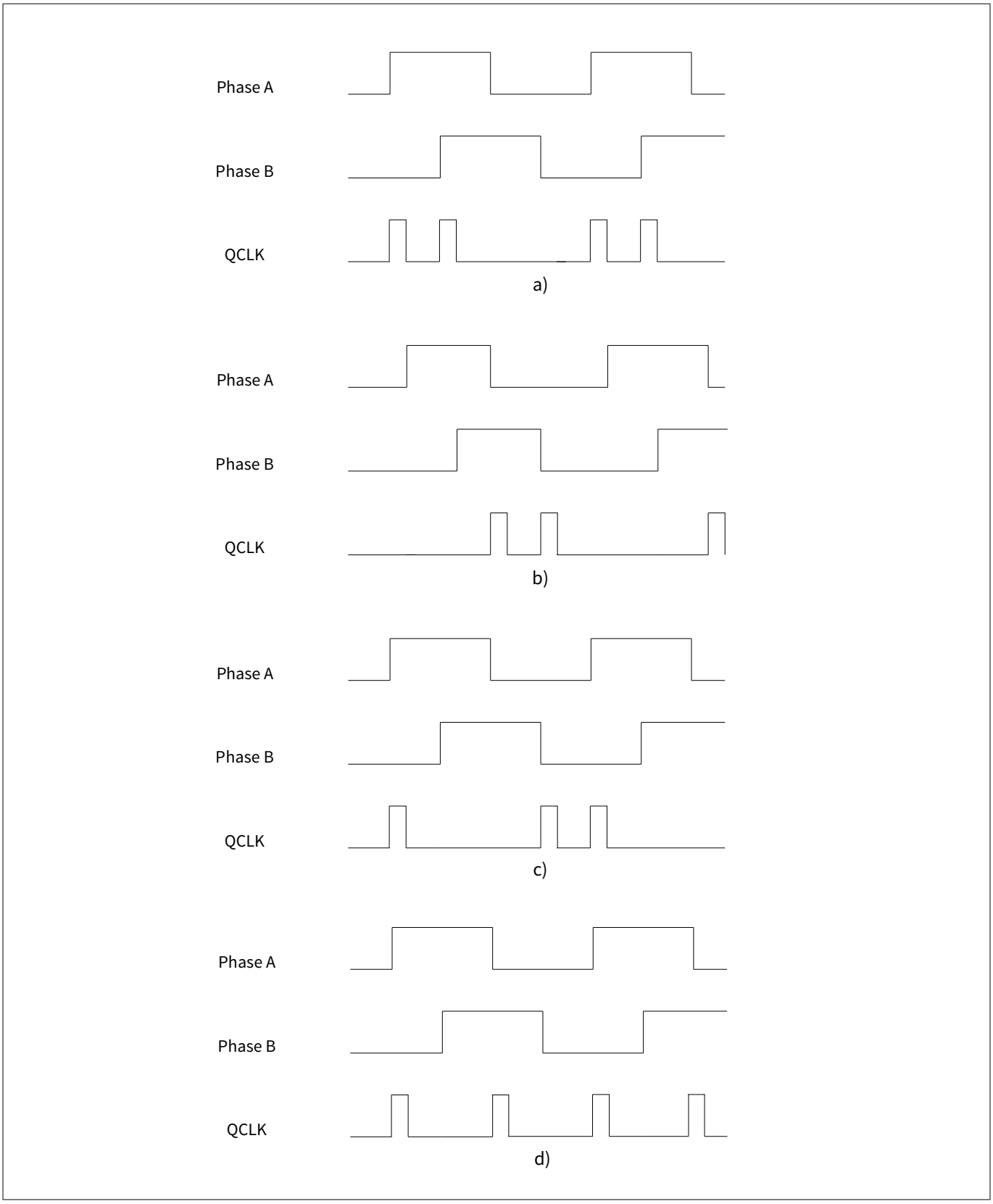


Figure 459 X2 clock generation: a) PAEM=RISE/PBEM=RISE; b) PAEM=FALL/PBEM=FALL; c) PAEM=RISE/PBEM=FALL; d) PAEM=BOTH/PBEM=NONE

The following diagram shows the X4 clock generation behavior.

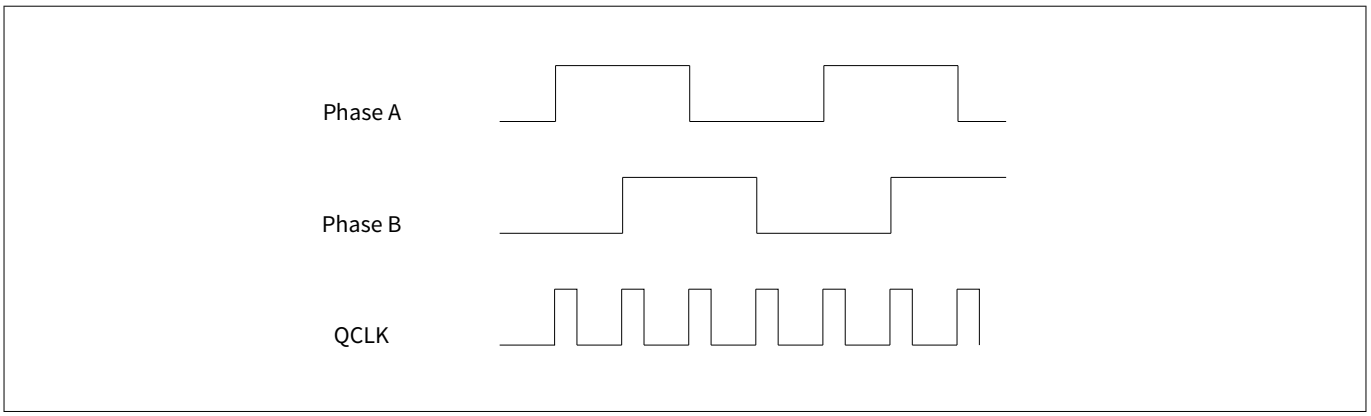


Figure 460 X4 clock generation PAEM=BOTH/PBEM=BOTH

A period clock is also generated for velocity measurement operations that can be forwarded to any of the MOTIFx.OUTy outputs. The decoding of this clock is also affected by the QDC.PAEM and QDC.PBEM fields. The direction of the motor rotation is decoded from the Phase A and Phase B signals - QDIR - and it can also be forwarded to any of the MOTIFx.OUTy outputs. The direction signal is asserted HIGH when the motor is rotating clockwise and LOW when it is turning in the counterclockwise direction.

The index control logic memorizes which was the first edge after the assertion of the index signal, so the same quadrature transition is used for index event operations. It also memorizes the direction of the first index, so that it can control when the revolution increment signal should be asserted. Two index events can be forwarded to the MOTIFx.OUTy outputs - the INDX event and the PRGINDX event - Index Control for a complete description.

An error signal, connected to a flag (and if enable an interrupt can be generated) can also be generated when a wrong phase pair is detected (this event is only available as an interrupt and it cannot be forward to the MOTIFx.OUTy outputs).

The quadrature decoder control uses the information of the current and previous phase pair to decode the direction and clocks. Both phase signals pass through an edge detection logic from which the outputs are going to be used against the valid/invalid transitions stages, see Figure 461. There is the possibility to reset the decoder state machine (but not the flags and retention mmio registers) by writing a 1_B into the PCONF.ENABLE field. The quadrature decoder current state, previous state, and current index value can be read via the PDBG register.

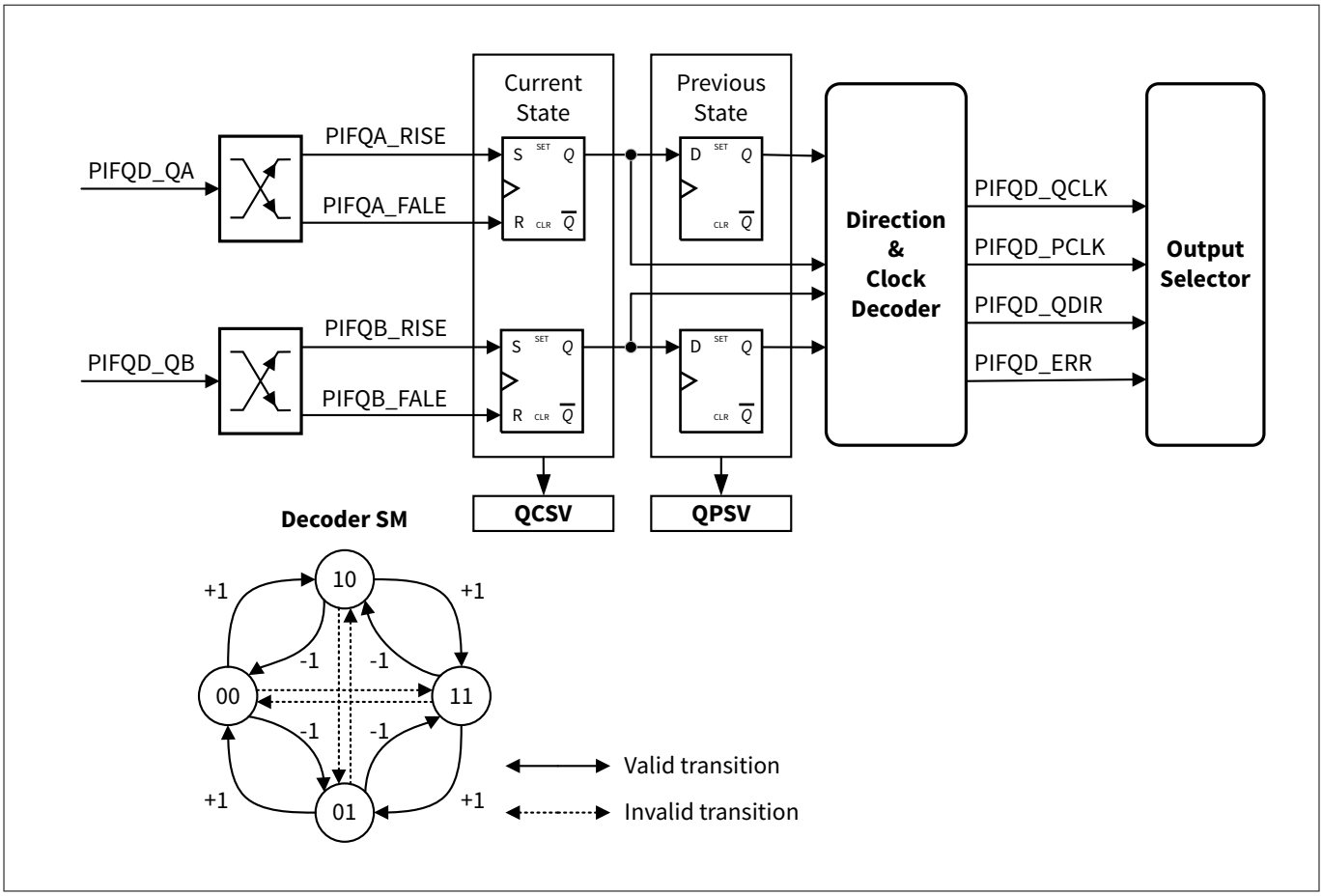


Figure 461 Quadrature decoder states

Direction count mode

Some position encoders do not have the phase signals as outputs, instead, they provide two signals that contain the clock and direction information. This is known as the Direct Count Mode.

When using a position encoder of this type, the user should set the PCONF.QDCM bit field to 1_B (enabling the direction count mode). In this case, the user must select to which of the input groups - MOTIFx.INy - the clock and direction signals are connected. This is done via the QDC.PACS and QDC.PBDS control fields. The input signals - direction and clock - are synchronized with the MOTIF module clock and shall be low pass filtered if enabled before they are forwarded to any of the MOTIFx.OUTy outputs, via the output selector. Notice that the direction input is not obligatory. One can disable it by setting the QDC.PBDS to 00_B. The output signals PCLK, INDX, PRGINDX, ERR are set to constant zero in this mode.

33.5.4.1 Quadrature clock and direction decoding

The quadrature decoder unit outputs two clocks. One clock is used for position control and therefore is generated in each edge transition of the phase signal - QCLK. The second clock is used for velocity measurements and is immune to possible glitches on the phase signals that can be present at very slow rotation speeds causing motor vibrations - PCLK.

The glitches due to motor vibrations are not necessarily electrical noise glitches. This type of vibration glitch is slow enough that it cannot be de-glitched by input filters, refer Figure 463.

The PCLK is not generated when motor direction changes (transition on QDIR). Exception is that at the initial start of motor operation. PCLK is generated for the first transition of phase A or phase B.

The decoding of the direction signal is done following the rules in Figure 461. Figure 462 shows all the valid transitions of phase A and phase B signals. Figure 464 shows the difference between the two clock signals in the case that some glitches are present in the phase signals.

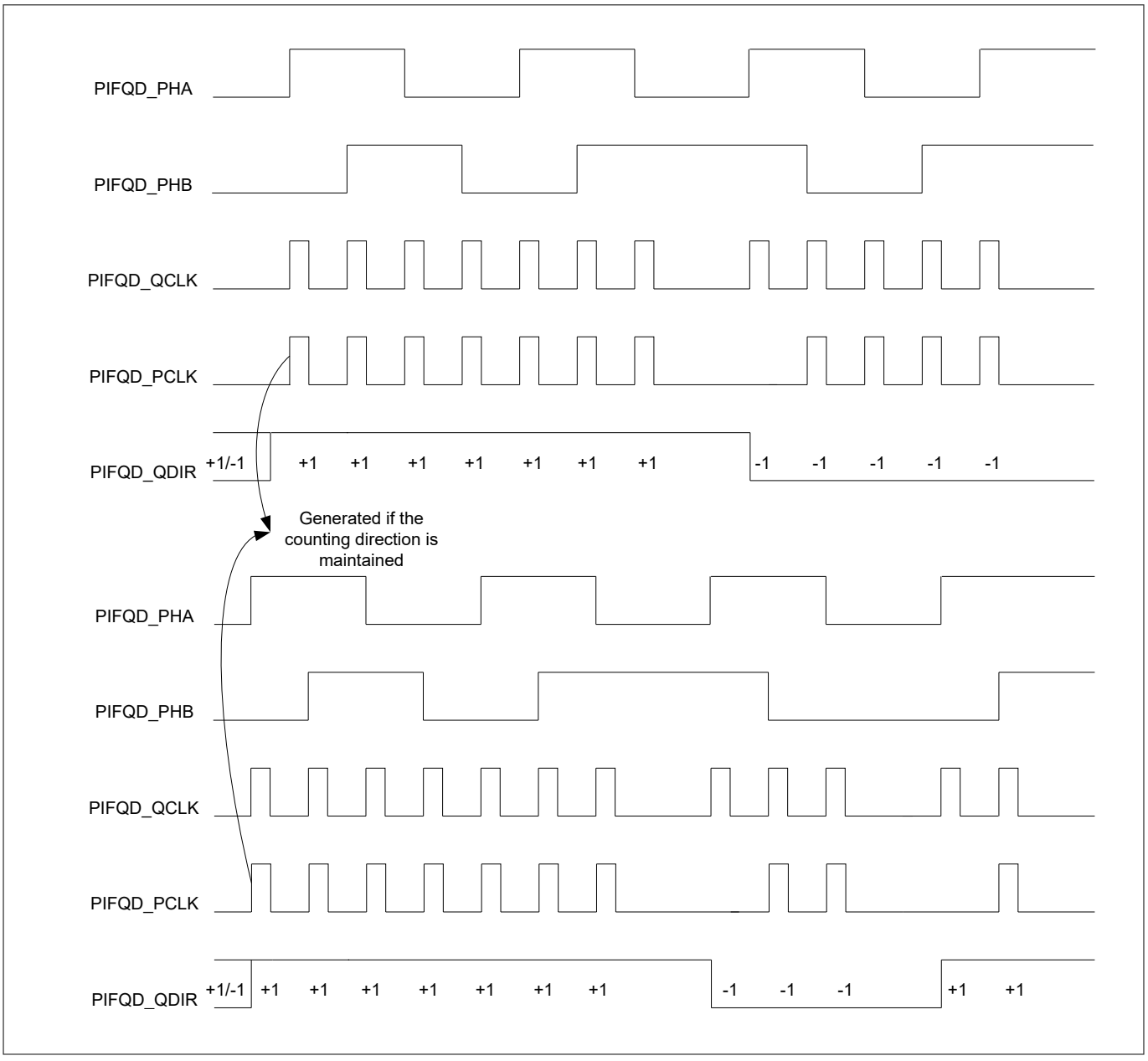


Figure 462 Quadrature clock and direction timings

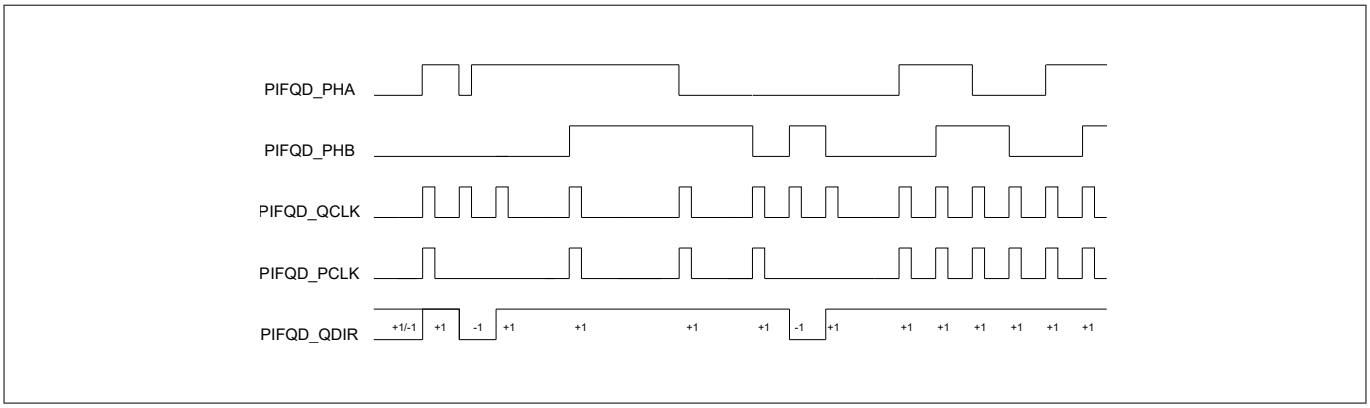


Figure 463 Quadrature clock with jitter

33.5.4.2 Index control

The index control logic has two different output events. One is asserted every time that an index signal is detected (and the motor shaft rotation is the same) - INDX – and therefore it can be used in a revolution counter. The software can monitor not only the position of the shaft but also the total number of revolutions that have occurred.

The activity of the other event - PRGINDX - can be programmed via the QDC.ICM field. Depending on the value set in the QDC.ICM, this signal can be generated:

- In every index signal occurrence
- Only on the first index signal occurrence
- Disabled - this output is never asserted

This is useful for applications that need to reset the counters on every index event or only at the first one.

The Index control logic memorizes the immediate next phase edge that follows an index so the generated signals have always the same reference. If the first-phase edge after the index is the rising edge of the phase B signal, then the index signals are going to be generated in the next index event with the rising edge of the phase B signal if the direction is kept or with the falling edge of the phase B signal if the direction has changed.

***Note**: If a x2 clock generation (for example, where only the rising edge of Phase A or Phase B is used to generate the quadrature clock) index decoding follows the same behavior as mentioned above, no index signal pulse is lost.*

Figure 464 shows the timing diagram for the generation of the index signals. In this case the QDC.ICM = 10_B, which means that the PRGINDX event is going to be generated in every occurrence of the input index.

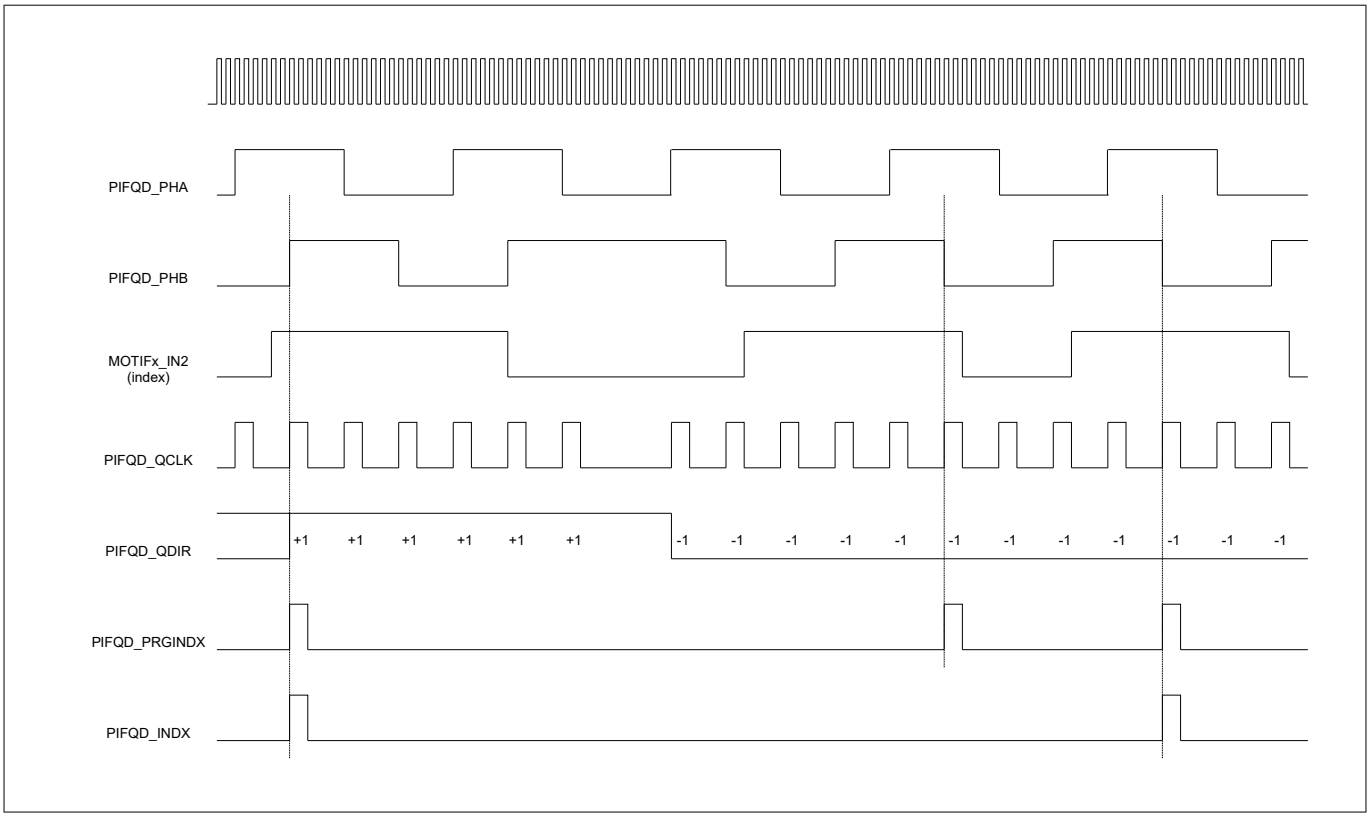


Figure 464 Index signals timing

33.5.5 Stand-alone multi-channel mode

The multi-channel mode can be used without the hall sensor control by setting the PCONF.FSEL = 10_B. The stand-alone multi-channel mode shall support two shadow transfer mechanisms for updating the multi-channel pattern output, one is called single shadow transfer, and the other is called Look-up table shadow transfer.

Single shadow transfer

The single shadow transfer mechanism is the same as the one described in hall sensor control section. The trigger for a pattern update can come from the software, by writing 1_B to MCMS.MNPS, or it can come from an external signal. This external signal should be mapped to the PIFMSET function by selecting the appropriate value in the PCONF.MSETS field, see Figure 451.

The synchronization between the update of the new multi-channel pattern and control signal may be also enabled by setting the MCMEC.MSBP to 0_B – the input signal MOTIFx.MSYNC is used to this function, see Figure 451.

The multi-channel pattern can be clear/load with the value preprogrammed in the MCPF.MCFV register into the MCM.MCMP - whenever the software writes a 1_B into the MCMC.MPC bitfield or if enabled when a fault is detected in the MOTIFx.EWHE module pin, see Figure 450. The value preprogrammed in the MCPF.MCFV is then applied to the MOTIFx.MOUT[31:0] outputs.

Look-up Table shadow transfer

The stand-alone multi-channel mode shall support an update of the output pattern modulation via look-up table (output LUT). To enable this operation, the LUTEN register shall be programmed to ‘1’.

The output LUT can be programmed by software, and can store up to 6 different modulation patterns, each of them consists of 32 bits. The hardware applies one modulation pattern at a time on MOTIF.MOUT[31:0] pin, this happens whenever a trigger for pattern update is received.

The following figure illustrates how the LUT is linked to the MOUT pin.

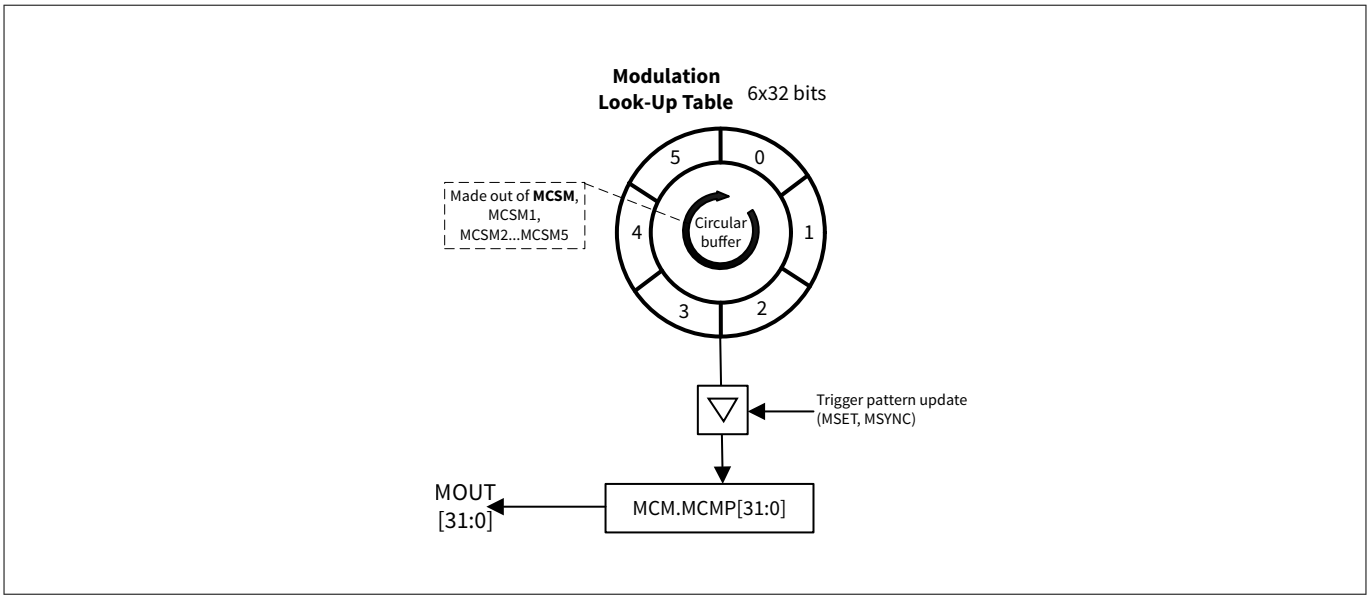


Figure 465 Stand-alone multi-channel with LUT support

The trigger for a pattern update can come from software, by writing 1_B to MCMS.MNPS, or it can come from an external signal. This external signal should then be mapped to the PIFMSET function by selecting the appropriate value in the PCONF.MSETS field. The synchronization between the update of the new multi-channel pattern and control signal may be also enabled by setting the MCMEC.MSBP to 0_B – the input signal MOTIFx.MSYNC is used to this function.

The hardware shall access the LUT content in a circular way. It starts from position 0 up to 5 and repeat again from 0. Every time a trigger for a pattern update is received the LUT address is incremented by 1.

The mapping between LUT addresses and LUT data registers is the following:

- LUT [0] corresponds to MCSM
- LUT [1] corresponds to MCSM1
- LUT [2] corresponds to MCSM2
- LUT [3] corresponds to MCSM3
- LUT [4] corresponds to MCSM4
- LUT [5] corresponds to MCSM5

The following timing diagram illustrates the operation of stand-alone multi-channel with LUT enabled. The MSET, MSYNC condition is used to increment by 1 the LUT address and also to transfer the content from LUT[x] to MOUT[31:0] output. The multi-channel pattern can be clear/load with the value preprogrammed in the MCPF.MCFV register into the MCM.MCMP - whenever the software writes a 1_B into the MCMC.MPC bitfield. Whenever the MOTIFx.EWHE input event is received, the MOTIF module is stopped and the value preprogrammed in the MCPF.MCFV is applied to MOUT[31:0] outputs. Software needs to set PCONF.HIDG=0, PCONF.FTS=0/2, MCMEC.MFCE=1 for this behavior.

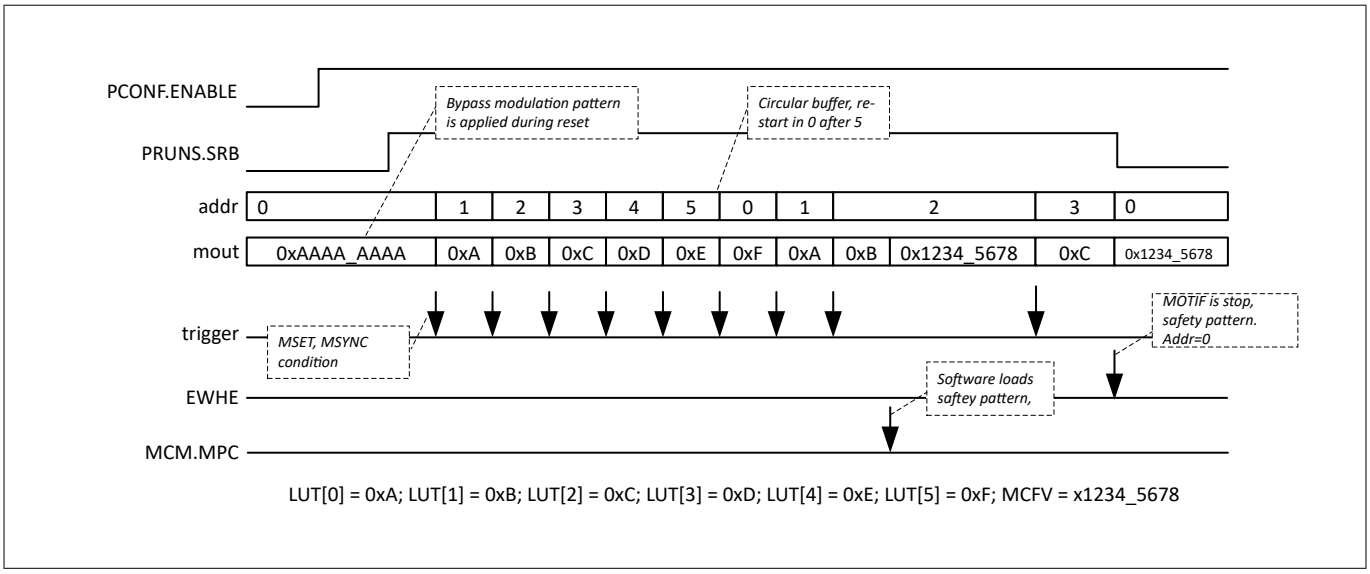


Figure 466 Timing stand-alone modulation with LUT support

***Note**: In single shadow transfer mode, the shadow transfer to MOTIF.MOUT pin always takes place from the MCSM register. Software must ensure to re-write a new pattern into this register after shadow transfer.*

In LUT mode, the software shall program the output LUT only once and the hardware shall loop through the 6-address starting from address 0. The process is stopped when MOTIF is stopped.

33.5.6 Modulation interface from MOTIF

Each MOTIF module has a modulation output. This is a 32bit output containing the multi-channel pattern. The modulation output is used in hall sensor mode with multichannel mode, stand-alone multi-channel mode and quadrature with stand-alone multi-channel mode.

The modulation interface is a direct wire connection from each MOTIF module to a predefined set of group counters. This is done to reduce the routing congestion on top level.

Each MOTIF modulation output can connect with a maximum of 32 counters channels. For all these counters, line_out and line_compl_out can be modulated via MOTIF in a synchronous way.

Table 348 shows how MOTIF0.MOUT[31:0] is connected to GRP1.COUNTER[7…0]. 4 bits, which are required to modulate line_out and line_compl_out signals.

Table 348 MOTIF0 modulation output mapping to GRP1.COUNTERs

MOTIF Modulation Output	COUNTER modulation input	Output line modulation
MOTIF0.MOUT[1:0]	GROUP1.COUNTER [0]	Line_out
MOTIF0.MOUT[3:2]	GROUP1.COUNTER [0]	Line_compl_out
MOTIF0.MOUT[5:4]	GROUP1.COUNTER [1]	Line_out
MOTIF0.MOUT[7:6]	GROUP1.COUNTER [1]	Line_compl_out
MOTIF0.MOUT[9:8]	GROUP1.COUNTER [2]	Line_out
MOTIF0.MOUT[11:10]	GROUP1.COUNTER [2]	Line_compl_out
MOTIF0.MOUT[13:12]	GROUP1.COUNTER [3]	Line_out
MOTIF0.MOUT[15:14]	GROUP1.COUNTER [3]	Line_compl_out

(table continues...)

Table 348 (continued) MOTIF0 modulation output mapping to GRP1.COUNTERs

MOTIF Modulation Output	COUNTER modulation input	Output line modulation
MOTIF0.MOUT[17:16]	GROUP1.COUNTER [4]	Line_out
MOTIF0.MOUT[19:18]	GROUP1.COUNTER [4]	Line_compl_out
MOTIF0.MOUT[21:20]	GROUP1.COUNTER [5]	Line_out
MOTIF0.MOUT[23:22]	GROUP1.COUNTER [5]	Line_compl_out
MOTIF0.MOUT[25:24]	GROUP1.COUNTER [6]	Line_out
MOTIF0.MOUT[27:26]	GROUP1.COUNTER [6]	Line_compl_out
MOTIF0.MOUT[29:28]	GROUP1.COUNTER [7]	Line_out
MOTIF0.MOUT[31:30]	GROUP1.COUNTER [7]	Line_compl_out

The modulation logic is available on all counter groups, if enabled.

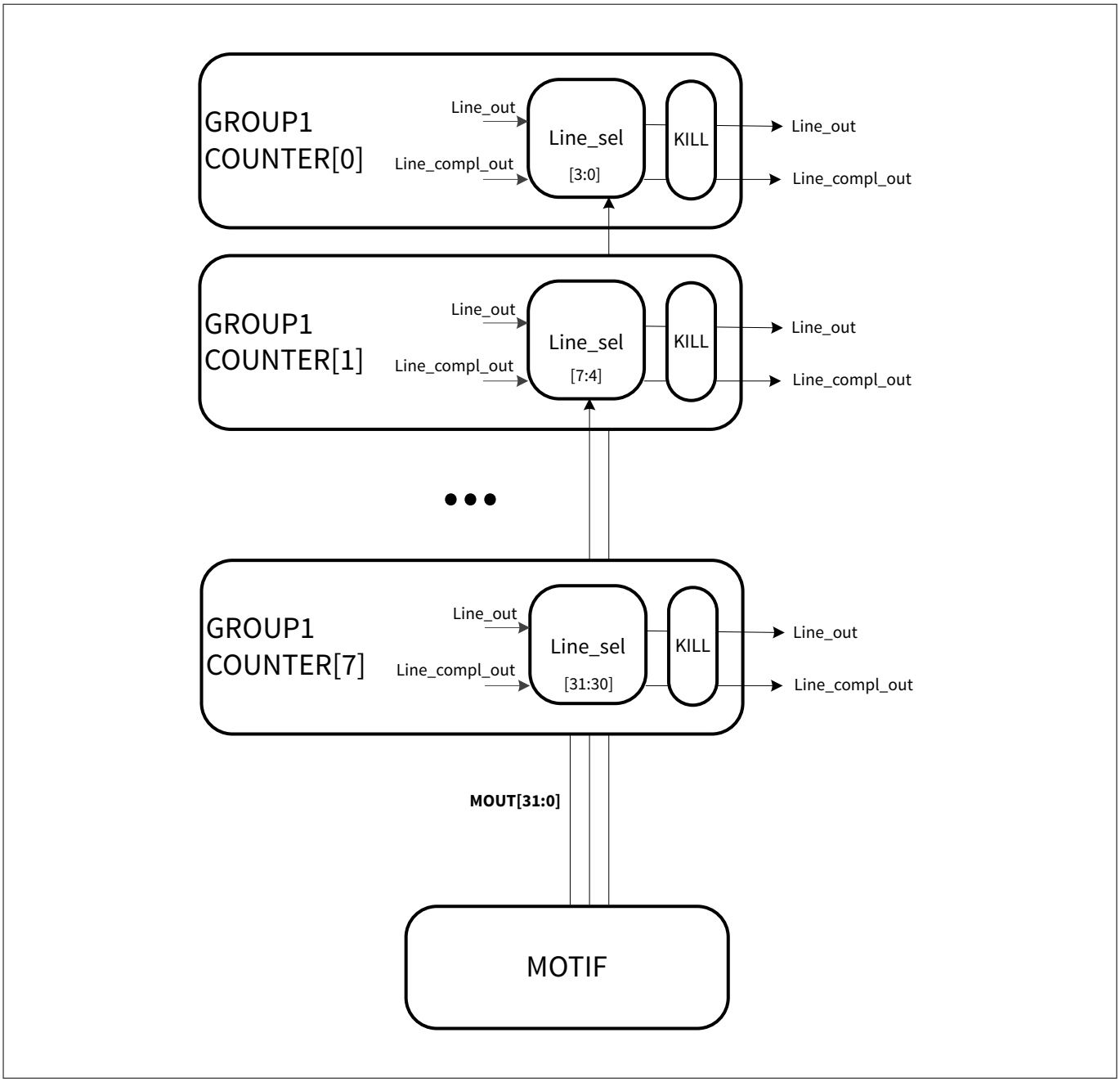


Figure 467 MOTIF0 modulation connection to group 1 counters

The 32-bit multi-channel pattern outputs can affect the PWM output of the associated timer counter in two different ways:

- AND gating the PWM - forcing the PWM to passive
- OR gating the PWM - forcing the Inv. PWM to active

Each counter PWM output receives two signals controlled by the multi-channel pattern outputs. One signal dictates if an OR or an AND is needed, and another signal is used as the second input for the function. The linking of the multi-channel pattern outputs with a timer PWM output is as described in the Table 349.

Table 349 Multi-channel pin influence on output

Multi-channel signal	Value	PWM output behavior on the counter
MOTIF0.MOUT[1]	0	PWM AND gated with MOTIFx.MOUT[0]
	1	Inv. PWM OR gated with MOTIFx.MOUT[0]
MOTIF0.MOUT[3]	0	PWM AND gated with MOTIFx.MOUT[2]
	1	Inv. PWM OR gated with MOTIFx.MOUT[2]
MOTIF0.MOUT[5]	0	PWM AND gated with MOTIFx.MOUT[4]
	1	Inv. PWM OR gated with MOTIFx.MOUT[4]
MOTIF0.MOUT[7]	0	PWM AND gated with MOTIFx.MOUT[6]
	1	Inv. PWM OR gated with MOTIFx.MOUT[6]
MOTIF0.MOUT[9]	0	PWM AND gated with MOTIFx.MOUT[8]
	1	Inv. PWM OR gated with MOTIFx.MOUT[8]
MOTIF0.MOUT[11]	0	PWM AND gated with MOTIFx.MOUT[10]
	1	Inv. PWM OR gated with MOTIFx.MOUT[10]
MOTIF0.MOUT[13]	0	PWM AND gated with MOTIFx.MOUT[12]
	1	Inv. PWM OR gated with MOTIFx.MOUT[12]
MOTIF0.MOUT[15]	0	PWM AND gated with MOTIFx.MOUT[14]
	1	Inv. PWM OR gated with MOTIFx.MOUT14]
MOTIF0.MOUT[17]	0	PWM AND gated with MOTIFx.MOUT[16]
	1	Inv. PWM OR gated with MOTIFx.MOUT[16]
MOTIF0.MOUT[19]	0	PWM AND gated with MOTIFx.MOUT[18]
	1	Inv. PWM OR gated with MOTIFx.MOUT[18]
MOTIF0.MOUT[21]	0	PWM AND gated with MOTIFx.MOUT[20]
	1	Inv. PWM OR gated with MOTIFx.MOUT[20]
MOTIF0.MOUT[23]	0	PWM AND gated with MOTIFx.MOUT[22]
	1	Inv. PWM OR gated with MOTIFx.MOUT[22]
MOTIF0.MOUT[25]	0	PWM AND gated with MOTIFx.MOUT[24]
	1	Inv. PWM OR gated with MOTIFx.MOUT[24]
MOTIF0.MOUT[27]	0	PWM AND gated with MOTIFx.MOUT[26]
	1	Inv. PWM OR gated with MOTIFx.MOUT[26]
MOTIF0.MOUT[29]	0	PWM AND gated with MOTIFx.MOUT[28]
	1	Inv. PWM OR gated with MOTIFx.MOUT[28]
MOTIF0.MOUT[31]	0	PWM AND gated with MOTIFx.MOUT[30]
	1	Inv. PWM OR gated with MOTIFx.MOUT[30]

If MOTIF gets disabled via PCONF.ENABLE=0, the modulation output from MOTIF gets reset to a default value. The reset value bypasses the PWM inside the counter and has not effect on the PWM signal.

MOTIFx.MOUT = 0xAAAA_AAAA

If MOTIF gets stopped via PRUNS.SRB=0, the modulation output from MOTIF will keep the existing value.

33.5.6.1 Modulation inside counter logic

The PWM signal conditioning can be done via MOTIF. The counter MMIO LINE_SEL register needs to be configured to select MOTIF as the input source for modulation.

Figure 468 illustrates the line out generation logic of PWM, PWM_DT modes.

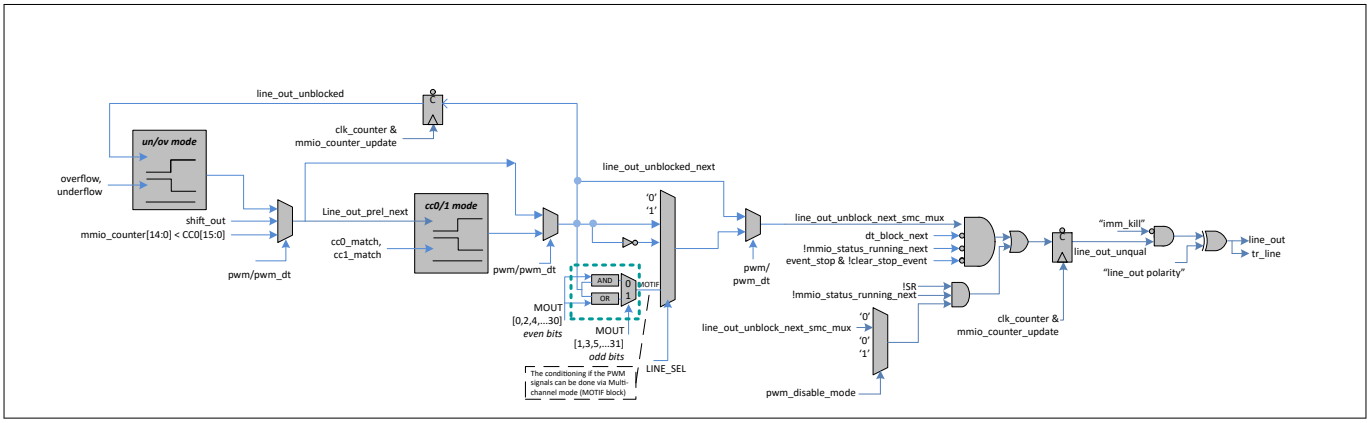


Figure 468 line_out generation logic

MOTIF interface only supports 0, 1 and PWM currently. The MOTIF output mapping would be:

Table 350 MOTIF output mapping for line_out

Motif output mapping	Line_out
mout[1],mout[0]	
00	0
01	PWM
10	Reserved
11	1

Similarly line_compl_out would also be updated as:

Table 351 MOTIF output mapping for line_compl_out

Motif output mapping	Line_compl_out
mout[3],mout[2]	
00	0
01	PWM
10	Reserved
11	1

33.5.7 Synchronous start

The MOTIF module has a synchronous start event that can be forwarded to any of the MOTIFx.OUTy outputs via the output selector. This event can be used together with the TCPWM counters for a complete synchronous start

of both modules. The synchronous start is linked with the run bit of the MOTIF module, which means that every time that the run bit is set a transition is generated that can be forward to one of the outputs.

By using the synchronous start output, the software does not perform two independent accesses to start the MOTIF and the TCWPM counters and therefore is guaranteed that both modules start their operation at the exact same time.

33.5.8 Output selector

The output selector receives the signals generated by the three main sub units comprised inside the MOTIF:

- signals from the quadrature (or direction count) control logic
- signals from the hall sensor control logic
- signals from the multi-channel control logic

The first stage of the output selector, as seen in Figure 469 will gate the signals that are not being used depending on the function that has been selected by the PCONF.FSEL field (this field dictates which MOTIF sub unit is being currently used - quadrature mode, hall sensor mode, multi-channel mode). If a specific sub unit is not being used, t. Generate signals coming from this control logic will be disabled.

On the second stage of the output selector, one can use the node pointers to forward the specific signal/event, generate by a sub unit to the wanted MOTIF output pin – MOTIFx.OUT[0...6]. The node pointer configurations are handled via the QOSC, HOSC and MOSC registers, for the quadrature decoder, hall sensor and multi-channel control logic respectively.

An OR stage will group all the signals/events that have been forward to the same MOTIF output pin - for example, if the QCLK and Start (from the quadrature control logic) have been configured in their node pointer, to be forward to MOTIFx.OUT0. Then at this specific pin, one should expect to see both events ORed (an event is going to be seen whenever a QCLK is generated or when a Start is detected).

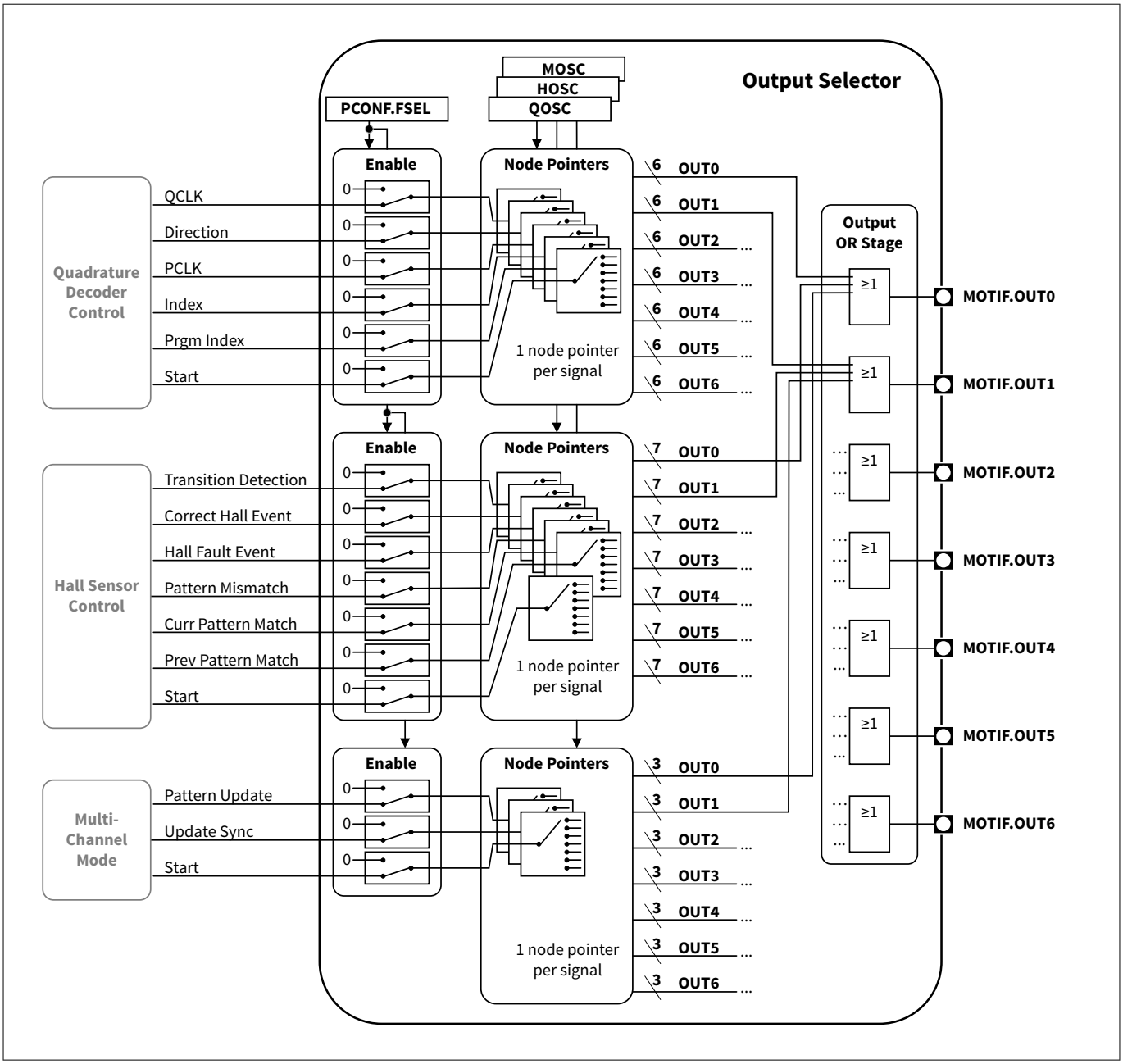


Figure 469 Output selector logic diagram

33.5.9 Using the MOTIF

The MOTIF module needs to be linked with a TCPWM counters module to perform the full set of functions in each of the possible modes (because it does not contain built-in counters/timers).

To operate the MOTIF in the quadrature decoder mode (at least) five counters are needed. The hall sensor mode needs at least 3 counters to control a brushless DC motor and also (at least) two counters to control delay. The stand-alone multi-channel mode linking configuration depends heavily on the use cases and therefore the number of counters used is freely chosen by the user.

***Note**: The configurations that are defined in the examples described in this section do not reflect product configuration. User shall refer to datasheet and feature chapter for actual implementation for the device.*

Hall sensor mode usage:

When using the hall sensor mode of the MOTIF, the multi-channel mode is also working to provide the multi-channel pattern to TCWPM counters. The counter module performs the multi-channel PWM Modulation. For this purpose, the counter needs to be configured via LINE_SEL register to select MOTIF as the signal conditioning source for PWM signal.

In Figure 470, the hall sensor mode is used with 2 groups of counters with five counters in total. Group 1 is configured with advance motor control features (three counters). Group 0 is configured as a standard General-purpose counter (two counters). Group 0 counters is being are used to control the delays between the hall transition detection of the hall inputs and the actual sampling, and also to control the delay between a correct hall event and the multi-channel pattern update enable.

The rising edge of Group 0 counter 0 tr_out0 is used as the finish trigger for the first delay. It connects to MOTIF input HSD. While Group 0 counter 0 tr_out1 is used as the delay between the hall input sampling and the modulation pattern update, it connects to MOTIF input MSET. Group 0 Counter 0 is configured in single shot mode, so that the time delay can be re triggered every time that a request from the MOTIF occurs.

The Group 0 Counter 1 is being used in capture mode, to capture the time between the correct hall events (storing this way the motor speed between two correct hall events). The correct hall event of the MOTIF is used as the capture trigger for the counter while the pattern mismatch event is used as stop the Group 0 Counter 0 and Group 0 Counter 1. The capture and stop triggers are configured in the specific timer slices as active on the rising edge.

The Group 1 counters are used to generate the PWM signals to control the motor and therefore, the multi-channel pattern outputs MOTIFx.MOUT[31:0] are linked to this unit.

To close the multi-channel loop, an output of the Group 1 counters needs to be connected to the MOTIF module (tr_out0 period match), so that the multi-channel pattern is updated synchronously with the PWM period. All MOTIF input and output events are connected via trigger multiplexer, whereas MOTIF modulation output is connected internally via hardware connection to TCPWM counters Group 1.

Counter configuration:

- 2 Groups of counters
  - G0_CNTx standard counter (two counters)
  - G1_CNTx with advanced motor control (three counters)

G0_CNT0 is used to delay hall input sampling and to generate MSET signal to MOTIF (one-shot timer mode).

- G0_CNT1 is used capture mode
- G1_CNT[2:0] is used to generate 3-phase PWM (PWM right-aligned mode)

MOTIF configuration:

- OUT0 = edge detection
- OUT1 = correct hall event
- OUT2 = pattern mismatch
- OUT3 = wrong hall event
- OUT4 = start

MOTIF <> TCPWM connectivity

- OUT0 is used to reload G0_CNT0 via a 1-to-1 trigger
- OUT1 is used to capture G0_CNT1 via 1-to-1 trigger (it is also used to reload G0_CNT1 since **TCPWM does not have a capture clear feature**)
- OUT2 is used to stop G0_CNTx via group trigger
- OUT3 is used to stop G1_CNTx via group trigger
- OUT4 is used to reload G1_CNTx via group trigger
- G0_CNT0 TROUT0 (CC0 match) is routed to HSD

- G0_CNT0 TROUT1 (period match) is routed to MSET
- G1_CNT0 TROUT0 (period match) is routed to MSYNC

PWM and MOTIF are synchronously started by making use of the RUN bit via OUT4.

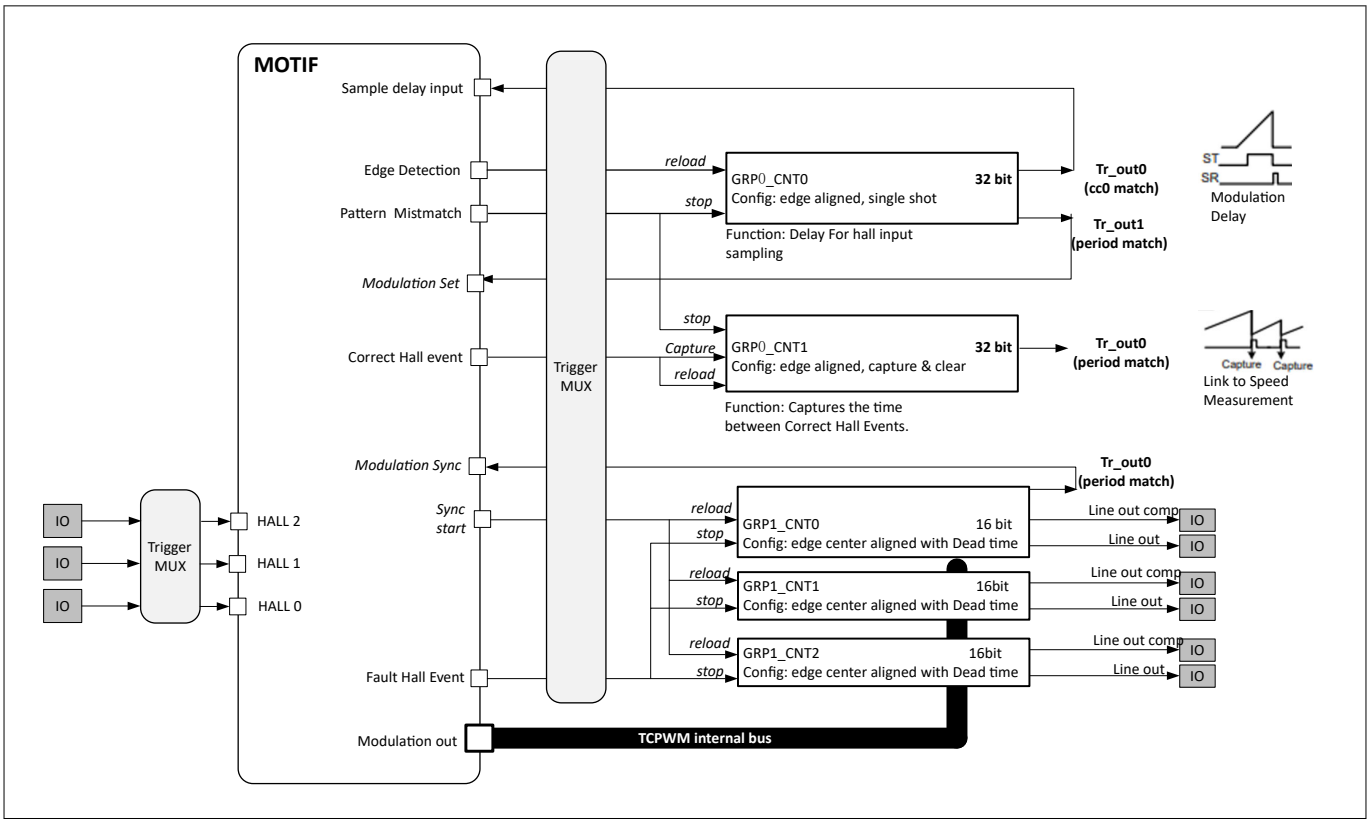


Figure 470 Hall sensor mode usage - profile 1

Quadrature decoder mode usage:

The quadrature decoder mode can be used in a very flexible way when connected with a TCPWM Counter module. The connection profile depends on the number of functions that the user wants to perform in parallel: position control, revolution control, and velocity measurement.

Quadrature decoder usage - Tick and revolution compare plus velocity between N ticks

In Figure 471, the MOTIF is linked with four TCPWM counters via a trigger multiplexer. In this profile, the user in quadrature decoder mode has a counter being used for position control (comparison), one counter for revolution counting, and two aggregated counters used for velocity measurement.

Counter 0 is connected to the QCLK and direction events. This imposes to configure the QCLK as counting functionality and direction as an up/down counting function, inside Counter 0. This counter is then used to track the actual position of the system and the compare channel can be configured to trigger the required actions, when the position reaches a certain value.

Counter 1 is connected to the INDX event (Index event). This event is used as a motor revolution counter. The compare channel can be programmed to trigger an interrupt every time that the motor performs N revolutions.

Counter 2 and Counter 3 are used to perform velocity measurements. Counter 2 uses the PCLK as a counting event. The compare channel of this slice is then used to trigger a capture event in Counter 3. The last one is using the module clock and therefore in every capture event, the actual system ticks are captured. This way the user knows how much time has elapsed between N phase periods and can calculate the corresponding velocity profile.

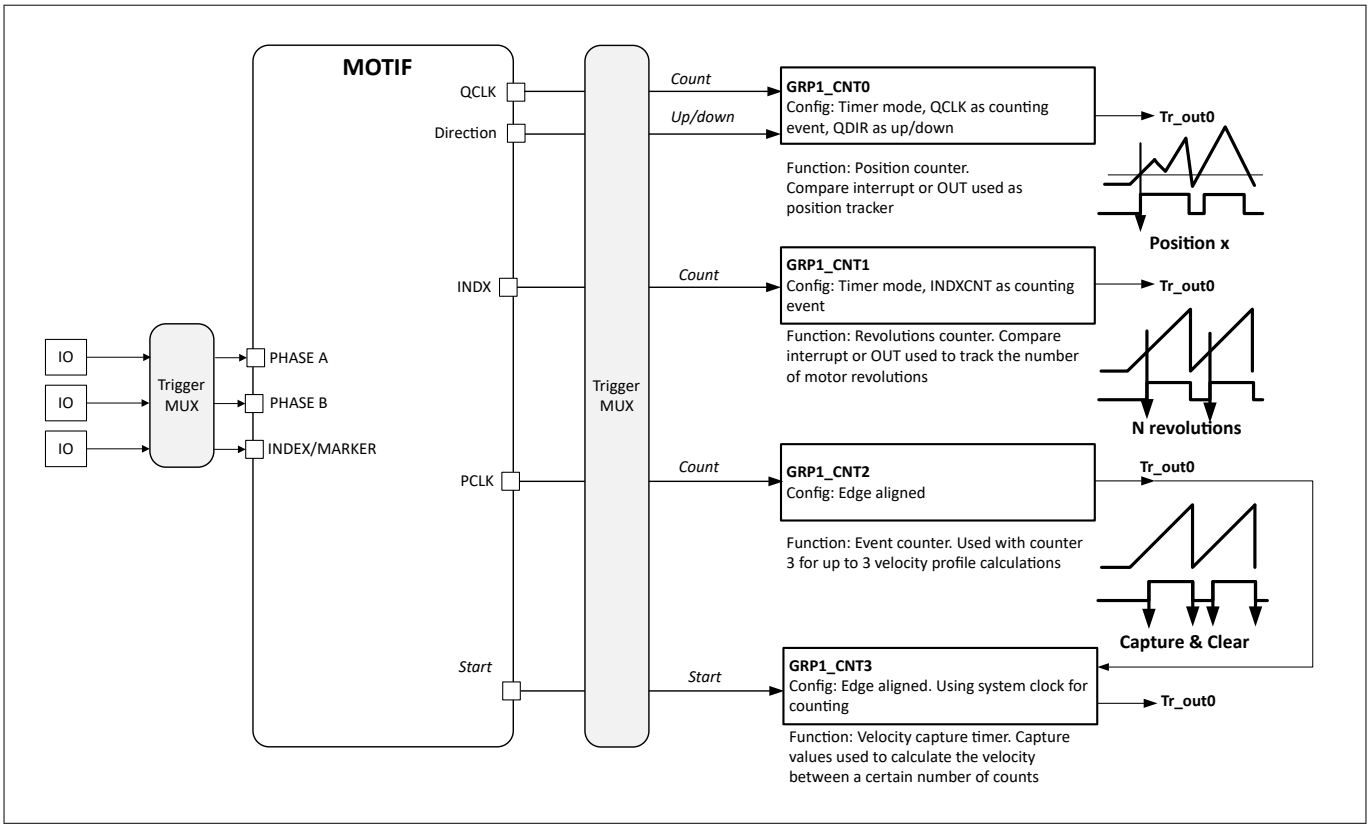


Figure 471 Quadrature decoder mode usage - profile 1

Quadrature decoder usage - Extended tick comparison plus velocity between N ticks

The profile demonstrated in the figure below enables the usage of 2 compare channels for the position control. This profile is especially useful for multi-operations during just one motor revolution.

Counter 0 and Counter 1 are using the QCLK event as a counting function and the direction as an up/down function. The comparing channels in each counter are then programmed with different compare values.

Counter 2 and Counter 3 are used in the same manner as profile 1, see Figure 471

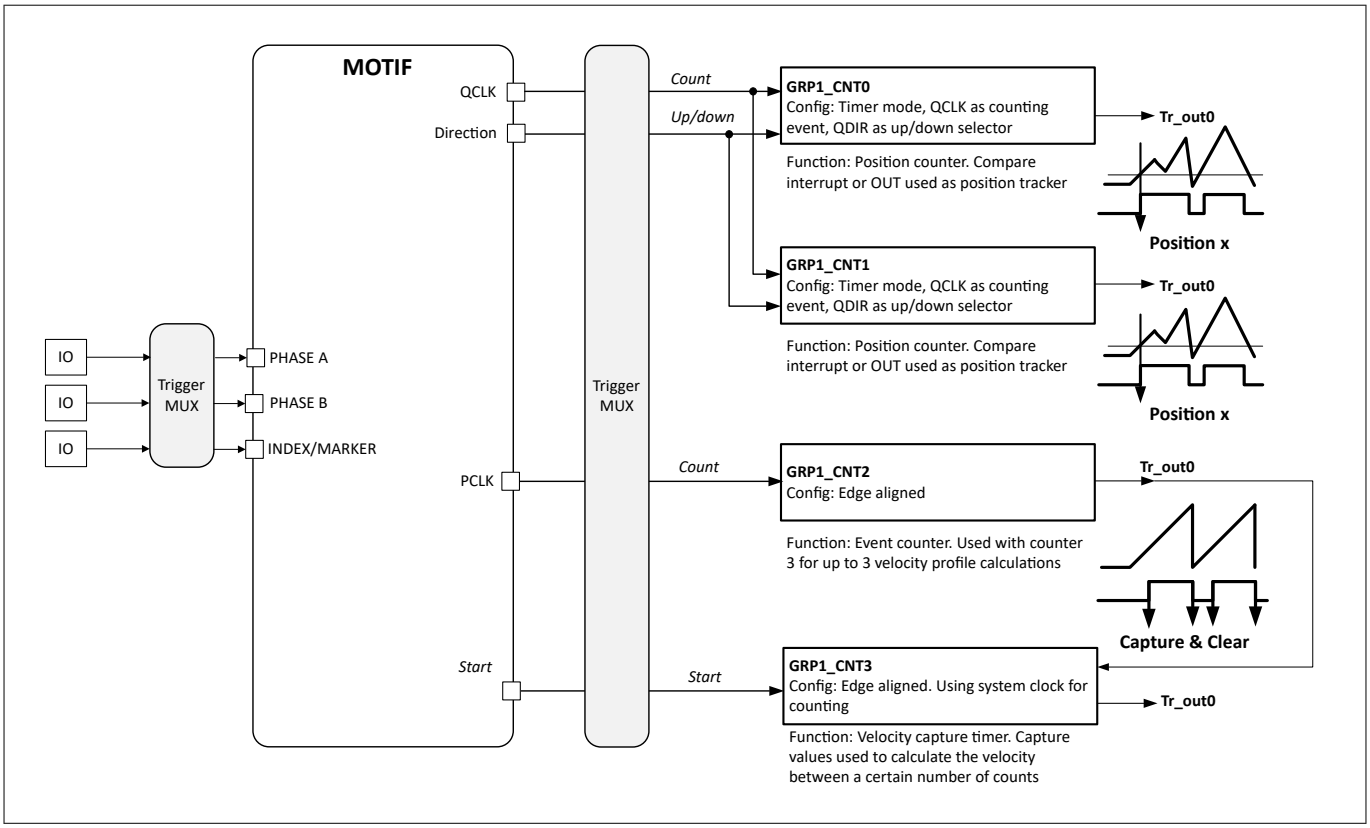


Figure 472 Quadrature decoder mode usage - profile 2

Quadrature decoder usage - Tick and revolution comparison with index clear plus velocity between N ticks

In some applications it is useful to use the index marker as a clear signal for the position and velocity control. The PRGINDX event can be configured to be generated only at the first index marker or at all maker hits. This PRGINDX event is then connected to the specific counter and used as a clear function. The following figure shows the adaptation of profile 1 with the PRGINDX event used as a clear function.

The same procedure can be used in profile 2, so that we can have the 2 compare channels plus the velocity measurement and the PRGINDX as clear signal.

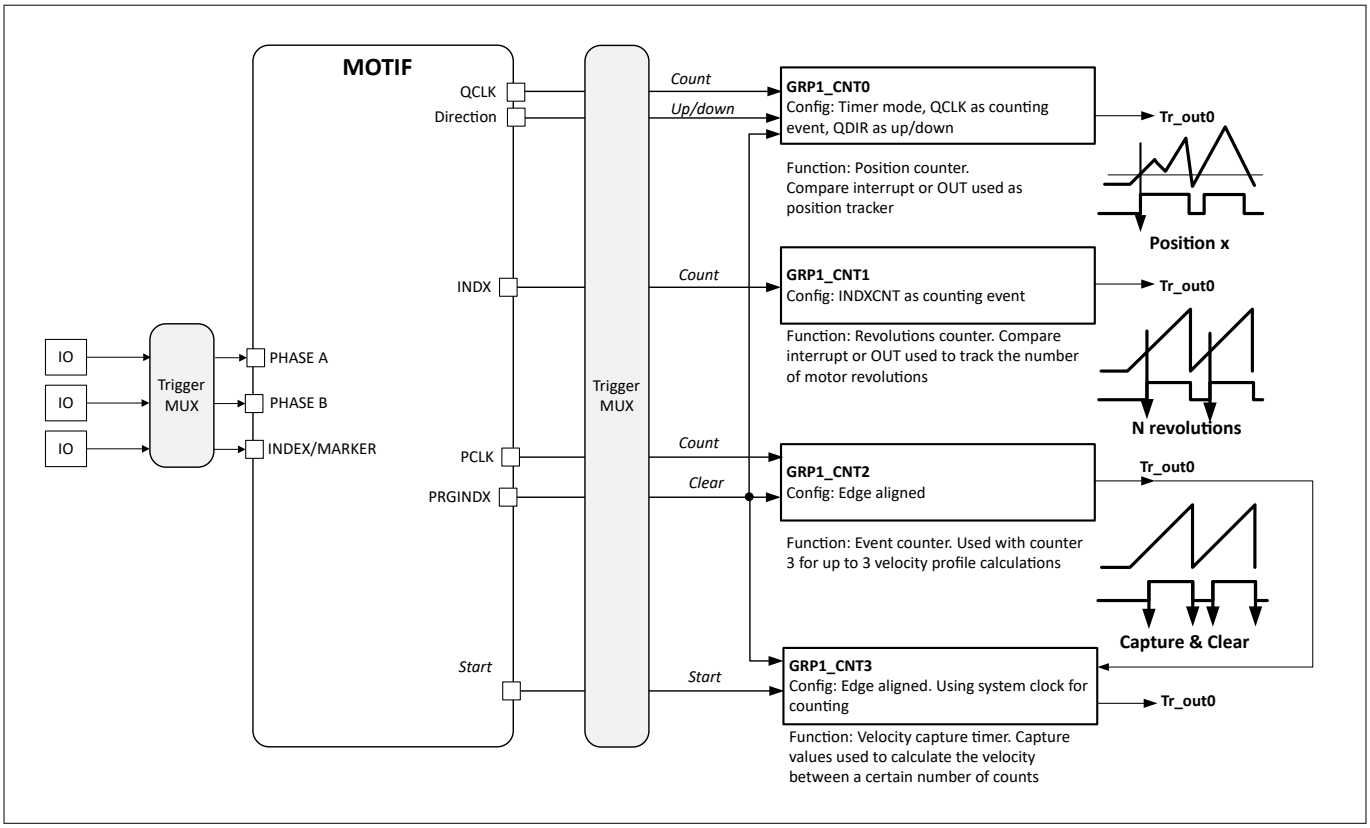


Figure 473 Quadrature decoder mode usage - profile 3

Quadrature decoder usage - Tick comparison plus micro tick velocity for slow rotating speeds

When the motor rotating speed is slow, it may not be suitable to discard the time between each occurrence of a rotary encoder tick.

In a fast-rotating system, the time between ticks is normally discarded and the velocity calculation can be done, by considering the number of ticks that have been elapsed since the last ISR. But in a slow rotating system, considering the number of ticks may not be enough (because of the associated error), and the software needs also to take into consideration, the time between the last tick and the actual ISR trigger.

The following timing diagram shows a slow rotating system, where the ISR for the velocity calculation is triggered in a periodic way. In this case, because of the small number of ticks between each ISR trigger, the software needs to know not only the number of elapsed ticks but also the elapsed time between the last tick and the ISR occurrence.

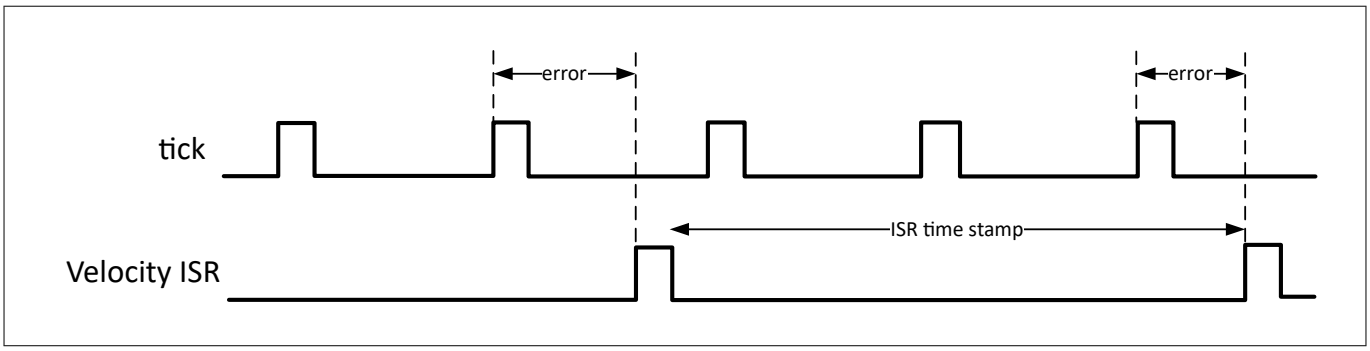


Figure 474 Example slow rotating system

It is possible to build a profile with the MOTIF and one counter module to perform a control loop that is immune to this slow velocity calculation pitfall. Counter 0, is used to monitor the current position of the motor shaft.

Three additional counters are needed to build the slow velocity calculation loop: Counter 1, Counter 2 and Counter 3.

Counter 1, is counting the number of ticks (PCLK) that are elapsed between each velocity ISR occurrence. Counter 2 is counting the time between each tick (PCLK) occurrence. This Counter is cleared and started within every tick and therefore it always keeps the timing info between the last and the current tick. Counter 3, is controlling the periodicity of the velocity ISR. Every time that a velocity ISR is triggered, Counter 3 also triggers a capture in Counter 2 and a capture & clear in Counter 1.

With this mechanism, every time that the software reads back the captured values from Counter 1 and Counter 2, it can calculate the speed based on:

- The number of ticks elapsed since the last ISR
- Plus the amount of time elapsed between the last tick and the ISR

This control loop therefore offers a very accurate way to calculate the motor speed within systems with low velocity.

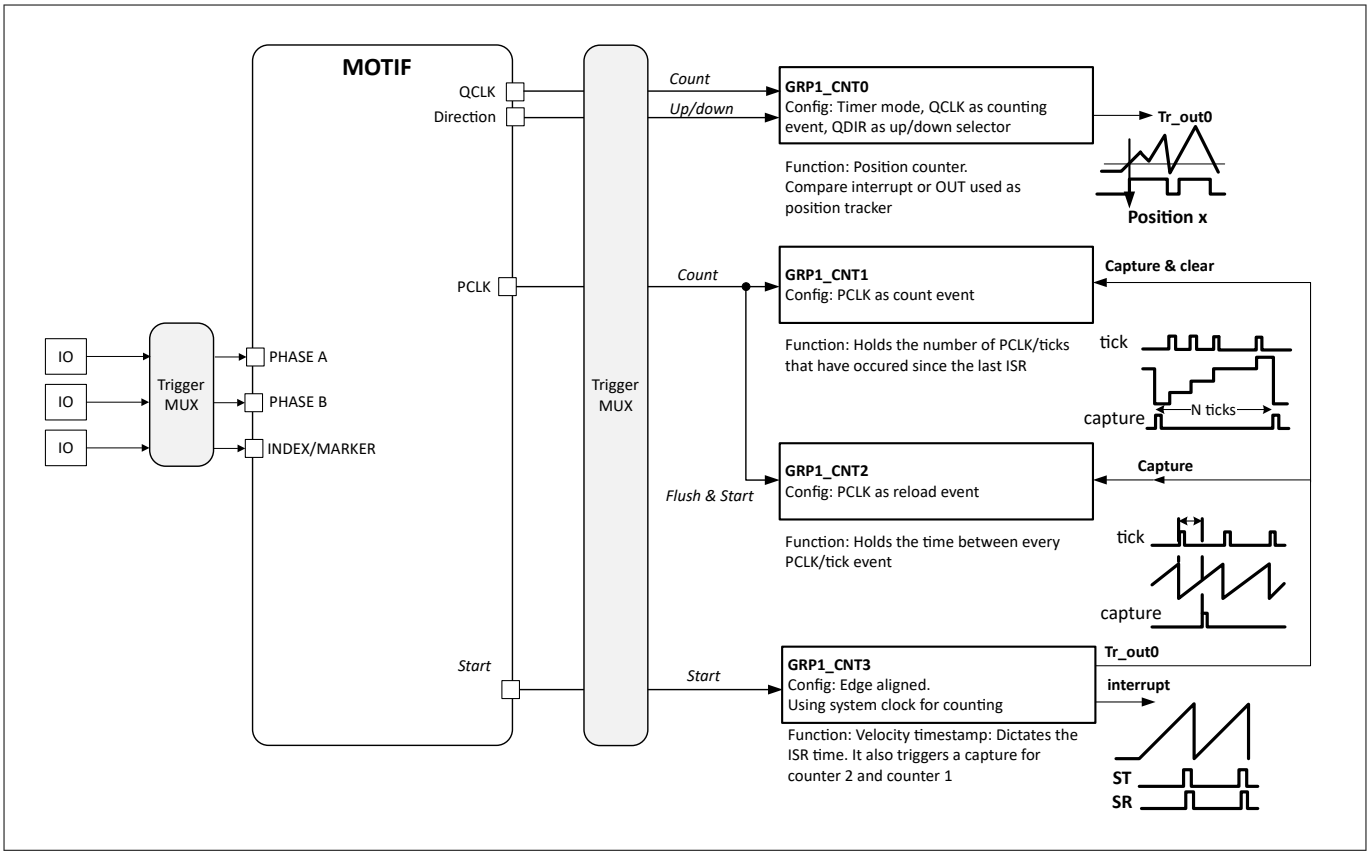


Figure 475 Quadrature decoder mode usage - profile 4

Stand-alone multi-channel mode usage

The multi-channel mode can be used in the MOTIF module without being linked with the hall sensor mode. This is especially useful for performing generic control loops that need to be perfectly synchronized.

The stand-alone multi-channel mode is very generic and depends very much on the control loop specified by the user. A generic profile for the multi-channel mode is to use a set of PWM counters to perform the signal generation and a General-purpose counter linked with an external pin that is used as a trigger for updating the multi-channel pattern.

On Figure 476 a General-purpose counter is used to control the delay between the update of an external signal (for example, an external sensor) and the update of the multi-channel pattern. Note that this external signal can also be connected to the MOTIF module if no timing delay is needed or it can be just controlled by software, by writing a one into the MCMS.MNPS field.

One of the PWM counter tr_out outputs can then be chosen to act as a synchronization trigger between the generated signal and the update of the multi-channel pattern.

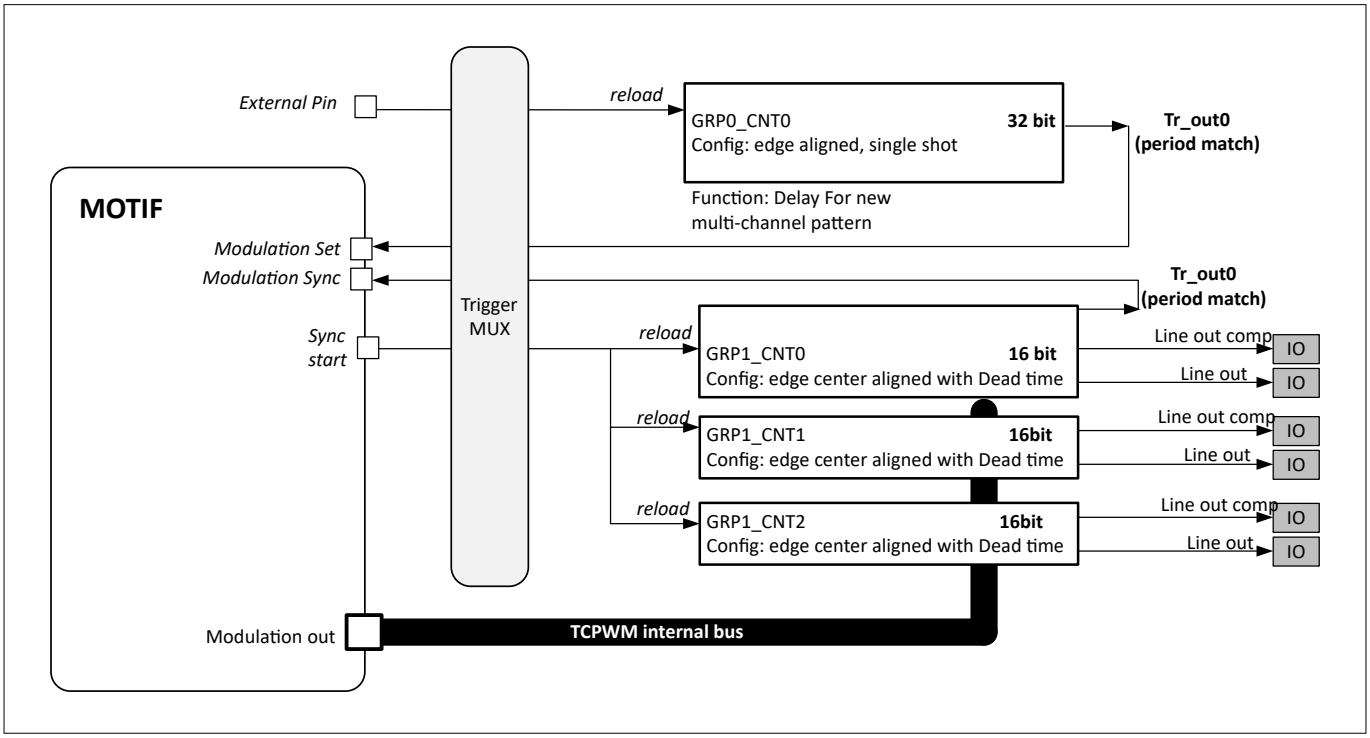


Figure 476 Stand-alone multi-channel mode usage

PWM output controlled by MOTIF modulation pattern

Figure 477 shows different types of PWM switching schemes supported by MOTIF together with the TCPWM counter.

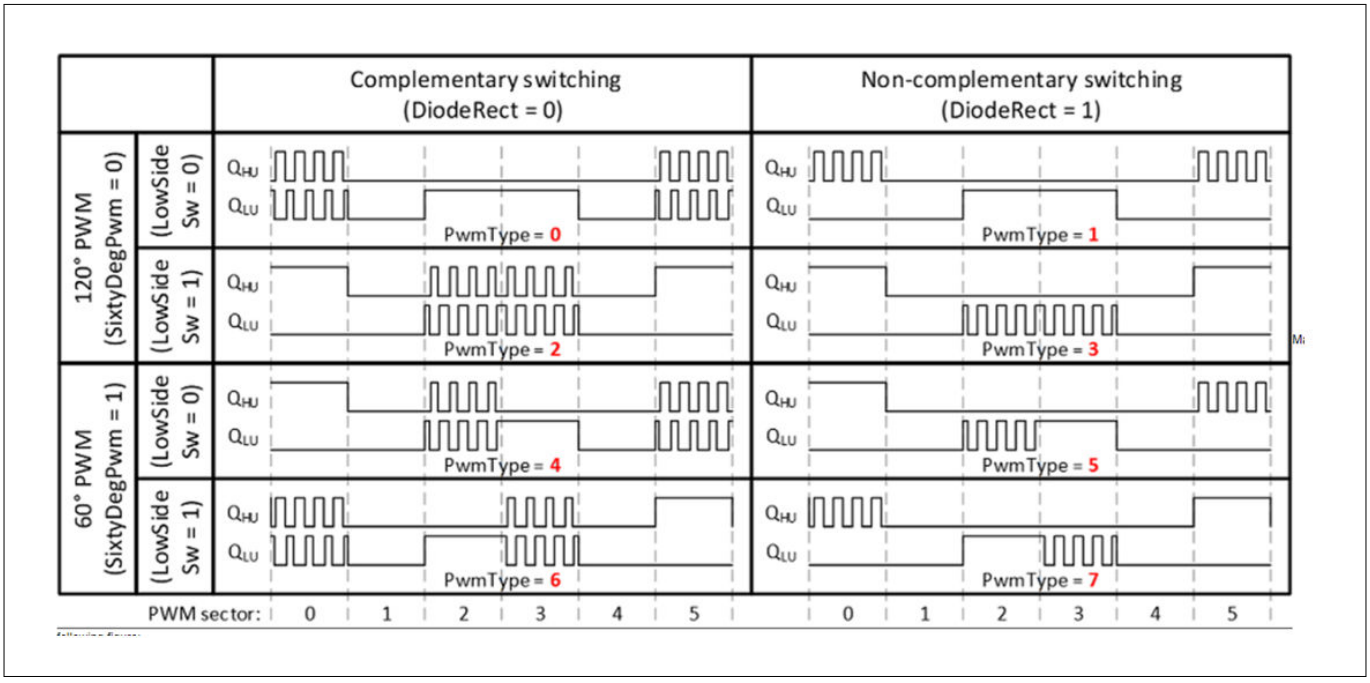


Figure 477 PWM switching modulation via MOTIF

33.5.10 Service request generation

The MOTIF has several interrupt sources that are linked to the different operation modes: hall sensor mode, quadrature decoder mode, and stand-alone multi-channel mode.

The interrupt flags for the hall sensor mode are described in Hall sensor mode flags while the interrupt flags of the Quadrature Decoder Mode are described in Quadrature decoder flags.

The flags associated with the multi-channel function are available in all the modes. This is because the multi-channel mode can operate in parallel with the quadrature decoder mode and is needed whenever the hall sensor mode is activated.

Each of the interrupt sources, can be routed to the MOTIFx.SR0 or MOTIFx.SR1 output, depending on the value programmed in the PFLGE register.

33.5.10.1 Hall sensor mode flags

The Hall Sensor Control contains six flags that can be configured to generate an interrupt request pulse, see Figure 478.

The six interrupt sources are:

- transition at the Hall Inputs (PFLG.HIES)
- occurrence of a correct hall event (PFLG.CHES)
- occurrence of a wrong hall event (PFLG.WHES)
- occurrence of a current pattern match event (PFLG.CPES)
- occurrence of a previous pattern match event (PFLG.PPES)
- shadow transfer of the Multi-Channel pattern (PFLG.MSTS)

The last one is triggered every time the Multi-Channel pattern is updated (PIFMST), which means that the MOTIFx.MOUT[31:0] output was updated with a new value.

Each of this interrupt sources can be enabled/disabled individually. The SW also has the possibility to set and clear the specific flags by writing a 1B into the specific fields of SPFLG and RPFLG registers. By enabling an interrupt source, an interrupt pulse is generated every time that a flag set operation occurs, independently if the flag is already set or not.

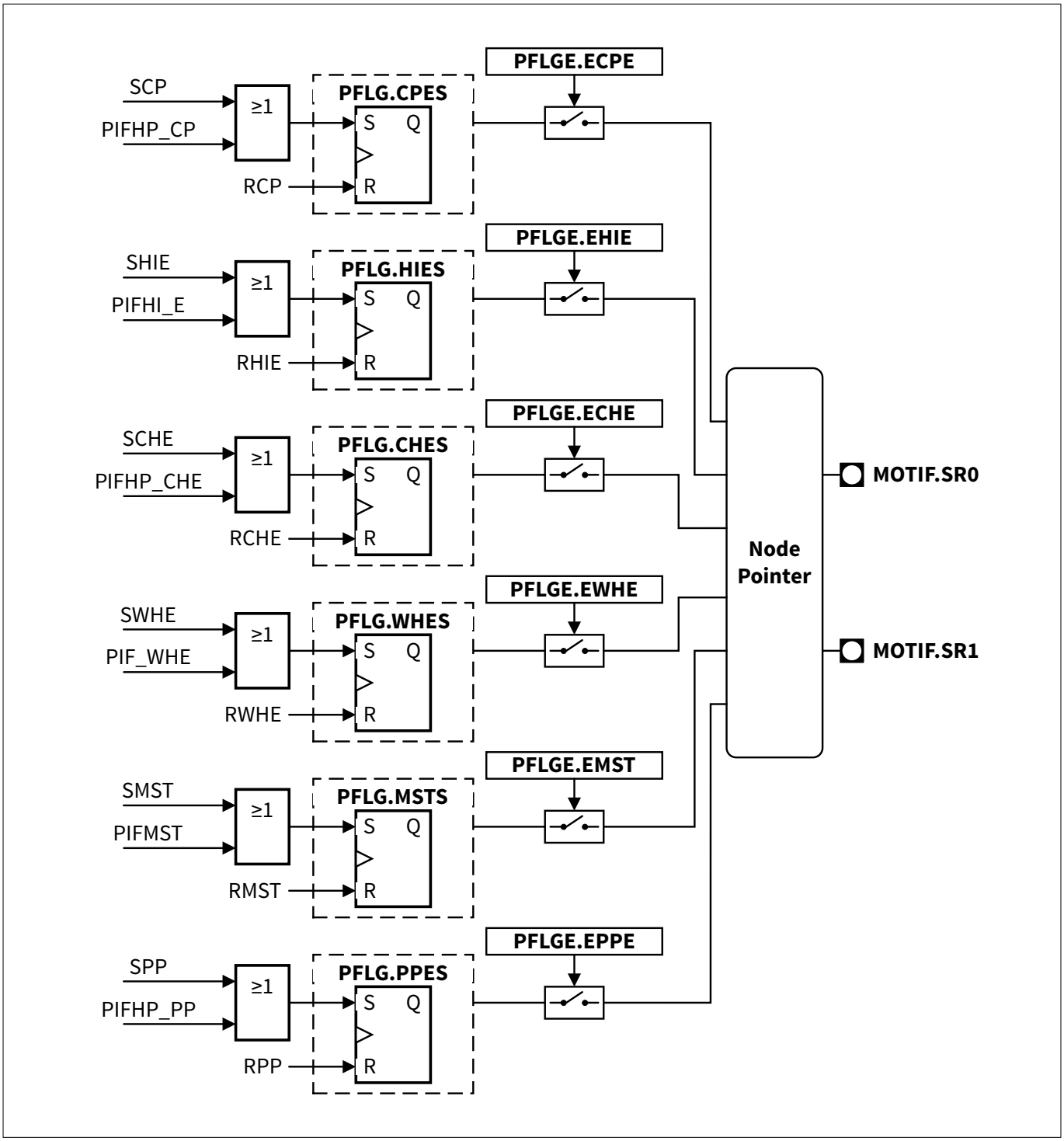


Figure 478 Hall sensor mode flags

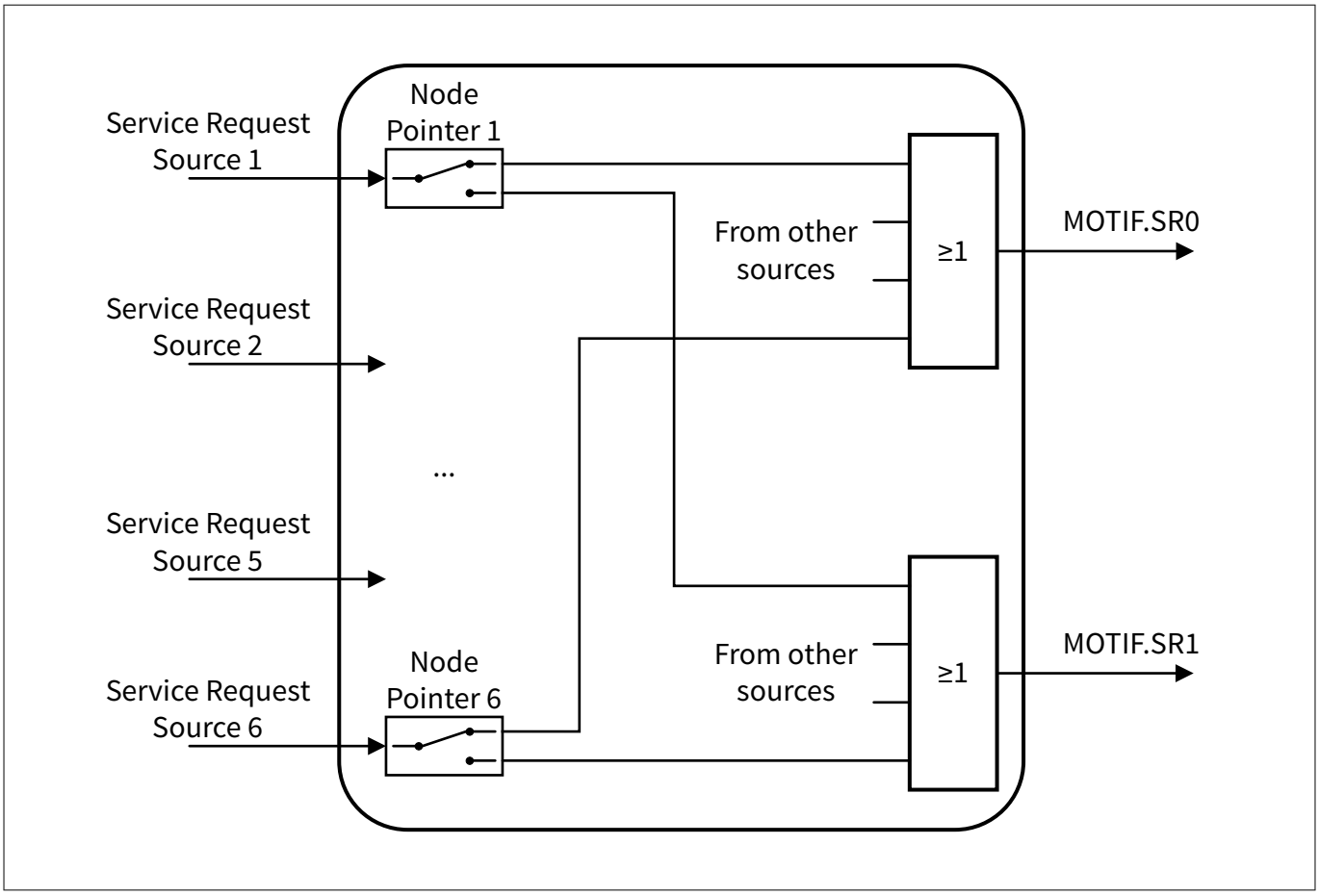


Figure 479 Interrupt node pointer overview - hall sensor mode

33.5.10.2 Quadrature decoder flags

The Quadrature Decoder Mode has five flags that can be enabled individually as interrupt sources (besides these five flags, the ones that are linked to the usage of Multi-Channel mode are also available: Multi-Channel Pattern update (PFLG.MSTS) and Wrong Hall event (PFLG.WHES). This can be useful if one selects the Quadrature Mode and the Multi-Channel stand-alone functionality.

These five flags are:

- index event detection (PFLG.INDXS)
- phase detection error (PFLG.ERRS)
- quadrature clock generation (PFLG.CNTS)
- period clock generation (PFLG.PCLKS)
- direction change (PFLG.DIRS)

By enabling an interrupt source, an interrupt pulse is generated every time that a flag set operation occurs, independently if the flag is already set or not.

The index event detection flag is set every time that an index is detected.

The phase error flag is set when one invalid transition on the phase signals is detected, see Figure 461

The quadrature clock and period clock flags are generated accordingly with the timings shown in Figure 462

The direction change flag is set, every time that an inversion of the motor rotation happens.

Each flag can be set/cleared individually by SW, by writing into the specific field on the registers SPFLG and RPFLG, see Figure 480

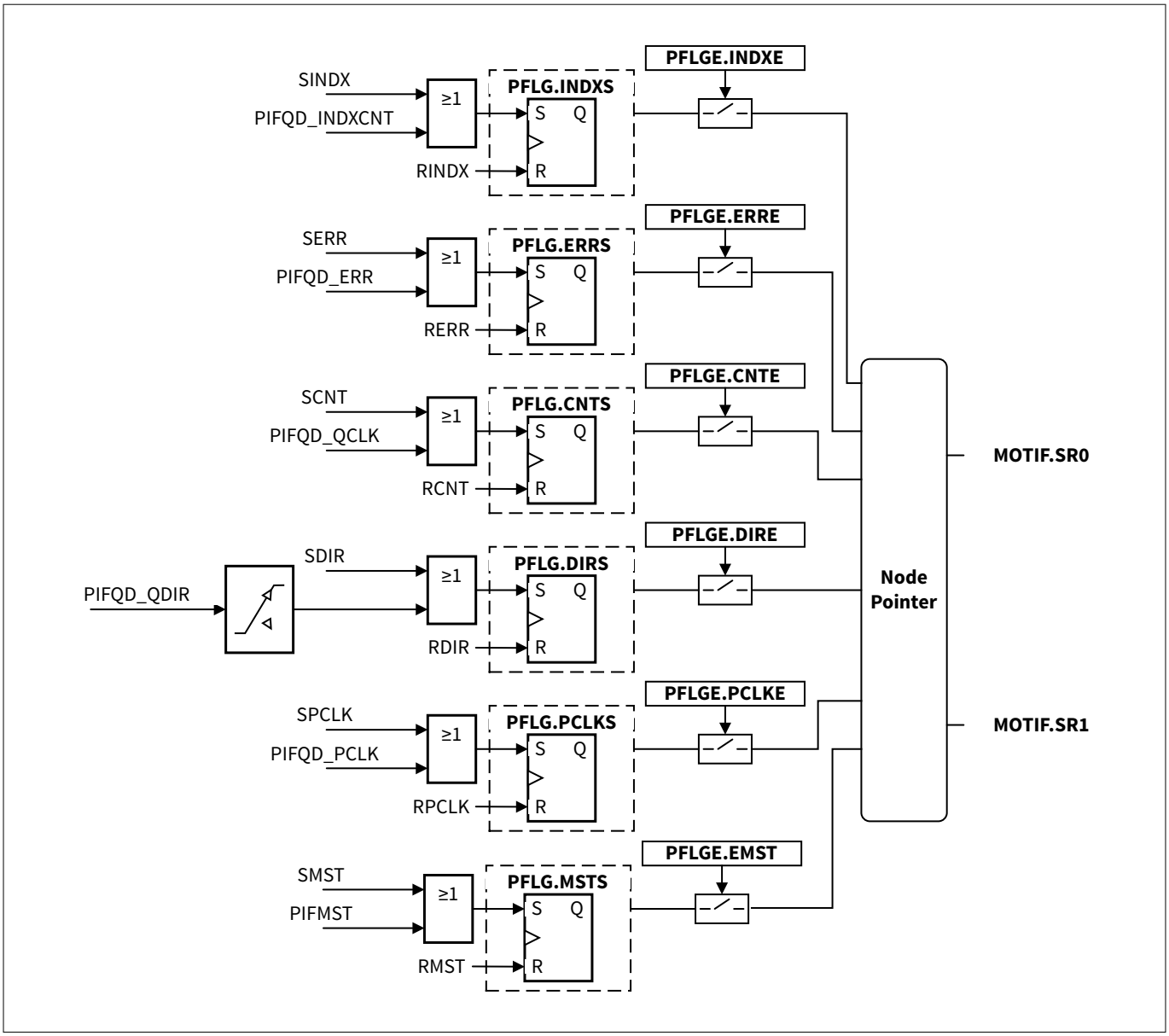


Figure 480 Quadrature decoder flags

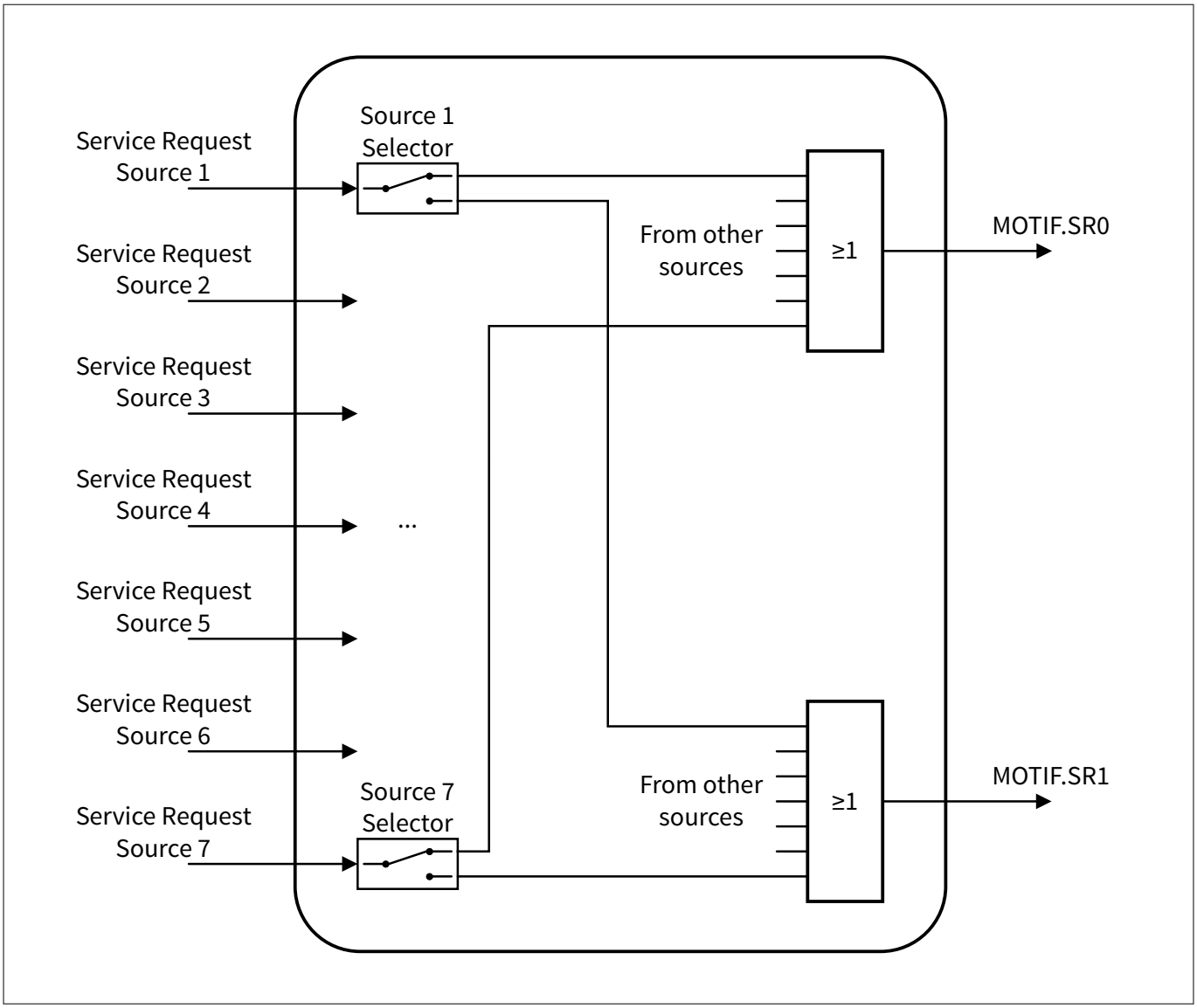


Figure 481 Interrupt node pointer overview - quadrature decoder mode

33.5.11 MOTIF debug behavior

The TCPWM MOTIF supports debugging. It can be configured per MOTIF if the MOTIF operation continues or pauses in a debug state (for example, after running to a break point), which can be realized by gating the counter clock “clk_motif” when entering the debug state defined by PSUS.SUS.

The MOTIF registers can still be accessed by the CPU. This mode is useful for debugging purposes, for example, where the current device status should be frozen to get a snapshot of the internal values. In debug mode, the entire MOTIF is halted. The debug mode is non-intrusive concerning the register bits. This means register bits are not modified by hardware when entering or leaving the suspended mode. Entry into debug mode can be configured at the kernel level by means of the register PSUS. The module is only functional after the debug signal becomes inactive.

The TCPWM has an input “tr_debug_freeze” which signalizes entering the debug state. It is connected via a PERI trigger multiplexer to the 2 cross-trigger interface outputs of the CPUSS. Depending on which CPU shall be debugged, the debug state output trigger of that CPU can be selected as “tr_debug_freeze” input of the TCPWM. Since this signal is routed through the trigger multiplexers, it needs to be synchronized (as all trigger signals are considered to be asynchronous due to potential long wires). For these MOTIF which are controlled

by the CPU to be debugged and which should be stopped when entering the debug mode defined by PSUS.SUS register.

***Note**: The MOTIF input trigger synchronizer logic also works with "clk_motif". During debug state input triggers are not synchronized.*

33.5.12 Initialization and System dependencies

33.5.12.1 Initialization

The initialization sequence for an application that is using the MOTIF should be the following:

1. Apply system reset
2. Release system reset
3. Enable the MOTIF via register PCONF.ENABLE=1

   ***Note**: Once enabled, the reset "rst_motif_act_n" gets released and motif operation logic receives "clk_motif". The LPF is active and running, so MOTIF inputs tr_motif_in[0], tr_motif_in[1], tr_motif_in[2] are filtered out if the filter is enabled.*

4. Configure MOTIF registers via AHB access
5. Read PDBG.LPFxV until tr_motif_in[0], tr_motif_in[1], tr_motif_in[2] inputs are equal to one
6. Start MOTIF via PRUNS.SRB=1

The internal state machines shall go out of idle state and run into operation mode.

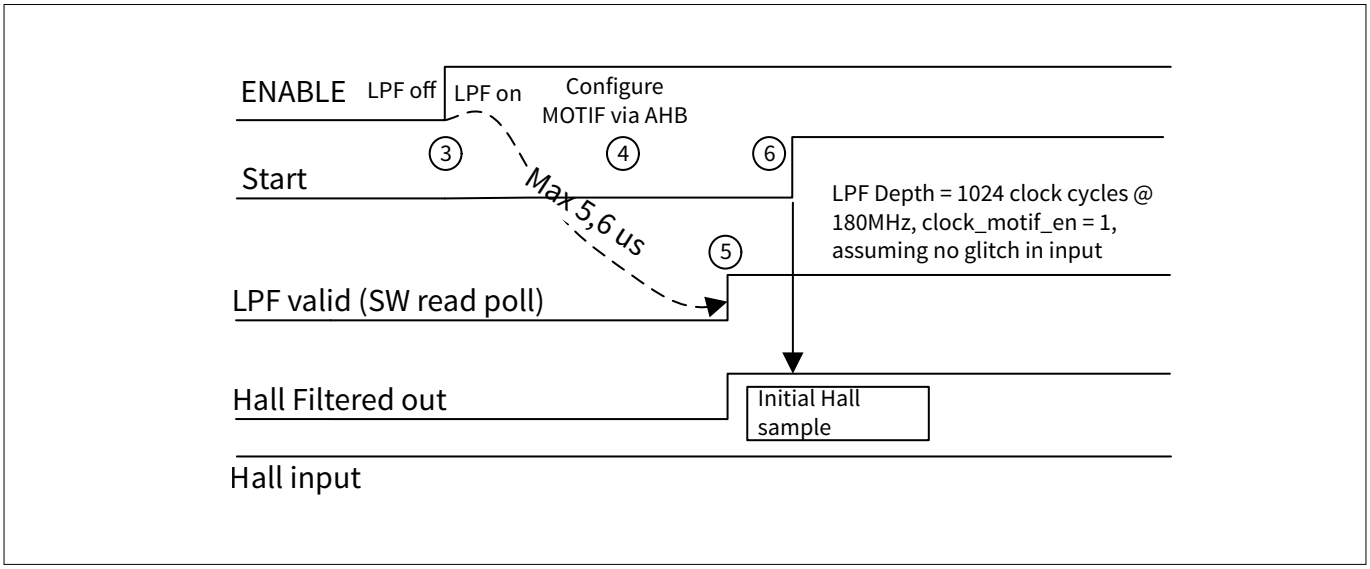


Figure 482 Initialization sequence for Hall mode

The initialization sequence is the same for hall mode and quad mode. In stand-alone multichannel mode, 5. can be omitted as LPF is not used.

Stop MOTIF is done via PRUNS.SRB=0, at this time, the state machines shall go to idle state, output events gets reset to default value, interrupt output get reset to default value and modulation output keep the existing value.

Disable MOTIF is done via PCONF.ENABLE=0, at this time, all non-retention registers get reset to the default value "rst_motif_act_n" is active. The MOTIF operational logic will not receive a clock, "clk_motif" is gated. The

MOTIF trigger synchronization will not receive a clock, “clk_peri_motif” is gated. The MOTIF MMIO registers get an active clock, “clk_ahb_motif”.

33.5.12.2 System dependencies

MOTIF may have different dependencies regarding module and bus clock frequencies. These dependencies should be addressed in the device clock unit and system architecture chapters.

Dependencies between several peripherals, regarding different clock operating frequencies may also exist. This should be addressed before configuring the connectivity between the MOTIF and some other peripheral.

The following topics must be taken into consideration for good MOTIF and system operation:

- Module input triggers for the MOTIF must not exceed the module clock frequency (if the triggers are generated internally in the device)
- Frequency of the MOTIF outputs used as triggers/functions on other modules must be crosschecked on the end point
- Applying and removing MOTIF from reset can cause unwanted operations in other modules. This can occur if the modules are using MOTIF outputs as triggers/functions

33.6 Recovery

TCPWM can be recovered with any active reset event, such as:

- Power-on reset (POR)
- External reset (XRES)
- Watchdog timer reset (MCWDT and WDT)
- Brownout detection reset
- Overvoltage and overcurrent detection reset

There is no unexpected state in which the TCPWM can enter.

33.7 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 unless CTRL.PWM_DISABLE_MODE is set to a non-zero value. Results vary in this scenario based on the configuration.

33.8 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’, MOTIF_PCONF.ENABLE also is set to 0.

33.9 Registers summary

Table 352 List of TCPWM registers

Name	Description
TCPWMx_GRPy_CNTz_CTRL	Counter control register
TCPWMx_GRPy_CNTz_STATUS	Counter status register

(table continues...)

Table 352 (continued) List of TCPWM registers

Name	Description
TCPWMx_GRPy_CNTz_COUNTER	Counter count register
TCPWMx_GRPy_CNTz_CC0	Counter compare/capture 0 register
TCPWMx_GRPy_CNTz_CC0_BUFF	Counter buffered compare/capture 0 register
TCPWMx_GRPy_CNTz_CC1	Counter compare/capture 1 register
TCPWMx_GRPy_CNTz_CC1_BUFF	Counter buffered compare/capture 1 register
TCPWMx_GRPy_CNTz_PERIOD	Counter period register
TCPWMx_GRPy_CNTz_PERIOD_BUFF	Counter buffered period register
TCPWMx_GRPy_CNTz_LINE_SEL	Counter line selection register
TCPWMx_GRPy_CNTz_LINE_SEL_BUFF	Counter buffered line selection register
TCPWMx_GRPy_CNTz_DT	Counter PWM dead time register
TCPWMx_GRPy_CNTz_DT_BUFF	Counter buffered PWM dead time register
TCPWMx_GRPy_CNTz_PS	Counter prescalar register
TCPWMx_GRPy_CNTz_TR_CMD	Counter trigger command register
TCPWMx_GRPy_CNTz_TR_IN_SEL0	Counter input trigger selection register 0
TCPWMx_GRPy_CNTz_TR_IN_SEL1	Counter input trigger selection register 1
TCPWMx_GRPy_CNTz_TR_IN_EDGE_SEL	Counter input trigger edge selection register
TCPWMx_GRPy_CNTz_TR_PWM_CTR	Counter trigger PWM control register
TCPWMx_GRPy_CNTz_TR_OUT_SEL	Counter output trigger selection register
TCPWMx_GRPy_CNTz_INTR	Interrupt request register
TCPWMx_GRPy_CNTz_INTR_SET	Interrupt set request register
TCPWMx_GRPy_CNTz_INTR_MASK	Interrupt mask register
TCPWMx_GRPy_CNTz_INTR_MASKED	Interrupt masked request register
TCPWMx_GRPy_CNTz_TR_ONE_SYNC_BYPASS	Sync bypass register for one to one trigger
TCPWMx_GRPy_CNTz_HRPWM_CTRL	Counter control register for HRPWM feature
TCPWMx_GRPy_CNTz_CTRL2	Counter control register 2
TCPWMx_TR_ALL_SYNC_BYPASS	Trigger Sync bypass for group trigger

***Note**: In TCPWMx_GRPy_CNTz, 'x' signifies the TCPWM instance number, 'y' is the group number and 'z' is the counter in the respective TCWPM group*

Table 353 MOTIF register summary

Name	Description
TCPWMx_MOTIF_GRPy_MOTIF0_PCONF	Global control register
TCPWMx_MOTIF_GRPy_MOTIF0_PSUS	Suspend configuration
TCPWMx_MOTIF_GRPy_MOTIF0_PRUNS	MOTIF runs bit set

(table continues...)

Table 353 (continued) MOTIF register summary

Name	Description
TCPWMx_MOTIF_GRPy_MOTIF0_PRUN	MOTIF run bit status
TCPWMx_MOTIF_GRPy_MOTIF0_HIST	Hall inputs sample trigger
TCPWMx_MOTIF_GRPy_MOTIF0_HMEC	Hall mode extra config
TCPWMx_MOTIF_GRPy_MOTIF0_HALP	Hall current and expected patterns
TCPWMx_MOTIF_GRPy_MOTIF0_HALPS	Hall current and expected shadow patterns
TCPWMx_MOTIF_GRPy_MOTIF0_HOSC	Hall sensor output config
TCPWMx_MOTIF_GRPy_MOTIF0_MCM	Multi-channel mode pattern
TCPWMx_MOTIF_GRPy_MOTIF0_MCSM	Multi-channel mode shadow pattern LUT0
TCPWMx_MOTIF_GRPy_MOTIF0_MCMS	Multi-channel mode control set
TCPWMx_MOTIF_GRPy_MOTIF0_MCMC	Multi-channel mode control clear
TCPWMx_MOTIF_GRPy_MOTIF0_MCMF	Multi-channel mode flag status
TCPWMx_MOTIF_GRPy_MOTIF0_MCPF	Multi-channel pattern fault
TCPWMx_MOTIF_GRPy_MOTIF0_MOSC	Multi-channel output config
TCPWMx_MOTIF_GRPy_MOTIF0_QDC	Quadrature decoder configuration
TCPWMx_MOTIF_GRPy_MOTIF0_QOSC	Quadrature output config
TCPWMx_MOTIF_GRPy_MOTIF0_MCMEC	Multi-channel extra config
TCPWMx_MOTIF_GRPy_MOTIF0_PFLG	MOTIF interrupt status
TCPWMx_MOTIF_GRPy_MOTIF0_PFLGE	MOTIF interrupt enables
TCPWMx_MOTIF_GRPy_MOTIF0_SPFLG	Interrupt set register
TCPWMx_MOTIF_GRPy_MOTIF0_RPFLG	Interrupt clear register
TCPWMx_MOTIF_GRPy_MOTIF0_MCSM1	Multi-channel mode shadow pattern LUT1
TCPWMx_MOTIF_GRPy_MOTIF0_MCSM2	Multi-channel mode shadow pattern LUT2
TCPWMx_MOTIF_GRPy_MOTIF0_MCSM3	Multi-channel mode shadow pattern LUT3
TCPWMx_MOTIF_GRPy_MOTIF0_MCSM4	Multi-channel mode shadow pattern LUT4
TCPWMx_MOTIF_GRPy_MOTIF0_MCSM5	Multi-channel mode shadow pattern LUT5
TCPWMx_MOTIF_GRPy_MOTIF0_CLUT	Hall mode LUT config
TCPWMx_MOTIF_GRPy_MOTIF0_SLUT	Hall mode LUT status
TCPWMx_MOTIF_GRPy_MOTIF0_PDBG	MOTIF debug register
TCPWMx_MOTIF_GRPy_MOTIF0_PLP0S	MOTIF low pass 0 status
TCPWMx_MOTIF_GRPy_MOTIF0_PLP1S	MOTIF low pass 1 status
TCPWMx_MOTIF_GRPy_MOTIF0_PLP2S	MOTIF low pass 2 status

***Note**: In TCPWMx_MOTIF_GRPy, 'x' signifies the TCPWM instance number, 'y' is the group number in the respective TCWPM group. For PPCA, x = 0/1, y = 2.*

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

34 PPCA Coordinate Rotation Digital Computer (CORDIC)

34.1 Features

PSC3M/P8, PSC3M/P7 supports two CORDIC modules. CORDIC is a peripheral module to support the CPU with hardware calculations for trigonometric, hyperbolic, complex numbers, and related functions. The CPU can access CORDIC through a peripheral AHB bus. The completion of an operation is signaled via status bits, an interrupt request line, or a trigger done line.

The CORDIC algorithm is a useful convergence method. It allows the performance of vector rotation not only in the Euclidean plane but also in the hyperbolic planes. The CORDIC algorithm is an iterative process where truncation errors are inherent. Higher accuracy is achieved in the CORDIC coprocessor with 28 iterations per calculation and a kernel data width of at least 32 bits. The main advantage of using this algorithm is its fast calculation speed compared to software and high accuracy. The generalized CORDIC algorithm has the following CORDIC equations:

Table 354 CORDIC algorithm equations

Equations	Label
$x_{n+1} = x_n - m \cdot d_n y_n 2^{-n}$	Eq.1
$y_{n+1} = y_n + d_n x_n 2^{-n}$	Eq.2
$z_{n+1} = z_n - d_n e_n$	Eq.3

The m factor controls the vector rotation and selects the set of angles for the circular and hyperbolic functions. The n factor is the CORDIC calculation's iterative number.

m = 1: Circular mode with $e_n = \text{atan}(2^{-n})$

m = -1: Hyperbolic mode with $e_n = \text{atanh}(2^{-n})$

CORDIC feature list:

- Modes of operation
  - Supports all CORDIC operating modes for solving circular (trigonometric) and hyperbolic functions
  - Integrated look-up tables (LUTs) for all operating modes
- Circular vectoring mode: Extended support for values of initial X and Y data up to the full range of $[-2^{23}, (2^{23}-1)]$ for solving angle and magnitude
- Circular rotation mode: Extended support for values of initial Z data up to the full range of $[-2^{23}, (2^{23}-1)]$, representing angles in the range $[-\pi, ((2^{23}-1)/2^{23})\pi]$ for solving trigonometry
- 24-bit accessible data width
  - 32-bit kernel data width plus 2 sign extension bits and 1 overflow bit for X and Y each
  - 29-bit kernel data width plus 1 sign extension bit and 1 overflow bit for Z
  - KEEP bit to retain the last value in the MXCORDIC_1_0_KEEP register for a new calculation
- 28 iterations per calculation (27 in hyperbolic mode): 4 iterations per clock cycle, resulting in 7 clock cycles per calculation plus 3 clock cycles from a set of start (ST) bits to a set of end-of-calculation flags, excluding the time taken for write and read access of data bytes
- Two's complement data processing
  - X and Y data are generally accepted as integer or rational numbers; X and Y must be of the same data form
  - Only exception: X result data with user-selectable option for unsigned result

- Truncation error
  - The result of a CORDIC calculation may return an approximation due to the truncation of LSBs
  - Good accuracy of the CORDIC calculated result data, especially in circular mode
- Interrupt
  - On completion of a calculation
  - Interrupt enabling and corresponding flag

34.2 Architecture

The CORDIC uses an AMBA 5 AHB slave interface, an interrupt and a trigger interface.

32-bit AMBA 5 AHB Peri slave interface: The MXCORDIC MMIOs are accessed by this interface.

Interrupt interface: Sends interrupt requests to the host MCU (M33), usually to signal a completed job or an error during execution.

Trigger interface: Sends trigger requests to the Datawire (DW) or Direct Memory Access (DMA) to signal a start of a job and completion of a job (on separate start and done ports). The start trigger indicates that it can be reconfigured for next CORDIC calculation until it completes the current calculation. The DW or DMA can be used to configure and start the CORDIC and to read back the result.

34.2.1 Block Diagram

The CORDIC block diagram contains two major blocks: MMIO and CORDIC_KERNEL.

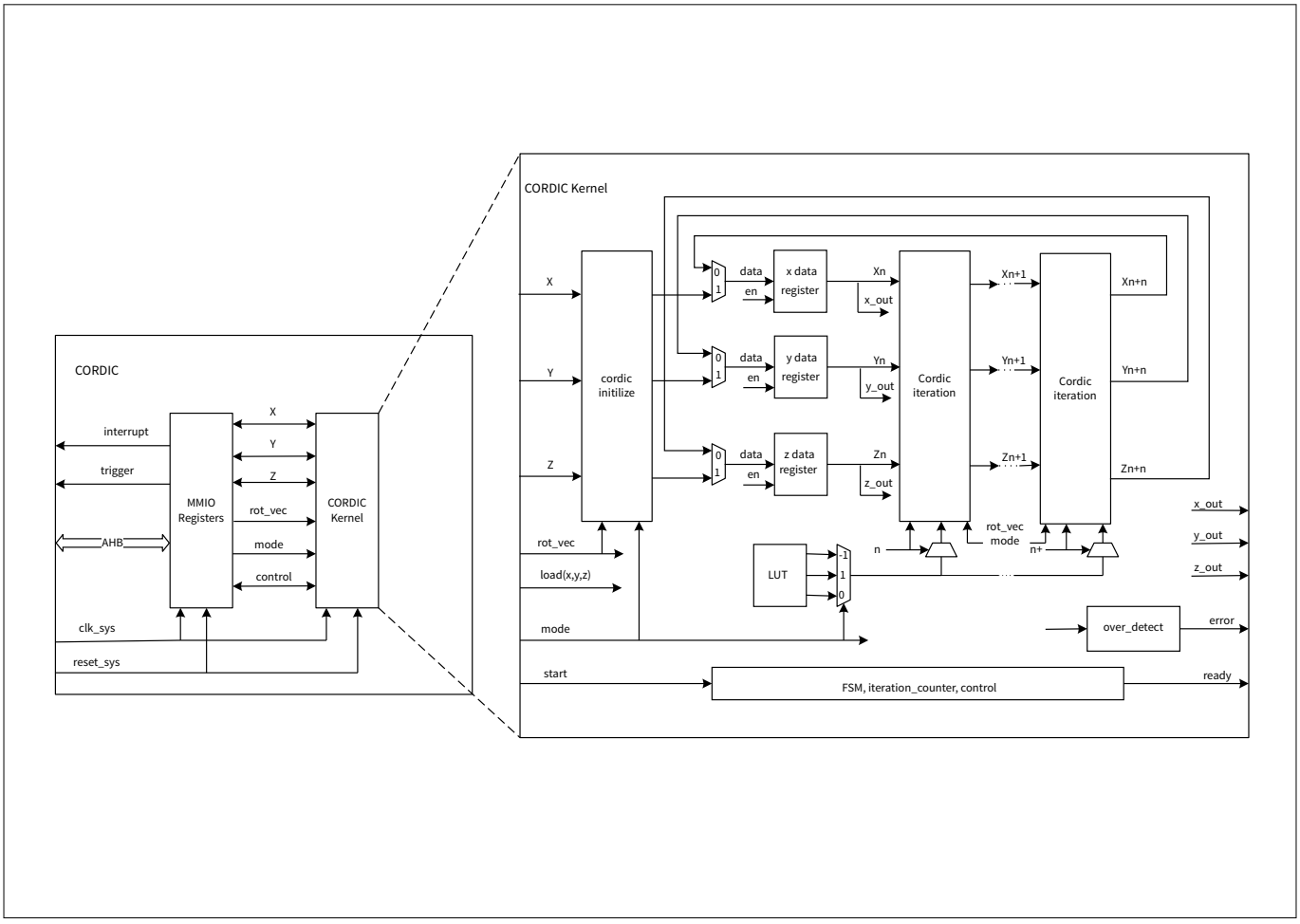


Figure 483 CORDIC block diagram

34.2.1.1 MMIO

The MMIO block contains all control, status and data registers. The MMIOs are accessed by the AHB slave interface.

34.2.1.2 CORDIC_KERNEL

The CORDIC_KERNEL implements the iterative CORDIC algorithm. The X, Y, and Z data registers are configured with initial values via the MMIO interface. When the calculation starts, those registers hold the intermediate results calculated by the CORDIC_ITERATION blocks. The CORDIC_ITERATION block implements one iteration of the CORDIC algorithm using combination shifters and adders. Four CORDIC_ITERATIONs are instantiated in sequence. After seven clock cycles, the final results (28 iterations) are available in the X, Y, and Z data registers. The sequence is controlled by a finite state machine (FSM) and an iteration counter.

The look-up tables (LUT) prove the required constants for the CORDIC algorithm in circular, hyperbolic and linear mode.

The CORDIC_INITIALIZE block increased the X, Y and Z bit width for the internal computation to avoid overflow and to limit truncation errors. The block additionally performs a pre-rotation in circular mode to increase the range of convergence (see the Domains of Convergence section for details).

34.3 Functional Description

34.3.1 Operation

The CORDIC coprocessor can be used for circular (trigonometric) and hyperbolic functions in either rotation or vectoring mode. The modes are selectable by software via the MXCORDIC_1_0_CON control register. Initialization of the kernel data register is enabled by clearing the respective KEEPx bits of the register MXCORDIC_1_0_KEEP. If MXCORDIC_1_0_CON.ST_MODE = 1, writing '1' to MXCORDIC_1_0_START_CMD.ST starts a new calculation. Otherwise, by default, if MXCORDIC_1_0_CON.ST_MODE = 0, a new calculation starts after a write access to register MXCORDIC_1_0_CORDZ. Each calculation involves a MXCORDIC_1_0_CON.N_ITER*4 number of CORDIC iterations. Bit MXCORDIC_1_0_STAT.BSY is set while a calculation is in progress to indicate busy status. At the end of a calculation, hardware clears it. The MXCORDIC_1_0_START_CMD.ST bit is cleared immediately after a new CORDIC calculation is started.

As the first step in starting a CORDIC calculation (the initial data is loaded from the data registers when KEEP is 0 and from the result registers when KEEP is 1), the initial data is loaded from the data registers CORDx to the internal kernel data registers. During the calculation, the kernel data registers always hold the latest intermediate data. On completion of the calculation, the result data is loaded into the result registers CORRx.

The data registers CORDx function as shadow registers, which can be written to without affecting an ongoing calculation. Values are transferred to the kernel data registers only on a valid setting of the MXCORDIC_1_0_START_CMD.ST bit or if MXCORDIC_1_0_CON.ST_MODE = 0 after write access to the X data register CORDX (provided the KEEP bit of the corresponding data is not set). The result data can be read from the result registers CORRx at the end of the calculation (BSY is no longer active). The result data must be read before the start of the next calculation. Any read access on the result registers CORRx while MXCORDIC_1_0_STAT.BSY is set (and the kernel is still running a calculation) causes the kernel to issue a bus wait until MXCORDIC_1_0_STAT.BSY is reset. This ensures that any read access on the result registers CORRx returns a valid result.

At the end of each calculation, MXCORDIC_1_0_STAT.BSY returns to 0. The MXCORDIC_1_0_INTR.CDEOC bit, which denotes the CORDIC end-of-calculation, is set. The interrupt request signal is activated if the interrupt is enabled by MXCORDIC_1_0_INTR_MASK.CDEOC = 1. The MXCORDIC_1_0_INTR.CDEOC flag must be cleared manually via software by writing MXCORDIC_1_0_INTR.CDEOC = 1. The result data in X, Y, and Z are internally checked, and in cases of data overflow, the MXCORDIC_1_0_INTR.CDERR bit is set. An interrupt request signal is

activated if the interrupt is enabled by MXCORDIC_1_0_INTR_MASK.CDERR = 1. The MXCORDIC_1_0_INTR.CDERR flag must be cleared manually via software by writing MXCORDIC_1_0_INTR.CDERR = 1.

At the end of each calculation, the hardware trigger is set to '1' for two clock cycles. This signal can be used to trigger the DW to read back the CORDIC results.

Setting the bit MXCORDIC_1_0_START_CMD.ST during an ongoing calculation (if MXCORDIC_1_0_CON.ST_MODE = 1) while MXCORDIC_1_0_STAT.BSY is set has no effect. To start a new calculation, bit MXCORDIC_1_0_START_CMD.ST must be set again at a later time when MXCORDIC_1_0_STAT.BSY is no longer active. In the same manner, changing the operating mode during a running calculation (as indicated by MXCORDIC_1_0_STAT.BSY) has no effect.

In automatic start mode (MXCORDIC_1_0_CON.ST_MODE = 0), if a write access to CORDX occurs while MXCORDIC_1_0_STAT.BSY is set, the new calculation is not started, similar to the case where MXCORDIC_1_0_CON.ST_MODE = 1.

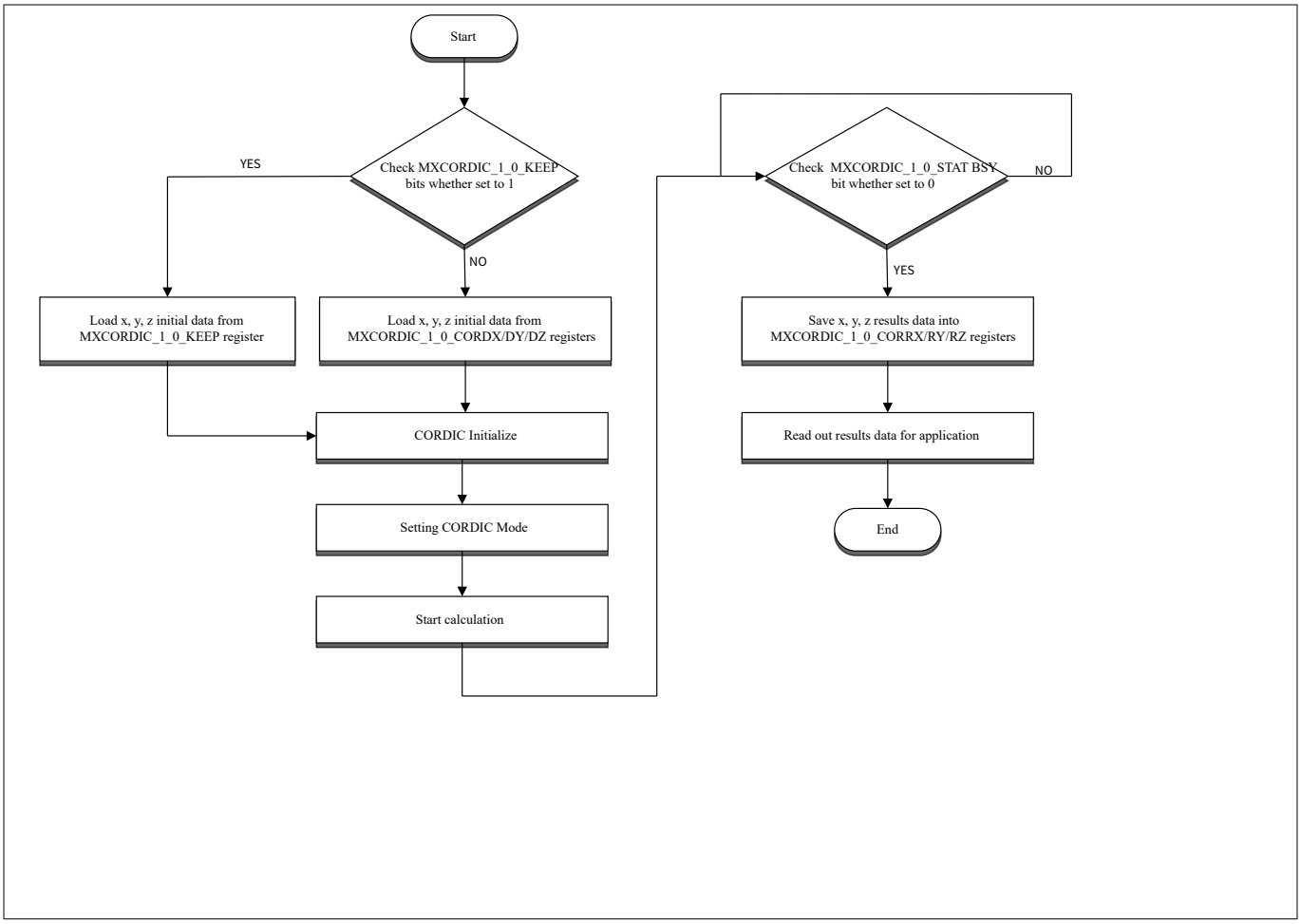


Figure 484 CORDIC operation

34.3.2 Normalized Result Data

In all operating modes, the CORDIC coprocessor returns normalized result data for X and Y, as shown in the following equation:

X or Y Result Data = (CORDIC Calculated Data)/MPS

***Note**: MPS is the Magnitude PreScaler.*

On the other hand, the interpretation of the Z result data differs, which is also dependent on the CORDIC function used.

For **circular** and **hyperbolic** functions, the accessible Z result data is a normalized integer value; angles in the range [-π, $((2^{23}-1)/2^{23})\pi$] are represented by [-2^{23},$(2^{23}-1)$]. The CORDIC coprocessor expects the Z data to be interpreted with this scaling:

Input Z Initial Data = Real Z Initial Value (in radians) · 0x800000/π

Real Z Result Value (in radians) = Z Result Data · π/0x800000

The CORDIC calculated data includes an inherent gain factor, K, resulting from the rotation or vectoring. The value K is different for each CORDIC function, as shown in the following table:

Table 355 Approximated CORDIC Gain K

Function	Approximated Gain K
Circular	1.646760258121
Hyperbolic	0.828159360960

34.3.3 Operating Modes

The following figure gives an overview of the CORDIC coprocessor operating modes. In this table, X, Y, and Z represent the initial data.

	Rotation Mode	Vectoring Mode
	$d_n = sign(z_n), z_n \rightarrow 0$	$d_n = -sign(y_n), y_n \rightarrow 0$
Circular m = 1 $e_n = \text{atan}(2^{-n})$ n = 0, 1, 2, …, N $K \approx 1.64676$	x → $K[xcos(z) - ysin(z)]$ y → $K[ycos(z) + xsin(z)]$ z → 0 For $\cos(v)$r and $sin(v)$, set $x = 1/K$, $y = 0$ $\tan(v) = \sin(v)/\cos(v)$	x → $K\sqrt{x^2 + y^2}$ y → 0 z → $z + \text{atan}(y/x)$ For $\text{atan}(v)$, set $x = 1, z = 0$ $\text{acos}(w) = \text{atan}\left[\sqrt{1 - w^2}/w\right]$ $\text{asin}(w) = \text{atan}\left[w/\sqrt{1 - w^2}\right]$
Hyperbolic m = -1 $e_n = \text{atanh}(2^{-n})$ n = 1, 2, 3, 4, 4, 5, …13, 13, …, N $K' \approx 0.82815$	x → $K'[xcosh(z) - ysinh(z)]$ y → $K'[ycosh(z) + xsinh(z)]$ z → 0 For $\cosh(v)$ and $\sinh(v)$, set $x = 1/K', y = 0$ $\tanh(v) = \frac{\sinh(v)}{\cosh(v)}$ $e^v = \sinh(v) + \cosh(v)$ $w^t = e^{tln(w)}$	x → $K'\sqrt{x^2 - y^2}$ y → 0 z → $z + \text{atanh}(y/x)$ For $\text{atanh}(w)$, set $x = 1, z = 0$ $\ln(w) = 2\,\text{atanh}\left(\frac{w-1}{w+1}\right)$ $\sqrt{w} = \sqrt{(w + 0.25)^2 + (w - 0.25)^2}$ $\text{acosh}(w) = \ln\left(w + \sqrt{1 - w^2}\right)$ $\text{asinh}(w) = \ln(w + \sqrt{1 + w^2})$

Figure 485 CORDIC operation modes

Usage notes:

- For solving the respective functions, initialize the CORDIC data (X, Y, and Z) with meaningful initial values within the domain of convergence to ensure result convergence. The 'useful domain' listed in the table covers the supported domain of convergence for the CORDIC algorithm and excludes the non-meaningful range(s) for the function
- Function limitations must be considered; for example, setting initial X = 0 for atan(Y/X) is not meaningful. Violations of such function limitations may yield incoherent CORDIC result data
- All data inputs are processed and handled as two's complements. The only exception is the user-option for X result data (only) to be read as an unsigned value
- The only case where the result data is always positive and larger than the initial data is the X result data (only) in the circular vectoring mode; therefore, the user may prefer to use the MSB bit as a data bit instead of a sign bit. By setting X_USIGN = 1, the X result data is processed as unsigned data
- For circular and hyperbolic functions, and due to the corresponding fixed LUT, the Z data is always handled as a signed integer Q(1,28), accessible as Q(1,23). The LUTs contain scaled integer values Q(1,28) of atan(2^{-n}) for n = 0, 1, 2, ..., 27 and atanh(2^{-n}) for n = 1, 2, ..., 27, such that angles in the range [-π,((2^{23}-1)/2^{23})π] are represented by integer values ranging [-1,(2^{23}-1)/2^{23}]. Therefore, Z data is limited (not considering the domain of convergence) to represent angles [-π, ((2^{23}-1)/2^{23})π] for these CORDIC functions. Any calculated value of Z outside of this range will result in an overflow error
- The MPS setting has no effect on Z data. Ensure proper initialization of Z initial data to prevent overflow and incorrect result data
- The CORDIC coprocessor is designed such that, with the correct user setting of MPS > 1, there is no internal overflow of the X and Y data, and the read result data is complete. However, note that in these cases, the higher the MPS setting, the lower the resolution of the result data due to the loss of LSB bit(s)
- The hyperbolic rotation mode is limited in terms of result accuracy since the initial Y data must be set to zero. In other words, the CORDIC coprocessor cannot return an accurate result for cosh(Z)+/-sinh(Z) in a single calculation

34.3.4 Domains of Convergence

This section does not cover the theory of the inherent limits of CORDIC convergence, which vary with different CORDIC functions in rotation or vectoring mode. For the convergence of result data, there are limitations to the magnitude or value of the initial data and the corresponding useful data form, depending on the operating mode used. The following are generally applicable regarding the convergence of CORDIC result data.

Rotation mode: Z data must converge towards 0. The initial Z data must be equal to or smaller than $\Sigma d_n \cdot e_n$, where e_n is always decreasing for iteration n. In other words, |Z| ≤ Sum of LUT. In a circular function, this means the |Z| ≤ integer value represents 1.74 radians. In a hyperbolic function, the |Z| ≤ integer value represents 1.11 radians.

Vectoring mode: Y data must converge towards 0. The values of initial X and Y are limited by the Z function, which is dependent on the corresponding LUT. In a circular function, |atan(Y/X)| ≤ 1.74 radians. In a hyperbolic function, |atanh(Y/X)| ≤ 1.11 radians. In vectoring mode, the additional requirement is that X > 0.

Circular rotation mode: The full range of Z input [-2^{23}, (2^{23}-1)] representing angles [-π, ((2^{23}-1)/2^{23})π] is supported. There are no limitations on initial X and Y inputs, except for overflow considerations, which can be overcome with MPS settings.

Circular vectoring mode: The full range of X and Y inputs [-2^{23},(2^{23}-1)] are supported, while the Z initial value should satisfy |Z| ≤ π/2 to prevent possible Z result data overflow.

While the operating modes of the CORDIC coprocessor are generally bound by these convergence limits, there are exceptions for the circular rotation and circular vectoring modes, which use additional pre- and post-processing logic to support a wider range of inputs.

***Note**: Considerations should also be given to function limitations, such as the meaning of the result data (for example, dividing by zero is not meaningful). The 'useful domain', included within Figure 485 for each of the main functions, attempts to cover both CORDIC convergence and the useful range of the function.*

***Note**: Input values may be within the domain of convergence; however, this does not guarantee a fixed level of accuracy of the CORDIC result data.*

34.3.5 Overflow Considerations

Besides the domain of convergence, consider the limitations on the magnitude of input data to prevent result data overflow.

In all operating modes, the CORDIC coprocessor handles data overflow in the same way. Prevent overflow for X and Y data by setting the MPS bit correctly; the bit's value is partly based on the CORDIC coprocessor operating mode and the application data.

The MPS setting has no effect on the Z data. For circular and hyperbolic functions, any value of Z outside of the range $[-\pi, ((2^{23}-1)/2^{23})\pi]$ cannot be represented and results in a Z data overflow error. Note that kernel data Z has values in the range $[-\pi, ((2^{23}-1)/2^{23})\pi]$ scaled to the range $[-2^{23}, (2^{23}-1)]$, so the written and read values of Z data are always normalized. For a linear function where Z is a real value, the magnitude of Z must not exceed 4 integer bits.

The CORDIC coprocessor accepts (initial) data X, Y, and Z inputs in two's complement format. The result data is also in two's complement format. The only exception is for the X result data in circular vectoring mode. The X result data has a default data format of two's complement, but the user can select via bit CON.X_USIGN = 1 for the X result data to be read as an unsigned value. This option prevents a potential overflow of the X result data (taken together with the MPS setting), as the MSB bit is now a data bit. Note that setting bit CON.X_USIGN = 1 is only effective when operating in the circular vectoring mode, which always yields result data that is positive and larger than the initial data.

34.3.6 Data Format

The CORDIC coprocessor accepts initial data (24-bit) X, Y, and Z inputs in two's complement format. The result data (24-bit) is also in two's complement format. The only exception is for the X result data in circular vectoring mode. The X result data has a default data format of two's complement, but the user can select via bit CON.X_USIGN = 1 for the X result data to be read as an unsigned value. This option prevents a potential overflow of the X result data (taken together with the MPS setting), as the MSB bit is now a data bit. Note that setting bit CON.X_USIGN = 1 is only effective when operating in the circular vectoring mode, which always yields result data that is positive and larger than the initial data.

***Note**: The X, Y, and Z initial data registers (MXCORDIC_1_0_CORDX, MXCORDIC_1_0_CORDY, and MXCORDIC_1_0_CORDZ) and result data registers (MXCORDIC_1_0_CORRX, MXCORDIC_1_0_CORRY, and MXCORDIC_1_0_CORRZ) all use the bit[31:8] to save data, and bit[7:0] is reserved.*

Generally, the input data for X and Y can be an integer or a rational number (fraction). However, in any calculation, the data form must be the same for both X and Y. Also, in the case of fractions, X and Y must have the same number of bits for decimal places. The Z data is always handled as an integer based on the normalization factor for a circular or hyperbolic function.

See the Normalized Result Data section for details on data normalization.

34.3.7 Performance

The CORDIC calculation time increases linearly with increased precision. Increased precision is achieved with a greater number of iterations, which requires an increased width of the data parameters. The CORDIC coprocessor uses barrel shifters for data shifting. For a fixed number of 28 iterations per calculation, the total time from the start of the calculation to the instant the EVFR.CDEOC flag is set is 28/4+3 = 10 kernel clock cycles. Note that the EVFR.CDERR flag is valid only for one kernel clock cycle after the onset of EVFR.CDEOC. This timing for one complete calculation is applicable to those modes that involve additional data processing.

***Note**: In hyperbolic modes, the iterations start from n = 1, and the iterations n = 4 and 13 must be repeated to obtain convergence. As a result, a total of n = 27 iterations are executed, except n = 28.*

***Note**: The above timing excludes the time taken for software loading of initial data and reading of the final result data to and from the six data registers.*

34.3.8 Look-Up Tables

This section describes the look-up tables used in the CORDIC coprocessor kernel. The LUTs are 29 bits wide. Each entry of the atan and atanh LUT is divided into 1 sign bit (MSB), followed by a 28-bit fractional part, Q(1,28).

The Z data is a normalized representation of the actual angle. The internal scaling is such that $[-\pi, ((2^{23}-1)/2^{23})\pi]$ is equivalent to $[-2^{28},(2^{28}-1)]$. The last 5 LSB bits are truncated, as 24-bit data is transferred to the data bus when addressed. From the user's point of view, the angles $[-\pi,((2^{23}-1)/2^{23})\pi]$ are therefore represented by the range $[-2^{23},(2^{23}-1)]$.

Table 356 Circular mode look-up table

Iteration No. n	Scaled atan(2^{-n}) in hex	Iteration No. n	Scaled atan(2^{-n}) in hex
0	4000000	14	145F
1	25C80A4	15	A30
2	13F670B	16	518
3	A2223B	17	28C
4	5161A8	18	146
5	28BAFC	19	A3
6	145EC4	20	51
7	A2F8B	21	29
8	517CA	22	14
9	28BE6	23	A
10	145F3	24	5
11	A2FA	25	3
12	517D	26	1
13	28BE	27	1

In hyperbolic mode, the iterations n = 4 and n = 13 are repeated, while the iterations n = 0 and n = 27 are skipped. In order to converge at the same number of clock cycles as the circular mode, n = 27 is skipped.

Table 357 Hyperbolic mode look-up table

Iteration No. n	Scaled atanh(2^{-n}) in hex	Iteration No. n	Scaled atanh(2^{-n}) in hex
0	-	14	145F
1	2CC2F12	15	A30
2	14D01AC	16	518
3	A3D4E0	17	28C
4	5197FC	18	146
5	28C1C7	19	A3
6	145F9D	20	51
7	A2FA6	21	29
8	517CE	22	14
9	28BE6	23	A
10	145F3	24	5
11	A2FA	25	3
12	517D	26	1
13	28BE	27	-

34.3.9 Interrupt Generation

The CORDIC interrupt request signal is generated as illustrated in the following diagram:

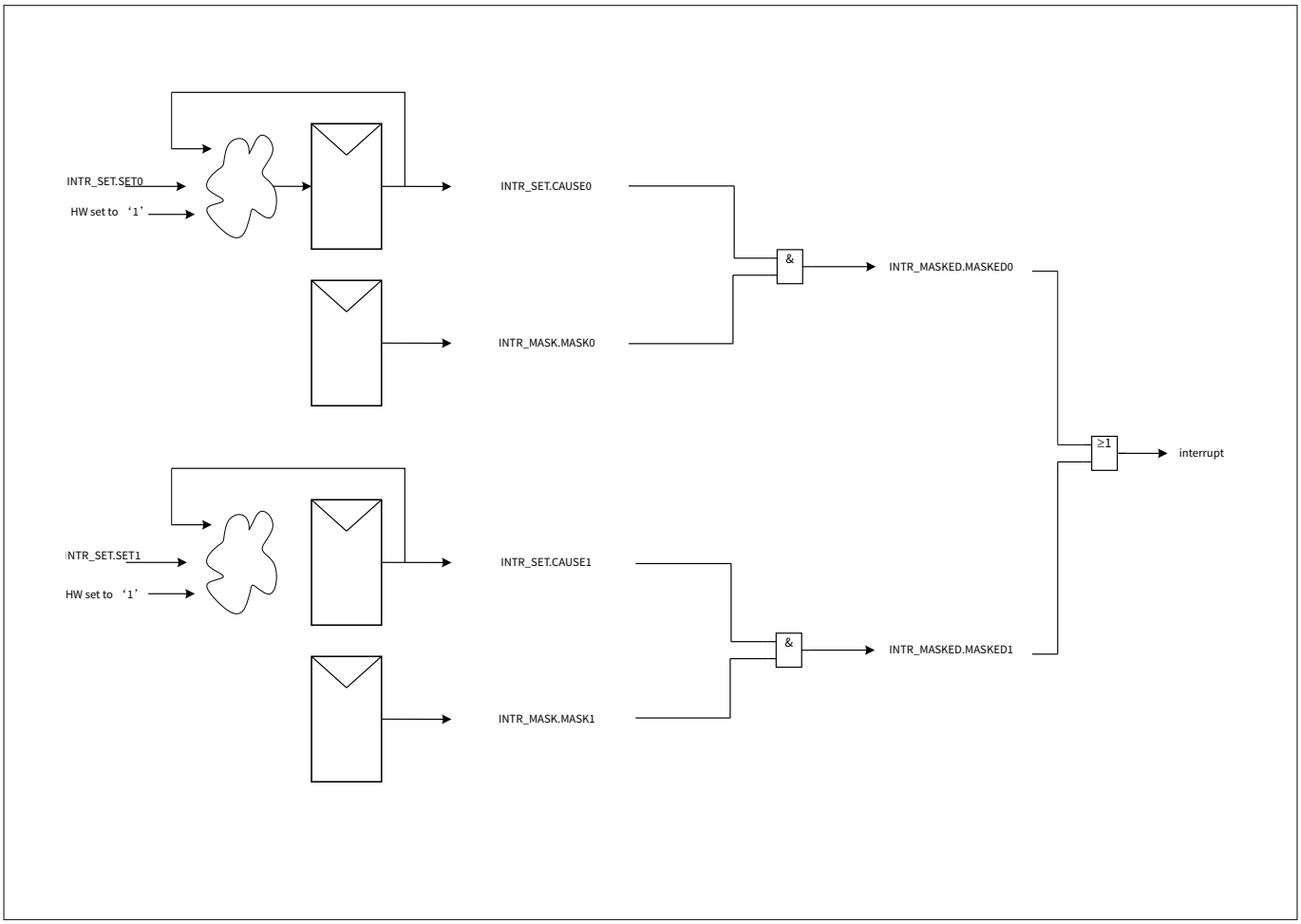


Figure 486 CORDIC interrupts generation

Hardware can set the MXCORDIC_1_0_INTR register interrupt flag bits to 1 when the event signal is detected. The MXCORDIC_1_0_INTR.CDEOC is the end of the calculation event flag. The MXCORDIC_1_0_INTR.CDERR is the calculation error event flag. Software can set the MXCORDIC_1_0_INT_SET register interrupt cause bits to 1. The MXCORDIC_1_0_INTR_MASK register can hold the interrupt mask bits of co-located interrupt cause bits in MXCORDIC_1_0_INT; the co-located interrupt cause bit only causes an interrupt when its mask bit is '1'. The MXCORDIC_1_0_INTR_MASKED register is a logical AND of the MXCORDIC_1_0_INT_INTR and MXCORDIC_1_0_INTR_MASK registers. The user can read out the interrupt request signal directly via the MXCORDIC_1_0_INTR_MASKED register.

***Note**: For the PSC3M/P8, PSC3M/P7 device due to faster calculation by the CORDIC hardware, the use of non-blocking function APIs may not give any speed advantages. It is suggested that the user use the blocking functions directly.*

34.3.10 Notes on MMIO Access

- All registers support only 32-bit access
  - 8- or 16-bit accesses will result in an AHB error (HRESP = 1)
- The result data from CORDIC calculation will be written to the respective result registers. Any read access on these registers while STAT.BSY is set (a calculation is still running) will cause the kernel to issue a bus wait until BSY is reset.

34.4 Power, Clock, and Reset

34.4.1 Power Supply

CORDIC is available only in active power mode. If the system is in DeepSleep mode, the power domain in which CORDIC is placed is powered down. All flops inside CORDIC are reset without retention. There are no power-related interfaces (P- or Q-channels) in the CORDIC. The CORDIC IP does not implement any internal power domains or power switches.

34.4.2 Clock

CORDIC runs on a single clock input (*clk_sys*). This clock is synchronous to the m33syscpuss clock. The clock can be gated centrally (in MXSPERI) by the SL_CTL.ENABLED MMIO.

34.4.3 Reset

CORDIC has one asynchronous reset signal. The reset can be asserted asynchronously, but it has to be de-asserted synchronously to the system clock. There is no reset synchronization inside this module. The reset synchronization is done inside MXSPERI.

34.5 Registers summary

Table 358 CORDIC registers summary

Register	Function
MXCORDIC_1_0_CTL	module enable/disable register
MXCORDIC_1_0_ID	module identification register
MXCORDIC_1_0_INTR	interrupt cause register
MXCORDIC_1_0_INTR_SET	interrupt set register
MXCORDIC_1_0_INTR_MASK	interrupt mask register
MXCORDIC_1_0_INTR_MASKED	Interrupt Masked Register
MXCORDIC_1_0_KEEP	CORDIC keep register
MXCORDIC_1_0_CON	CORDIC control register
MXCORDIC_1_0_CORDX	CORDIC X data register
MXCORDIC_1_0_CORDY	CORDIC Y data register
MXCORDIC_1_0_CORDZ	CORDIC Z data register

(table continues...)

Table 358 (continued) CORDIC registers summary

Register	Function
MXCORDIC_1_0_CORRX	CORDIC X result register
MXCORDIC_1_0_CORRY	CORDIC Y result register
MXCORDIC_1_0_CORRZ	CORDIC Z result register
MXCORDIC_1_0_STAT	CORDIC status register
MXCORDIC_1_0_START_CMD	CORDIC start command register

35 Low-power comparator (LPCOMP)

PSC3M/P8, PSC3M/P7 MCU has two low-power comparators, which can perform analog-signal comparison in all system power modes. The low-power comparator output can be:

- Inspected by the CPU
- Used as an interrupt/wake-up source to the CPU when in CPU Sleep mode
- Used as a wake-up source to system resources when in System DeepSleep or Hibernate mode, or fed to a GPIO or trigger multiplexer (see Trigger multiplexer (Trigger MUX) chapter) as an asynchronous or synchronous signal (level or pulse)

35.1 Features

The PSC3M/P8, PSC3M/P7 MCU low-power comparators have the following features:

- Programmable power and speed
- Ultra-low-power mode support
- Each comparator features a one-sided hysteresis option
- Rising edge, falling edge, and combined rising and falling edge detection at the comparator output
- Local reference voltage generation
- Wake-up source from low-power modes

35.2 Architecture

The following figure shows the block diagram for the low-power comparator.

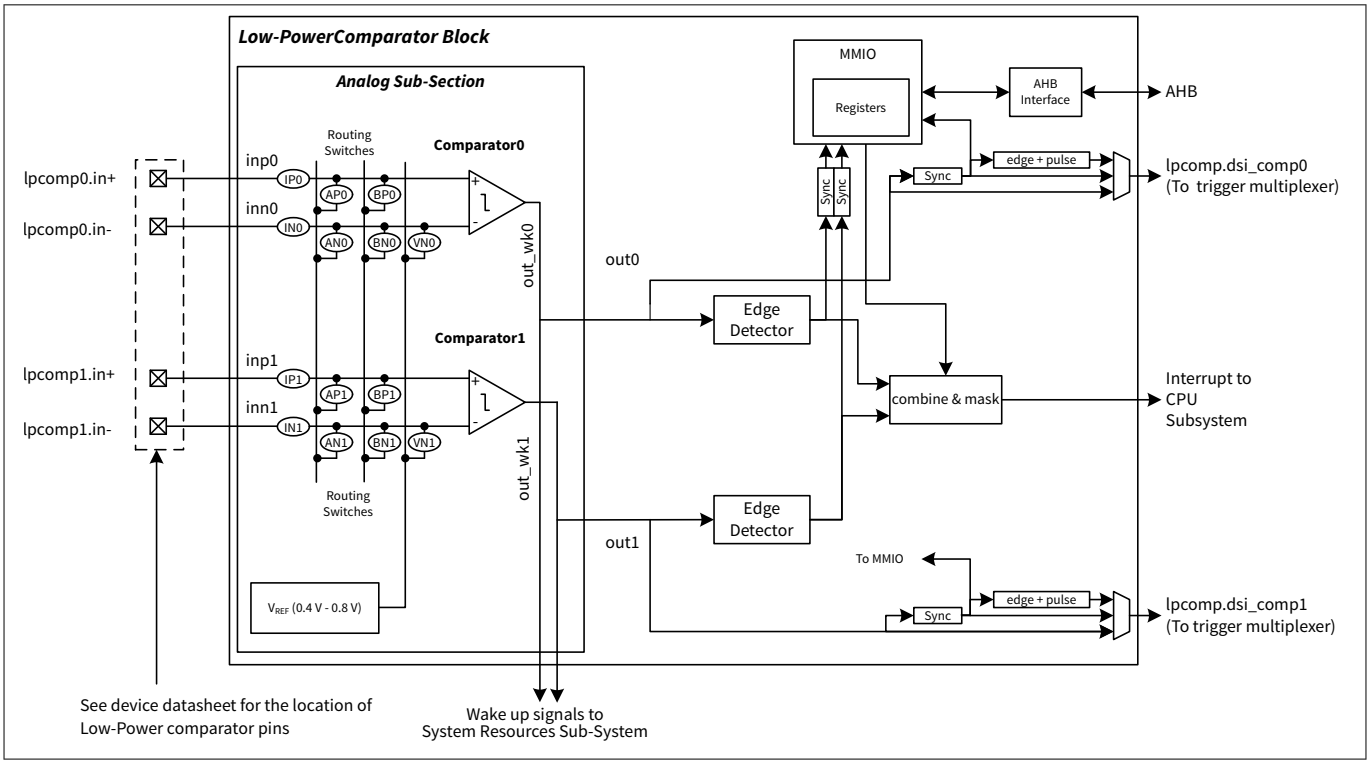


Figure 487 Low-power comparator block diagram

The block, on a higher level, consists of two subsections – analog and digital logic. The analog subsection consists of two comparators with the option of hysteresis, switches to make connection to the inputs, and the reference voltage source. The output from the comparators is driven to the System Resources Subsystem (SRSS) directly for device wake-up from Hibernate mode.

The comparator output is also driven to the digital subsection. This subsection provides different output options, such as synchronous level and pulse output, and options to trigger interrupts on positive, negative, or both edges of the comparator outputs to wake up the device from Sleep and DeepSleep modes.

The following sections describe the operation of the PSC3M/P8, PSC3M/P7 MCU low-power comparator, including input configuration, power and speed modes, output and interrupt configuration, hysteresis, and wake-up from low-power modes.

35.2.1 Input configuration

Low-power comparators operate with the following input options:

- Compare two voltages from external pins
- Compare external signals with a locally generated reference voltage. Note that this voltage is not a precision reference and can vary from 0.4 V to 0.8 V

The internal routing switches (IP0, AP0, BP0, IN0, AN0, BN0, VN0, IP1, AP1, BP1, IN1, AN1, BN1, VN1) shown in Figure 487 are controlled using the LPCOMP_CMP0_SW, LPCOMP_CMP1_SW, LPCOMP_CMP0_SW_CLEAR, and LPCOMP_CMP1_SW_CLEAR registers. See the register reference manual to understand how to control these switches.

If the inverting input of a comparator is routed to a local voltage reference, set the LPREF_EN bit in the LPCOMP_CONFIG register to enable the voltage reference. See the register reference manual for the details of these registers.

35.2.2 Output and interrupt configuration

The output can be configured to one of the following options using the control registers (LPCOMP_CMP0_CTRL and LPCOMP_CMP1_CTRL):

- Direct comparator output
- Synchronous comparator output - level
- Synchronous comparator output - pulse

The DSI_BYPASS0 and DSI_BYPASS1 bits of the control register select the direct or synchronous comparator output.

The DSI_LEVEL0 and DSI_LEVEL1 bits of the control register select between pulse or level output. Pulse output is two clk_sys cycles, and it can be generated on a positive edge, a negative edge, or on both edges configured using the INTTYPE0 and INTTYPE1 bits of the control register. Table 359 summarizes the configuration.

Table 359 Output configuration

DSI_BYPASSx	DSI_LEVELx	Output
1	x	Direct comparator output
0	1	Synchronous comparator output - level
0	0	Synchronous comparator output - pulse

In addition, comparator outputs can be routed to GPIOs and other on-chip peripherals, such as DMA and TCPWMs, using the trigger multiplexer. See the Trigger multiplexer (Trigger MUX) chapter for more details.

Firmware readouts of comparator 0 and comparator 1 outputs are available at the OUT0 and OUT1 bits of the LPCOMP_STATUS register (Table 360). The output of each comparator is connected to a corresponding edge detector block. This block determines the edge that triggers the interrupt. The edge selection and interrupt enable are configured using the INTTYPE0 and INTTYPE1 bitfields in the LPCOMP_CMP0_CTRL and LPCOMP_CMP1_CTRL registers for comparator 0 and comparator 1, respectively. Using the INTTYPE0 and

INTTYPE1 bits, the interrupt type can be selected to disabled, rising edge, falling edge, or both edges, as described in Table 360.

During an edge event, the comparator triggers an interrupt. The interrupt request is registered in the COMP0 bit and COMP1 bit of the LPCOMP_INTR register for comparator 0 and comparator 1, respectively. Both comparator 0 and comparator 1 share a common interrupt signal output (see Table 360), which is the logical OR of the two interrupts and mapped as the low-power comparator block's interrupt in the CPU NVIC. Refer to the interrupts section for details. If both comparators are used in a design, read the COMP0 and COMP1 bits of the LPCOMP_INTR register in the interrupt service routine to know which one triggered the interrupt. Alternatively, the COMP0_MASK bit and the COMP1_MASK bit of the LPCOMP_INTR_MASK register can be used to mask the comparator 0 and comparator 1 interrupts to the CPU. Only the masked interrupts are serviced by the CPU. After the interrupt is processed, clear the interrupt by writing a '1' to the COMP0 and COMP1 bits of the LPCOMP_INTR register in the firmware. If the interrupt to the CPU is not cleared, it says active regardless of the next compare events and interrupts the CPU continuously. Refer to the Interrupts chapter for details.

The LPCOMP_INTR_SET register bits [1:0] can be used to assert an interrupt for firmware debugging.

In the DeepSleep mode, the wake-up interrupt controller (WIC) can be activated by a comparator edge event, which then wakes up the CPU. Similarly, in the Hibernate mode, the LPCOMP can wake up the system resources subsystem(SRSS), which includes the clocks. Thus, the LPCOMP can monitor a specified signal in low-power modes. See the Power supply and monitoring chapter and the Device power modes chapter for more details.

Table 360 Output and interrupt configuration

Register[Bit_Pos]	Bit_Name	Description
LPCOMP_STATUS[0]	OUT0	Current/Instantaneous output value of Comparator 0
LPCOMP_STATUS[16]	OUT1	Current/Instantaneous output value of Comparator 1
LPCOMP_CMP0_CTRL[7:6]	INTTYPE0	Sets on which edge Comparator 0 will trigger an IRQ 00: Disabled 01: Rising edge 10: Falling edge 11: Both rising and falling edges
LPCOMP_CMP1_CTRL[7:6]	INTTYPE1	Sets on which edge Comparator 1 will trigger an IRQ 00: Disabled 01: Rising edge 10: Falling edge 11: Both rising and falling edges
LPCOMP_INTR[0]	COMP0	Comparator 0 Interrupt: hardware sets this interrupt when Comparator 0 triggers Write a '1' to clear the interrupt
LPCOMP_INTR[1]	COMP1	Comparator 1 Interrupt: hardware sets this interrupt when Comparator 1 triggers Write a '1' to clear the interrupt
LPCOMP_INTR_SET[0]	COMP0	Write a '1' to trigger the software interrupt for Comparator 0
LPCOMP_INTR_SET[1]	COMP1	Write a '1' to trigger the software interrupt for Comparator 1
LPCOMP_INTR_MASK[0]	COMP0_MASK	Write a ‘1’ to enable the Comparator 0 interrupt
LPCOMP_INTR_MASK[1]	COMP1_MASK	Write a ‘1’ to enable the Comparator 1 interrupt

(table continues...)

Table 360 (continued) Output and interrupt configuration

Register[Bit_Pos]	Bit_Name	Description
LPCOMP_INTR_MASKED[0]	COMP0_MASKED	If 1, indicates that Comparator 0 interrupt has triggered
LPCOMP_INTR_MASKED[1]	COMP1_MASKED	If 1, indicates that Comparator 1 interrupt has triggered

35.2.3 Power mode and speed configuration

The low-power comparators can operate in three power modes:

- Normal
- Low-power
- Ultra-low-power

The power or speed setting for comparator 0/1 is configured using the MODE0/1 bitfield of the LPCOMP_CMPx_CTRL register, where x = 0,1, respectively. The power consumption and response time vary depending on the selected power mode; power consumption is highest in fast mode and lowest in ultra-low-power mode, and response time is fastest in fast mode and slowest in ultra-low-power mode. Refer to the device datasheet for specifications for the response time and power consumption for various power settings.

The comparators can also be enabled or disabled using these bitfields, as described in Table 361.

***Note**: The output of the comparator may glitch when the power mode is changed while the comparator is enabled. To avoid this, disable the comparator before changing the power mode.*

Table 361 Comparator power mode selection bits

Register[Bit_Pos]	Bit_Name	Description
LPCOMP_CMP0_CTRL[1:0]	MODE0	Comparator 0 power mode selection 00: Off 01: Ultra-low-power operating mode. Use this mode when the device is in DeepSleep or Hibernate mode 10: Low-power operating mode 11: Normal, full-speed, full-power operating mode See the datasheet for electrical specifications in each power mode.
LPCOMP_CMP1_CTRL[1:0]	MODE1	Comparator 1 power mode selection 00: Off 01: Ultra-low-power operating mode. Use this mode when the device is in DeepSleep or Hibernate mode 10: Low-power operating mode 11: Normal, full-speed, full-power operating mode See the datasheet for electrical specifications in each power mode.

Additionally, the entire low-power comparator system can be enabled or disabled globally using the LPCOMP_CONFIG[31] bit. See the register reference manual for details on these bitfields.

35.2.4 Hysteresis

For applications that compare signals close to each other and slow-changing signals, hysteresis helps to avoid oscillations at the comparator output when the signals are noisy. For such applications, a fixed hysteresis may be enabled in the comparator block. See the device datasheet for the hysteresis voltage range.

The hysteresis level is enabled/disabled by using the HYST0 and HYST1 bitfields in the LPCOMP_CMP0_CTRL and LPCOMP_CMP1_CTRL registers for comparator 0 and comparator 1, as described in Table 362.

Table 362 Hysteresis control bits

Register[Bit_Pos]	Bit_Name	Description
LPCOMP_CMP0_CTRL[5]	HYST0	Enable/Disable hysteresis to Comparator 0 1: Enable Hysteresis 0: Disable Hysteresis See the datasheet for the hysteresis voltage range.
LPCOMP_CMP1_CTRL[5]	HYST1	Enable/Disable hysteresis to Comparator 1 1: Enable Hysteresis 0: Disable Hysteresis See the datasheet for the hysteresis voltage range.

35.2.5 Wake up from low-power modes

The comparator is operational in the device's low-power modes, including Sleep, DeepSleep, and Hibernate modes. The comparator output can wake the device from Sleep and DeepSleep modes. The comparator output can generate interrupts in DeepSleep mode when enabled in the LPCOMP_CONFIG register. Do not set the INTTYPEx bits in the LPCOMP_CMPx_CTRL register to disabled, and set the INTR_MASKx bit in the LPCOMP_INTR_MASK register for the corresponding comparator to wake the device from low-power modes.

The comparators have the following restrictions when operating in DeepSleep and Hibernate modes:

- Comparators should be operated in ultra-low-power mode
- Input to the comparator should come from the dedicated pins. The internal reference, VREF, is available and can be connected to the inverting inputs of the comparator

***Note**: Set the LPREF_EN bit in the LPCOMP_CONFIG register to '1', to use the block in DeepSleep and Hibernate modes. This bit controls the local reference voltage VREF as well as the bias current required for the block's operation in low-power modes.*

35.2.6 Comparator clock

The comparator uses the system main clock clk_sys as the clock which is from CLK_HF[1] for interrupt and output synchronization. See the Clocking system chapter for more details.

35.2.7 Using low-power comparator

Follow these steps to configure the comparator:

1. Configure the internal routing switches to connect the inputs to the comparator. See the LPCOMP_CMP0_SW, LPCOMP _CMP1_SW, LPCOMP_CMP0_SW_CLE AR, and LPCOMP_CMP1_SW_CLEAR registers in the registers reference manual to understand how to control the internal routing switches
2. Enable the LPCOMP block by writing into the ENABLED bit of the LPCOMP_CONFIG register

3. Configure the hysteresis and the output type – direct, synchronous-level, or synchronous-pulse – using the LPCOMP_CMP0_CTRL or LPCOMP_CMP1_CTRL register
4. If the internal reference VREF is being used and connected to an inverting input, enable it by writing into the LPREF_EN bit in the LPCOMP_CONFIG register
5. Set the power mode of the comparator by writing into the mode bits of the LPCOMP_CMP0_CTRL or LPCOMP_CMP1_CTRL register. If operation is required during DeepSleep mode and Hibernate mode, make sure that the power mode is configured to ultra-low-power and the LPREF_EN bit of the LPCOMP_CONFIG register is set
6. Provide a delay so that the comparator stabilizes before routing the output or enabling the interrupt in the next step. The duration of the delay depends on the power mode configured. See the device datasheet for the delay numbers
7. Route the output to the GPIOs or to other peripherals by configuring the trigger multiplexer
8. If an interrupt is required, configure the interrupt type by writing into the LPCOMP_CMP0_CTRL or LPCOMP_CMP1_CTRL register, and enable the interrupt by writing into the LPCOMP_INTR_MASK register

35.3 Registers summary

Table 363 Low-power comparator register summary

Register	Function
LPCOMP_CONFIG	LPCOMP global configuration register
LPCOMP_INTR	LPCOMP interrupt register
LPCOMP_INTR_SET	LPCOMP interrupt set register
LPCOMP_INTR_MASK	LPCOMP interrupt request mask register
LPCOMP_INTR_MASKED	LPCOMP masked interrupt output register
LPCOMP_STATUS	Output status register
LPCOMP_CMP0_CTRL	Comparator 0 configuration register
LPCOMP_CMP1_CTRL	Comparator 1 configuration register
LPCOMP_CMP0_SW	Comparator 0 switch control register
LPCOMP_CMP1_SW	Comparator 1 switch control register
LPCOMP_CMP0_SW_CLEAR	Comparator 0 switch control clear register
LPCOMP_CMP1_SW_CLEAR	Comparator 1 switch control clear register

36 Supply Voltage Glitch Sensor (SVGS)

36.1 Introduction

The supply voltage glitch sensor (SVGS) is a peripheral that detects supply voltage manipulations by attackers attempting to disrupt device operation. The SVGS module provides a robust and reliable way to monitor the supply voltage and detect potential glitches or attacks. The block implements high-speed comparators that supervise the microcontroller's core supply (VCCD) and provide protection against disturbances on the operational supply (VDDQ), ensuring immunity of status retention against noise events up to 2 µs.

The voltage glitch sensors are widely deployed in power systems, electronic devices, and industrial control systems to detect abnormal voltage transients. When a glitch is detected, the sensor can trigger alarms or initiate corrective actions to protect the system from potential damage.

36.2 Features

The SVGS module consists of four continuous-time comparators that monitor the supply voltage (VCCD) and detect any overshoot, undershoot, noise, or other anomalies. The module provides a range of features, including:

- VCCD overshoot monitor with a configurable threshold
- VCCD undershoot monitor with a configurable threshold, using a filtered VCCD signal
- VCCD noise monitor with a configurable threshold
- VDDQ undershoot monitor, which detects voltage levels below the brownout threshold

The application software can enable or disable the block and configure interrupts (`svgs_interrupt_IRQ` - IRQ141) for event notification through the registers, allowing it to determine the appropriate action based on the reported event.

The SVGS operates only in Active mode and is disabled in other modes to save power.

36.3 Architecture

The SVGS module consists of four comparators (each with independent enable and independent mask):

- **Under-Voltage Comparison (UVC)**: Detects undershoot events of VCCD
- **Over-Voltage Comparison (OVC)**: Detects overshoot events of VCCD
- **Noise Voltage Comparison (NVC)**: Detects noise amplitudes of VCCD with respect to its DC value
- **Under-Voltage Comparison with Filtered supply (UVFC)**: Detects undershoot events of filtered VCCD

Each comparator has a configurable reference voltage and can be enabled or disabled independently. It can also operate either in half-power mode (lower power and slower detection) or full-power mode (higher power and faster detection).

The SVGS is intended for deployment by the Boot ROM and the secure application on the CM33-Main CPU, and should be configured as secure-only accessible.

***Note**: It is recommended to reset (reboot) the SVGS block after a detected event to ensure reliable continued operation.*

36.4 Functional description

The SVGS_MAIN register block contains control, status, interrupt, and configuration registers. Comparator trip/trim values are stored in SFLASH, from where BootROM loads and applies them during startup.

The application firmware may enable/disable the SVGS module and configure fault or interrupt handling through `svgs_interrupt_IRQ` (IRQ141).

The SVGS module includes several registers that control its operation and provide status information.

The SVGS_CTL register specifically controls the overall operation of the SVGS module.

Table 364 SVGS_CTL register layout and functional description

Register [Bit_Pos]	Bit_Name	Access	Description
SVGS_CTL[3:0]	ENABLE	RW	0101 = Disable the SVGS; all other values enable the module. ***Note**: The SVGS will also be disabled during DEEPSLEEP for power savings.*
SVGS_CTL[8]	FAULT_EN	RW	0 = Disable fault generation 1 = A fault will be generated to the CPUSS fault handler for any enabled comparator. ***Note**: This fault will not be registered if the reset is not masked for the triggered comparator. Reset takes precedence.*

The SVGS_STATUS register indicates the captured events and the internal state of the block.

Table 365 SVGS_STATUS register layout and functional description

Register [Bit_Pos]	Bit_Name	Access	Description
SVGS_STATUS[0]	UVC_CAPTURED	R	1 = UVC event occurred since the last reset.
SVGS_STATUS[1]	OVC_CAPTURED	R	1 = OVC event occurred since the last reset.
SVGS_STATUS[2]	UVFC_CAPTURED	R	1 = UVFC event occurred since the last reset.
SVGS_STATUS[3]	NVC_CAPTURED	R	1 = NVC event occurred since the last reset.
SVGS_STATUS[4]	HVC_CAPTURED	R	1 = HVC event occurred since the last reset.
SVGS_STATUS[7:5]	-	-	Reserved
SVGS_STATUS[8]	EN_SEQ_ACTIVE	R	1 = The hardware enable sequence is active. Software should verify that this sequence has been completed before performing any security-critical operations, ensuring that the SVGS is fully enabled.
SVGS_STATUS[9]	SVGS_ENABLED	R	1 = The SVGS enable sequence is complete, and the module is operating according to the loaded settings 0 = The module is disabled or currently undergoing the enable sequence.

SVGS_CMP_CFG_RO is a read-only register that reports the enable, power, and mask configuration currently applied to all comparators.

Table 366 SVGS_COMP_CFG register layout and functional description

Register [Bit_Pos]	Bit_Name	Access	Description
SVGS_CMP_CFG_RO[0]	UVC_EN	R	1 = Enable UVC
SVGS_CMP_CFG_RO[1]	OVC_EN	R	1 = Enable OVC
SVGS_CMP_CFG_RO[2]	UVFC_EN	R	1 = Enable UVFC
SVGS_CMP_CFG_RO[3]	NVC_EN	R	1 = Enable NVC
SVGS_CMP_CFG_RO[8]	UVC_PWR	R	1 = UVC in full-power mode 0 = UVC in half-power mode
SVGS_CMP_CFG_RO[9]	OVC_PWR	R	1 = OVC in full-power mode 0 = OVC in half-power mode
SVGS_CMP_CFG_RO[10]	UVFC_PWR	R	1 = UVFC in full-power mode 0 = UVFC in half-power mode
SVGS_CMP_CFG_RO[11]	NVC_PWR	R	1 = NVC in full-power mode 0 = NVC in half-power mode
SVGS_CMP_CFG_RO[16]	UVC_MASK	R	1 = UVC masked from affecting svgs_glitch_event
SVGS_CMP_CFG_RO[17]	OVC_MASK	R	1 = OVC masked from affecting svgs_glitch_event
SVGS_CMP_CFG_RO[18]	UVFC_MASK	R	1 = UVFC masked from affecting svgs_glitch_event
SVGS_CMP_CFG_RO[19]	NVC_MASK	R	1 = NVC masked from affecting svgs_glitch_event
SVGS_CMP_CFG_RO[20]	HVC_MASK	R	1 = HVC masked from affecting svgs_glitch_event
SVGS_CMP_CFG_RO[27:24]	SVGS_UVFC_TC	R	Time constant of the UVFC functionality. Each increment in the configured setting increases the time constant. The mapping is as follows: 0h sets the time constant to 20 ns 1h sets the time constant to 24.4 ns 2h sets the time constant to 29.8 ns 3h sets the time constant to 36.4 ns 4h sets the time constant to 44.4 ns 5h sets the time constant to 54.3 ns 6h sets the time constant to 66.2 ns 7h sets the time constant to 81 ns 8h sets the time constant to 98.8 ns Ah sets the time constant to 147.3 ns Bh sets the time constant to 179.9 ns Ch sets the time constant to 219.7 ns Dh sets the time constant to 268.3 ns Eh sets the time constant to 327.5 ns Fh sets the time constant to 400 ns

The SVGS_CMP_TRIP_RO register defines trip thresholds for UVC, OVC, UVFC, and NVC comparators, including their supported voltage steps and ranges.

Table 367 SVGS_CMP_TRIP_RO register layout and functional description

Register [Bit_Pos]	Bit_Name	Access	Description
SVGS_CMP_TRIP_RO[5:0]	SVGS_UVC_TRIP	R	Configured threshold settings for UVC functionality in steps of 6.2 mV. 000000 = 760.00 mV 000001 = 766.2 mV …. 111111 = 1150 mV
SVGS_CMP_TRIP_RO[9:8]	SVGS_OVC_TRIP	R	Configured threshold settings for OVC functionality in steps of 25 mV. 00 = 1250 mV 01 = 1275 mV 10 = 1300 mV 11 = 1325 mV
SVGS_CMP_TRIP_RO[21:16]	SVGS_UVFC_TRIP	R	Configured threshold settings for UVFC functionality in steps of 6.2 mV. 000000 = 760.00 mV 000001 = 766.2 mV …. 111111 = 1150 mV
SVGS_CMP_TRIP_RO[27:24]	SVGS_NVC_TRIP	R	Configured threshold settings for NVC functionality in steps of 10 mV. 0000 = 30 mV 0001 = 40 mV …. 1111 = 180 mV

The SVGS_INTR register provides interrupt status information.

Table 368 SVGS_INTR register layout and functional description

Register [Bit_Pos]	Bit_Name	Access	Description
SVGS_INTR[0]	UVC_GLITCH	RW1C	UVC glitch detected interrupt
SVGS_INTR[1]	OVC_GLITCH	RW1C	OVC glitch detected interrupt
SVGS_INTR[2]	UVFC_GLITCH	RW1C	UVFC glitch detected interrupt
SVGS_INTR[3]	NVC_GLITCH	RW1C	NVC glitch detected interrupt
SVGS_INTR[4]	HVC_GLITCH	RW1C	HVC glitch detected interrupt

***Note**: A field marked RW1C can be read normally, but on write:*

- ***Writing 1** → clears the bit (hardware sets it to 0)*
- ***Writing 0** → has no effect*

The SVGS block must be reset (disabled and then enabled) to clear the interrupt. Otherwise, the event status remains set, and the interrupt will continue to trigger.

The SVGS_INTR_SET register sets the interrupt status.

Table 369 SVGS_INTR_SET register layout and functional description

Register [Bit_Pos]	Bit_Name	Access	Description
SVGS_INTR_SET[0]	UVC_GLITCH	RW1S	Set an interrupt for UVC_GLITCH interrupt.
SVGS_INTR_SET[1]	OVC_GLITCH	RW1S	Set an interrupt for OVC_GLITCH interrupt.
SVGS_INTR_SET[2]	UVFC_GLITCH	RW1S	Set an interrupt for UVFC_GLITCH interrupt.
SVGS_INTR_SET[3]	NVC_GLITCH	RW1S	Set an interrupt for NVC_GLITCH interrupt.
SVGS_INTR_SET[4]	HVC_GLITCH	RW1S	Set an interrupt for HVC_GLITCH interrupt.

***Note**: A field marked RW1S can be read normally, but on write:*

- ***Writing 1** → sets the bit (hardware writes 1)*
- ***Writing 0** → has no effect*

The SVGS_INTR_MASK register masks the interrupts.

Table 370 SVGS_INTR_MASK register layout and functional description

Register [Bit_Pos]	Bit_Name	Access	Description
SVGS_INTR_MASK[0]	UVC_GLITCH	RW	Mask for UVC_GLITCH interrupt.
SVGS_INTR_MASK[1]	OVC_GLITCH	RW	Mask for OVC_GLITCH interrupt.
SVGS_INTR_MASK[2]	UVFC_GLITCH	RW	Mask for UVFC_GLITCH interrupt.
SVGS_INTR_MASK[3]	NVC_GLITCH	RW	Mask for NVC_GLITCH interrupt.
SVGS_INTR_MASK[4]	HVC_GLITCH	RW	Mask for HVC_GLITCH interrupt.

The SVGS_INTR_MASKED register provides the masked interrupt status.

Table 371 SVGS_INTR_MASKED register layout and functional description

Register [Bit_Pos]	Bit_Name	Access	Description
SVGS_INTR_MASKED[0]	UVC_GLITCH	R	Logical and of corresponding request and mask bits.
SVGS_INTR_MASKED[1]	OVC_GLITCH	R	Logical and of corresponding request and mask bits.
SVGS_INTR_MASKED[2]	UVFC_GLITCH	R	Logical and of corresponding request and mask bits.
SVGS_INTR_MASKED[3]	NVC_GLITCH	R	Logical and of corresponding request and mask bits.
SVGS_INTR_MASKED[4]	HVC_GLITCH	R	Logical and of corresponding request and mask bits.

Initialization sequence

During the bootup and initialization, the SVGS trims are loaded by the BootROM. The user application can make the SVGS block operational by enabling it.

- Enable the SVGS module by writing to the SVGS_CTL register

- Wait for the SVGS module to complete its enable sequence, indicated when the SVGS_STATUS register reads 0x200
- Check the SVGS_STATUS register to ensure that the SVGS module is enabled and functioning correctly
- SVGS_STATUS.EN_SEQ_ACTIVE is 1 while the SVGS module is performing an initialization. When SVGS is enabled, it sets SVGS_ENABLED = 1 and EN_SEQ_ACTIVE = 0
- Start normal operation once the SVGS module is fully active

Interrupt handling

To handle interrupts generated by the SVGS module, follow these steps:

- Enable the SVGS interrupt (`svgs_interrupt_IRQ` - IRQ141) by setting the appropriate bit in the SVGS_INTR_MASK register
- Set up an interrupt handler to handle the SVGS interrupt
- In the interrupt handler, check the SVGS_INTR register to determine which comparator triggered the interrupt
- Perform the necessary action to handle the interrupt, such as resetting/rebooting the system or taking corrective action

Fault handling

To handle faults generated by the SVGS module, follow these steps:

- Enable fault generation by setting the FAULT_EN bit in the SVGS_CTL register
- Set up a fault handler to handle the SVGS fault
- In the fault handler, check the SVGS_STATUS register to determine which comparator triggered the fault
- Perform the necessary corrective action to handle the fault condition

Event status is latched in SVGS_STATUS and SVGS_INTR/SVGS_INTR_MASKED regardless of reset, fault, or interrupt events.

36.5 Registers summary

SVGS protected run time registers (accessible by non-privileged software).

Table 372 Register list

Name	Access	Function
SVGS_CTL	RW	SVGS control register
SVGS_STATUS	R	SVGS status register
SVGS_CMP_CFG_RO	R	SVGS comparator configuration
SVGS_CMP_TRIP_RO	R	SVGS comparator trip-point configuration
SVGS_TRIM_RO	R	SVGS trim register
SVGS_INTR	RW1C	SVGS interrupt register
SVGS_INTR_SET	RW1S	SVGS interrupt set register
SVGS_INTR_MASK	RW	SVGS interrupt mask register
SVGS_INTR_MASKED	R	SVGS interrupt masked register

Glossary

ADC
Analog-to-digital converter

AES
Advanced encryption standard

AFE
Analog front end

AMUX
Analog multiplexer

ATOP
Analog top

BLDC
Brushless direct current

BOD
Brownout detect

CAN
Controller area network

CLB
Custom logic block

CORDIC
Coordinate Rotation Digital Computer

CSV
Clock supervision

CTI
Cross trigger interface

DAC
Digital-to-analog converter

DCMP
Digital comparator

DCSG
Differential analog comparator with slope generator

DFU
Device firmware upgrade

DMA
Direct memory access

DSI
Digital signal interconnect

DSP
Digital signal processor

ECC
Error correcting code

ECO
External crystal oscillator

ENOB
Effective number of bits

EOS
End of sampling

EPU
Event processing unit

ETB
Embedded Trace Buffer

ETM
Embedded Trace Macrocell

FET
Field effect transistor

FIFO
First in, First out

FOC
Field-oriented control

FPU
Floating point unit

GPI
General purpose input

GPIO
General-purpose input/output

HPPASS
High-performance programmable analog subsystem

HRPWM
High-resolution pulse-width modulator

HSIOM
High-speed I/O matrix

HWFILT3P3Z
Hardware filter - 3 poles 3 zeros

I-cache
Instruction-cache

I2C
Inter-integrated circuit

IHO
Internal high-speed oscillator

ILO
Internal low-speed oscillator

IMO
Internal main oscillator

IPC
Inter-processor communication

IRQ
Interrupt request

ISR
Interrupt service routine

ITM
Instrumentation Trace Macrocell

LPComp
Low-power comparator

LUT
Lookup table

LVD
Low-voltage detection

LVTTL
Low-voltage transistor-transistor logic

MCU
Microcontroller

MCWDT
Multi-counter watchdog timer

MOTIF
Motion interface

MPU
Memory protection unit

NVIC
Nested Vectored Interrupt Controller

PAL
Programmable array logic

PLD
Programmable logic device

PLL
Phase-locked loops

PMBus
Power management bus

POR
Power-on reset

PPCA
Programmable Power Control Accelerator

ROM
Read-only memory

RSA
Rivest-Shamir-Adleman, a public-key cryptography algorithm

RTC
Real-time clock

RWW
Read-While-Write

S/H
Sample/Hold

SAR
Successive Approximation Register

SAU
Secure Attribution Unit

SCB
Serial communication blocks

SHA
Secure Hash Algorithm

SIL
Safety integrity levels

SMBus
System management bus

SOC
Start of conversion

SPI
Serial Peripheral Interface

SRAM
Static random access memory

SRSS
System resources subsystem

SVGS
Supply voltage glitch sensor

TCPWM
Timer/counter pulse-width modulator

TRNG
True Random Number Generator

TZ
TrustZone

UART
Universal Asynchronous Transmitter Receiver

WCO
Watch crystal oscillator

WIC
Wakeup Interrupt Controller

XRES
External reset input pin

Revision history

Document revision	Date	Description of changes
*D	2026-06-25	Public release.
*E	2026-09-21	Updated document title. Updated device name from **PSOC™ Control C3 (P/M)(8/7)** to **PSC3M/P8, PSC3M/P7** throughout the document. Updated Detailed features. Updated Flash geometry.

Trademarks

PSOC™, formerly known as PSoC™, is a trademark of Infineon Technologies. Any references to PSoC™ in this document or others shall be deemed to refer to PSOC™.

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2026-09-21

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2026 Infineon Technologies AG

All Rights Reserved.

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference

IFX-pfg1736137658118

Important notice

Products which may also include samples and may be comprised of hardware or software or both ("Product(s)") are sold or provided and delivered by Infineon Technologies AG and its affiliates ("Infineon") subject to the terms and conditions of the frame supply contract or other written agreement(s) executed by a customer and Infineon or, in the absence of the foregoing, the applicable Sales Conditions of Infineon. General terms and conditions of a customer or deviations from applicable Sales Conditions of Infineon shall only be binding for Infineon if and to the extent Infineon has given its express written consent.

For the avoidance of doubt, Infineon disclaims all warranties of non-infringement of third-party rights and implied warranties such as warranties of fitness for a specific use/purpose or merchantability. Infineon shall not be responsible for any information with respect to samples, the application or customer's specific use of any Product or for any examples or typical values given in this document.

If this document is marked as 'Preliminary', 'Target', 'Draft' or 'Proposal', Infineon reserves the right to change all information given in this document at any time without notice. Subject to the development and release of a Product for series supply by Infineon, the technical specifications of the Product are set forth in the relevant datasheet provided by Infineon without such marking.

The data contained in this document is exclusively intended for technically qualified and skilled customer representatives. It is the responsibility of the customer to evaluate the suitability of the Product for the intended application and the customer's specific use and to verify all relevant technical data contained in this document in the intended application and the customer's specific use. The customer is responsible for properly designing, programming, and testing the functionality and safety of the intended application, as well as complying with any legal requirements related to its use.

Unless otherwise explicitly approved by Infineon, Products may not be used in any application where a failure of the Products or any consequences of the use thereof can reasonably be expected to result in personal injury.

However, the foregoing shall not prevent the customer from using any Product in such fields of use that Infineon has explicitly designed and sold it for, provided that the overall responsibility for the application lies with the customer. Infineon expressly reserves the right to use its content for commercial text and data mining (TDM) according to applicable laws, e.g. Section 44b of the German Copyright Act (UrhG). If the Product includes security features:

Because no computing device can be absolutely secure, and despite security measures implemented in the Product, Infineon does not guarantee that the Product will be free from intrusion, data theft or loss, or other breaches ("Security Breaches"), and Infineon shall have no liability arising out of any Security Breaches.

If this document includes or references software:

The software is owned by Infineon under the intellectual property laws and treaties of the United States, Germany, and other countries worldwide. All rights reserved. Therefore, you may use the software only as provided in the software license agreement accompanying the software.

If no software license agreement applies, Infineon hereby grants you a personal, non-exclusive, non-transferable license (without the right to sublicense) under its intellectual property rights in the software (a) for software provided in source code form, to modify and reproduce the software solely for use with Infineon hardware products, only internally within your organization, and (b) to distribute the software in binary code form externally to end users, solely for use on Infineon hardware products. Any other use, reproduction, modification, translation, or compilation of the software is prohibited.

For further information on the Product, technology, delivery terms and conditions, and prices, please contact your nearest Infineon office or visit www.infineon.com.