

User Guide

megawin

MG32-M0
User Guide

Version 5.2
Date 2025/12/11

List of contents

1. Document Using	34
1.1. Documentation Conventions	34
1.2. Block Diagram Glossary	34
1.3. Power Operation Mode Indicator	34
2. Chip and System	35
2.1. Chip Overview	35
2.2. Applicable Chips	36
2.2.1. Chip Implementation Summary	36
2.3. Chip Main Block	38
2.3.1. MG32F02A128/U128/A064/U064 Main Block	38
2.3.2. MG32F02V032 Main Block	39
2.3.3. MG32F02N128/K128/N064/K064 Main Block	40
2.4. CPU Core	41
2.4.1. CPU Features	41
2.4.2. ARM Cortex-M0 Processor	41
2.5. Memory Organization	42
2.5.1. CPU Memory Map	43
2.5.2. Peripheral Memory Boundary	44
2.5.3. Boot Modes	48
2.5.4. Flash Code Lock	48
2.6. Chip Debug	49
2.6.1. Chip DAP	49
2.6.2. SWD Interface	49
2.6.3. SWD and ICP Interface Circuit	50
3. System Power	51
3.1. Introduction	51
3.2. Features	51
3.3. Implementation	51
3.3.1. Power Device Implementation	51
3.4. Power Operation Mode	51
3.5. Power Supply	53
3.6. Power Controller Block	54
3.7. Power Voltage Detect	54
3.7.1. POR/LVR Detector	54
3.7.2. Brown-Out Detector	54
3.7.3. Power Voltage Detection Threshold	55
3.8. Interrupt and Reset	56
3.8.1. PW Interrupt Flags	56
3.8.2. PW Reset Events	57
3.9. Register Protect and Lock	57
3.10. Chip Power Mode Control	58
3.10.1. CPU Power Down	58
3.10.2. Internal Devices Control	58
3.10.3. Device Power Enable Control in Power-Down Modes	59
3.10.4. System Power Operation Flow	61
3.11. Wakeup Control	62
3.11.1. Wakeup Event Sources	62
3.11.2. Wakeup and Interrupt	63

3.11.3. CPU Power Down and Wakeup Control Flow	67
3.12. Chip Power Application Circuit	69
4. System Reset	70
4.1. Introduction	70
4.2. Features	70
4.3. Reset Source Controller	71
4.4. Chip Reset	74
4.4.1. Chip Reset Levels	74
4.4.2. Power-On Reset and Chip Reset Timing	75
4.4.3. External Reset Timing	75
4.5. Reset Event	77
4.5.1. Reset Event Source	77
4.5.2. Reset Event Control	77
4.6. Register Protect and Lock	79
4.7. Module Reset	80
4.7.1. Module Software Reset	80
4.7.2. GPIO Reset Control	81
4.7.3. USB Software Reset Control	81
4.8. External Reset Application Circuit	82
5. System Clock	83
5.1. Introduction	83
5.2. Features	83
5.3. Implementation	83
5.3.1. Embedded Clock PLL Implementation	83
5.4. Clock Source Controller	84
5.5. Interrupt and Reset	85
5.5.1. CSC Interrupt Flags	85
5.5.2. CSC Clock Status	85
5.5.3. CSC Reset Events	86
5.6. Register Protect and Lock	86
5.7. System Clock Control	86
5.7.1. System Clock Source	86
5.7.2. High Speed Clock and Low Speed Clock	87
5.7.3. PLL Clock	88
5.7.4. Internal System Clocks	89
5.8. Module Process Clock Control	90
5.8.1. Module Process Clock Select	90
5.8.2. Module Process Clock Enable	91
5.8.3. Device Clock Enable Control in Power-Down Modes	93
5.9. Module Event Clock	94
5.10. Internal Clock Output Control	94
5.11. Crystal Oscillating Circuit	95
6. System Common Control	96
6.1. Introduction	96
6.2. Features	96
6.3. Interrupt and Event	97
6.4. Chip Manufacture ID	97
6.5. System Backup Register	97
7. System Memory	98

7.1.	Introduction	98
7.2.	Features	98
7.2.1.	Embedded Memory	98
7.2.2.	Memory Controller	98
7.3.	Implementation	98
7.3.1.	Embedded Memory Implementation	98
7.4.	Memory Controller	99
7.5.	Enabling and Clock	100
7.6.	Interrupt and Event	100
7.6.1.	Memory Controller Interrupt Control and Reset	100
7.6.2.	MEM Interrupt Flags	100
7.7.	Register Protect and Lock	101
7.8.	Boot and On-Chip Memory	101
7.8.1.	Memory Boot Modes	101
7.8.2.	On-Chip Flash Memory	102
7.8.3.	On-Chip Data RAM	103
7.9.	Memory Controller Function	104
7.9.1.	Flash Memory Access	104
7.9.2.	Hardware Option Byte Flash Memory	106
7.9.3.	ICP/ISP/IAP for Flash Memory	107
7.9.4.	Memory Access Restriction	107
7.9.5.	CPU Code Execution and Hold	108
7.9.6.	Memory Access Error Management	108
7.10.	Flash Memory Configuration for Different Product	109
8.	Hardware Option	110
8.1.	Introduction	110
8.2.	Hardware Option Byte	110
8.3.	CFG Option Register	112
8.3.1.	CFG Register Protect and Lock	112
8.3.2.	Manufacturer ADC Calibration Value	112
8.3.3.	Manufacturer Temperature Sensor Calibration Value	112
8.3.4.	Manufacturer OPA Calibration Value	112
9.	GPIO (General Purpose IO)	113
9.1.	Introduction	113
9.2.	Features	113
9.3.	Implementation	114
9.3.1.	GPIO Implementation	114
9.4.	Control Block	114
9.5.	IO Mode	115
9.5.1.	IO Mode Control Block	115
9.5.2.	IO Configuration	116
9.6.	IO Structure	117
9.6.1.	Analog IO Structure	118
9.6.2.	Digital Input Structure	118
9.6.3.	Push-Pull Output Structure	118
9.6.4.	Open-Drain Output Structure	119
9.6.5.	Quasi-Bidirectional IO Structure	120
9.7.	IO Port Access	121
9.7.1.	GPIO IO Control	121

9.7.2. Set and Clear Control	121
9.7.3. Bit Like IO Control	121
9.8. Alternate Function Select	122
9.8.1. Alternate Function Select Control	122
9.8.2. Alternate Function Select of Special Pins	122
9.8.3. GPIO AFS Locking.....	124
9.9. GPIO Application Circuit.....	125
9.9.1. GPIO Input and Output	125
9.9.2. GPIO Input Filter	126
9.9.3. GPIO Output High Speed and Drive Strength.....	126
10. Interrupt.....	127
10.1. Introduction	127
10.2. Features	127
10.3. Interrupt Structure	128
10.3.1. Interrupt Sources	128
10.3.2. Exception types.....	129
10.3.3. Exception handlers	130
10.3.4. Interrupt Priority	130
10.3.5. Lockup on Cortex-M0.....	130
10.4. Interrupt Control Block.....	131
10.5. Nested Vectored Interrupt Controller	132
10.5.1. NVIC Function.....	132
10.5.2. NVIC Exception Control	132
10.5.3. Level-sensitive and pulse interrupts.....	133
10.5.4. Hard Fault handling.....	134
10.6. Wakeup Interrupt Controller	134
10.7. External Interrupt Controller	135
10.7.1. EXIC Interrupt Control.....	135
10.7.2. External Port Input Interrupt.....	137
11. GPL (General Purpose Logic)	139
11.1. Introduction	139
11.2. Features.....	139
11.3. Implementation.....	139
11.3.1. Chip Implementation	139
11.4. Control Block.....	140
11.5. Clock	141
11.5.1. GPL Clock Control	141
11.6. GPL Function Control.....	142
11.6.1. Byte Order Change	142
11.6.2. Bit Order Change	143
11.6.3. Data Invert	143
11.6.4. Parity Check.....	143
11.6.5. Cyclic Redundancy Check	144
11.6.6. Hardware Divider	145
11.7. GPL DMA Operation.....	146
11.7.1. DMA Module Configure.....	146
11.7.2. GPL DMA Control	146
12. DMA (Direct Memory Access)	147
12.1. Introduction	147

12.2. Features	147
12.3. Implementation	147
12.3.1. Chip Implementation	147
12.4. Control Block	148
12.5. IO Lines	149
12.5.1. IO Signals	149
12.5.2. IO Configure	149
12.6. Enabling and Clock	149
12.7. Interrupt and Event	149
12.7.1. DMA Interrupt Control and Status	149
12.7.2. DMA Interrupt Flags	149
12.8. DMA Control	151
12.8.1. DMA Source and Destination	151
12.8.2. DMA Channel Arbitrator	154
12.8.3. DMA Channel Operation	154
12.8.4. DMA SRAM Using	156
12.9. DMA Transaction	157
12.9.1. DMA Transaction Configuration and Sequence	157
12.9.2. DMA Transaction under SLEEP	157
12.9.3. Peripheral DMA RX Request and Acknowledge	158
12.9.4. Peripheral DMA TX Request and Acknowledge	159
12.9.5. Peripheral DMA Transaction Hold	159
12.9.6. Peripheral Interrupt Flag Control	160
12.10. DMA External Request Trigger Input	163
13. EMB (External Memory Bus)	164
13.1. Introduction	164
13.2. Features	164
13.3. Implementation	164
13.3.1. Chip Implementation	164
13.4. Control Block	165
13.5. IO Lines	165
13.5.1. IO Signals	165
13.5.2. IO Configure	166
13.6. Enabling and Clock	166
13.6.1. EMB Global Enable	166
13.6.2. EMB Clock Control	166
13.7. Interrupt and Event	167
13.7.1. EMB Interrupt Control and Status	167
13.7.2. EMB Interrupt Flags	167
13.8. EMB IO Control	168
13.8.1. EMB Clock and Command Signals Control	168
13.8.2. EMB Address and Data Signals Control	171
13.8.3. Signal Mapping Suggestion for External Devices	171
13.9. EMB Memory Control	172
13.9.1. EMB Memory Space	172
13.9.2. AHB to External Memory Transactions	172
13.9.3. EMB Write Protection	174
13.9.4. EMB Timing Control	174
13.10. EMB Address and Data Interface Mode	176

13.10.1. EMB 16bit MA and 16bit MD	178
13.10.2. EMB 16bit MA and multiplexed 16bit MD	179
13.10.3. EMB Multiplexed 16bit MAD with 2 Address Phase	180
13.10.4. EMB 16bit MA and 8bit MD	181
13.10.5. EMB 16bit MA and multiplexed 8bit MAD	182
13.10.6. EMB Multiplexed 8bit MAD with 2 Address Phase	183
13.11. EMB Device Interface and Timing	184
13.11.1. SRAM Interface and Access Timing	184
13.11.2. NOR-Flash Interface and Access Timing	187
13.11.3. NAND-Flash Interface and Access Timing	190
13.11.4. 8080 LCD Interface and Access Timing	192
13.12. EMB DMA Operation	195
13.12.1. DMA Module Configure	195
13.12.2. EMB DMA Control	195
13.12.3. EMB Interrupt Flag Control during DMA	195
13.13. EMB Application Note	195
13.13.1. Pin Suggestion for EMB Signal	195
13.13.2. EMB and SPI Flash to LCD Transaction	197
14. APB (APB Common Control)	198
14.1. Introduction	198
14.2. Features	198
14.3. Implementation	198
14.3.1. Chip Implementation	198
14.4. Interrupt and Event	199
14.4.1. APB Interrupt Control and Status	199
14.4.2. APB Interrupt Flags	199
14.5. Timer Common Control	200
14.5.1. Timer Synchronous Enable Control	200
14.5.2. Timer Common Trigger/Clock Source Select	200
14.6. OBM Control	202
14.6.1. OBM Output Channel	203
14.6.2. OBM Break Channel	204
14.6.3. OBM Operation Mode	204
14.7. IR Control	205
14.7.1. IR Control Interface	205
14.7.2. IR Transmission Modulation	206
14.8. NCO Control	207
14.8.1. NCO Block and Clock Input	207
14.8.2. NCO Clock Output	207
14.9. Multi-Function Signal	209
15. APX (APB Extended Control)	210
15.1. Introduction	210
15.2. Features	210
15.3. Implementation	210
15.3.1. Chip Implementation	210
15.4. Interrupt and Event	211
15.4.1. APX Interrupt Control and Status	211
15.4.2. APX Interrupt Flags	211
15.5. CCL Control	212

15.5.1. CCL Block	212
15.5.2. CCL Input and Truth Table.....	212
15.5.3. CCL Filter and Edge Detect	214
15.5.4. CCL Sequential Logic and Output.....	214
15.6. ASB Control	215
15.6.1. ASB Block	215
15.6.2. ASB Connection for Application	216
15.6.3. ASB Channel Control	217
15.6.4. ASB Data Transmit	218
15.6.5. ASB Bus Reset	219
15.6.6. ASB Channel Synchronous Mode.....	220
15.7. SDT Control.....	221
15.7.1. SDT Block	221
15.7.2. SDT Input and State	221
15.7.3. SDT Output	222
15.8. APX DMA Operation.....	222
15.8.1. DMA Module Configure.....	222
15.8.2. APX DMA Control	222
15.8.3. APX Interrupt Flag Control during DMA	222
16. I2C (Inter Integrated Circuit).....	223
16.1. Introduction	223
16.2. Features	223
16.3. Implementation	224
16.3.1. Chip Implementation	224
16.3.2. Modules' Functions	224
16.4. Control Block.....	225
16.5. IO Lines	225
16.5.1. IO Signals	225
16.5.2. IO Configure.....	225
16.6. Enabling and Clock	226
16.6.1. I2C Global Enable	226
16.6.2. I2C Clock Control	226
16.7. Interrupt and Event.....	227
16.7.1. I2C Interrupt Control and Status	227
16.7.2. I2C Subrange Interrupt.....	227
16.7.3. I2C Interrupt Flags	228
16.7.4. I2C Control Flags	229
16.8. I2C Connection for Application	230
16.9. I2C Fundamental Control	231
16.9.1. I2C Fundamental Protocol	231
16.9.2. I2C Control Mode	231
16.9.3. I2C Slave Address	231
16.9.4. I2C Data Register and Shift Buffer	231
16.9.5. I2C Access Command	232
16.9.6. I2C START/STOP/Data-Change.....	232
16.10. I2C Byte Mode Control.....	233
16.10.1. I2C Byte Mode Transmit and Receive.....	233
16.10.2. I2C Event Code.....	233
16.10.3. I2C Byte Mode Control Flow	234

16.11. I2C Buffer Mode Control	243
16.11.1. I2C Buffer Mode Data Buffer	243
16.11.2. I2C Data Buffer Control	244
16.11.3. I2C Buffer Mode Access Command	244
16.11.4. I2C Master Transmit and Receive	245
16.11.5. I2C Slave Transmit and Receive	247
16.12. I2C Function Control	249
16.12.1. I2C Signal Drive Control Timing	249
16.12.2. I2C Slave SCL Stretching	250
16.12.3. Wakeup from STOP	251
16.12.4. I2C Timeout Timer Control	251
16.12.5. I2C NACK Control	252
16.13. I2C Error Management	253
16.13.1. I2C Bus Error Event	253
16.13.2. I2C Master Transmission NACK Detect	253
16.13.3. I2C Slave Data Overrun under SCL Stretching Disable	254
16.14. I2C DMA Operation	255
16.14.1. DMA Module Configure	255
16.14.2. I2C DMA Control	255
16.14.3. I2C Interrupt Flag Control during DMA	255
17. UART (Universal Asynchronous Receiver Transmitter)	256
17.1. Introduction	256
17.2. Features	256
17.3. Implementation	257
17.3.1. Chip Implementation	257
17.3.2. Modules' Functions	258
17.4. Control Block	259
17.4.1. Advanced UART Control Block	259
17.4.2. Basic UART Control Block	260
17.5. IO Lines	260
17.5.1. IO Signals	260
17.5.2. IO Configure	261
17.6. Enabling and Clock	261
17.6.1. UART Global Enable	261
17.6.2. UART Clock Control	261
17.6.3. UART PSC Timeout Signal Output	263
17.7. Interrupt and Event	264
17.7.1. UART Interrupt Control and Status	264
17.7.2. UART Subrange Interrupt	265
17.7.3. UART Interrupt Flags	266
17.8. UART Module IO Control	268
17.8.1. UART IO Control	268
17.8.2. UART IO Mode	269
17.8.3. UART Loop Back Mode	269
17.9. UART Connection for Application	270
17.9.1. UART Connect for UART/IrDA Mode	270
17.9.2. UART Connect for LIN Mode	270
17.9.3. UART Connect for SmartCard Mode	271
17.9.4. UART Connect for SPI Mode	271

17.9.5. UART Connect for SYNC Mode	272
17.10. UART Character Format	273
17.10.1. UART Data Character Format Setting.....	273
17.10.2. UART Parity Bit.....	274
17.10.3. UART Idle-Line Format Setting	274
17.10.4. UART Break Condition Format Setting	274
17.11. UART Fundamental Control	275
17.11.1. UART Control Mode Setting.....	275
17.11.2. UART Operation Mode Setting	275
17.11.3. UART Transmit	276
17.11.4. UART Receive	277
17.12. UART Data Buffer	278
17.12.1. UART Data Buffer Control.....	279
17.12.2. UART Data Inverse	279
17.12.3. UART Shift Buffer.....	279
17.12.4. UART Data Buffer Clear.....	279
17.13. UART Multi-Processor	280
17.13.1. UART Multi-Processor Operation Mode.....	280
17.13.2. UART Multi-Processor and Mute mode.....	280
17.13.3. UART Multi-Processor Address Setting	280
17.13.4. UART Multi-Processor Slave Address Transmit	281
17.14. UART Break Condition Transmit	282
17.15. UART Baud-Rate Control.....	283
17.15.1. UART Baud-Rate Timer On/Off Control	283
17.15.2. UART Baud-Rate Calibration	283
17.15.3. UART BR Timeout Signal Output.....	285
17.15.4. UART Baud-Rate Setting Examples	285
17.16. UART Data Receive and Sampling	286
17.16.1. UART Data Sampling.....	286
17.16.2. UART Receive Noise Character	286
17.16.3. UART Receive Oversampling	287
17.16.4. UART Receive Bit Time Error Tolerance	287
17.17. UART TMO Timeout Control	289
17.17.1. UART TMO Timeout Timer On/Off Control	290
17.17.2. UART Guard Time and Timeout Detection	290
17.17.3. UART Auto Calibration Timeout.....	291
17.17.4. UART Break Condition Timeout.....	292
17.17.5. UART TMO Timeout Signal Output.....	292
17.18. UART Error Management	293
17.18.1. UART Break and Parity/Frame Error Detection	293
17.18.2. UART TX Error Detect and Resend Control.....	294
17.18.3. UART RX Parity Error Detect and Retry Control (for SmartCard)	295
17.19. UART Mute Mode Control	296
17.20. UART Synchronous Mode	297
17.20.1. UART Synchronous Mode Configure	297
17.20.2. UART Synchronous Mode Timing.....	297
17.20.3. UART Synchronous Mode NSS Control.....	298
17.21. UART SmartCard Clock Output	300
17.21.1. SmartCard Clock Frequency Calculation	300

17.21.2. UART SmartCard Clock Output Setting Examples.....	301
17.22. UART IrDA Control.....	302
17.22.1. UART IrDA Timing	302
17.22.2. UART IrDA Data Sampling.....	303
17.22.3. UART IrDA Busy Flag	303
17.23. UART DE Control.....	304
17.24. UART Hardware Flow Control.....	305
17.24.1. UART Connection for Hardware Flow Control	305
17.24.2. UART CTS Hardware Flow Control.....	305
17.24.3. UART RTS Hardware Flow Control.....	306
17.24.4. UART Software Flow Control	307
17.25. UART Receive Hardware Hold and Capture Control	307
17.26. UART DMA Operation.....	308
17.26.1. DMA Module Configure.....	308
17.26.2. UART DMA Control.....	308
17.26.3. UART Interrupt Flag Control during DMA.....	309
18. SPI (Serial Peripheral Interface).....	310
18.1. Introduction	310
18.2. Features.....	310
18.3. Implementation.....	311
18.3.1. Chip Implementation	311
18.3.2. Module's Functions	312
18.4. Control Block.....	313
18.5. IO Lines.....	313
18.5.1. IO Signals	313
18.5.2. IO Configure.....	314
18.6. Enabling and Clock	314
18.6.1. SPI Global Enable.....	314
18.6.2. SPI Clock Control.....	314
18.7. Interrupt and Event.....	315
18.7.1. SPI Interrupt Control and Status	315
18.7.2. SPI Interrupt Flags	315
18.8. SPI Module IO Control	317
18.8.1. SPI IO Control.....	317
18.8.2. SPI IO Mode	318
18.8.3. SPI Loop Back Mode	318
18.9. SPI Connection for Application.....	319
18.9.1. Full Duplex Communication	319
18.9.2. Simplex Communication	319
18.9.3. Half Duplex Communication.....	320
18.9.4. Multi-Slave Communication	321
18.9.5. Two Master Communication	322
18.10. SPI Fundamental Control.....	323
18.10.1. SPI Clock	323
18.10.2. SPI Transmit	324
18.10.3. SPI Receive	324
18.10.4. SPI Fundamental Timing.....	325
18.10.5. SPI High Speed Operation.....	326
18.10.6. SPI DTR Mode	328

18.10.7. SPI NSS Mode	330
18.11. SPI Data Buffer	332
18.11.1. SPI Data Buffer Control.....	332
18.11.2. SPI Data Frame	333
18.11.3. SPI Transmitted Data Frame Size	333
18.11.4. SPI Received Data Frame Size	334
18.11.5. SPI Data Buffer Clear.....	335
18.12. SPI Data Modes.....	336
18.12.1. SPI Data Mode – SPI.....	336
18.12.2. SPI Data Mode – SPI-1.....	337
18.12.3. SPI Data Mode – SPI-2.....	337
18.12.4. SPI Data Mode – SPI-2C	337
18.12.5. SPI Data Mode – SPI-4.....	338
18.12.6. SPI Data Mode – SPI-4C	339
18.12.7. SPI Data Mode – SPI-4D	339
18.12.8. SPI Data Mode – SPI-8.....	340
18.13. SPI Supported Serial Flash.....	341
18.14. SPI Receive Overrun	342
18.15. SPI Transmit Error	342
18.16. SPI Master Mode Change Detect	343
18.17. SPI DMA Operation.....	344
18.17.1. DMA Module Configure.....	344
18.17.2. SPI DMA Control.....	344
18.17.3. SPI Interrupt Flag Control during DMA.....	344
19. USB (Universal Serial Bus)	345
19.1. Introduction	345
19.2. Features.....	345
19.3. Implementation.....	346
19.3.1. Chip Implementation	346
19.3.2. Module's Functions	346
19.4. Control Block.....	347
19.5. IO Lines.....	348
19.5.1. IO Signals	348
19.5.2. Bus State	348
19.6. Power and Clock	349
19.6.1. USB Global Enable	349
19.6.2. USB Power Control.....	349
19.6.3. USB Clock Control	349
19.6.4. USB Reset Control.....	350
19.7. Interrupt and Event.....	351
19.7.1. USB Interrupt Control and Status.....	351
19.7.2. USB Subrange Interrupt – Bus and Error.....	352
19.7.3. USB Subrange Interrupt –Endpoint.....	354
19.8. USB Connection for Application	356
19.9. USB Fundamental Control	356
19.9.1. USB Analog Front End.....	356
19.9.2. USB Endpoint Configuration	357
19.9.3. USB Packet Buffer	357
19.9.4. USB Buffer Mode	357

19.10. USB Operation	359
19.10.1. USB Transfer Construction	359
19.10.2. USB Transfer Type	360
19.10.3. USB Transaction Packet	361
19.10.4. USB Bus Reset and Endpoint Reset	363
19.10.5. USB Suspend and Resume	363
19.10.6. USB Remote Wakeup	363
19.10.7. USB Link Power Management	364
19.11. USB DMA Operation	366
19.11.1. DMA Module Configure	366
19.11.2. USB DMA Control	366
19.11.3. USB Interrupt Flag Control during DMA	366
19.12. USB Application Circuit	367
20. CAN (Controller Area Network)	368
20.1. Introduction	368
20.2. Features	368
20.3. Implementation	368
20.3.1. Chip Implementation	368
20.4. Control Block	369
20.5. IO Lines	369
20.5.1. IO Signals	369
20.5.2. IO Configure	369
20.6. Enabling and Clock	370
20.6.1. CAN Global Enable	370
20.6.2. CAN Clock Control	370
20.7. Interrupt and Event	371
20.7.1. CAN Interrupt Control and Status	371
20.7.2. CAN Interrupt Flags	371
20.8. CAN Module IO Control	374
20.8.1. CAN IO Control	374
20.8.2. CAN IO Mode	374
20.9. CAN Connection for Application	374
20.10. CAN Fundamental Control	375
20.10.1. Operation Mode	375
20.10.2. CAN Clock	376
20.10.3. CAN Bit Timing	377
20.10.4. CAN Bit Synchronization	378
20.10.5. CAN Error Detect	379
20.10.6. CAN Error Mode	380
20.11. CAN Message Frames	382
20.11.1. CAN Data Frame	382
20.11.2. CAN Remote Frame	383
20.11.3. CAN Error Frame	383
20.11.4. CAN Overload Frame	384
20.12. CAN Message Transfer	385
20.12.1. CAN Message Buffer	385
20.12.2. CAN Transmit	386
20.12.3. CAN Acceptance Filter	386
20.12.4. CAN Receive	387

20.13. CAN Test Mode	389
20.13.1. CAN Test Mode – Loop Back Mode.....	389
20.13.2. CAN Test Mode – Silent Mode.....	389
20.13.3. CAN Test Mode – Loop Back Combined with Silent Mode	389
20.13.4. CAN Self Reception Request Mode	390
21. Timer (General Timer)	391
21.1. Introduction	391
21.2. Features.....	391
21.3. Implementation.....	392
21.3.1. Chip Implementation	392
21.3.2. Modules' Functions	393
21.4. Control Block.....	394
21.5. IO Lines.....	397
21.5.1. IO Signals	397
21.5.2. IO Configure.....	397
21.6. Enabling and Clock	398
21.6.1. Timer Enable.....	398
21.6.2. Timer Process Clock Control	398
21.6.3. Timer Module Clock and Trigger Source.....	398
21.7. Interrupt and Event.....	400
21.7.1. Timer Update Event	400
21.7.2. Timer Interrupt Control and Status.....	400
21.7.3. Timer Interrupt Flags.....	402
21.8. Timer Operation	403
21.8.1. Timer Operation Mode	403
21.8.2. Timer and Counter Control.....	404
21.8.3. Timer Operation Mode and Control Validation	404
21.8.4. Repetition Counter	405
21.9. Timer Trigger Control Block.....	406
21.9.1. Timer Trigger Input Events.....	406
21.9.2. Timer Trigger Output Events.....	407
21.10. Timer Input/Output Channels	408
21.10.1. TM3x Timer Input/Output Channels Block	408
21.10.2. TM2x Timer Input/Output Channels Block	409
21.10.3. Timer Input Channel Control	409
21.10.4. Timer Clock Output	410
21.11. Timer Capture and Compare Block.....	412
21.11.1. Timer Channel Mode.....	412
21.11.2. Software Input Capture and Output Compare Generation	412
21.11.3. Timer Capture and Compare Register Control.....	412
21.12. Timer Input Capture	413
21.12.1. Capture Data and Edge Selection.....	413
21.12.2. Capture Data Overrun.....	413
21.12.3. Capture Control and Status.....	414
21.12.4. Duty Capture Control	417
21.13. Timer Output Compare and PWM.....	419
21.13.1. Compare Reload Register.....	419
21.13.2. Compare Output State	419
21.13.3. 16-bit Compare/PWM Mode.....	419

21.13.4. Two 8-bit Compare/PWM Mode	420
21.13.5. Output Compare/PWM Control and Status	420
21.13.6. Timer Output Compare and PWM Timing	421
21.13.7. PWM Dead-Time Control	425
21.14. Timer Output Control Block	426
21.14.1. TM2x Timer Output Control Block	426
21.14.2. TM3x Timer Output Control Block	427
21.14.3. Timer Output Enable Preload Control	428
21.15. Timer Break Control Block	429
21.15.1. Break Enable and Event Sources	429
21.15.2. Break Control Mode	429
21.16. QEI Control Block	431
21.16.1. QEI Control Mode	431
21.16.2. QEI Input Signals	432
21.16.3. QEI Control Mode 1, 2	432
21.16.4. QEI Control Mode 3, 4, 5	433
21.17. Timer DMA Operation	434
21.17.1. DMA Module Configure	434
21.17.2. Timer DMA Control	434
21.17.3. Timer Interrupt Flag Control during DMA	435
22. IWDT (Independent WatchDog Timer)	436
22.1. Introduction	436
22.2. Features	436
22.3. Control Block	436
22.4. Enabling and Clock	437
22.4.1. IWDT Global Enable	437
22.4.2. IWDT Clock Control	437
22.5. Interrupt and Event	437
22.5.1. IWDT Interrupt Control and Status	437
22.5.2. IWDT Interrupt Flags	437
22.6. Register Protect and Lock	438
22.7. IWDT Function Control	438
22.7.1. IWDT Timer Control	438
22.7.2. Operation in STOP	439
22.7.3. Wakeup from STOP	439
22.7.4. IWDT Event Timing	439
23. WWDT (Window WatchDog Timer)	440
23.1. Introduction	440
23.2. Features	440
23.3. Control Block	440
23.4. Enabling and Clock	441
23.4.1. WWDT Global Enable	441
23.4.2. WWDT Clock Control	441
23.5. Interrupt and Event	441
23.5.1. WWDT Interrupt Control and Status	441
23.5.2. WWDT Interrupt Flags	441
23.6. WWDT Register Protect	442
23.7. WWDT Function Control	442
23.7.1. WWDT Timer Control	442

23.7.2. WWDT Module Reset Control	442
23.7.3. WWDT Window Timing	443
24. RTC (Real Time Clock)	444
24.1. Introduction	444
24.2. Features	444
24.3. Control Block	444
24.4. IO Lines	445
24.4.1. IO Signals	445
24.4.2. IO Configure	445
24.5. Enabling and Clock	445
24.5.1. RTC Global Enable	445
24.5.2. RTC Clock Control	445
24.6. Interrupt and Event	446
24.6.1. RTC Interrupt Control and Status	446
24.6.2. RTC Interrupt Flags	446
24.6.3. RTC Clock Status	446
24.7. Register Protect and Lock	446
24.8. RTC Function Control	447
24.8.1. RTC Alarm	447
24.8.2. RTC Time Stamp	447
24.8.3. RTC Timer Capture and Reload	447
24.8.4. RTC Output	447
24.8.5. Operation in STOP	447
24.8.6. Wakeup from STOP	447
25. LCD (Liquid Crystal Display Controller)	448
25.1. Introduction	448
25.2. Features	448
25.3. Implementation	449
25.3.1. Chip Implementation	449
25.4. Control Block	450
25.5. IO Lines	451
25.5.1. IO Signals	451
25.5.2. IO Configure	451
25.6. Enabling and Clock	451
25.6.1. LCD Global Enable	451
25.6.2. LCD Clock Control	451
25.7. Interrupt and Event	452
25.7.1. LCD Interrupt Control and Status	452
25.7.2. LCD Interrupt Flags	452
25.8. LCD Fundamental Control	454
25.8.1. LCD Clock	454
25.8.2. LCD Analog Control	455
25.8.3. LCD Power Source and Pin Connections	456
25.8.4. LCD COM and SEG Control	458
25.9. LCD Drive Timing	460
25.9.1. LCD Clock and Frame	460
25.9.2. LCD Drive 1/2 Bias	462
25.9.3. LCD Drive 1/3 Bias	464
25.9.4. LCD Drive 1/4 Bias	466

25.9.5. LCD Drive Static	470
25.9.6. LCD Dead Timing	472
25.9.7. LCD Blinking and Segment Off	473
25.9.8. LCD Drive Timing Phase Inverse	474
25.10. LCD Data Control.....	476
25.10.1. LCD Display Data	476
25.10.2. LCD SOF and UDCF Interrupt	477
26. OPA (Operational Amplifier)	478
26.1. Introduction	478
26.2. Features.....	478
26.3. Implementation.....	478
26.3.1. Chip Implementation	478
26.4. Control Block.....	479
26.5. IO Lines.....	480
26.5.1. IO Signals	480
26.5.2. IO Configure.....	480
26.6. Power and Clock	480
26.6.1. OPA Power Control.....	480
26.6.2. OPA Clock Control.....	480
26.7. OPA Operational Amplifier	481
26.7.1. Input Channels	481
26.7.2. Input Offset Trimming.....	481
26.7.3. Operational Amplifier Output.....	481
27. ADC (Analog-to-Digital Converter)	482
27.1. Introduction	482
27.2. Features.....	482
27.3. Implementation.....	483
27.3.1. Chip Implementation	483
27.3.2. Modules' Functions	484
27.4. Control Block.....	485
27.5. IO Lines.....	486
27.5.1. IO Signals	486
27.5.2. IO Configure.....	486
27.6. Power and Clock	486
27.6.1. ADC Power Control.....	486
27.6.2. ADC Clock Control.....	486
27.7. Interrupt and Event.....	488
27.7.1. ADC Interrupt and Reset Event.....	488
27.7.2. ADC Interrupt Flags	488
27.8. ADC Operation.....	490
27.8.1. Single-End Mode	490
27.8.2. ADC Input Channels	490
27.8.3. Input Dynamic Voltage Range and Code Range	492
27.8.4. ADC Conversion	493
27.8.5. ADC PGA Control	494
27.8.6. ADC Calibration	495
27.8.7. ADC Operation in SLEEP	495
27.9. ADC Conversion Sequence	496
27.9.1. ADC Conversion Configuration and Sequence	496

27.9.2. ADC Conversion Hold	497
27.9.3. ADC Conversion Overrun	497
27.10. ADC Conversion Mode	497
27.10.1. One Shot and Continuous Conversion	497
27.10.2. Channel Scan Conversion	499
27.10.3. Scan Loop Conversion	500
27.10.4. Input Channel Change	500
27.10.5. ADC Conversion Flow	501
27.11. ADC Output Control	502
27.11.1. Digital Offset Adjuster	502
27.11.2. Signed Code Converter	503
27.11.3. Resolution and Data Alignment	504
27.11.4. ADC Data Register	504
27.11.5. Voltage Window Detect	505
27.11.6. Output Code Limit	506
27.12. ADC Data Sum	507
27.12.1. ADC Data Sum Accumulate	507
27.12.2. ADC Data Sum Register	508
27.12.3. ADC Data Sum Overrun	508
27.12.4. ADC Conversion Timing of Data Sum	509
27.13. ADC Wait and Auto-Off	511
27.14. Temperature Sensor	513
27.15. ADC DMA Operation	514
27.15.1. DMA Module Configure	514
27.15.2. ADC DMA Control	514
27.15.3. ADC Interrupt Flag Control during DMA	514
27.16. ADC Application Circuit	516
28. CMP (Analog Comparator)	518
28.1. Introduction	518
28.2. Features	518
28.3. Implementation	519
28.3.1. Chip Implementation	519
28.3.2. Modules' Functions	519
28.4. Control Block	520
28.5. IO Lines	522
28.5.1. IO Signals	522
28.5.2. IO Configure	522
28.6. Power and Clock	522
28.6.1. CMP Power Control	522
28.6.2. CMP Clock Control	522
28.7. Interrupt and Event	523
28.7.1. CMP Interrupt Control	523
28.7.2. CMP Interrupt Flags	523
28.8. CMP Analog Comparator	524
28.8.1. Input Channels	524
28.8.2. Internal Voltage Reference and Source	524
28.8.3. Analog Comparator-0	525
28.8.4. Analog Comparator-1	525
28.8.5. Response Time Select	526

28.8.6. Comparator Output	526
28.9. CMP Internal Voltage Reference	526
28.9.1. IVREF and IVREF2	526
28.9.2. IVREF Output Voltage	527
28.10. CMP Input Hysteresis Voltage	528
28.11. CMP Input Networks	529
29. DAC (Digital-to-Analog Converter)	531
29.1. Introduction	531
29.2. Features	531
29.3. Implementation	531
29.3.1. Chip Implementation	531
29.4. Control Block	532
29.4.1. Voltage DAC Control Block	532
29.5. IO Lines	533
29.5.1. IO Signals	533
29.5.2. IO Configure	533
29.6. Power and Clock	533
29.7. Interrupt and Event	534
29.7.1. DAC Interrupt Control	534
29.7.2. DAC Interrupt Flags	534
29.8. DAC Output Voltage	535
29.8.1. Voltage DAC Output	535
29.9. DAC Conversion	536
29.9.1. DAC Data Register Written	536
29.9.2. DAC Event Trigger	538
29.9.3. DAC Data Resolution and Alignment	538
29.10. DAC DMA Operation	539
29.10.1. DMA Module Configure	539
29.10.2. DAC DMA Control	539
29.10.3. DAC Interrupt Flag Control during DMA	539
29.11. DAC Application Circuit	540
29.11.1. Voltage DAC Application	540
30. Appended Information	541
30.1. Word Glossary	541
30.2. Modules Interconnection	542
31. Revision History	543

List of figures

Figure 1-1. Block Diagram Glossary	34
Figure 1-2. Power Operation Mode Indicator	34
Figure 2-1. Chip Main Block – MG32F02A128/U128/A064/U064	38
Figure 2-2. Chip Main Block – MG32F02V032	39
Figure 2-3. Chip Main Block – MG32F02N128/K128/N064/K064	40
Figure 2-4. ARM Cortex-M0 Processor	41
Figure 2-5. Memory Organization Map	42
Figure 2-6. CPU Debugging Connection Block Diagram	49
Figure 2-7. SWD and ICP Interface Circuit	50
Figure 3-1. Power Mode State	52
Figure 3-2. Power Supply Diagram	53
Figure 3-3. Power Controller	54
Figure 3-4. Power Voltage Detect	55
Figure 3-5. Power Controller Interrupt and Reset Event	56
Figure 3-6. Internal Power LDO Control	58
Figure 3-7. Device Power Enable Control in Power-Down Modes	60
Figure 3-8. System Power Operation Flow	61
Figure 3-9. STOP Wakeup Time	63
Figure 3-10. Wakeup and Interrupt Control - MG32F02A128/U128/A064/U064	64
Figure 3-11. Wakeup and Interrupt Control - MG32F02V032	65
Figure 3-12. Wakeup and Interrupt Control - MG32F02N128/K128/N064/K064	66
Figure 3-13. CPU Power Down and Wakeup Control Flow	67
Figure 3-14. WFE Control Flow Example	68
Figure 3-15. Power Supply Circuit	69
Figure 4-1. Reset Source Control Main Block	71
Figure 4-2. Reset Source Control Main Block	72
Figure 4-3. Reset Source Control Main Block	73
Figure 4-4. Chip Reset Timing	75
Figure 4-5. External Reset Timing	76
Figure 4-6. Module Software Reset	80
Figure 4-7. GPIO Reset Control	81
Figure 4-8. Reset Circuit	82
Figure 5-1. Clock Source Control Block	84
Figure 5-2. Clock Source Controller Interrupt	85
Figure 5-3. PLL Control Block	88
Figure 5-4. Module Process Clock Select	90
Figure 5-5. Module Process Clock Control – Unsupported Clock Running in STOP	91
Figure 5-6. Module Process Clock Control – Supported Clock Running in STOP	92
Figure 5-7. XTAL Oscillating Circuit	95
Figure 6-1. System Interrupt Control	97
Figure 7-1. Memory Control Block	99
Figure 7-2. Memory Interrupt/Reset Event Control	100
Figure 7-3. MCU Boot Modes	101
Figure 7-4. Flash Memory Address Mapping	102
Figure 7-5. Option Byte Memory Mapping and Register Load	106
Figure 7-6. Hardware Option Byte Load Sequence	106
Figure 7-7. ICP/ISP/IAP Access Memory Restriction	107
Figure 7-8. Memory Access Error Management	109

Figure 7-9. Flash Memory Size and Configuration for Different Product.....	109
Figure 9-1. GPIO Control Block	114
Figure 9-2. IO Mode Control Block.....	115
Figure 9-3. IO Control Structure Diagram	117
Figure 9-4. IO Pad Structure – Analog IO	118
Figure 9-5. IO Pad Structure – Digital Input	118
Figure 9-6. IO Pad Structure – Push-Pull.....	119
Figure 9-7. IO Pad Structure – Open Drain.....	119
Figure 9-8. IO Pad Structure – Quasi-Bidirectional	120
Figure 9-9. IO Port Access Block	121
Figure 9-10. Pin Alternate Function Select Block.....	122
Figure 9-11. Alternate Function Select of Special Pins.....	123
Figure 9-12. GPIO Input Application Circuit	125
Figure 9-13. GPIO Output Application Circuit	125
Figure 10-1. Interrupt Function Block.....	131
Figure 10-2. NVIC Exception Control Block	132
Figure 10-3. NVIC Interrupt Control	133
Figure 10-4. WIC Control Block	134
Figure 10-5. External Interrupt Controller.....	135
Figure 10-6. GPIO STOP Mode Wakeup Block	136
Figure 10-7. External Port Interrupt Block.....	138
Figure 11-1. General Purpose logic	140
Figure 11-2. GPL Process Clock Control	141
Figure 11-3. Byte Order Big/Little Endian Change Diagram.....	142
Figure 11-4. Bit Order Reverse Change Diagram	143
Figure 11-5. CRC Control Block.....	144
Figure 11-6. CRC Polynomials.....	144
Figure 11-7. Hardware Divider	145
Figure 11-8. GPL DMA Control Block.....	146
Figure 12-1. DMA Control Block	148
Figure 12-2. Module Process Clock Control – DMA.....	149
Figure 12-3. DMA Interrupt Control	150
Figure 12-4. DMA Source and Destination Control	151
Figure 12-5. DMA Channel Select Priority	154
Figure 12-6. DMA SRAM Using Suggestion	156
Figure 12-7. Peripheral DMA RX Request and Acknowledge Control Timing	158
Figure 12-8. Peripheral DMA TX Request and Acknowledge Control Timing.....	159
Figure 12-9. DMA External Request Trigger Input Timing.....	163
Figure 13-1. External Memory Bus Controller.....	165
Figure 13-2. EMB Process Clock Control	166
Figure 13-3. EMB Interrupt Control	167
Figure 13-4. EMB Module IO Control - MCLK, MWE, MOE.....	168
Figure 13-5. EMB Module IO Control – MCE, MALE, MALE2, MBW0, MBW1, MAM1.....	169
Figure 13-6. EMB Clock and Control Signals Polarity.....	170
Figure 13-7. EMB Module IO Control – MA, MAD.....	171
Figure 13-8. EMB Memory Map	172
Figure 13-9. EMB AHB to External Memory Transactions	173
Figure 13-10. EMB Access Control Timing State	175
Figure 13-11. EMB Address/Data Interface – 16bit MA + 16bit MD	178

Figure 13-12. EMB Address/Data Timing – 16bit MA + 16bit MD	178
Figure 13-13. EMB Address/Data Interface – 16bit MA + 16bit MAD	179
Figure 13-14. EMB Address/Data Timing – 16bit MA + 16bit MAD	179
Figure 13-15. EMB Address/Data Interface – 16bit MAD with 2 Address Phase	180
Figure 13-16. EMB Address/Data Timing – 16bit MAD with 2 Address Phase	180
Figure 13-17. EMB Address/Data Interface – 16bit MA + 8bit MD	181
Figure 13-18. EMB Address/Data Timing – 16bit MA + 8bit MD	181
Figure 13-19. EMB Address/Data Interface – 16bit MA + 8bit MAD	182
Figure 13-20. EMB Address/Data Timing – 16bit MA + 8bit MAD	182
Figure 13-21. EMB Address/Data Interface – 8bit MAD with 2 Address Phase	183
Figure 13-22. EMB Address/Data Timing – 8bit MAD with 2 Address Phase	183
Figure 13-23. EMB SRAM Interface	184
Figure 13-24. EMB SRAM 16-bit Data Timing	185
Figure 13-25. EMB SRAM 8-bit Data Write Timing	186
Figure 13-26. EMB NOR-Flash Interface	187
Figure 13-27. EMB NOR-Flash 16-bit Data Timing	188
Figure 13-28. EMB NOR-Flash 8-bit Data Timing	189
Figure 13-29. EMB NAND-Flash Interface	190
Figure 13-30. EMB NAND-Flash 8/16-bit Data Timing	191
Figure 13-31. EMB LCD Interface	192
Figure 13-32. EMB LCD Register 16-bit Bus Access Timing	192
Figure 13-33. EMB LCD Register 8-bit Bus Access Timing	193
Figure 13-34. EMB LCD GRAM 8/16-bit Bus Access Timing	194
Figure 13-35. EMB and SPI Flash to 8-bit LCD Connection	197
Figure 14-1. APB Interrupt Control	199
Figure 14-2. Timer Synchronous Enable Control	200
Figure 14-3. Timer Common Trigger/Clock Source Selection	201
Figure 14-4. Output Signal Break and Modulation Control	202
Figure 14-5. IR Control Interface	205
Figure 14-6. NCO Block	207
Figure 14-7. NCO Output	208
Figure 14-8. MF Signal MUX	209
Figure 15-1. APX Interrupt Control	211
Figure 15-2. CCL Block	212
Figure 15-3. ASB Block	215
Figure 15-4. ASB Connection - ARGB Mode	216
Figure 15-5. ASB Connection – SHIFT Mode	216
Figure 15-6. ASB Output Mode	217
Figure 15-7. ASB Output Waveform	218
Figure 15-8. ASB RESET code Generation Timing	219
Figure 15-9. ASB Channel Synchronous Mode Waveform Example	220
Figure 15-10. SDT Block	221
Figure 15-11. SDT Block	221
Figure 16-1. I2C Main Control Block – I2C0/1	225
Figure 16-2. I2C Process Clock Control	226
Figure 16-3. I2C Status and Interrupt Control – I2C0/1	227
Figure 16-4. I2C Subrange Interrupt Control – I2C0/1	228
Figure 16-5. I2C Connection for Single Master Communication	230
Figure 16-6. I2C Connection for Multi-Master or Slave Mode Communication	230

Figure 16-7. I2C Fundamental Protocol	231
Figure 16-8. I2C START/STOP/Data-Change Timing	232
Figure 16-9. Flow Chart Glossary	234
Figure 16-10. I2C Master Transmitter Mode Flow Chart	235
Figure 16-11. I2C Master Receiver Mode Flow Chart	237
Figure 16-12. I2C Slave Transmitter Mode Flow Chart	239
Figure 16-13. I2C Slave Receiver Mode Flow Chart	241
Figure 16-14. I2C Slave Receiver General Call Mode Flow Chart	242
Figure 16-15. I2C Data Buffer Mode Control Block – I2C0/I2C1	243
Figure 16-16. I2C Access Command Bits	244
Figure 16-17. I2C Master Transmit Access Sequence	245
Figure 16-18. I2C Master Receive Access Sequence	246
Figure 16-19. I2C Slave Transmit Access Sequence	247
Figure 16-20. I2C Slave Receive Access Sequence	248
Figure 16-21. I2C Signal Drive Control Timing	249
Figure 16-22. I2C Slave SCL Stretching	250
Figure 16-23. I2C TMO Timeout Timer Control	251
Figure 16-24. I2C Bus Error Event	253
Figure 16-25. I2C Master Transmission NACK Detect	253
Figure 16-26. I2C Slave Data Overrun under SCL Stretching Disable	254
Figure 17-1. Advanced UART Main Control Block – URT0/1/2	259
Figure 17-2. Basic UART Main Control Block – URT4/5/6/7	260
Figure 17-3. UART Process Clock Control	261
Figure 17-4. Advanced UART Clock Control	262
Figure 17-5. Basic UART Clock Control – URT4/5/6/7	263
Figure 17-6. UART Status and Interrupt Control – URT0/1/2	264
Figure 17-7. UART Status and Interrupt Control – URT4/5/6/7	265
Figure 17-8. UART Subrange Interrupt Control – URT0/1/2	265
Figure 17-9. UART Subrange Interrupt Control – URT4/5/6/7	266
Figure 17-10. UART RX/TX IO Control	268
Figure 17-11. UART Other IO Control	269
Figure 17-12. UART Connect for UART/IrDA Mode	270
Figure 17-13. UART Connect for LIN Mode	270
Figure 17-14. UART Connect for SmartCard Mode	271
Figure 17-15. UART Connect for SPI Master Mode	271
Figure 17-16. UART Connect for SYNC Mode	272
Figure 17-17. UART Character Format	273
Figure 17-18. UART Idle-Line and Break Condition Format	274
Figure 17-19. UART Data Transmit and Receive	277
Figure 17-20. UART Data Mode Control – URT0/1/2	278
Figure 17-21. UART Data Buffer Mode Control – URT4/5/6/7	279
Figure 17-22. UART Multi-Processor Operation Mode	280
Figure 17-23. UART Multi-Processor Address Setting Example	281
Figure 17-24. UART Multi-Processor Slave Address Transmit	281
Figure 17-25. UART Break Condition Transmit	282
Figure 17-26. UART Baud-Rate Timer Control Block	283
Figure 17-27. UART Auto Baud-Rate Control Mode Timing	284
Figure 17-28. UART Receive Data Sampling	286
Figure 17-29. UART Receive Oversampling Majority Points	287

Figure 17-30. UART Receive Bit Time Error Tolerance.....	288
Figure 17-31. UART Receive Bit Time Error Tolerance Examples	288
Figure 17-32. UART TMO Timeout Timer Control Block	289
Figure 17-33. UART Guard Time and Timeout Detection.....	290
Figure 17-34. UART Auto Calibration Timeout Error Detection	291
Figure 17-35. UART Break and Parity/Frame Error Detection	293
Figure 17-36. UART TX Error Detect and Resend Control	294
Figure 17-37. UART RX Parity Error Detect and Retry Control.....	295
Figure 17-38. UART Mute Mode Control Timing	296
Figure 17-39. UART Synchronous SYNC Mode Timing – 1 bidirectional data line	297
Figure 17-40. UART Synchronous SPI Mode Timing	298
Figure 17-41. UART Synchronous Mode Hardware NSS Timing	299
Figure 17-42. UART SmartCard Clock Output	300
Figure 17-43. UART IrDA Timing	302
Figure 17-44. UART Busy and IrDA Busy Status	303
Figure 17-45. UART Drive Enable Timing	304
Figure 17-46. UART Connection for Hardware Flow Control	305
Figure 17-47. UART CTS Hardware Flow Control Timing.....	305
Figure 17-48. UART RTS Hardware Flow Control and RX Overrun Timing.....	306
Figure 17-49. UART Software Flow Control.....	307
Figure 17-50. UART Receive Hardware and Capture Hold Event	308
Figure 18-1. SPI Main Control Block.....	313
Figure 18-2. SPI Process Clock Control	314
Figure 18-3. SPI Status and Interrupt.....	315
Figure 18-4. SPI Data IO Control.....	317
Figure 18-5. SPI Clock/NSS IO Control	318
Figure 18-6. SPI Full Duplex Communication	319
Figure 18-7. SPI Simplex Communication	319
Figure 18-8. SPI Half Duplex Communication	320
Figure 18-9. SPI Half Duplex Communication with 8 Data Lines	321
Figure 18-10. SPI Multi-Slave Communication	321
Figure 18-11. SPI Two Master Communication.....	322
Figure 18-12. SPI Master and Slave Swap Communication	322
Figure 18-13. SPI Master Clock Output and Input	323
Figure 18-14. SPI Fundamental Timing with NSS.....	325
Figure 18-15. SPI Fundamental Timing without NSS.....	326
Figure 18-16. SPI Master High Speed Connection Delay	326
Figure 18-17. SPI Master High Speed Timing.....	327
Figure 18-18. SPI Slave High Speed Timing.....	328
Figure 18-19. SPI Master DTR Mode Timing	329
Figure 18-20. SPI Master Mode Hardware NSS Timing.....	331
Figure 18-21. SPI Data Buffer Mode Control – SPI0.....	332
Figure 18-22. SPI Transmitted Data Frame Size Control.....	333
Figure 18-23. SPI Received Data Frame Size Control.....	335
Figure 18-24. SPI Data Mode - separated I/O (SPI)	336
Figure 18-25. SPI Data Mode - 1 Bidirectional data line	337
Figure 18-26. SPI Data Mode – 2 Bidirectional data lines with spread data.....	337
Figure 18-27. SPI Data Mode – 2 Bidirectional data lines with copied data.....	337
Figure 18-28. SPI Data Mode – 4 Bidirectional data lines with spread data.....	338

Figure 18-29. SPI Data Mode – 4 Bidirectional data lines with copied data	339
Figure 18-30. SPI Data Mode – 8 Bidirectional data lines with two duplicated 4 data lines	339
Figure 18-31. SPI Data Mode – 8 Bidirectional data lines with spread data.....	340
Figure 18-32. SPI Receive Overrun	342
Figure 18-33. SPI Slave Mode Transmit Error	342
Figure 18-34. SPI Master Mode Change Detect	343
Figure 19-1. USB Main Block.....	347
Figure 19-2. USB Process Clock Control.....	349
Figure 19-3. USB SYNC Control Block	350
Figure 19-4. USB Interrupt Control - Root.....	351
Figure 19-5. USB Global Event Status	352
Figure 19-6. USB Subrange Interrupt Control – Bus and Error.....	353
Figure 19-7. USB Subrange Interrupt Control – Endpoint 0.....	354
Figure 19-8. USB Subrange Interrupt Control – Endpoint 1~7.....	354
Figure 19-9. USB Bus Connection	356
Figure 19-10. USB Analog Front End.....	356
Figure 19-11. USB Packet Buffer Map Example	358
Figure 19-12. USB Transfer Construction (Full Speed).....	359
Figure 19-13. USB Transfer Type - Control.....	360
Figure 19-14. USB Transfer Type – Bulk and Interrupt	360
Figure 19-15. USB Transfer Type - Isochronous.....	361
Figure 19-16. USB Transaction Packet Structure and Flow	362
Figure 19-17. USB LPM State Transition Diagram.....	364
Figure 19-18. USB LPM Packets in an Extension Token Transaction	365
Figure 19-19. USB Application Circuit.....	367
Figure 19-20. USB PCB Layout Suggestion on 2-layer 1.6mm FR4 PCB	367
Figure 20-1. CAN Main Control Block	369
Figure 20-2. CAN Process Clock Control.....	370
Figure 20-3. CAN Clock Source.....	370
Figure 20-4. CAN Interrupt.....	371
Figure 20-5. CAN Status	372
Figure 20-6. CAN Module IO Control	374
Figure 20-7. CAN Connection	374
Figure 20-8. CAN Operation Mode	375
Figure 20-9. CAN Clock Control	376
Figure 20-10. CAN Bit Timing	377
Figure 20-11. CAN Bit Synchronization.....	378
Figure 20-12. CAN Error Mode	380
Figure 20-13. CAN Data Frame	382
Figure 20-14. CAN Remote Frame	383
Figure 20-15. CAN Error Frame.....	383
Figure 20-16. CAN Overload Frame	384
Figure 20-17. CAN Message Buffer Control	385
Figure 20-18. CAN Acceptance Filter.....	386
Figure 20-19. CAN Test Mode – Loop Back Mode.....	389
Figure 20-20. CAN Test Mode – Silent Mode.....	389
Figure 20-21. CAN Test Mode - Loop Back Combined with Silent Mode	389
Figure 20-22. CAN Self Reception Request Mode	390
Figure 21-1. Timer Main Block ~ TM0x/TM1x	394

Figure 21-2. Timer Main Block ~ TM2x	395
Figure 21-3. Timer Main Block ~ TM3x	396
Figure 21-4. Timer Process Clock Control	398
Figure 21-5. Timer Clock and Trigger Source	399
Figure 21-6. Timer Event Control and Timer Flags	400
Figure 21-7. Timer Status and Interrupt Control – TM0x/1x	401
Figure 21-8. Timer Status and Interrupt Control – TM2x/3x	401
Figure 21-9. Timer Operation Mode Control - Cascade Mode	403
Figure 21-10. Timer Operation Mode Control - Separate Mode	403
Figure 21-11. Timer Operation Mode Control - Full-Counter Mode	404
Figure 21-12. Timer Repetition Counter Control Block	405
Figure 21-13. Timer Trigger Input Control Block	406
Figure 21-14. Timer Clock and Trigger Control	406
Figure 21-15. Timer Trigger Output Control Block	407
Figure 21-16. Timer Input / Output Channels ~ TM3x	408
Figure 21-17. Timer Input / Output Channels ~ TM2x	409
Figure 21-18. Timer Clock Output Block	410
Figure 21-19. Timer CKO Clock Output Timing	411
Figure 21-20. Timer Input Capture and Output Compare Block	412
Figure 21-21. Timer Input Capture Block	413
Figure 21-22. Timer Input Capture Timing for Full-Counter Mode	415
Figure 21-23. Timer Input Capture Timing for Cascade and Separate Modes	416
Figure 21-24. Timer Duty Capture Timing for Full-Counter Mode	417
Figure 21-25. Timer Duty Capture Timing for Cascade and Separate Modes	418
Figure 21-26. Timer 16-bit Output Compare/PWM Block	419
Figure 21-27. Timer Two 8-bit Compare/PWM Output Block	420
Figure 21-28. Timer Output Compare Timing with Up Counting	421
Figure 21-29. Timer Edge Aligned PWM Timing with Up Counting	422
Figure 21-30. Timer Edge Aligned PWM Timing with Down Counting	423
Figure 21-31. Timer Center Aligned PWM Timing	424
Figure 21-32. Timer PWM Dead-time Control Timing	425
Figure 21-33. Timer Output Control Block ~ TM2x	426
Figure 21-34. Timer Output Control Block ~ TM3x	427
Figure 21-35. Timer OC Output Enable Preload Control	428
Figure 21-36. Timer Break Input Source	429
Figure 21-37. Timer Break Event Control Timing	430
Figure 21-38. Timer QEI Control Block	431
Figure 21-39. Timer QEI Control Mode 1, 2	432
Figure 21-40. Timer EXUD Control Mode 3, 4, 5	434
Figure 22-1. IWDTC Control Block	436
Figure 22-2. IWDTC Process Clock Control	437
Figure 22-3. IWDTC Event Timing	439
Figure 23-1. WWDTC System Window Watch Dog Timer	440
Figure 23-2. WWDTC Process Clock Control	441
Figure 23-3. WWDTC Window Timing	443
Figure 24-1. RTC Real Time Clock Control	444
Figure 24-2. RTC Process Clock Control	445
Figure 25-1. LCD Main Control Block	450
Figure 25-2. LCD Process Clock Control	451

Figure 25-3. LCD Clock Source	452
Figure 25-4. LCD Status and Interrupt	452
Figure 25-5. LCD Clock Control.....	454
Figure 25-6. LCD Analog Control.....	455
Figure 25-7. LCD Bias with Internal Charge Pump	456
Figure 25-8. LCD CPRF Interrupt and PRDYF Status	457
Figure 25-9. LCD Bias with Internal AVDD.....	457
Figure 25-10. LCD Bias with External Power	458
Figure 25-11. LCD COM/SEG Connection.....	458
Figure 25-12. LCD Drive Output Control.....	459
Figure 25-13. LCD Clock and Frame Timing.....	461
Figure 25-14. LCD Drive Timing –1/2Bias, 1/2Duty (Type-A).....	462
Figure 25-15. LCD Drive Timing –1/2Bias, 1/2Duty (Type-B).....	463
Figure 25-16. LCD Drive Timing –1/3Bias, 1/4Duty (Type-A).....	464
Figure 25-17. LCD Drive Timing –1/3Bias, 1/4Duty (Type-B).....	465
Figure 25-18. LCD Drive Timing –1/4Bias, 1/4Duty (Type-A).....	466
Figure 25-19. LCD Drive Timing –1/4Bias, 1/4Duty (Type-B).....	467
Figure 25-20. LCD Drive Timing –1/4Bias, 1/8Duty (Type-A).....	468
Figure 25-21. LCD Drive Timing –1/4Bias, 1/8Duty (Type-B).....	469
Figure 25-22. LCD Drive Timing – Static (Type-A)	470
Figure 25-23. LCD Drive Timing – Static (Type-B)	471
Figure 25-24. LCD Dead Timing – In Frame (Type-A)	472
Figure 25-25. LCD Dead Timing – In Frame (Type-B)	472
Figure 25-26. LCD Dead Timing – In Duty (Type-A)	473
Figure 25-27. LCD Blinking Timing	474
Figure 25-28. LCD Drive Timing Inverse (Type-A)	474
Figure 25-29. LCD Drive Timing Inverse (Type-B)	475
Figure 25-30. LCD Display Data Map	476
Figure 25-31. LCD SOF and UDCF Interrupt Timing	477
Figure 26-1. OPA Main Block.....	479
Figure 26-2. OPA Process Clock Control	480
Figure 27-1. ADC Block Diagram with Single-End Mode	485
Figure 27-2. ADC Process Clock Control.....	487
Figure 27-3. ADC Input Clock	487
Figure 27-4. ADC Interrupt Control	489
Figure 27-5. ADC and Analog Input Multiplexer	490
Figure 27-6. ADC Input from DAC Output.....	492
Figure 27-7. ADC Dynamic Voltage Range.....	492
Figure 27-8. ADC Conversion Sequence	493
Figure 27-9. ADC PGA Control	494
Figure 27-10. ADC Conversion Start Control	496
Figure 27-11. ADC Conversion Mode – One Channel	498
Figure 27-12. ADC Conversion Mode – Channel Scan.....	499
Figure 27-13. ADC Conversion Mode – Scan Loop	500
Figure 27-14. ADC Conversion Flow.....	501
Figure 27-15. ADC Output Control Block	502
Figure 27-16. ADC Voltage Window Detect	505
Figure 27-17. ADC Output Code Limit	506
Figure 27-18. ADC Data Sum Mode – One Shot or Channel Scan Discontinuous	509

Figure 27-19. ADC Data Sum Mode – Channel/Loop Scan	510
Figure 27-20. ADC Wait without Auto-Off Mode	511
Figure 27-21. ADC Wait with Auto-Off Mode Timing	512
Figure 27-22. ADC Input from TS Output.....	513
Figure 27-23. ADC Application Circuit with VREF+ Pin.....	516
Figure 27-24. ADC Application Circuit without VREF+ Pin.....	516
Figure 28-1. CMP Main Block - MG32F02A128/U128/A064/U064	520
Figure 28-2. CMP Main Block - MG32F02N128/K128/N064/K064	521
Figure 28-3. CMP Process Clock Control	522
Figure 28-4. CMP Interrupt Control.....	523
Figure 28-5. CMP Analog Comparator CMP0 Block	525
Figure 28-6. CMP Analog Comparator CMP1 Block	525
Figure 28-7. CMP Comparator IVREF Ladder Block	526
Figure 28-8. CMP IVREF Output External Control Block	528
Figure 28-9. CMP Comparator Hysteresis Voltage	528
Figure 28-10. CMP Input Networks Example.....	529
Figure 29-1. DAC Main Block – Voltage DAC	532
Figure 29-2. DAC Process Clock Control.....	533
Figure 29-3. DAC Interrupt Control	534
Figure 29-4. DAC Conversion – Output Update by Data Register Written	537
Figure 29-5. DAC Conversion – Output Update by Event Trigger.....	538
Figure 29-6. DAC Application Circuit – Voltage DAC	540

List of tables

Table 2-1. Chip Implementation Table	36
Table 2-2. CPU Memory Address Map	43
Table 2-3. Peripheral Memory Boundary Address – MG32F02A128/U128/A064/U064	44
Table 2-4. Peripheral Memory Boundary Address – MG32F02V032	45
Table 2-5. Peripheral Memory Boundary Address – MG32F02N128/K128/N064/K064	47
Table 3-1. Power Device Implementation	51
Table 3-2. Power Operation Mode	52
Table 3-3. Voltage Detect Threshold	56
Table 3-4. Sleep and DeepSleep Entering	58
Table 3-5. Power Mode Selection	58
Table 3-6. LDO Control	59
Table 3-7. Internal Device Power Control	59
Table 3-8. Device Power-On Control	60
Table 3-9. Wakeup Events Source	62
Table 3-10. WFI and WFE Action for Wakeup and ISR	63
Table 3-11. WFE Events for Setting Event Register	67
Table 4-1. Reset Control Function for Different Reset Level	74
Table 4-2. Reset Event Source	77
Table 4-3. Reset VS Lock Function for Modules' Registers	79
Table 5-1. Embedded PLL Implementation	83
Table 5-2. Module Running and Clock Enable Control	91
Table 5-3. Modules' Available Clock Enable Register	92
Table 5-4. Internal Device Clock Control	93
Table 5-5. Modules' Available Input Event Clock Sources Table	94
Table 5-6. Internal Total Equivalent Capacitance Example for XOSC circuit	95
Table 5-7. Reference Capacitance of C1 & C2 for crystal oscillating circuit	95
Table 6-1. System Interrupt Control Source	97
Table 7-1. Embedded Memory Implementation	98
Table 7-2. Memory Boot Mode Select	101
Table 7-3. Flash Memory Access Control and Key	104
Table 7-4. Memory Access Restriction vs Boot Mode	107
Table 7-5. CPU Hold Control under Flash Memory Access	108
Table 9-1. GPIO Implementation	114
Table 9-2. IO Operation Mode Control	116
Table 9-3. Port C IO Mode Default Setting	124
Table 10-1. Interrupt Sources	128
Table 10-2. CPU Lockup Exit Events	130
Table 10-3. Hard Fault Handling Events on Cortex-M0	134
Table 11-1. GPL Implementation	139
Table 12-1. DMA Implementation	147
Table 12-2. DMA Memory Source Support	152
Table 12-3. DMA Channel Source Request Selection	152
Table 12-4. DMA Channel Destination Request Selection	153
Table 12-5. DMA Channel Operation Type Support	153
Table 12-6. DMA Transfer Number/Start Address and Burst Size Setting Note	155
Table 12-7. Peripheral Module Interrupt Flag Control for DMA – MG32F02A128/U128/A064/U064	160
Table 12-8. Peripheral Module Interrupt Flag Control for DMA – MG32F02V032	161
Table 12-9. Peripheral Module Interrupt Flag Control for DMA –MG32F02N128/K128/N064/K064	162

Table 13-1. EMB Implementation	164
Table 13-2. EMB Interface and Device Pin Mapping	171
Table 13-3. EMB AHB to External Memory Supported Transactions	173
Table 13-4. EMB Address/Data Interface Mode Setting	176
Table 13-5. EMB Internal and External Signal Mapping	177
Table 13-6. EMB Interrupt Flag Control for DMA Function	195
Table 13-7. EMB Interface Signal and Suggestion Pin Table	195
Table 14-1. APB Implementation	198
Table 14-2. Timer Common ITR6/ITR7 Signals Table	200
Table 14-3. OBM Block Output Channel Signals Table	203
Table 14-4. OBM Block Break Channel Signals Table	204
Table 14-5. IR Clock/Data Signals Table	206
Table 14-6. Multi-Function Signals Table	209
Table 15-1. APX Implementation	210
Table 15-2. CCL Input Mux Signal Selection	213
Table 15-3. CCL Truth Table Setting Examples for Three Inputs	213
Table 15-4. CCL Sequential I/O Table	214
Table 15-5. APX Interrupt Flag Control for DMA Function	222
Table 16-1. I2C Implementation	224
Table 16-2. I2C Modules' Functions	224
Table 16-3. I2C Event Code Table	233
Table 16-4. I2C Master Transmitter Mode Event Table	234
Table 16-5. I2C Master Receiver Mode Event Table	236
Table 16-6. I2C Slave Transmitter Mode Event Table	238
Table 16-7. I2C Slave Receiver Mode Event Table	240
Table 16-8. I2C Miscellaneous Event Table	242
Table 16-9. I2C Signal Output Driving Setting	250
Table 16-10. I2C TMO Timeout Timer On/Off Control	252
Table 16-11. I2C Interrupt Flag Control for DMA Function	255
Table 17-1. UART Implementation	257
Table 17-2. UART Modules' Functions	258
Table 17-3. UART Pin Swap Function Mapping	268
Table 17-4. UART Data Character Format Setting	273
Table 17-5. UART Parity Bit Definitions	274
Table 17-6. UART Control Mode vs. Character Data Control Register	275
Table 17-7. UART Module vs. Character Data Control Register	275
Table 17-8. UART Operation Mode Setting	275
Table 17-9. UART TX Stop Bit Length Setting	276
Table 17-10. UART Break Condition Send and Detect Control	282
Table 17-11. UART BR Baud-Rate Timer On/Off Control	283
Table 17-12. UART General Using Baud-Rate Setting Examples	285
Table 17-13. UART Received Data Oversampling and Noise Detection	286
Table 17-14. UART TMO Timeout Timer On/Off Control	290
Table 17-15. UART Calibration Timeout Condition	291
Table 17-16. UART Calibration Timeout Time	292
Table 17-17. UART Multi-Processor Address Matched vs Mute Mode Control	296
Table 17-18. UART Synchronous Clock Mode Table	297
Table 17-19. UART Synchronous Mode NSS Timing Table	299
Table 17-20. UART SmartCard Clock Output Setting Examples	301

Table 17-21. UART IrDA Received Data Oversampling and Sampling Mode	303
Table 17-22. UART Data Buffer vs RHF Status when Multi-Processor Address Unmatched	308
Table 17-23. UART Interrupt Flag Control for DMA Function	309
Table 18-1. SPI Implementation	311
Table 18-2. SPI Modules' Functions	312
Table 18-3. SPI Clock Mode Table	323
Table 18-4. SPI NSS Control Table	330
Table 18-5. SPI Master NSS Timing Table	330
Table 18-6. SPI Data Control Table	336
Table 18-7. SPI Flash Type List	341
Table 18-8. SPI Flash Control Table	341
Table 18-9. SPI Interrupt Flag Control for DMA Function	344
Table 19-1. USB Endpoint Functions	346
Table 19-2. USB Bus State and Flag	348
Table 19-3. USB Buffer Mode Control	358
Table 19-4. USB Link Power Manager States	364
Table 19-5. USB Summary Similarities/Differences between L1 and L2	365
Table 19-6. USB Interrupt Flag Control for DMA Function	366
Table 20-1. CAN Implementation	368
Table 20-2. CAN Initial Mode Action	375
Table 20-3. CAN Error Code Capture Value Definitions	379
Table 20-4. CAN RX and TX Buffer Data Definitions	385
Table 20-5. CAN RX Acceptance Filter Definitions	387
Table 20-6. CAN RX FIFO Reset Action	388
Table 21-1. Timer Implementation	392
Table 21-2. Timer Modules' Functions	393
Table 21-3. Timer Operation Mode and Control Validation	404
Table 21-4. Timer Internal Trigger Signals Table	407
Table 21-5. Timer Channel Input Signals Table	410
Table 21-6. Timer Capture and Compare Register Control	413
Table 21-7. Timer Input Capture Mode	414
Table 21-8. Timer Output Compare/PWM Mode	420
Table 21-9. Timer Output Break or Stop Control	429
Table 21-10. Timer QEI External Input Up/Down Control Mode	431
Table 21-11. Timer QEI Encoder States	433
Table 21-12. Timer QEI Encoder State Transition	433
Table 21-13. Timer Interrupt Flag Control for DMA Function	435
Table 22-1. IWDG Register Write Protection Table	438
Table 25-1. LCD Implementation	449
Table 25-2. LCD Power Rail Pin Control	455
Table 25-3. LCD Power Source and R-Ladder Control Mode	456
Table 25-4. LCD Bias and Duty Configure Suggestion	460
Table 25-5. LCD Frame Rate and Clock Configure Examples	460
Table 25-6. LCD Clock Frequency Range Examples	461
Table 26-1. OPA Implementation	478
Table 27-1. ADC Implementation-1	483
Table 27-2. ADC Implementation-2	483
Table 27-3. ADC Module Functions	484
Table 27-4. ADC Channel Definitions	491

Table 27-5. ADC Gain Calculation - x1 ~ x4.....	494
Table 27-6. ADC Gain Calculation - x1 ~ x128.....	495
Table 27-7. ADC Conversion Mode Control	497
Table 27-8. ADC Digital Offset Process	502
Table 27-9. ADC Single-End Data Format Definitions.....	503
Table 27-10. ADC Differential Data Format Definitions	503
Table 27-11. ADC Data Alignment Definitions	504
Table 27-12. ADC Data Code Example.....	504
Table 27-13. ADC Data Sum and Conversion Mode Control	507
Table 27-14. ADC Data Sum Flags vs. Conversion Mode.....	507
Table 27-15. ADC SUM Value Range	508
Table 27-16. ADC Auto-Off Mode Start-Up Cycle vs. Conversion Mode	512
Table 27-17. ADC Interrupt Flag Control for DMA Function	514
Table 27-18. ADC DMA Mode vs. Conversion Mode.....	515
Table 28-1. CMP Implementation.....	519
Table 28-2. Analog Comparator Modules' Functions.....	519
Table 28-3. CMP Comparator IVREF R-Ladder Output	527
Table 29-1. DAC Implementation	531
Table 29-2. DAC Data Alignment Definitions.....	538
Table 29-3. DAC Interrupt Flag Control for DMA Function	539
Table 30-1. Word Glossary	541
Table 30-2. Modules Interconnection Table.....	542

1. Document Using

1.1. Documentation Conventions

The following signed text examples are used to indicate special signification in this document.

- PA[15:0]** : Pin name in dark-green foreground color
DAC_TRG0 : GPIO AFS IO name in orange foreground color
I2Cx_EN : Register name in dark-red foreground color
CK_HS : Internal signal in blue foreground color
[Notify] : The following descriptions of the "[Notify]" in underline and italic are used to notify user.

1.2. Block Diagram Glossary

Figure 1-1. Block Diagram Glossary

	Chip Pin and Pin Name		Voltage, Current Reference Source
	AFS IO and IO Name		Analog, Digital Multiplex
	Module IO and IO Name		On/Off Switch - default Enable
	Module External Signal		On/Off Switch - default Disable
	Module Internal Signal		Digital Comparator
	Register Name and Control Indicated Line		Clock Divider
	Rising, Falling Edge Signals		Signal Inverter/Polarity Control
	Pulse, Level Signal		Signal Toggler
	<Note-1> and description		Flag set (blue) and clear (red)
	See <Note-1>, <Note-2> below block diagram for more information		Branch Item
<Note> : General descriptions to notify for this diagram. <Note-1> : Descriptions to notify for *1 in the diagram. <Note-2> : Descriptions to notify for *2 in the diagram.			Step or Separated Item

1.3. Power Operation Mode Indicator

The following diagram is showing the power operation mode indicator of module.

Figure 1-2. Power Operation Mode Indicator

Power Operation Modes			Indicator
① ON	CPU is able running in full speed.		The module can be running in all power operation modes.
② SLEEP	CPU is stopped and all peripheral modules can be configurable to continue running.		The module can be running in ON and SLEEP modes only.
③ STOP	CPU and all peripheral modules are stopped except analog comparator, RTC and IWDG modules can be configurable to continue running.		The module can be running in ON and SLEEP modes but can wakeup from STOP mode.

2. Chip and System

2.1. Chip Overview

The **MG32F02** series is the single-chip 32-bit microcontroller based on a high performance Core ARM® 32-bit Cortex®-M0 CPU with embedded Nested Vectored Interrupt Controller (NVIC).

The **MG32F02** series has up to 128K bytes of embedded main flash memory for code and data, programmable memory size of embedded system flash memory for boot load code and 64 bytes of embedded option-byte flash memory for chip configuration. The all flash memory can be programmed either in serial writer mode (ICP, In-Circuit-Programming). Also, the main flash memory can be programmed in ISP (In-System Programming) mode or SRAM (Boot on SRAM) mode. ICP and ISP allow the user to download new code without removing the microcontroller from the actual end product; IAP means that the device can write non-volatile data in the flash memory while the application program is running. There needs no external high voltage for programming due to its built-in charge-pumping circuitry.

The **MG32F02** series retains all features of the ARM® 32-bit Cortex®-M0 with maximum 16K bytes of SRAM, 5 I/O ports, 32 external interrupts source with 4-level interrupt controller and seven 8/16-bits timer/counters. In addition, the **MG32F02** series has a System Tick Timer, two Watchdog Timers, three Advance timer modules with IC/OC/PWM, four Basic timer modules for universal using, on-chip crystal oscillator for 32.768 KHz to 25MHz, two high precision internal oscillators IHRCO for 11.059/12MHz and ILRCO for 32 KHz, one 12-bit ADC, four programmable threshold comparator, one Operational Amplifier with optional analog comparator function and one 12-bit voltage mode DAC.

Also, the **MG32F02** series support multiple and flexible communicate interface for production application. It provides alternate function pins those are including of GPIO, I2C, SPI, CAN, UART, Timer with IC/PWM, ADC, Analog Comparator, DAC, EMB, LCD, NCO, CCL and SWD(on chip debug). It has maximum 73 GPIO pins and provides programmable IO type - quasi-bidirectional , push-pull output , open-drain output , input only(Hi-z) with optional pull-high. In addition, it is built-in internal de-bounce circuit to deglitch noise for worse signals.

One direct memory access (DMA) controller is used to improve data transfer between peripherals and memory and memory to memory. The data can be transfer by DMA controller and does not cost any CPU time.

One external memory bus (EMB) controller is used to access external SRAM, NOR/NAND flash or 8080 interface LCD display panel. It supports multiple address bus and data bus multiplex modes. Also it supports synchronous or asynchronous timing with programmable cycle time for external devices.

For **MG32F02U** series, the chip provides a USB (Universal Serial Bus) full-speed device with relocated endpoint addresses. It is fully compliable with USB specification 2.0 and 1.1 to support various USB applications. The USB block contains an on-chip 3.3V regulator, a USB transceiver which transmits and receives differential USB signal, a USB Core to perform NRZI encoding and decoding, bit stuffing, CRC generation and checking, serial-parallel data transforming, data flow between USB data buffer and CPU.

For **MG32F02N/K** series, one Liquid Crystal Display (LCD) controller is used to drive external STN LCD with embedded LCD data RAM and supports LCD display on SLEEP and STOP modes. It provides maximum 8 lines of common and 40 lines of segment those are total 44 pins with GPIO alternate function. It supports 1/2, 1/3, 1/4 bias voltage and 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8 duty control. The controller can operate type-A or type-B frame control mode optionally with programmable dead time between frames or duty phases. Also it supports LCD blinking display.

For power management and reset control, the **MG32F02** series is built-in a power supervisor including of a Low Voltage Detector(LVD), maximum three Brown-out Detectors(BOD0/BOD1/BOD2), a Power-On Reset(POR) , a Low-voltage Reset(LVR). The **MG32F02** series has multiple power-down modes to reduce the power consumption: Sleep mode and Stop mode.

In the Sleep mode the CPU is frozen while the peripherals and the interrupt system are still operating. In the Stop mode the RAM and SFRs' value are saved and all other functions are inoperative; most importantly, in the Sleep mode the chip can be waked up by many interrupt or reset sources(POR/LVR/BOD0/BOD1/BOD2).

2.2. Applicable Chips

This User Guide is applicable to the chips in following.

- MG32F02A128/U128/A064/U064
- MG32F02V032
- MG32F02N128/K128/N064/K064

2.2.1. Chip Implementation Summary

The following table is showing the implemented functions and modules of chips.

Table 2-1. Chip Implementation Table

Chip Functions	MG32F02A128 MG32F02A064	MG32F02U128 MG32F02U064	MG32F02V032	MG32F02N128 MG32F02N064	MG32F02K128 MG32F02K064	Comment
CPU Core	ARM Cortex M0	ARM Cortex M0	ARM Cortex M0	ARM Cortex M0	ARM Cortex M0	with one NVIC and one single-cycle 32-bit multiplier
Flash ROM	128KB; 64KB	128KB; 64KB	32KB	128KB; 64KB	128KB; 64KB	memory space of AP+IAP+ISP
SRAM	16KB; 10KB	16KB	4KB	16KB; 10KB	16KB; 10KB	Upper 2K-byte suggestion for DMA (16/10 KB SRAM chip only)
Package	LQFP80/64; LQFP64/48	LQFP80/64; LQFP64/48	LQFP32, QFN32 TSSOP20	LQFP80/64; LQFP64/48	LQFP80/64; LQFP64/48	
IO Number	73/59; 59/44	70/56; 56/41	29/29/17	73/59; 59/44	73/59; 59/44	IO number by package
Max. Ext. Interrupt	73/59; 59/44	70/56; 56/41	29/29/17	73/59; 59/44	73/59; 59/44	independently external edge detect interrupt lines
Max. CPU Frequency	48MHz	48MHz	48MHz	48MHz	48MHz	
Internal Clock Source	ILRCO+IHRCO	ILRCO+IHRCO	ILRCO+IHRCO	ILRCO+IHRCO	ILRCO+IHRCO	12MHz(default) & 11.059MHz option for IHRCO
Voltage Detector	LVR+BOD0/1/2	LVR+BOD0/1/2	LVR+BOD0/1/2	LVR+BOD0/1/2	LVR+BOD0/1/2	Low Voltage Reset (LVR), BOD1: 4.2/3.6/2.4/2.0V, BOD2: 1.7V
Timers	16-bit*2: TM00/01 32-bit*5: TM1x/2x/36	16-bit*2: TM00/01 32-bit*5: TM1x/2x/36	16-bit*2: TM00/01 32-bit*4: TM1x/20/36	16-bit*2: TM00/01 32-bit*5: TM1x/2x/36	16-bit*2: TM00/01 32-bit*5: TM1x/2x/36	support Full-Counter, Cascade , Separate modes
IC/OC/PWM Channels	8-CH (16-bit) or 16-CH(8-bit)	8-CH (16-bit) or 16-CH(8-bit)	6-CH (16-bit) or 12-CH(8-bit)	8-CH (16-bit) or 16-CH(8-bit)	8-CH (16-bit) or 16-CH(8-bit)	IC: Input Capture, OC: Output Compare (support normal + complement output)
Complement PWM	7-CH	7-CH	5-CH	7-CH	7-CH	
QEI Support Mode	Mode 1,2,3,4,5	Mode 1,2,3,4,5	Mode 1,2,3,4,5	Mode 1,2,3,4,5	Mode 1,2,3,4,5	QEI : Quadrature Encoder Interface
Repetition Counter	8-Bit	8-Bit	8-Bit	8-Bit	8-Bit	Repetition counter for CKO and PWM
WDT	IWDT+WWDT	IWDT+WWDT	IWDT+WWDT	IWDT+WWDT	IWDT+WWDT	IWDT: Independent Watch Dog Timer, WWDT: System Window Watch Dog Timer
RTC	32-Bit	32-Bit	32-Bit	32-Bit	32-Bit	
ADC	12-Bit , 16-CH 1.5Msps	12-Bit , 16-CH 1.5Msps	12-Bit , 8-CH 1.0Msps	12-Bit , 16-CH 1.5Msps	12-Bit , 16-CH 1.5Msps	at room temperature
ACMP Units	2	2	-	2	2	fast rail-to-rail analog comparators with two R-ladder voltage reference
DAC	voltage DAC 12-Bit , 1-CH	voltage DAC 12-Bit , 1-CH	-	-	-	
DAC Conversion Rate	1Msps	1Msps	-	-	-	
DAC Output Buffer	yes	yes	-	-	-	
OPA Units	-	-	-	1	1	
I2C Units	2	2	2	2	2	optional Byte/Buffer mode, with/without clock stretching
UART Units	Advanced *3 Basic *4	Advanced *3 Basic *4	Advanced *2 Basic *1	Advanced *3 Basic *4	Advanced *3 Basic *4	URT0~2 can configurable to UART, Multi-processor, SPI, IrDA, LIN, ISO-7816 (SmartCard) and support Hardware flow control; URT4~7 are

						basic UART.
UART as SPI	Master/Slave *3	Master/Slave *3	Master/Slave *2	Master/Slave *3	Master/Slave *3	configurable and shared in advanced UART module
SPI Units	1	1	1	1	1	master/slave mode with or without NSS control
USB Units (#1)	-	USB Device *1	-	-	-	USB full speed device is with USB 2.0 compliant and embedded 3.3v LDO for USB operation power.
USB Endpoints/Buffer	-	8/512-Bytes	-	-	-	all endpoints are with both IN and Out directions
CAN Units (#2)	-	-	-	CAN 2.0 A/B 1Mbit/s *1	-	
EMB	16/8-Bit Bus	16/8-Bit Bus	-	-	-	support SRAM,NOR/NAND flash,8088 LCD IF
LCD	-	-	-	Configurable 44 Pins COM or SEG	Configurable 44 Pins COM or SEG	4COM*40SEG, 6COM*38SEG, 8COM*36SEG
LCD Duty	-	-	-	Static, 1/2, 1/3, 1/4, 1/8	Static, 1/2, 1/3, 1/4, 1/8	
LCD Bias	-	-	-	Static, 1/2, 1/3, 1/4	Static, 1/2, 1/3, 1/4	
LCD Power	-	-	-	VDD, CP, EXT	VDD, CP, EXT	CP: Charge Pump
DMA Channels	5-CH	5-CH	4-CH	5-CH	5-CH	single/block/demand mode for external pin trigger
DMA Memory Source	SRAM, EMB	SRAM, EMB	SRAM, Flash	SRAM, EMB	SRAM, EMB	Flash is not supported as DMA destination; EMB: external memory through EMB interface
CRC	CRC8+16+32	CRC8+16+32	CRC8+16+32	CRC8+16+32	CRC8+16+32	CRC8/CRC16/CCITT16/CRC32 fixed polynomial
HW Divider	32bit/8-clocks	32bit/8-clocks	-	-	-	signed/unsigned hardware divider (Dividend and Divisor Bits/Cycles)
OBM Units	2	2	2	2	2	Output Signal Break and Modulation
NCO Units	1	1	1	1	1	numerically controlled oscillator, Output frequency <= 1/2 Input frequency
CCL Units	2	2	2	2	2	configurable custom logic
SDT Units	1	1	1	1	1	sequential state detector
ASB Units	-	-	4-CH	4-CH	4-CH	ARGB LED serial bus
Flash Regions	AP,IAP,ISP	AP,IAP,ISP	AP,IAP,ISP	AP,IAP,ISP	AP,IAP,ISP	Application flash, In-Application-Programming data flash, In-System-Programming flash (ISP: In-System-Programming data flash)
Flash Program IF	ICP,ISP,CPU	ICP,ISP,CPU	ICP,ISP,CPU	ICP,ISP,CPU	ICP,ISP,CPU	In-Chip-Programming interface(U1 writer), In-System-Programming interface(UART), CPU firmware directly programming

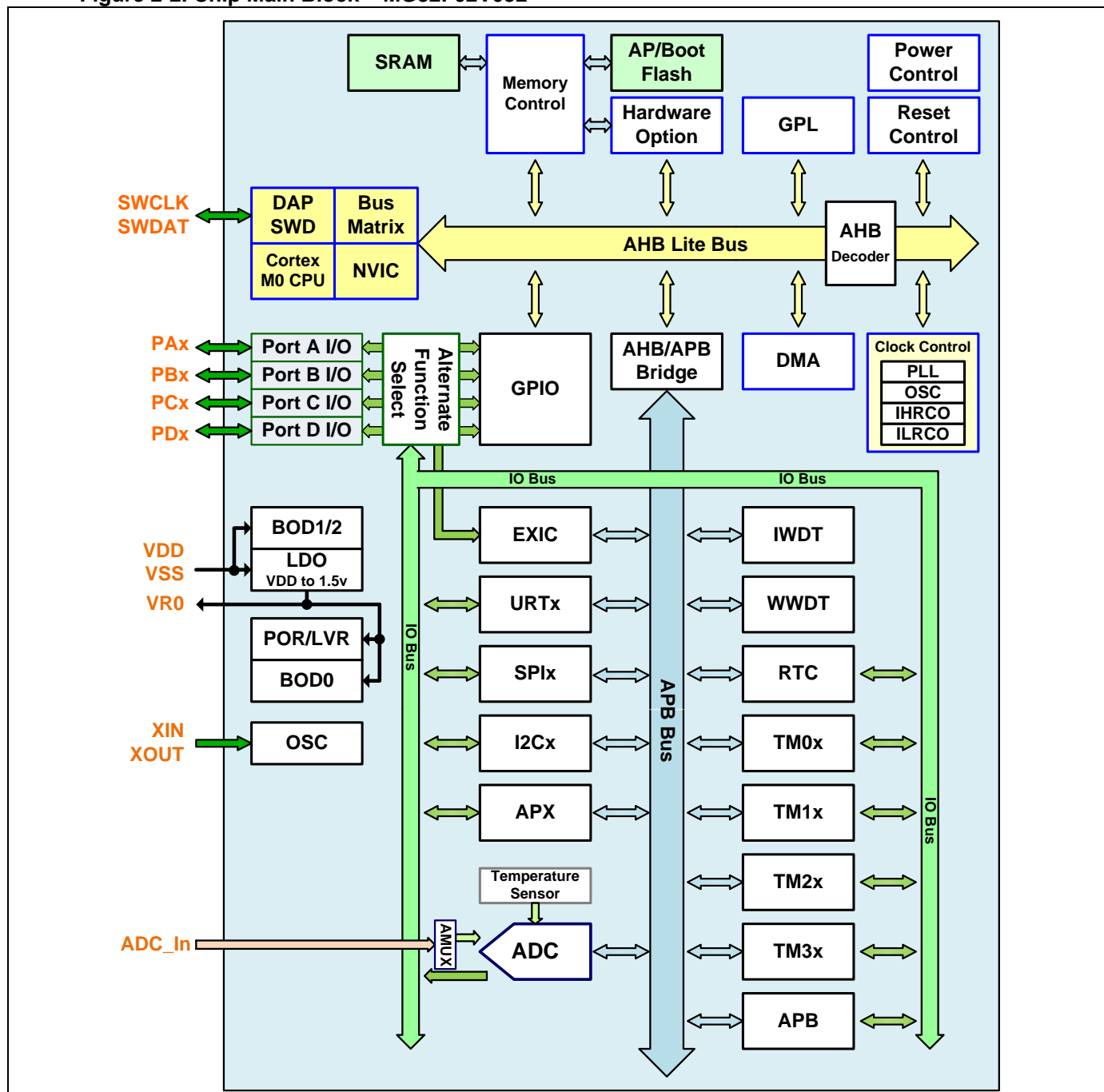
<Note> "-": not supported or not implemented

#1: Only MG32F02Uxxx series is supported of USB function.

#2: Only MG32F02Nxxx series is supported of CAN function.

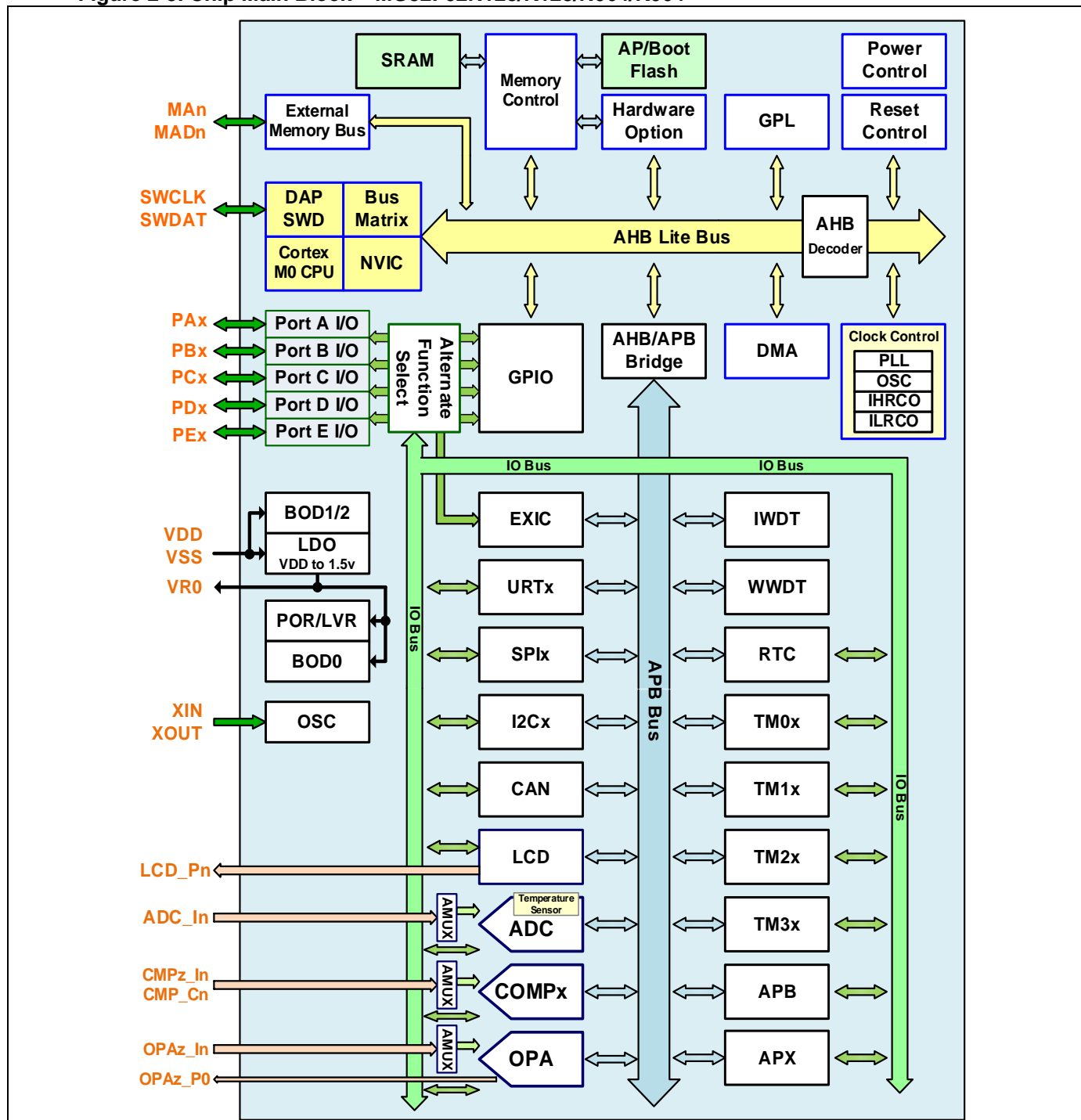
2.3.2. MG32F02V032 Main Block

Figure 2-2. Chip Main Block – MG32F02V032



2.3.3. MG32F02N128/K128/N064/K064 Main Block

Figure 2-3. Chip Main Block – MG32F02N128/K128/N064/K064



2.4. CPU Core

The chip is embedded a CPU core of Cortex®-M0 processor. The processor is a configurable, multistage, 32-bit RISC processor. It has an AMBA AHB-Lite interface and includes an NVIC component. It also has optional DAP hardware debug functionality.

The processor can execute Thumb code and is compatible with other Cortex®-M profile processor. The profile supports two modes -Thread mode and Handler mode. Handler mode is entered as a result of an exception. An exception return can only be issued in Handler mode. Thread mode is entered on Reset, and can be entered as a result of an exception return.

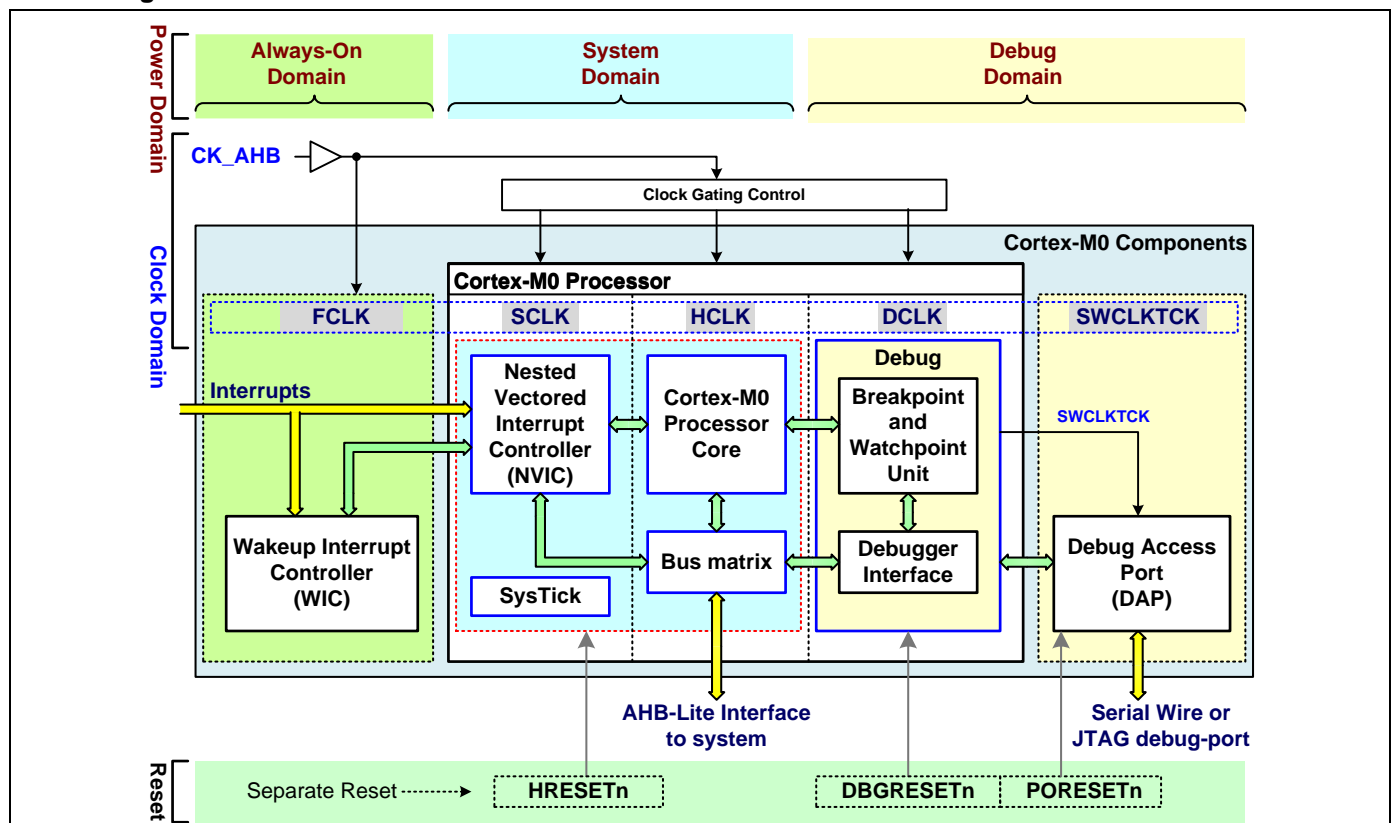
2.4.1. CPU Features

- ARM® 32-bit Cortex®-M0 CPU
- Operation frequency up to 48MHz
- Built-in one NVIC for 32 external interrupt inputs with 4-level priority
- Built-in one 24-bit system tick timer
- Built-in one single-cycle 32-bit multiplier
- Built-in one SWD serial wire debugger with 2 watch points and 4 breakpoint
- The ARMv6-M Thumb® instruction set

2.4.2. ARM Cortex-M0 Processor

The following diagram is showing the block of ARM® Cortex®-M0 Processor. Refer the ARM® “Cortex-M0 Devices Generic User Guide” for more information about Cortex®-M0 CPU.

Figure 2-4. ARM Cortex-M0 Processor



2.5. Memory Organization

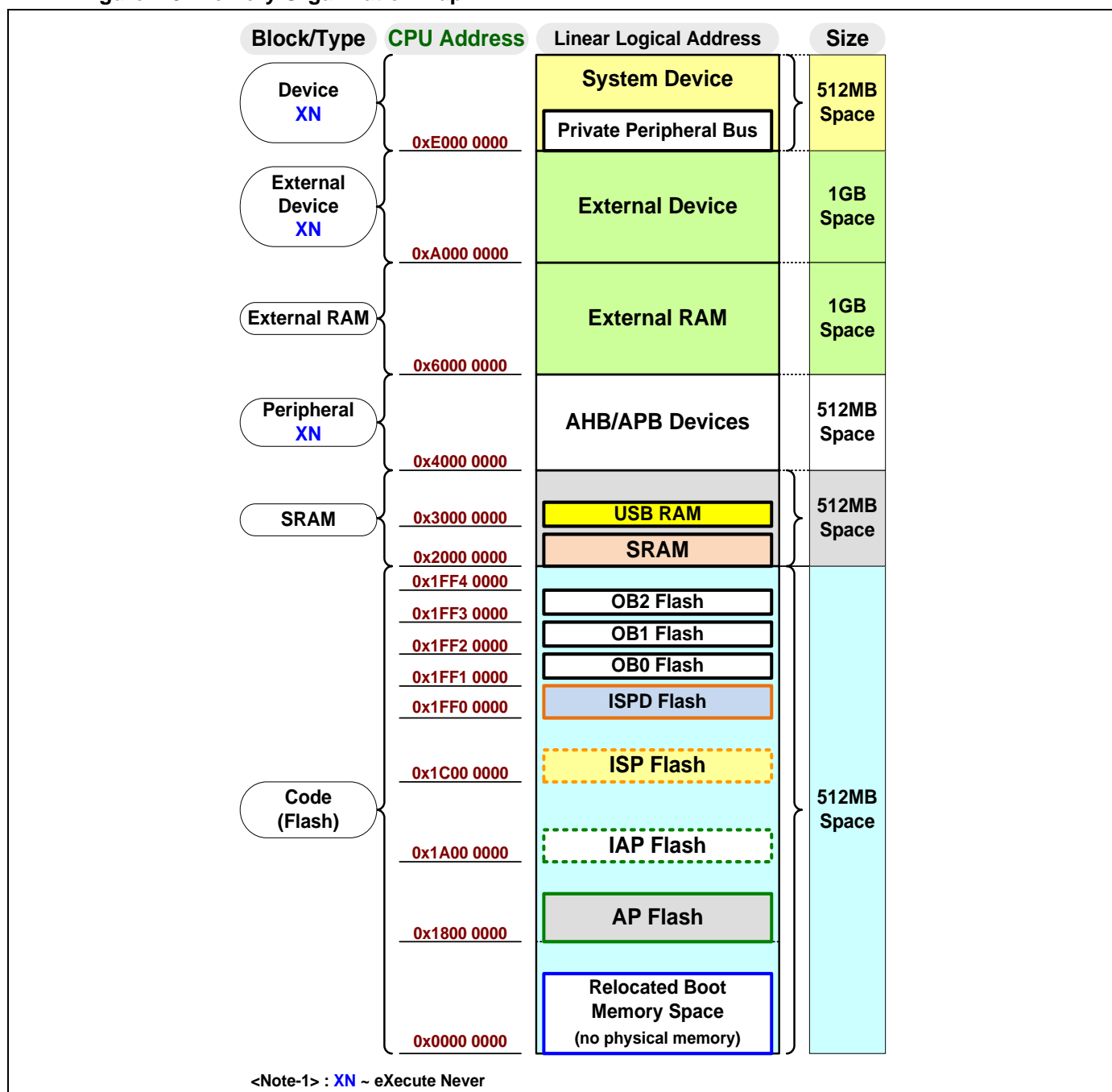
There are maximum **16K** bytes of SRAM built in the chip. The chip has up to **128K** bytes of embedded main flash memory for code and data, programmable memory size of embedded system flash memory for boot load code and **64** bytes of embedded option-byte (**OB**) flash memory for chip configuration. Others, there are many modules' independent hardware control registers and locate at the memory space of AHB/APB devices. Please refer the chip Data Sheet for the actual embedded SRAM and flash memory size.

Please refer the table of “[Chip Implementation Table](#)” in the section of “Applicable Chips” for more detail information about MCU chip embedded flash and SRAM configuration.

User can configure the whole flash to store for his Application Program (AP) code, In-System-Program (ISP) code and In-Application-Program (IAP) memory. User can adjust the size for the three flash memories.

The following diagram is showing the memory organization map of CPU with total 4G bytes address space. There are separated eight memory blocks and the memory size is 512M-byte for each block. The block is signed “XN” which is not able to execute code. The USB RAM is embedded only for MG32F02U series.

Figure 2-5. Memory Organization Map



2.5.1. CPU Memory Map

The following table is showing the memory address map of CPU. The block is signed “XN” which is not able to execute code.

Table 2-2. CPU Memory Address Map

Block Index	Block Name	XN	Boundary address		Size	Address Space	Note
			Start address	End address			
7	System Device	XN	0xE010 0000	0xFFFF FFFF	511MB	VENDOR_SYS	
			0xE000 0000	0xE00F FFFF	1MB	Private Peripheral Bus(PPB)	M0 Reserved Cortex M0 internal peripherals
6	External Device	XN	0xC000 0000	0xDFFF FFFF	512MB	Reserved	External memory (SRAM, Flash)
5	External Device	XN	0xA000 0000	0xBFFF FFFF	512MB	Reserved	External memory (SRAM, Flash)
4	External RAM		0x8000 0000	0x9FFF FFFF	512MB	Reserved	External memory (SRAM, Flash)
3	External RAM		0x6000 0000	0x7FFF FFFF	512MB	Reserved	External memory (SRAM, Flash)
2	Peripheral	XN	0x4000 0000	0x5FFF FFFF	512MB	APB/AHB	APB/AHB modules
1	SRAM		0x3000 0200	0x3FFF FFFF	256MB	Reserved	
			0x3000 0000	0x3000 01FF	512B	USB SRAM (*2)	
			0x2000 4000	0x2FFF FFFF	256MB	Reserved	
			0x2000 3800	0x2000 3FFF	2KB	SRAM (*2)	Upper 2K-byte suggestion for DMA
			0x2000 0000	0x2000 37FF	14KB		
0	Code		0x1FF3 0400	0x1FFF FFFF	831KB	Reserved	
			0x1FF3 0040	0x1FF3 03FF	960B	OB Flash-2 (*2)	
			0x1FF3 0000	0x1FF3 003F	64B		Hardware Option byte-2 (64-byte)
			0x1FF2 0400	0x1FF2 FFFF	63KB	Reserved	
			0x1FF2 0050	0x1FF2 03FF	944B	OB Flash-1 (*2)	
			0x1FF2 0040	0x1FF2 004F	16B		Unique ID (16-byte)
			0x1FF2 0000	0x1FF2 003F	64B		Hardware Option byte-1 (64-byte)
			0x1FF1 0400	0x1FF1 FFFF	63KB	Reserved	
			0x1FF1 0040	0x1FF1 03FF	960B	OB Flash-0 (*2)	
			0x1FF1 0000	0x1FF1 003F	64B		Hardware Option byte-0 (64-byte)
			0x1FF0 0400	0x1FF0 FFFF	63KB	Reserved	
			0x1FF0 0000	0x1FF0 03FF	1KB	ISPD Flash (*2)	ISP data flash
			0x1C02 1000	0x1FEF FFFF	63MB	Reserved	
			0x1C00 0000	0x1C01 FFFF	128KB	ISP Flash (*2)	Boot Flash memory (configurable size)
			0x1A02 1000	0x1BFF FFFF	32MB	Reserved	
			0x1A00 0000	0x1A01 FFFF	128KB	IAP Flash (*2)	Data Flash memory (configurable size)
			0x1802 1000	0x19FF FFFF	32MB	Reserved	
			0x1800 0000	0x1801 FFFF	128KB	AP Flash (*2)	Application Flash memory (configurable size by chip option)
			0x0002 1000	0x17FF FFFF	384MB	Reserved	
			0x0000 0000	0x0001 FFFF	128KB	Relocated memory space (*1)	Interrupt Vector 0x0000 00C0~0x0000 0000

XN : eXecute Never , 1 Block = 512MB

*1: Relocated memory space : Main flash memory, Boot flash memory or SRAM depending on BOOT configuration

*2: The table lists the maximum value. Refer the “Chip Selection Table” of Data Sheet about actual memory size.

2.5.2. Peripheral Memory Boundary

The following tables show the peripherals' boundary address in the related chip.

● MG32F02A128/U128/A064/U064 Peripheral Memory Boundary Address

Table 2-3. Peripheral Memory Boundary Address – MG32F02A128/U128/A064/U064

Address Type	Boundary address		Size	Sections / Groups Peripheral	Module	Note
	Start address	End address				
APB	0x5F01 0100	0x5FFF FFFF	16MB	APB	Reserved	
	0x5F01 0000	0x5F01 00FF	256B		APX	APB module extended control
	0x5F00 0100	0x5F00 FFFF	64KB		Reserved	
	0x5F00 0000	0x5F00 00FF	256B		APB	APB module global control
	0x5E00 0000	0x5EFF FFFF	16MB	Reserved	Reserved	
	0x5D04 0100	0x5DFF FFFF	16MB	WDT/RTC	Reserved	
	0x5D04 0000	0x5D04 00FF	256B		RTC	Real Time Clock
	0x5D01 0100	0x5D03 FFFF	192KB		Reserved	
	0x5D01 0000	0x5D01 00FF	256B		WWDT	Window WatchDog Timer
	0x5D00 0100	0x5D00 FFFF	64KB		Reserved	
	0x5D00 0000	0x5D00 00FF	256B		IWDT	Independent WatchDog Timer
	0x5C08 0100	0x5CFF FFFF	15MB	CMP/DAC	Reserved	
	0x5C08 0000	0x5C08 00FF	256B		DAC	Digital-to-Analog controller
	0x5C00 0100	0x5C07 FFFF	512KB		Reserved	
	0x5C00 0000	0x5C00 00FF	256B		CMP	Analog Comparator 0,1
	0x5B00 0100	0x5BFF FFFF	16MB	ADC	Reserved	
	0x5B00 0000	0x5B00 00FF	256B		ADC0	Analog-to-Digital controller
	0x5700 0000	0x5AFF FFFF	64MB	Reserved	Reserved	
	0x5686 0100	0x56FF FFFF	8MB	TM2x/3x	Reserved	
	0x5686 0000	0x5686 00FF	256B		TM36	32-bit Timer with 4 IC/OC/PWM
	0x5606 0100	0x5685 FFFF	8MB		Reserved	
	0x5606 0000	0x5606 00FF	256B		TM26	32-bit Timer with 2 IC/OC/PWM
	0x5600 0100	0x5605 FFFF	384KB		Reserved	
	0x5600 0000	0x5600 00FF	256B		TM20	32-bit Timer with 2 IC/OC/PWM
	0x5586 0100	0x55FF FFFF	8MB	TM0x/1x	Reserved	
	0x5586 0000	0x5586 00FF	256B		TM16	Basic32-bit Timer/Counter
	0x5580 0100	0x5585 FFFF	384KB		Reserved	
	0x5580 0000	0x5580 00FF	256B		TM10	Basic32-bit Timer/Counter
	0x5501 0100	0x557F FFFF	8MB		Reserved	
	0x5501 0000	0x5501 00FF	256B		TM01	Basic 16-bit Timer/Counter
	0x5500 0100	0x5500 FFFF	64KB		Reserved	
	0x5500 0000	0x5500 00FF	256B		TM00	Basic 16-bit Timer/Counter
	0x5401 0000	0x54FF FFFF	16MB	Reserved	Reserved	
	0x5400 0100	0x5400 FFFF	64KB	USB	Reserved	
	0x5400 0000	0x5400 00FF	256B		USB	USB bus controller
	0x5300 0100	0x53FF FFFF	16MB	SPI	Reserved	
	0x5300 0000	0x5300 00FF	256B		SPI0	SPI bus controller with data buffer
	0x5207 0100	0x52FF FFFF	16MB	UART	Reserved	
	0x5207 0000	0x5207 00FF	256B		URT7	Basic UART bus controller
	0x5206 0100	0x5206 FFFF	64KB		Reserved	
	0x5206 0000	0x5206 00FF	256B		URT6	Basic UART bus controller
	0x5205 0100	0x5205 FFFF	64KB		Reserved	
	0x5205 0000	0x5205 00FF	256B		URT5	Basic UART bus controller
	0x5204 0100	0x5204 FFFF	64KB		Reserved	
	0x5204 0000	0x5204 00FF	256B		URT4	Basic UART bus controller
	0x5202 0100	0x5203 FFFF	128KB		Reserved	
	0x5202 0000	0x5202 00FF	256B		URT2	Advance UART bus controller
	0x5201 0100	0x5201 FFFF	64KB		Reserved	
	0x5201 0000	0x5201 00FF	256B		URT1	Advance UART bus controller

	0x5200 0100	0x5200 FFFF	64KB		Reserved	
	0x5200 0000	0x5200 00FF	256B		URT0	Advance UART bus controller
	0x5101 0100	0x51FF FFFF	16MB		Reserved	
	0x5101 0000	0x5101 00FF	256B		I2C1	I2C bus controller
	0x5100 0100	0x5100 FFFF	64KB		Reserved	
	0x5100 0000	0x5100 00FF	256B		I2C0	I2C bus controller
	0x5000 0100	0x50FF FFFF	16MB		Reserved	
	0x5000 0000	0x5000 00FF	256B		EXIC	External Interrupt Controller
AHB	0x4FF0 0100	0x4FFF FFFF	1024KB	Chip	Reserved	
	0x4FF0 0000	0x4FF0 00FF	256B		CFG	Hardware option (NVR0/1/2)
	0x4F00 0100	0x4FEF FFFF	15MB		Reserved	
	0x4F00 0000	0x4F00 00FF	256B		WRI	Writer Interface Control
	0x4E00 0000	0x4EFF FFFF	16MB	Reserved	Reserved	
	0x4D02 0100	0x4DFF FFFF	16MB	Memory	Reserved	
	0x4D02 0000	0x4D02 00FF	256B		EMB	External Memory Bus Controller
	0x4D00 0100	0x4D01 FFFF	128KB		Reserved	
	0x4D00 0000	0x4D00 00FF	256B		MEM	Internal Memory Controller
	0x4C03 0100	0x4CFF FFFF	16MB	System	Reserved	
	0x4C03 0000	0x4C03 00FF	256B		SYS	System and Chip Control
	0x4C02 0100	0x4C02 FFFF	64KB		Reserved	
	0x4C02 0000	0x4C02 00FF	256B		PW	Power Management Controller
	0x4C01 0100	0x4C01 FFFF	64KB		Reserved	
	0x4C01 0000	0x4C01 00FF	256B		CSC	Clock Source Controller
	0x4C00 0100	0x4C00 FFFF	64KB		Reserved	
	0x4C00 0000	0x4C00 00FF	256B		RST	Reset Source Controller
	0x4BF0 0100	0x4BFF FFFF	1024KB	General Purpose	Reserved	
	0x4BF0 0000	0x4BF0 00FF	256B		DMA	Direct memory access
	0x4B00 0100	0x4BEF FFFF	15MB		Reserved	
	0x4B00 0000	0x4B00 00FF	256B		GPL	General Purpose Logic
	0x4500 0000	0x4AFF FFFF	96MB	Reserved	Reserved	Reserved for future design
	0x4404 0100	0x44FF FFFF	16MB	IO Configure	Reserved	
	0x4404 0000	0x4404 00FF	256B		PE	
	0x4403 0100	0x4403 FFFF	64KB		Reserved	
	0x4403 0000	0x4403 00FF	256B		PD	
	0x4402 0100	0x4402 FFFF	64KB		Reserved	
	0x4402 0000	0x4402 00FF	256B		PC	
	0x4401 0100	0x4401 FFFF	64KB		Reserved	
	0x4401 0000	0x4401 00FF	256B		PB	
	0x4400 0100	0x4400 FFFF	64KB		Reserved	
	0x4400 0000	0x4400 00FF	256B		PA	
	0x4100 0200	0x43FF FFFF	48MB	Reserved		Reserved for future design
	0x4100 0000	0x4100 01FF	512B	GPIO	IOP	IO Port Input/Output
	0x4000 0000	0x40FF FFFF	16MB	Reserved		Reserved for future design

● MG32F02V032 Peripheral Memory Boundary Address

Table 2-4. Peripheral Memory Boundary Address – MG32F02V032

Address Type	Boundary address		Size	Sections / Groups Peripheral	Module	Note
	Start address	End address				
APB	0x5F01 0100	0x5FFF FFFF	16MB	APB	Reserved	
	0x5F01 0000	0x5F01 00FF	256B		APX	APB module extended control
	0x5F00 0100	0x5F00 FFFF	64KB		Reserved	
	0x5F00 0000	0x5F00 00FF	256B		APB	APB module global control
	0x5E00 0000	0x5EFF FFFF	16MB	Reserved	Reserved	
	0x5D04 0100	0x5DFF FFFF	16MB	WDT/RTC	Reserved	
	0x5D04 0000	0x5D04 00FF	256B		RTC	Real Time Clock
	0x5D01 0100	0x5D03 FFFF	192KB		Reserved	
	0x5D01 0000	0x5D01 00FF	256B		WWDT	Window WatchDog Timer

	0x5D00 0100	0x5D00 FFFF	64KB	ADC	Reserved	
	0x5D00 0000	0x5D00 00FF	256B		IWDT	Independent WatchDog Timer
	0x5B00 0100	0x5CFF FFFF	32MB		Reserved	
	0x5B00 0000	0x5B00 00FF	256B		ADC0	Analog-to-Digital controller
	0x5700 0000	0x5AFF FFFF	64MB	Reserved	Reserved	
	0x5686 0100	0x56FF FFFF	8MB	TM2x/3x	Reserved	
	0x5686 0000	0x5686 00FF	256B		TM36	32-bit Timer with 4 IC/OC/PWM
	0x5600 0100	0x5685 FFFF	8MB		Reserved	
	0x5600 0000	0x5600 00FF	256B		TM20	32-bit Timer with 2 IC/OC/PWM
	0x5586 0100	0x55FF FFFF	8MB	TM0x/1x	Reserved	
	0x5586 0000	0x5586 00FF	256B		TM16	Basic32-bit Timer/Counter
	0x5580 0100	0x5585 FFFF	384KB		Reserved	
	0x5580 0000	0x5580 00FF	256B		TM10	Basic32-bit Timer/Counter
	0x5501 0100	0x557F FFFF	8MB		Reserved	
	0x5501 0000	0x5501 00FF	256B		TM01	Basic 16-bit Timer/Counter
	0x5500 0100	0x5500 FFFF	64KB		Reserved	
	0x5500 0000	0x5500 00FF	256B		TM00	Basic 16-bit Timer/Counter
	0x5401 0000	0x54FF FFFF	16MB	Reserved	Reserved	
	0x5300 0100	0x5400 FFFF	16MB	SPI	Reserved	
	0x5300 0000	0x5300 00FF	256B		SPI0	SPI bus controller with data buffer
	0x5204 0100	0x52FF FFFF	16MB		Reserved	
	0x5204 0000	0x5204 00FF	256B		URT4	Basic UART bus controller
	0x5201 0100	0x5203 FFFF	192KB		Reserved	
	0x5201 0000	0x5201 00FF	256B		URT1	Advance UART bus controller
	0x5200 0100	0x5200 FFFF	64KB		Reserved	
	0x5200 0000	0x5200 00FF	256B		URT0	Advance UART bus controller
	0x5101 0100	0x51FF FFFF	16MB	I2C	Reserved	
	0x5101 0000	0x5101 00FF	256B		I2C1	I2C bus controller
	0x5100 0100	0x5100 FFFF	64KB		Reserved	
	0x5100 0000	0x5100 00FF	256B		I2C0	I2C bus controller
	0x5000 0100	0x50FF FFFF	16MB	EXT Interrupt	Reserved	
	0x5000 0000	0x5000 00FF	256B		EXIC	External Interrupt Controller
AHB	0x4FF0 0100	0x4FFF FFFF	1MB	Chip	Reserved	
	0x4FF0 0000	0x4FF0 00FF	256B		CFG	Hardware option (NVR0/1/2)
	0x4F00 0100	0x4FEF FFFF	15MB		Reserved	
	0x4F00 0000	0x4F00 00FF	256B		WRI	Writer Interface Control
	0x4E00 0000	0x4EFF FFFF	16MB	Reserved	Reserved	
	0x4D00 0100	0x4DFF FFFF	16MB	System	Reserved	
	0x4D00 0000	0x4D00 00FF	256B		MEM	Internal Memory Controller
	0x4C03 0100	0x4CFF FFFF	16MB		Reserved	
	0x4C03 0000	0x4C03 00FF	256B		SYS	System and Chip Control
	0x4C02 0100	0x4C02 FFFF	64KB		Reserved	
	0x4C02 0000	0x4C02 00FF	256B		PW	Power Management Controller
	0x4C01 0100	0x4C01 FFFF	64KB		Reserved	
	0x4C01 0000	0x4C01 00FF	256B		CSC	Clock Source Controller
	0x4C00 0100	0x4C00 FFFF	64KB	General Purpose	Reserved	
	0x4C00 0000	0x4C00 00FF	256B		RST	Reset Source Controller
	0x4BF0 0100	0x4BFF FFFF	1MB		Reserved	
	0x4BF0 0000	0x4BF0 00FF	256B		DMA	Direct memory access
	0x4B00 0100	0x4BEF FFFF	15MB		Reserved	
	0x4B00 0000	0x4B00 00FF	256B		GPL	General Purpose Logic
	0x4500 0000	0x4AFF FFFF	96MB	Reserved	Reserved	Reserved for future design
	0x4403 0100	0x44FF FFFF	16MB	IO Configure	Reserved	
	0x4403 0000	0x4403 00FF	256B		PD	
	0x4402 0100	0x4402 FFFF	64KB		Reserved	
	0x4402 0000	0x4402 00FF	256B		PC	
	0x4401 0100	0x4401 FFFF	64KB		Reserved	
	0x4401 0000	0x4401 00FF	256B		PB	
	0x4400 0100	0x4400 FFFF	64KB		Reserved	

	0x4400 0000	0x4400 00FF	256B		PA	
	0x4100 0200	0x43FF FFFF	48MB	Reserved		Reserved for future design
	0x4100 0000	0x4100 01FF	512B	GPIO	IOP	IO Port Input/Output
	0x4000 0000	0x40FF FFFF	16MB	Reserved		Reserved for future design

● MG32F02N128/K128/N064/K064 Peripheral Memory Boundary Address

Table 2-5. Peripheral Memory Boundary Address – MG32F02N128/K128/N064/K064

Address Type	Boundary address		Size	Sections / Groups Peripheral	Module	Note
	Start address	End address				
APB	0x5F01 0100	0x5FFF FFFF	16MB	APB	Reserved	
	0x5F01 0000	0x5F01 00FF	256B		APX	APB module extended control
	0x5F00 0100	0x5F00 FFFF	64KB		Reserved	
	0x5F00 0000	0x5F00 00FF	256B		APB	APB module global control
	0x5E00 0000	0x5EFF FFFF	16MB	Reserved	Reserved	
	0x5D04 0100	0x5DFF FFFF	16MB	WDT/RTC	Reserved	
	0x5D04 0000	0x5D04 00FF	256B		RTC	Real Time Clock
	0x5D01 0100	0x5D03 FFFF	192KB		Reserved	
	0x5D01 0000	0x5D01 00FF	256B		WWDT	Window WatchDog Timer
	0x5D00 0100	0x5D00 FFFF	64KB		Reserved	
	0x5D00 0000	0x5D00 00FF	256B		IWDT	Independent WatchDog Timer
	0x5C00 0100	0x5CFF FFFF	16MB	CMP/DAC	Reserved	
	0x5C00 0000	0x5C00 00FF	256B		CMP	Analog Comparator 0,1
	0x5B00 0100	0x5BFF FFFF	16MB	ADC	Reserved	
	0x5B00 0000	0x5B00 00FF	256B		ADC0	Analog-to-Digital controller
	0x5A08 0100	0x5AFF FFFF	15MB	OPA	Reserved	
	0x5A08 0000	0x5A08 00FF	256B		OPA	Operational amplifier
	0x5A00 0100	0x5A07 FFFF	512KB	LCD	Reserved	
	0x5A00 0000	0x5A00 00FF	256B		LCD	Liquid Crystal Display controller
	0x5700 0000	0x59FF FFFF	48MB	Reserved	Reserved	
	0x5686 0100	0x56FF FFFF	8MB	TM2x/3x	Reserved	
	0x5686 0000	0x5686 00FF	256B		TM36	32-bit Timer with 4 IC/OC/PWM
	0x5606 0100	0x5685 FFFF	8MB		Reserved	
	0x5606 0000	0x5606 00FF	256B		TM26	32-bit Timer with 2 IC/OC/PWM
	0x5600 0100	0x5605 FFFF	384KB		Reserved	
	0x5600 0000	0x5600 00FF	256B		TM20	32-bit Timer with 2 IC/OC/PWM
	0x5586 0100	0x55FF FFFF	8MB	TM0x/1x	Reserved	
	0x5586 0000	0x5586 00FF	256B		TM16	Basic32-bit Timer/Counter
	0x5580 0100	0x5585 FFFF	384KB		Reserved	
	0x5580 0000	0x5580 00FF	256B		TM10	Basic32-bit Timer/Counter
	0x5501 0000	0x5501 00FF	256B		TM01	Basic 16-bit Timer/Counter
	0x5500 0100	0x5500 FFFF	64KB		Reserved	
	0x5500 0000	0x5500 00FF	256B		TM00	Basic 16-bit Timer/Counter
	0x5408 0100	0x54FF FFFF	15MB	CAN	Reserved	
	0x5408 0000	0x5408 00FF	256B		CAN0	Controller Area Network
	0x5300 0100	0x5407 FFFF	16MB	SPI	Reserved	
	0x5300 0000	0x5300 00FF	256B		SPI0	SPI bus controller with data buffer
	0x5207 0100	0x52FF FFFF	16MB	UART	Reserved	
	0x5207 0000	0x5207 00FF	256B		URT7	Basic UART bus controller
	0x5206 0100	0x5206 FFFF	64KB		Reserved	
	0x5206 0000	0x5206 00FF	256B		URT6	Basic UART bus controller
	0x5205 0100	0x5205 FFFF	64KB		Reserved	
	0x5205 0000	0x5205 00FF	256B		URT5	Basic UART bus controller
	0x5204 0100	0x5204 FFFF	64KB		Reserved	
	0x5204 0000	0x5204 00FF	256B		URT4	Basic UART bus controller
	0x5202 0100	0x5203 FFFF	128KB		Reserved	
	0x5202 0000	0x5202 00FF	256B		URT2	Advance UART bus controller
	0x5201 0100	0x5201 FFFF	64KB		Reserved	

	0x5201 0000	0x5201 00FF	256B		URT1	Advance UART bus controller
	0x5200 0100	0x5200 FFFF	64KB		Reserved	
	0x5200 0000	0x5200 00FF	256B		URT0	Advance UART bus controller
	0x5101 0100	0x51FF FFFF	16MB	I2C	Reserved	
	0x5101 0000	0x5101 00FF	256B		I2C1	I2C bus controller
	0x5100 0100	0x5100 FFFF	64KB		Reserved	
	0x5100 0000	0x5100 00FF	256B		I2C0	I2C bus controller
	0x5000 0100	0x50FF FFFF	16MB	EXT Interrupt	Reserved	
	0x5000 0000	0x5000 00FF	256B		EXIC	External Interrupt Controller
AHB	0x4FF0 0100	0x4FFF FFFF	1MB	Chip	Reserved	
	0x4FF0 0000	0x4FF0 00FF	256B		CFG	Hardware option (NVR0/1/2)
	0x4F00 0100	0x4FEF FFFF	15MB		Reserved	
	0x4F00 0000	0x4F00 00FF	256B		WRI	Writer Interface Control
	0x4E00 0000	0x4EFF FFFF	16MB	Reserved	Reserved	
	0x4D00 0100	0x4DFF FFFF	16MB	Memory	Reserved	
	0x4D00 0000	0x4D00 00FF	256B		MEM	Internal Memory Controller
	0x4C03 0100	0x4CFF FFFF	16MB	System	Reserved	
	0x4C03 0000	0x4C03 00FF	256B		SYS	System and Chip Control
	0x4C02 0100	0x4C02 FFFF	64KB		Reserved	
	0x4C02 0000	0x4C02 00FF	256B		PW	Power Management Controller
	0x4C01 0100	0x4C01 FFFF	64KB		Reserved	
	0x4C01 0000	0x4C01 00FF	256B		CSC	Clock Source Controller
	0x4C00 0100	0x4C00 FFFF	64KB		Reserved	
	0x4C00 0000	0x4C00 00FF	256B		RST	Reset Source Controller
	0x4BF0 0100	0x4BFF FFFF	1MB	General Purpose	Reserved	
	0x4BF0 0000	0x4BF0 00FF	256B		DMA	Direct memory access
	0x4B00 0100	0x4BEF FFFF	15MB		Reserved	
	0x4B00 0000	0x4B00 00FF	256B		GPL	General Purpose Logic
	0x4500 0000	0x4AFF FFFF	96MB	Reserved	Reserved	Reserved for future design
	0x4404 0100	0x44FF FFFF	16MB	IO Configure	Reserved	
	0x4404 0000	0x4404 00FF	256B		PE	
	0x4403 0100	0x4403 FFFF	64KB		Reserved	
	0x4403 0000	0x4403 00FF	256B		PD	
	0x4402 0100	0x4402 FFFF	64KB		Reserved	
	0x4402 0000	0x4402 00FF	256B		PC	
	0x4401 0100	0x4401 FFFF	64KB		Reserved	
	0x4401 0000	0x4401 00FF	256B		PB	
	0x4400 0100	0x4400 FFFF	64KB	Reserved	Reserved	
	0x4400 0000	0x4400 00FF	256B		PA	
	0x4100 0200	0x43FF FFFF	48MB	Reserved		Reserved for future design
	0x4100 0000	0x4100 01FF	512B	GPIO	IOP	IO Port Input/Output
	0x4000 0000	0x40FF FFFF	16MB	Reserved		Reserved for future design

2.5.3. Boot Modes

During chip startup, the hardware configuration option-byte (**OB**) is used to select one of the three boot options as follows.

- Boot from User Application Program (AP) Flash
- Boot from In-System-Program (ISP)
- Boot from embedded SRAM

Refer the register setting of **MEM_BOOT_MS** in the section of “[Memory Boot Modes](#)” and the hardware configuration of **BOOT_MS** in the section of “[Hardware Option Byte](#)” for more information.

2.5.4. Flash Code Lock

The flash code is default unlocked when the chip is manufactured from megawin manufactory. User can lock or unlock and program the flash by through the megawin ICP programmer. Refer the hardware configuration of **LOCK_DIS** in the section of “[Hardware Option Byte](#)” for more information.

2.6. Chip Debug

The chip is embedded the Cortex®-M0 CPU which is built-in the hardware a complete hardware debug solution, with extensive hardware breakpoint and watch-point options. This provides high system visibility of the processor, memory and peripherals through a 2-pin Serial Wire Debug (SWD) port.

Refer to the “Cortex-M0 Technical Reference Manual” for more information.

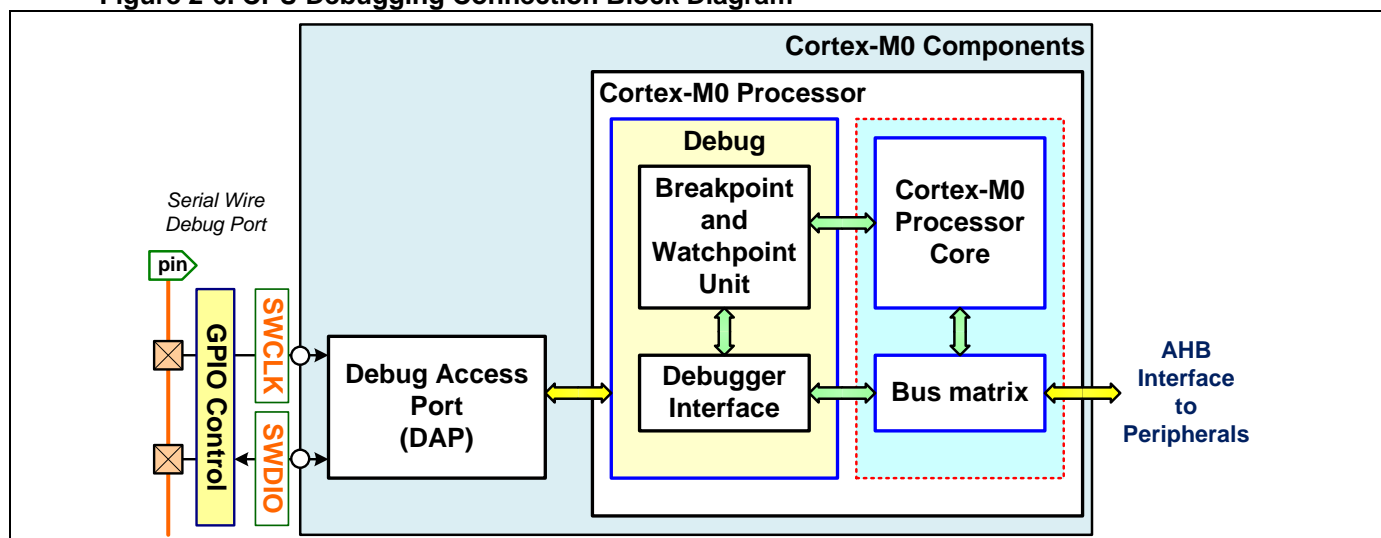
2.6.1. Chip DAP

The chip is implemented the Debug Access Port (DAP), Break point unit (BPU) and Data watch-point trigger (DWT) for hardware debug solution. The DAP can control the SWD interface. It connects to the debug interface of Cortex®-M0 CPU and Bus Matrix through AHP interface to internal peripheral modules.

User can allow stopping the core either on a given executing instruction (breakpoint) or data access (watch-point) by through the SWD interface. When the chip is stopped, user can monitor or access the CPU internal state. Also user can monitor or access the chip memory domain or the registers of any released peripheral module. The chip will restore the system state and resume the CPU instruction execution when user finish the monitoring and release the DAP.

The following diagram is showing the CPU debugging connection block diagram.

Figure 2-6. CPU Debugging Connection Block Diagram



2.6.2. SWD Interface

The SWD interface is including of SWCLK and SWDIO on PC4 and PC5 two pins.

- **SWCLK**
Serial wire clock input signal.
- **SWDIO**
Serial wire data input/output bidirectional signal.

These two pins of SWD are assigned as dedicated pins with internal pull-up for the chip debugger after Power-on reset or Cold reset. User can disable the SWD pins and release as GPIO or other alternate function pins in hardware option register of SWD_PIN by programming the hardware configuration OB flash. Refer the sections of “[Hardware Option Byte](#)”, “[Alternate Function Select of Special Pins](#)” for more information about SWD pin setting.

2.6.3. SWD and ICP Interface Circuit

The MCU chip implements an on-chip megawin proprietary “SWD” interface to allow both standard Cortex®-M0 Serial Wire Debug (SWD) interface and megawin In-Chip-Programming (ICP) interface. The SWD and ICP share the same interface to use a clock signal (SWCLK) and a bi-directional data signal (SWDIO) to transfer information between the MCU chip and a host system. The SWD interface is able to connected to the “megawin ARM ICE Adapter” for system debugging or the “megawin ARM Writer” to do ICP programming function for In-Chip flash memory and hardware configuration. The ICE adapter also supports ICP programming function. Refer the sections of “[Hardware Option Byte Flash Memory](#)” and “[Hardware Option Byte](#)” for more information about hardware configuration.

The SWD interface allows the SWCLK (PC4) and SWDIO (PC5) pins to be shared with user functions so that In-Chip flash memory programming function could be performed. This is practicable because ICP communication is performed when the device is in the halt state, where the on-chip peripherals and user software are stalled. In this halted state, the ICP interface can safely ‘borrow’ the SWCLK and SWDIO pins. In most applications, external resistors are required to isolate ICP interface traffic from the user application. A typical isolation configuration is shown in following diagram.

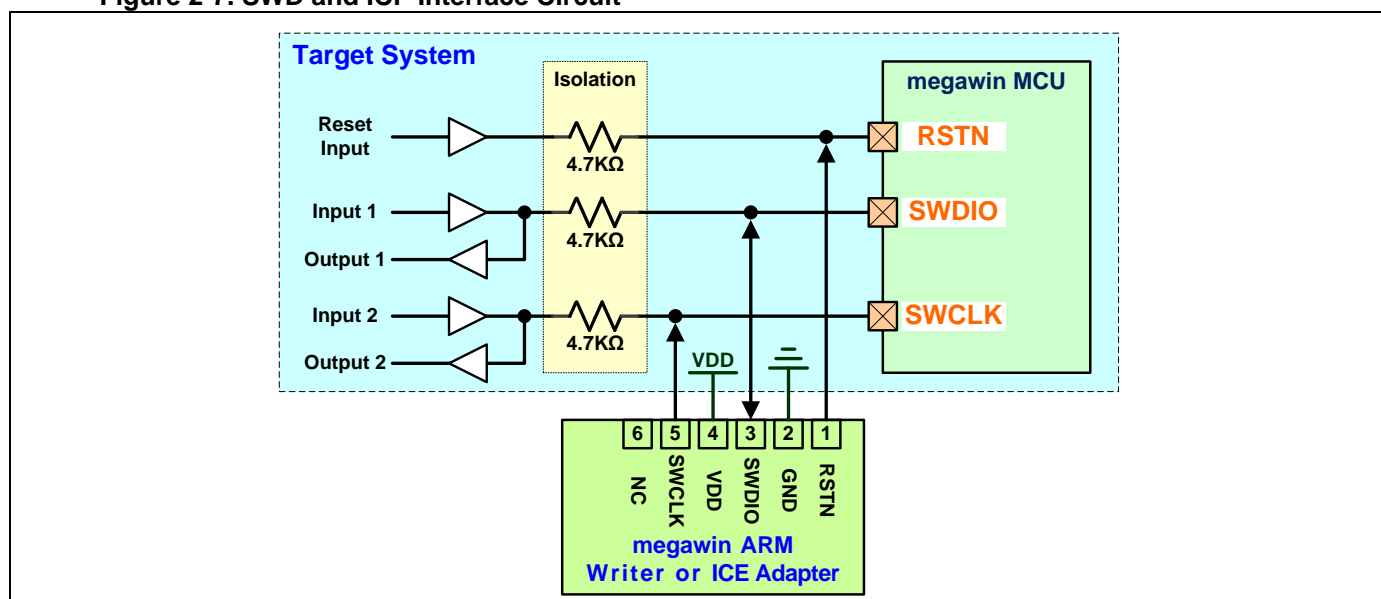
[Notify]: It is strongly recommended to build the SWD interface circuit on target system. It will reserve the whole capability for software programming and device options configured.

After power-on, the PC4 and PC5 pins are configured to SWCLK and SWDIO for SWD debugging function. This is possible because SWD communication is typically performed when the CPU is in the halt state, where the user software is stalled. In this halted state, the SWD interface can safely ‘use’ the SWCLK and SWDIO pins. As mentioned SWD interface isolation in following diagram, external resistors are required to isolate SWD interface traffic from the user application.

If user gives up the SWD function, software can configure the SWCLK and SWDIO to GPIO pins or other AFS (Alternate Function Select) IO function pins. When user would like to regain the SWD function, user can predict an event that triggers the software to switch the PC4 and PC5 pins back to SWCLK and SWDIO.

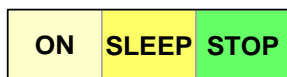
Refer the sections of “[Alternate Function Select of Special Pins](#)” for more information about SWD pin setting.

Figure 2-7. SWD and ICP Interface Circuit



3. System Power

3.1. Introduction



The module can be running in all power operation modes.

The chip power is implemented only by single power supply input and embedded one LDO to supply the internal core logic power. The chip supports one power controller (PW) to manage Power-on reset circuit, Low-voltage reset circuit, Brown-Out Detectors (BOD0/1/2), power down control and wakeup control.

It supports power-down modes: **SLEEP** mode and **STOP** modes. The power-down modes reduce chip power and provide the different power-saving scheme for chip application.

3.2. Features

- Built-in one embedded power regulator for core logic power
 - 1.5V output regulator
- Built-in three brown-out detectors – BOD0, BOD1, BOD2
- Built-in a power management controller with power-down and wakeup control
- Support three power operation modes
 - On(Normal) mode and SLEEP , STOP power down modes
- Support wake-up from SLEEP/STOP modes via multiple sources
 - Wake-up sources come from GPIO pins, IWDt, RTC, analog comparator, I2C, BODn

3.3. Implementation

3.3.1. Power Device Implementation

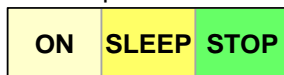
The following table is showing the implemented embedded power devices of chips.

Table 3-1. Power Device Implementation

Chip	Embedded Power Regulator		Embedded Power Detector		
	VCAP (Core LDO)	V33 (USB LDO)	BOD0 Threshold	BOD1 Threshold	BOD2 Threshold
MG32F02A128/A064 MG32F02V032	1.5V	-	1.4V	4.2V, 3.7V, 2.4V, 2.0V	1.7V
MG32F02U128/U064		3.3V			
MG32F02N128/N064 MG32F02K128/K064		-		4.2V, 3.6V, 2.4V, 2.0V	

3.4. Power Operation Mode

There are three power operation modes of **ON**, **SLEEP**, **STOP** to be supported in the power controller. The **PW_STATE** register provides the status of current operation mode for debugging.



- **ON mode**

In **ON** mode, the CPU is able running in full speed. All peripheral modules can use full power to do normally full function operation. These modules can enable or disable independent to save power consumption.

- **SLEEP mode**

In **SLEEP** mode, only the CPU is stopped and entering CPU sleep mode. All peripheral modules can be configurable to continue to operate or sleep.

In this mode, the chip can be waked up by the related interrupt or event occurs. Refer the section of [“Wakeup and Interrupt”](#) for more information.

- **STOP mode**

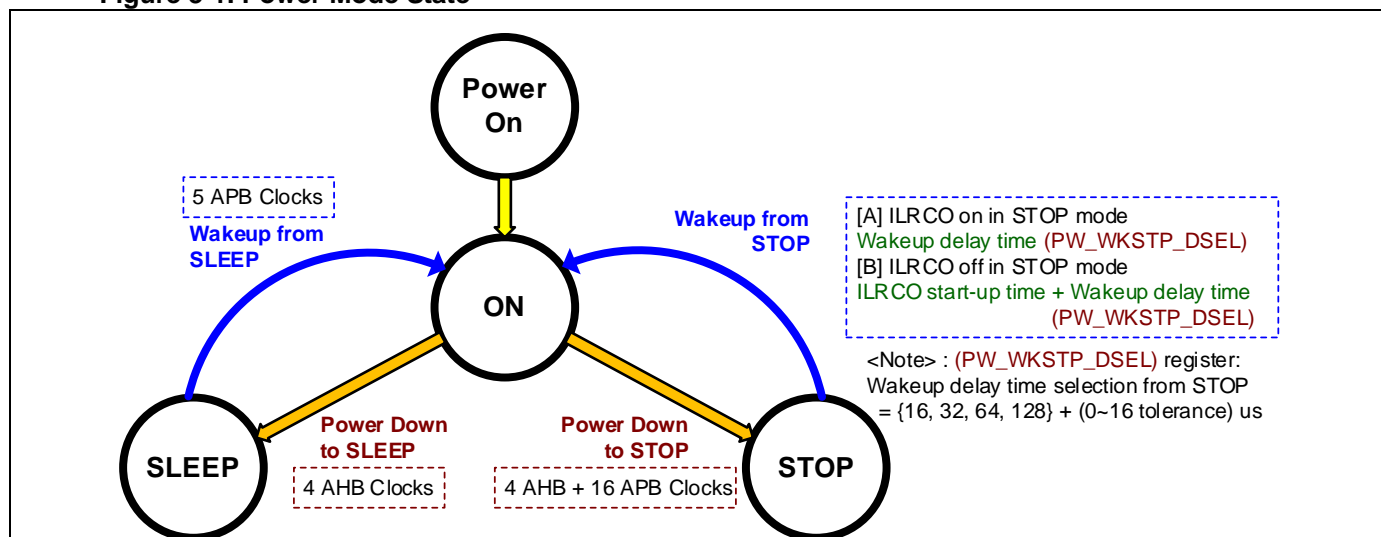
The **STOP** mode provides the lowest power consumption. The different from SLEEP mode is that CPU is entering CPU deep-sleep mode and all peripheral modules are disabled except some special modules or devices. These modules or devices can be configurable to continue to operate in STOP mode or not. They

include of IWDT, RTC, CMP modules and LVR, BOD0, BOD1, BOD2 devices. The internal voltage regulator is also running in low power mode.

Refer the section of “[Power Device Implementation](#)” about the chip support of Brown-Out Detectors.

In this mode, the chip can be waked up by some of the external input lines (GPIO) and some events detection. Refer the section of “[Wakeup and Interrupt](#)” for more information.

Figure 3-1. Power Mode State



Generally the ILRCO start-up time is about 2~30us by chip independent. Refer the sections of “[Chip Power Mode Control](#)” and “[Wakeup Control](#)” for more information about power down control and wakeup control.

The following table is showing the status of internal devices in different operation modes.

Table 3-2. Power Operation Mode

Power Operation Mode	Normal	Power-Down	
Internal Device	ON	SLEEP	STOP
Current Consumption	full	low	very low
ARM32 Cortex-M0	Normal mode	Sleep mode	Deep Sleep mode
Core LDO	full/low power (*1)	full/low power (*1)	full/low power (*2)
CPU Clock	run	stop	stop
AHB/APB/ILRCO Clock	run	run	stop
XOSC/ILRCO	normal	normal	Hardware Configure
IO Pins	normal	normal	normal
Peripheral Modules	normal	configurable	disabled except configurable IWDT, RTC, CMP, I2C, USB
Enter Power Mode		Execute WFI/WFE instruction command + SLEEPDEEP=0	Execute WFI/WFE instruction command + SLEEPDEEP=1
Wake up Events		All Interrupts or wakeup events	LVR, BOD0/BOD1/BOD2, CMP, RTC(from CK_LS, CK_UT), EXINT External interrupt pins (only level), IWDT(CK_ILRCO), I2C(slave address detection)

Note *1: LDO can select normal or low power mode by PW_LDO_ON register setting.

*2: LDO can select normal or low power mode by PW_LDO_STP register setting.

3.5. Power Supply

The chip power is implemented only by single power supply input for easy application PCB design. It is embedded one internal low dropout linear regulator (LDO) to generate about +1.5 volt voltage power VDDC for core logic power supply.

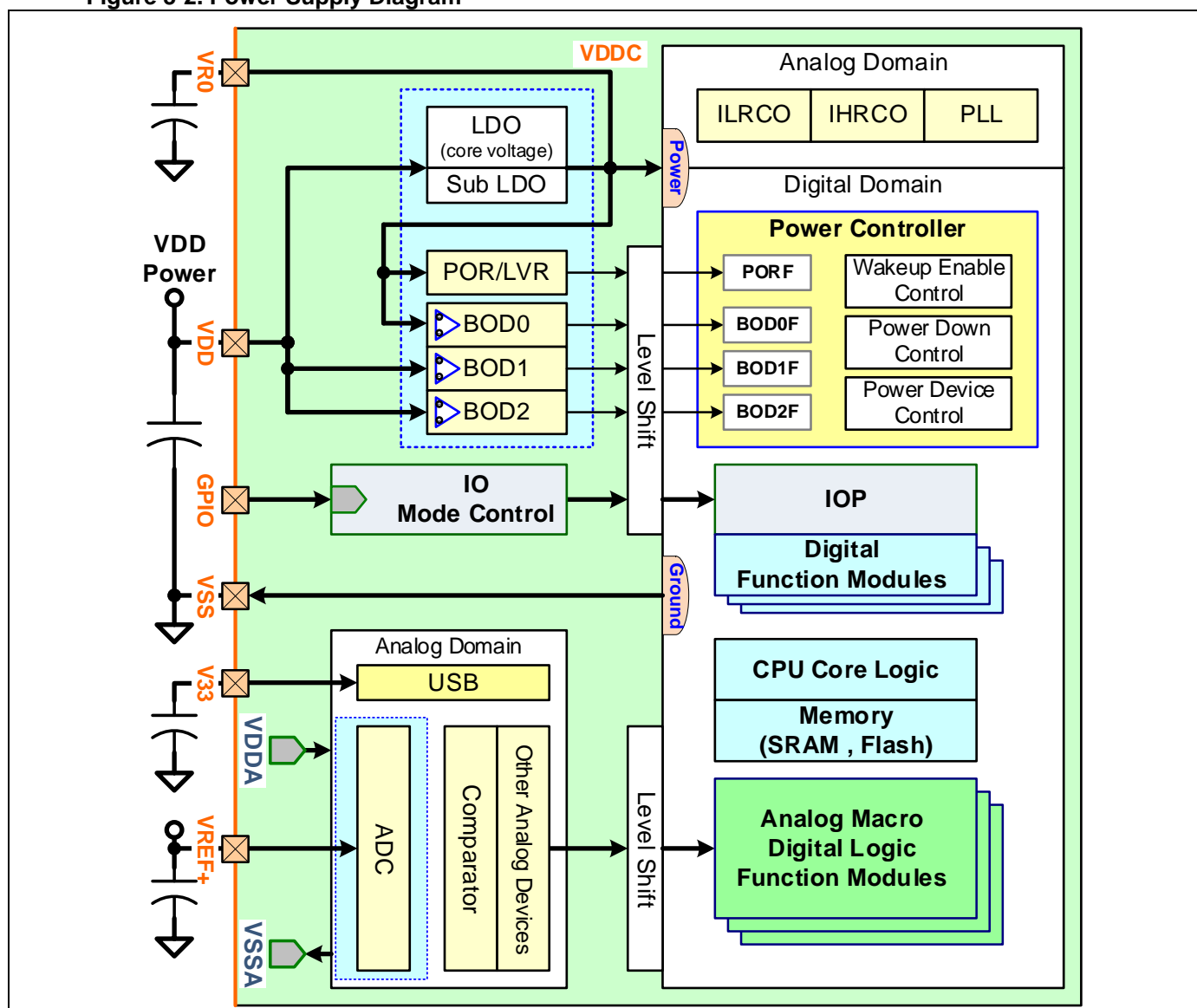
The **VDD** pin(s) is/are using for IO power supply input and internal LDO input. The **VSS** pin(s) is/are used to connect the external ground for internal reference ground of internal LDO, hard macros and digital logic. The **VCAP (VR0)** pin is the LDO output and it needs to connect bypass capacitors for normal operation. The **VREF+** pin is the input of ADC reference voltage which can connect to **VDD** pin for general application. The ADC voltage reference (**VREF+**) will be internally connected with VDD power if the chip package is without **VREF+** pin.

For **MG32F02U** series, the chip is embedded an on-chip 3.3V LDO regulator to provide the internal USB device power. The **V33** pin is the LDO output and it needs to connect bypass capacitor(s) for normal operation.

Refer the section of "[Chip Power Application Circuit](#)" for more information.

The following diagram is showing the power supplies of chip.

Figure 3-2. Power Supply Diagram

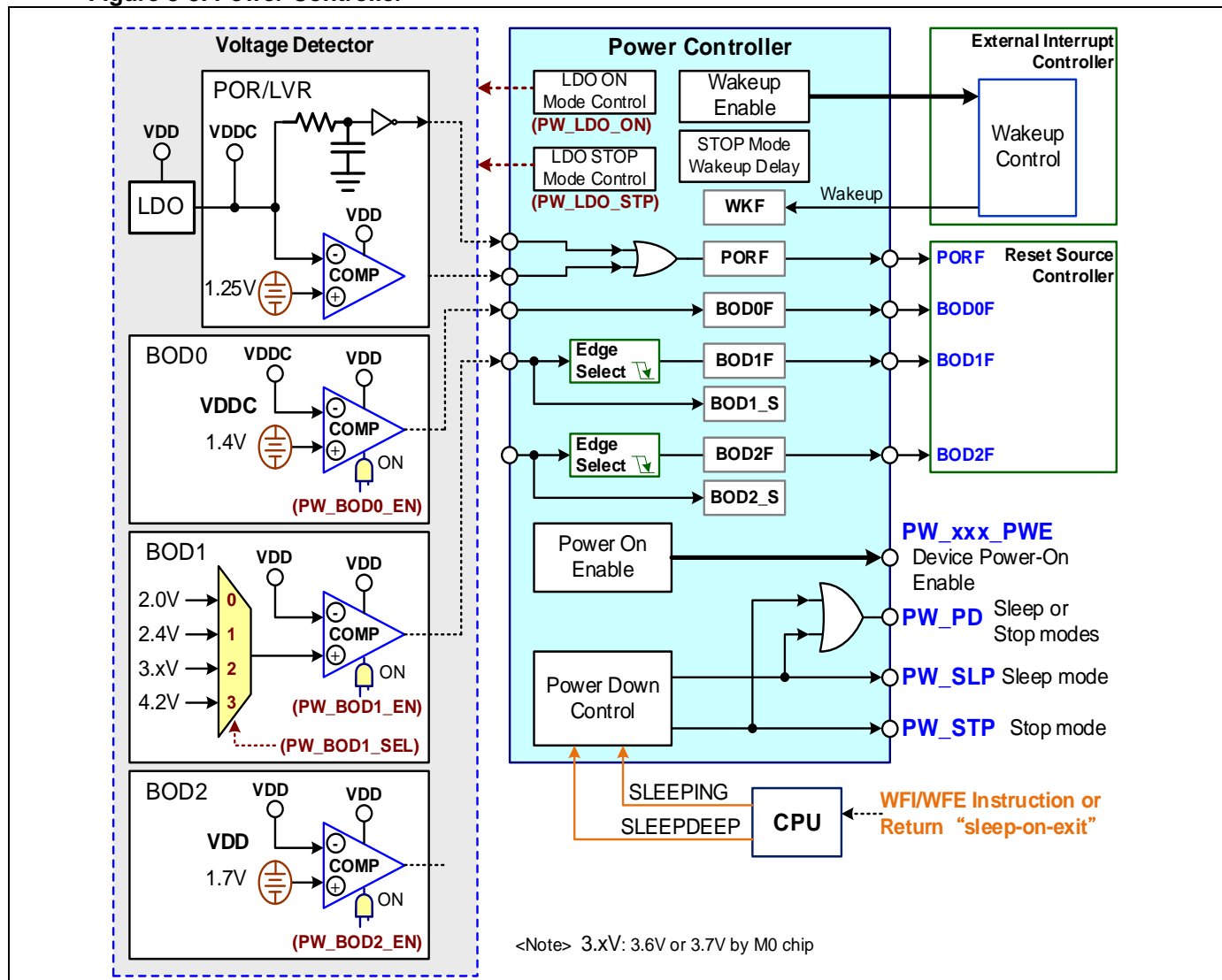


3.6. Power Controller Block

The chip supports one power controller (PW) to manage Power-on reset (POR) circuit, Low-voltage reset (LVR) circuit, Brown-Out Detectors (BOD0/1/2), power down control and wakeup control.

The following diagram is showing the Power control block.

Figure 3-3. Power Controller



3.7. Power Voltage Detect

3.7.1. POR/LVR Detector

The chip is embedded a power-on reset (POR) and a Low-voltage reset (LVR) circuits which are always active.

When the chip is powered up, the internal RC reset circuit will be active at the voltage threshold 0.7 volt (variant 0.6~0.8 volt by chip) of VDD. During power on state, the RC reset circuit needs to be restarted when VDD power drops below 0.3 volt and powers up again. Also the LVR can monitor the core power VDDC to protect chip for core logic operation. When core power voltage drops below LVR threshold, the LVR will be active until the core powers up upon 1.65 volt. The PORF flag (**PW_PORF**) is used to indicate the chip in power reset state.

3.7.2. Brown-Out Detector

In the chip, there are two or three Brown-Out Detectors (BOD0, BOD1 and BOD2) to monitor chip power. User can enable the BOD0, BOD1 or BOD2 detector independently in **PW_BOD0_EN**, **PW_BOD1_EN** or

PW_BOD2_EN register. Refer the section of “[Power Device Implementation](#)” about the chip support of Brown-Out Detectors.

BOD0 services the fixed detection level at core power $V_{DDC}=1.4V$. It uses to monitor the core power for internal flash memory write protection. When the core power voltage is below the BOD0 voltage threshold, it means the flash write function will be failed under this condition. Associated flag **PW_BOD0F** is set when BOD0 meets the detection threshold level. BOD0 provides the capability to interrupt CPU or to reset CPU by software configured.

If **SYS_IEA** is enabled in SYS module and **PW_BOD0_IE** is enabled in PW module, the BOD0F flag asserted event will generate a system flag interrupt. It can interrupt CPU either CPU in **ON** mode or **SLEEP** mode. The detect event also wakes up CPU in **STOP** modes if **PW_WKSTP_BOD0** is enabled.

In the chip also there is an extra Brown-Out Detector (BOD1) to monitor VDD power. They are same control as BOD0 for **PW_BOD1_IE** and **PW_WKSTP_BOD1** registers. It is different from BOD0 that BOD1 provides the programmable detection level at $V_{DD}=2.0 \sim 4.2V$ by setting **PW_BOD1_TH** register. Another one is that user can select the trigger edge of BOD1 output from rising edge, falling edge or dual edge for interrupt by setting **PW_BOD1_TRGS** register. Also it provides a status bit of **PW_BOD1_S** for the BOD1 comparator output.

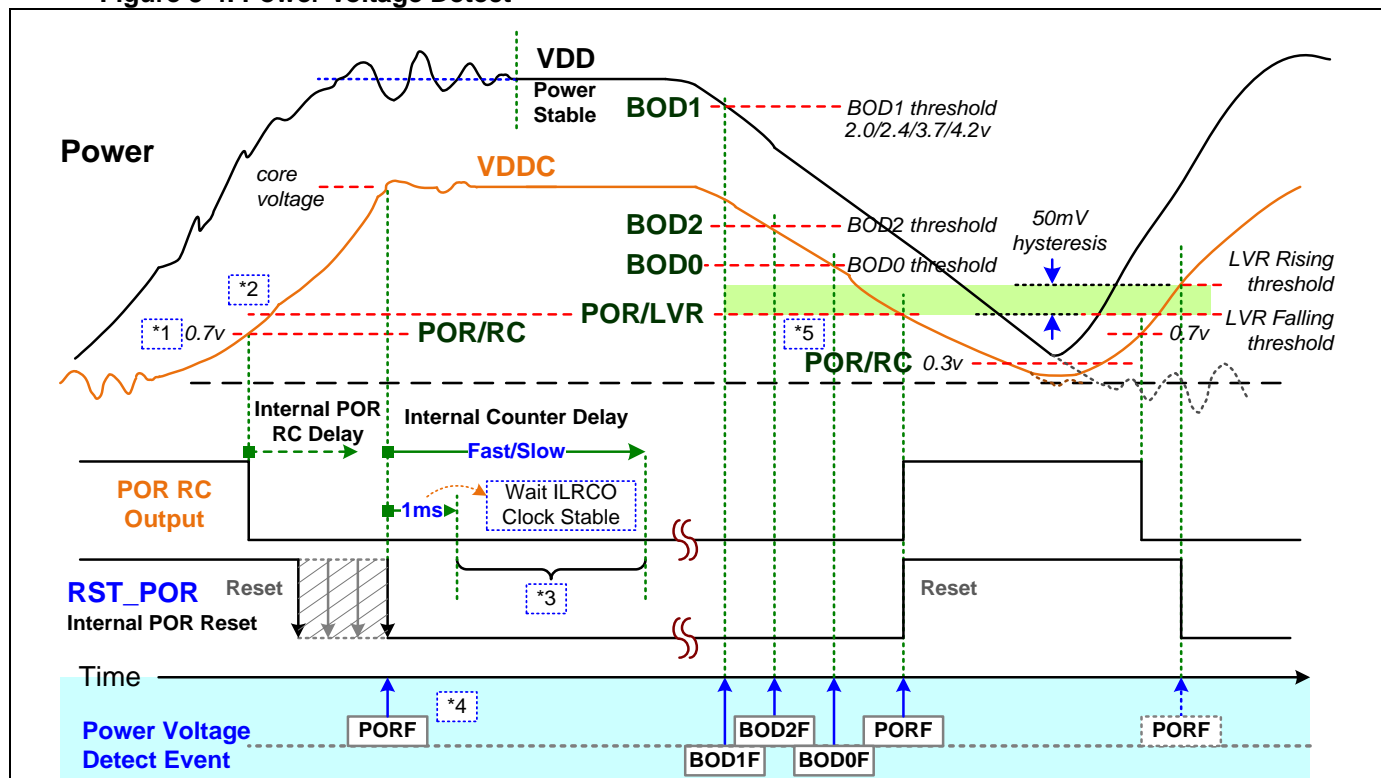
One another Brown-Out Detector (BOD2) is also used to monitor VDD power. They are same control as BOD1 for **PW_BOD2_IE** and **PW_WKSTP_BOD2** registers. It is different from BOD1 that BOD2 provides the fixed detection level at $V_{DD}=1.7V$. Another one is that user can select the trigger edge of BOD2 output from rising edge, falling edge or dual edge for interrupt by setting **PW_BOD2_TRGS** register. Also it provides a status bit of **PW_BOD2_S** for the BOD2 comparator output.

3.7.3. Power Voltage Detection Threshold

The following diagrams are showing the power voltage detection threshold during power up or power down cycle.

There is a time delay about 32ms or 4ms after POR reset for internal hardware configuration. The time delay is set by the hardware configuration **OB**. Refer the section of “[Power-On Reset and Chip Reset Timing](#)” in Reset chapter for more detail information. The power-on time start from POR reset start to CPU working is about 36ms or 8ms. The power-on time before the POR reset start is based on system power design.

Figure 3-4. Power Voltage Detect



<Note-1> : The POR/RC input buffer voltage threshold is not trimming and variant 0.6V~0.8V by chip.

<Note-2> : The POR/LVR voltage threshold is set by hardware default after power-on reset. Initial other power/clock devices.

<Note-3> : The POR/LVR voltage threshold is set by load OB (manufacture trimming) after internal reset of RST_POR/RST_C.

<Note-4> : The (**PW_PORF**) flag is set during power-up or VDD power drops below POR/LVR threshold. The flag must clear by firmware.

<Note-5> : The POR/RC circuit restart threshold low voltage is 0.3v. The power voltage needs drop down below it and restart POR.

The following table is showing the voltage detect threshold of chips. The hysteresis of LVR detect threshold voltage is about 50mV.

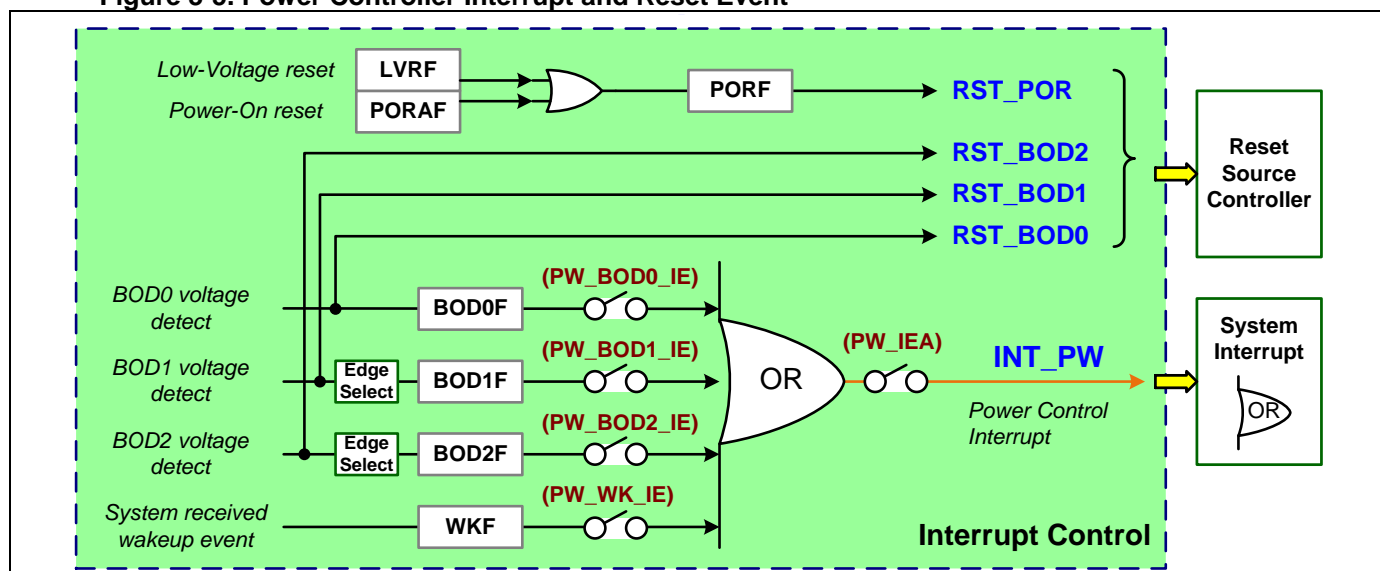
Table 3-3. Voltage Detect Threshold

Chip	VR0/VCAP Voltage (volt)	BOD0 Threshold (volt)	BOD1 Threshold (volt)	BOD2 Threshold (volt)	LVR Falling Threshold (volt)	LVR Rising Threshold (volt)
MG32F02A128/A064 MG32F02U128/U064 MG32F02V032	1.6V	1.4	2.0/2.4/3.7/4.2	1.7	1.25	1.3
MG32F02K128/K064 MG32F02N128/N064	1.6V	1.4	2.0/2.4/3.6/4.2	1.7	1.25	1.3

3.8. Interrupt and Reset

There are several signals of **INT_PW**, **RST_POR**, **RST_BOD0** and **RST_BOD1** to be generated in this PW control module. **INT_PW** sends to External Interrupt Controller (EXIC) to do as an interrupt event. **RST_POR**, **RST_BOD0**, **RST_BOD1** and **RST_BOD2** send to Reset Source Controller to do as a reset event.

Figure 3-5. Power Controller Interrupt and Reset Event



3.8.1. PW Interrupt Flags

The interrupt of **INT_PW** is OR with the voltage detection flags of BOD0, BOD1, BOD2 and wakeup event flag. It is outputted as one of the interrupt events of the system interrupt of **INT_SYS**. Refer the section of “[Interrupt and Event](#)” in the chapter of System Common Control for more information about system interrupt.

These interrupt flags are using for the interrupt service routine (ISR) flow control. Generally these flags are set by hardware and software clears them when the serviced job is completed in the related ISR. Each interrupt flag has one interrupt enable bit. User can enable or disable it. There is one interrupt all enable bit of **PW_IEA** to enable or disable all the interrupt sources for this module.

Generally these interrupt flags are set by hardware and clear by software writing 1. Refer the register descriptions for more information about the related interrupt flags and interrupt enable bits.

- **BOD0F**

BOD0 brown-out detection interrupt flag (**PW_BOD0F**). There is a related interrupt enable register bit of **PW_BOD0_IE**. The flag is active when the power voltage is lower than BOD0 threshold.

- **BOD1F**

BOD1 brown-out detection interrupt flag (**PW_BOD1F**). There is a related interrupt enable register bit of **PW_BOD1_IE**.

- **BOD2F**

BOD2 brown-out detection interrupt flag (**PW_BOD2F**). There is a related interrupt enable register bit of **PW_BOD2_IE**.

- **WKF**

System received wakeup event flag (**PW_WKF**). There is a related interrupt enable register bit of **PW_WK_IE**.

3.8.2. PW Reset Events

RST_POR, **RST_BOD0** and **RST_BOD1** signals send to Reset Source Controller (RST) to do as the warm reset events or cold reset events. These reset events can be enabled to reset the chip by setting the registers in RST.

Refer the descriptions of System Reset chapter for more information about the reset events and control.

- **PORF**

Power-On reset status flag (**PW_PORF**). The PORF flag is set during power-up or VDD power drops below POR/LVR threshold. Usually this flag can clear by firmware after power-on and use to indicate that the firmware flow is first time loop from power-on.

- **BOD0F, BOD1F, BOD2F**

These three flags are BOD0, BOD1 and BOD2 brown-out detection interrupt flag (**PW_BOD0F**, **PW_BOD1F** and **PW_BOD2F**).

3.9. Register Protect and Lock

After Power controller domain reset, all PW registers are write-access protected except **PW_STA**, **PW_KEY** registers. Write **0xA217** value to the **PW_KEY** register to unprotect the registers and process the control continuously. Oppositely write other value except **0xA217** value to protect the registers. Read the **PW_KEY** register to get the register value is Protected (=1) or Unprotected (=0).

Refer the table and descriptions of “[Register Protect and Lock](#)” in System Reset chapter for more information.

Special for hardware function control, the register of **PW_WKSTP_IWDT** is locked control by the **IWDT_LOCK** register. Another, the register of **PW_WKSTP_RTC** is locked control by the **RTC_LOCK** register. Refer the “Register Protect and Lock” in IWDT and RTC chapter for more information.

3.10. Chip Power Mode Control

There are three power operation modes of **ON**, **SLEEP**, **STOP** to be supported in the power controller. Refer the section of “[Power Operation Mode](#)” for more detail information.

3.10.1. CPU Power Down

For chip entering power down mode, the firmware must execute WFI or WFE instruction to force the CPU enters sleep mode or deep sleep mode. Then the chip will enter the power down mode of **SLEEP** or **STOP**. User can configure the CPU sleep mode by setting CPU register of **SLEEPDEEP** after firmware executes WFI or WFE instruction. Refer to the “Cortex-M0 Technical Reference Manual” for more information.

The following table is showing the CPU power down entering and wakeup conditions.

Table 3-4. Sleep and DeepSleep Entering

Event	SLEEPDEEP=0	SLEEPDEEP=1	Wakeup Conditions Description
WFE execution	CPU Sleep Wait Interrupts or Events	CPU DeepSleep Wait Interrupts or Events	Enabled Interrupts or all Interrupts(SEVONPEND=1) request, External event request (RXEV), Debug request event, Sleep mode reset (CPU reset), Event register=1 (not enter CPU Sleep or DeepSleep; interrupts or events have happened before WFE execution)
WFI execution Sleep-on-exit execution	CPU Sleep Wait Interrupts	CPU DeepSleep Wait Interrupts	Enabled Interrupt request, Debug request event, Sleep mode reset (CPU reset),

<Note-1> WFE(Wait for Event) / WFI(Wait for Interrupt) : CPU instructions

<Note-2> Sleep-on-exit: CPU SLEEPONEXIT bit =1, indicates CPU entering sleep when returning from Handle to Thread mode.

<Note-3> SLEEPDEEP: CPU SLEEPDEEP bit, 0= Sleep mode, 1=Deep sleep mode.

The following table is showing the relationship between CPU and system chip about power down modes.

Table 3-5. Power Mode Selection

Chip Power Mode	CPU	CPU Register SLEEPDEEP	Status
ON	Run	x	Normal
SLEEP	sleep	0	All modules can operate by configuration.
STOP	deep sleep	1	All modules are disabled except configurable IWDG, RTC, and CMP. The voltage regulator can be set normal or low power mode.

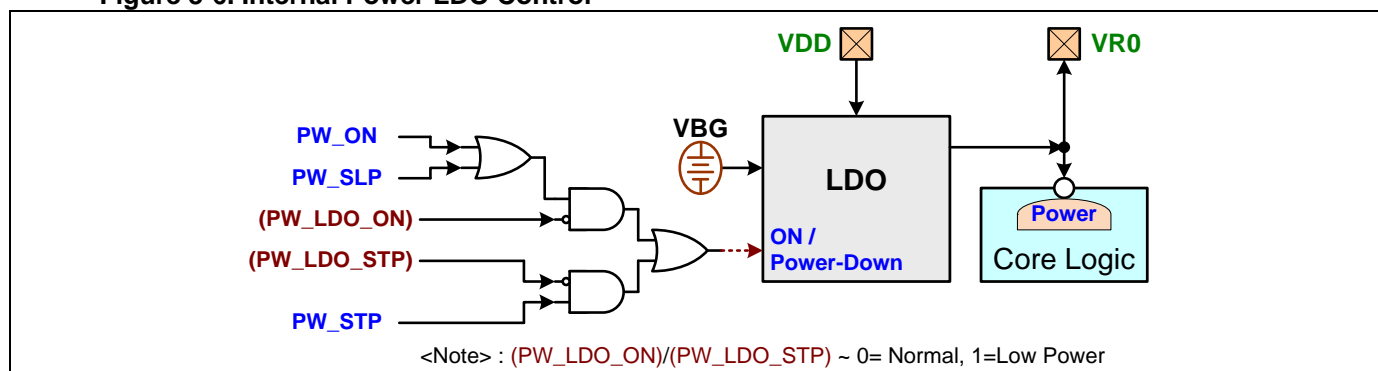
<Note> Enter SLEEP or STOP mode after execute WFI or WFE instruction command.

3.10.2. Internal Devices Control

❖ Internal Power LDO

The chip is embedded one internal low dropout linear regulator (LDO) to supply the internal core logic power. The LDO has two operation mode of normal and lower power. User can set the LDO operation mode in **PW_LDO_ON** register for **ON** mode and **SLEEP** mode. Also user can plan to set it in **PW_LDO_STP** register beforehand entering **STOP** mode. Refer the section of “[Device Power Enable Control in Power-Down Modes](#)” for more information about power enable control of other internal devices.

Figure 3-6. Internal Power LDO Control



For normal mode, the LDO is able to supply core power about maximum 30mA. For low power mode, the LDO is able to supply core power about maximum 0.5mA. User can select lower power mode for power saving when core power of the chip is operation below 0.5mA.

For MG32F02N128/N064/K128/K064, the LDO has extra lowest power mode. User can set the **PW_LCTL_SLP** and **PW_LCTL_STP** registers to configure LDO lowest power mode for **SLEEP** mode and **STOP** mode as following table.

Table 3-6. LDO Control

Power Mode	LDO Mode	PW Register			
		PW_LDO_ON	PW_LCTL_SLP	PW_LDO_STP	PW_LCTL_STP
ON	Normal	0	x	x	x
	Low	1	x	x	x
SLEEP	Normal	0	x	x	x
	Low	1	0	x	x
	Lowest	1	1	x	x
STOP	Normal	x	x	0	x
	Low	x	x	1	0
	Lowest	x	x	1	1

<Sign> x : Don't care

<Note> PW_LCTL_SLP/STP are only supported for MG32F02N128/N064/K128/K064.

❖ Internal voltage reference

The chip is built in an internal voltage reference (VBUF) source for ADC and analog comparator analog part. User need to enable the IVR for ADC or analog comparator operation in **PW_IVR_EN** register.

3.10.3. Device Power Enable Control in Power-Down Modes

The PW controller supports the power control planning ability for the internal devices in **SLEEP** or **STOP** modes. User can plan the internal devices are independently power-on or power-off beforehand entering **SLEEP** or **STOP** modes. When the CPU enters sleep mode or deep sleep mode by firmware execution WFI or WFE instruction, the chip will enter the power down mode of **SLEEP** or **STOP**.

The following table is showing the power control of internal devices in different operation modes.

Table 3-7. Internal Device Power Control

Internal Device	ON	SLEEP	STOP
Core LDO	on (normal or low-power)	on (normal or low-power)	on (normal or low-power)
Power-On Reset	on	on	PW Reg presetting
BOD0 Detector	Reg setting	Reg setting	PW Reg presetting
BOD1/2 Detector	Reg setting	Reg setting	PW Reg presetting
ILRCO/XOSC	Reg setting	Reg setting	Hardware setting (*1)
IHRCO/PLL	Reg setting	Reg setting	off
Analog Comparator	Reg setting	PW Reg presetting	PW Reg presetting
LCD	Reg setting	PW Reg presetting	Reg setting
OPA	Reg setting	PW Reg presetting	PW Reg presetting
USB	Reg setting	PW Reg presetting	PW Reg presetting

Note *1: Hardware auto set on/off by ILRCO or XOSC clock using

- POR

User can plan the power enable control of POR in **STOP** mode by presetting the **PW_STP_POR** register.

- BOD0 / BOD1 / BOD2

User can plan the power enable control of BOD0, BOD1 and BOD2 independently in **STOP** mode by presetting the **PW_STP_BOD0**, **PW_STP_BOD1** and **PW_STP_BOD2** registers.

- Analog Comparators

User can plan the power enable control of analog comparators independently in **SLEEP** mode by presetting the **PW_SLP_CMPn** registers. Also user can plan that in **STOP** mode by presetting the **PW_STP_CMPn**

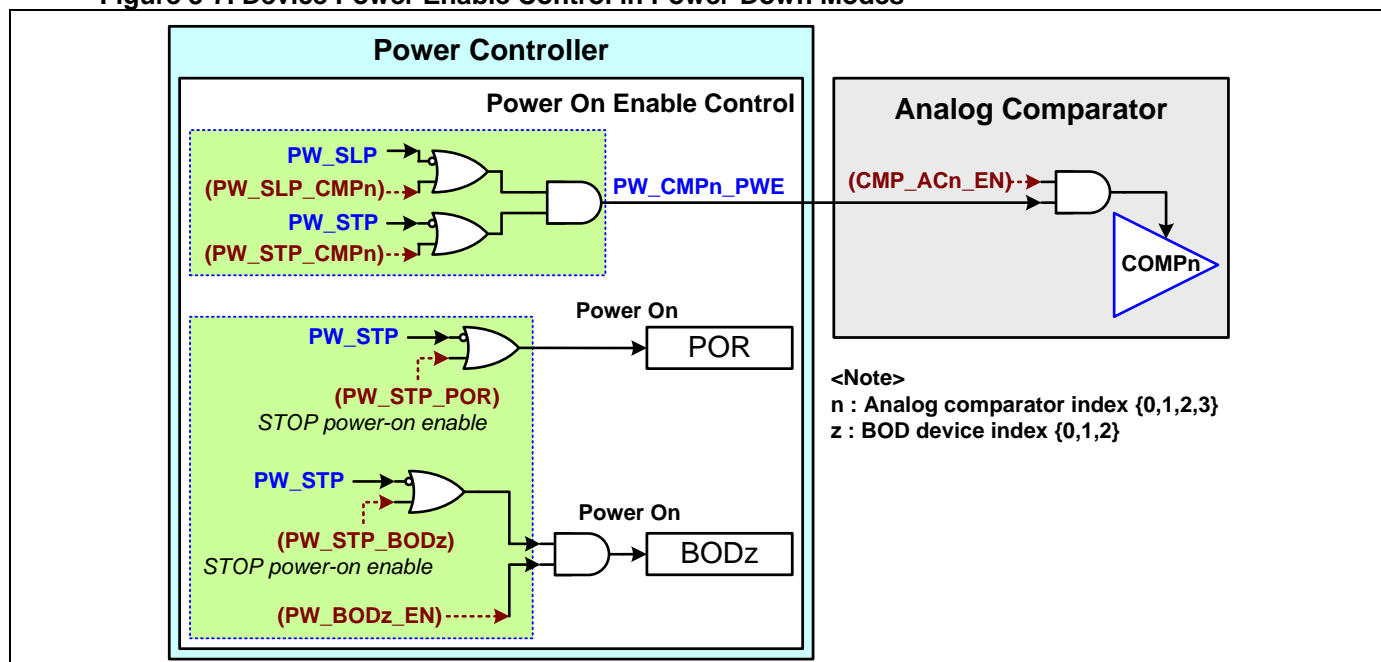
registers. (n = {0, 1, 2, 3})

- **USB**

User can plan the power enable control of USB macro independently in **SLEEP** mode by presetting the **PW_SLP_USB** registers. Also user can plan that in **STOP** mode by presetting the **PW_STP_USB** registers.

The following diagram is showing the power enable control of internal devices.

Figure 3-7. Device Power Enable Control in Power-Down Modes



The following table is showing internal device power state by different power mode and related power control registers' setting.

Table 3-8. Device Power-On Control

Power Mode	Signal			Device Support in SLEEP/STOP	Device Power-On Enable Register	Power-On Enable in SLEEP	Power-On Enable in STOP	Device State
	PW_SLP	PW_STP	PW_PD		PW_BODn_EN	PW_SLP_CMPn PW_SLP_USB	PW_STP_zzz	
ON	0	0	0	x	1	x	x	Off
				x	0	x	x	On
SLEEP	1	0	1	x	1	x	x	Off
				Yes	0	0	x	Off
				Yes	0	1	x	On
STOP	0	1	1	No	x	x	x	Off
				Yes	1	x	x	Off
				Yes	0	x	0	Off
				Yes	0	x	1	On

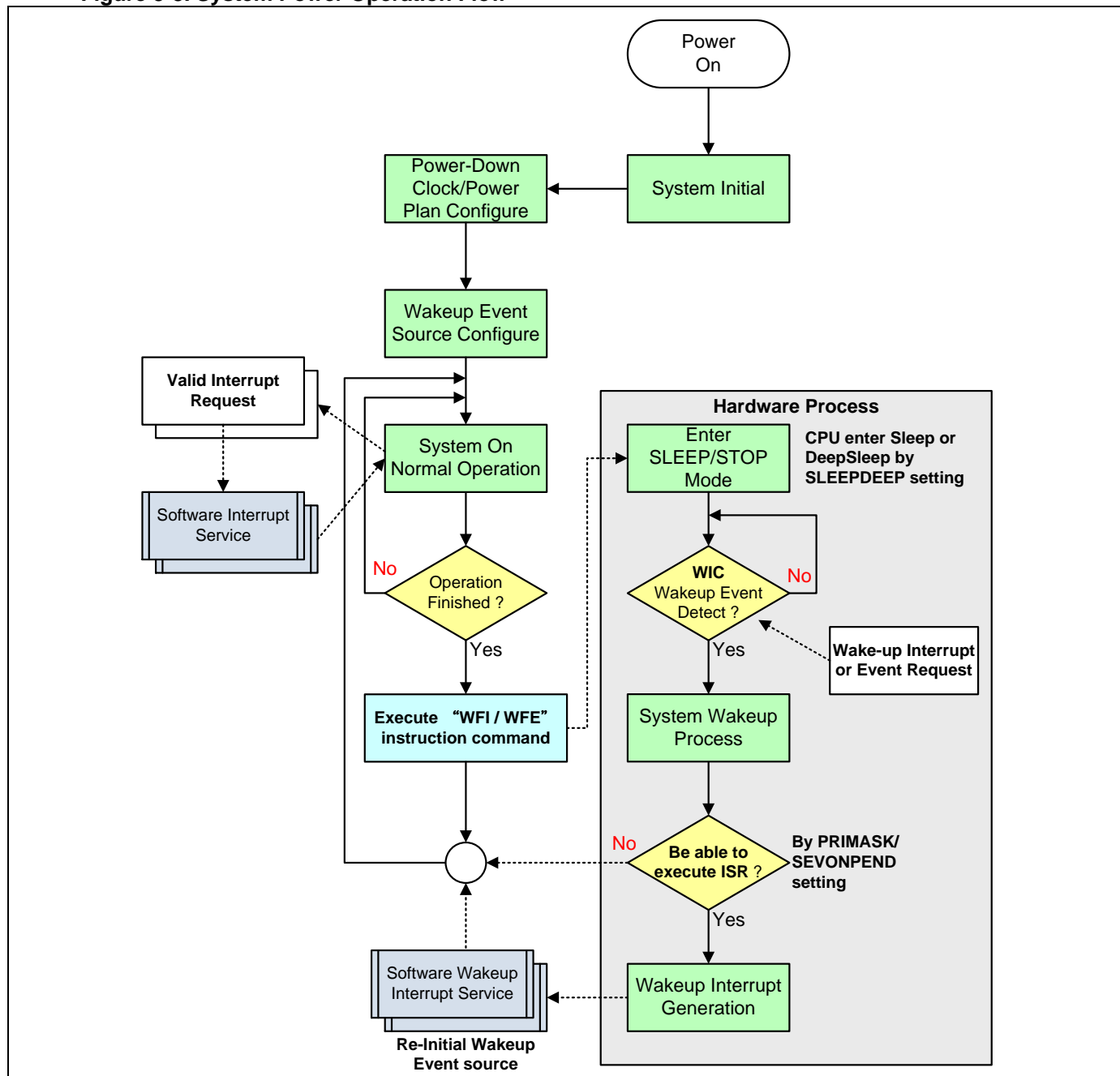
<Sign> x: Don't care, n: Device index, zzz: Device Name {BOD0,BOD1,BOD2, ...}

3.10.4. System Power Operation Flow

When CPU executes the **WFI** or **WFE** command, the chip is entering hardware process for going **SLEEP** or **STOP** mode. Also the CPU is entering Sleep or DeepSleep state until the chip detects a wakeup event. The chip will be waking up and CPU will run the next instruction of previously executed **WFI** or **WFE**. Refer the section of "[Wakeup Control](#)" for more information about chip wakeup control.

The following figure is showing the system power operation flow chart.

Figure 3-8. System Power Operation Flow



3.11. Wakeup Control

The PW controller supports programmable and flexible wakeup source control from **SLEEP** or **STOP** modes. User needs to plan the wakeup source beforehand entering **SLEEP** or **STOP** modes. Refer the section of [“RXEV and TXEV Control”](#) of Interrupt chapter for more information about the RXEV wakeup function.

3.11.1. Wakeup Event Sources

For **SLEEP** mode, all the interrupt events are able to do as the wakeup events as long as the related interrupt enable bits of module and NVIC are enabled. All the GPIO pins which are only PA/PB/PC/PD port and can configure as external interrupt are can do as the **SLEEP** wakeup source. The chip can detect edge or level trigger of input signal in EXIC module setting.

For **STOP** mode, the chip provides the wakeup sources of BOD0, BOD1, BOD2, analog comparators, RTC, IWDt, I2C, SWD debugging and GPIO pin trigger.

- **BOD0 / BOD1 / BOD2**

User can enable to wake up by BOD0/BOD1/BOD2 voltage detection from STOP mode in **PW_WKSTP_BOD0**, **PW_WKSTP_BOD1** and **PW_WKSTP_BOD2** registers.

- **SWD/Debug**

User can enable to wake up by SWD debugging communication from STOP mode in **PW_WKSTP_DBG** register.

- **GPIO**

User can enable to wake up by only level trigger detection of external interrupt GPIO pins from STOP mode in EXIC interrupt enable registers. Refer the “Wakeup Control” function of [“EXIC Interrupt Control”](#) section in Interrupt chapter for more information about the wakeup function.

- **Analog Comparators**

User can enable to wake up by analog comparator CMPn voltage detection independently from STOP mode in **PW_WKSTP_CMPn** registers. (n = {0, 1, 2, 3})

- **RTC**

User can enable to wake up by RTC module events from STOP mode in **PW_WKSTP_RTC** register.

- **IWDt**

User can enable to wake up by IWDt module events from STOP mode in **PW_WKSTP_IWDt** register.

- **I2C**

User can enable to wake up by I2C slave address detection from STOP mode in **PW_WKSTP_I2Cx** registers. (x = module index)

- **USB**

User can enable to wake up by USB bus state detection from STOP mode in **PW_WKSTP_USB** registers.

The following table is showing the wakeup event source in **SLEEP** and **STOP** operation modes.

Table 3-9. Wakeup Events Source

Internal Device	SLEEP		STOP		Comment
	Wakeup Enable	Wakeup Event	Wakeup Enable	Wakeup Event	
BOD0	interrupt enable setting	enabled interrupt event	PW Reg presetting	BOD0 event detection	
BOD1/2	interrupt enable setting	enabled interrupt event	PW Reg presetting	BOD1/2 event detection	
SWD/Debug	interrupt enable setting	enabled interrupt event	PW Reg presetting	SWD communication detection	
Ext INT pins	interrupt enable setting	enabled interrupt event (detect edge/level)	interrupt enable setting	enabled Interrupt event (detect level only)	
Analog Comparator	interrupt enable setting	enabled interrupt event	PW Reg presetting	Comparator output state change detection	
RTC	interrupt enable setting	enabled interrupt event	PW Reg presetting	RTC events detection	CK_LS,CK_UT in STOP
IWDt	interrupt enable setting	enabled interrupt event	PW Reg presetting	IWDt events detection	CK_ILRCO in STOP
I2Cx	interrupt enable	enabled interrupt	PW Reg presetting	Slave address	

	setting	event		detection only	
USB	interrupt enable setting	enabled interrupt event	PW Reg presetting	USB Rest/Resume	
Other Modules	interrupt enable setting	enabled interrupt event	Not support STOP wakeup		Not support STOP wakeup

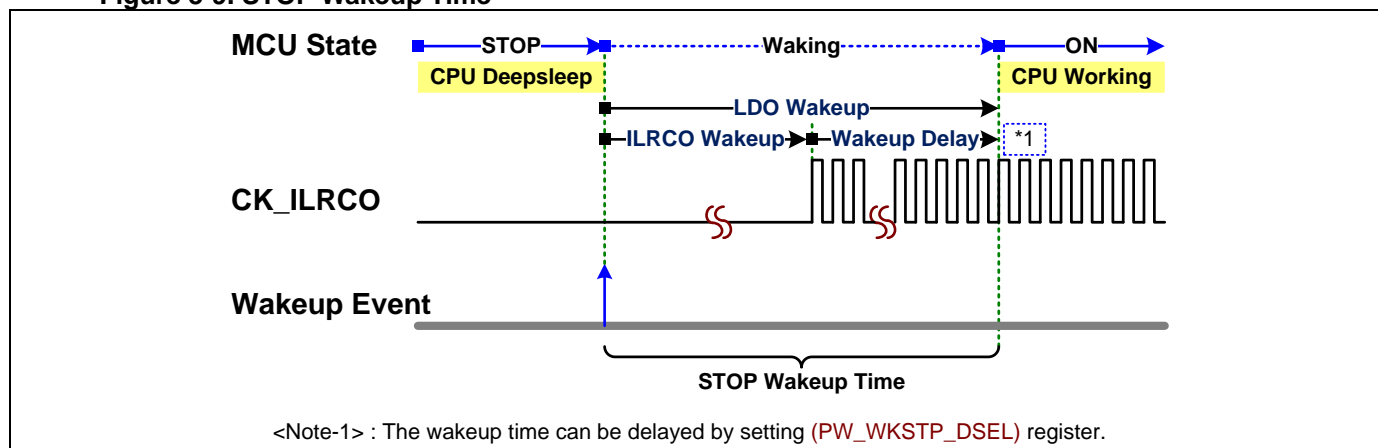
3.11.2. Wakeup and Interrupt

After wakeup sources are configured and the chip has entered power down mode by firmware execution **WFI** or **WFE** instruction, the chip will monitor these enabled wakeup source. PW controller will control these signals except external interrupt trigger events that can pass or inhibit to send to EXIC controller for wakeup control.

EXIC connects the wakeup signals to NIVC by through interrupt path to trigger wakeup process. When the WIC (wakeup interrupt controller) receives the valid wakeup signal, it will recover the clock system and wake up CPU. User can select the wakeup delayed time from 45~150us for steadying clock from **STOP** mode by setting **PW_WKSTP_DSEL** register. Also user can set the wakeup mode for steadying clock from **SLEEP** mode by setting **PW_WKSLP_MDS** register. When selects 'Normal', the MCU wakeup from **SLEEP** mode is about 5 AHB clock and MCU current consumption is normal in SLEEP mode. When selects 'Low Power', the MCU wakeup from **SLEEP** mode is slower (>20us) but MCU current consumption is lower in SLEEP mode.

The following diagram is showing the wakeup time diagram from **STOP** mode. When the ILRCO is not disabled during **STOP** mode for RTC, IWDG or others working, the ILRCO wakeup time is not necessary.

Figure 3-9. STOP Wakeup Time



The following table is showing the CPU wakeup conditions by the CPU instructions of WFI and WFE. Refer to the "Cortex-M0 Technical Reference Manual" for more information

Table 3-10. WFI and WFE Action for Wakeup and ISR

Execution	PRIMASK	SEVONPEND	Priority of interrupt	Wakeup	Request ISR Execution
WFI Sleep-on-exit	0	x	requested interrupt > current interrupt (*4)	yes	yes
			requested interrupt ≤ current interrupt	no	no
	1	x	requested interrupt > current interrupt (*5)	yes	no
			requested interrupt ≤ current interrupt	no	no
WFE	0	0	requested interrupt > current interrupt (*4)	yes	yes
			requested interrupt ≤ current interrupt	no	no
	0	1	requested interrupt > current interrupt (*4)	yes	yes
			requested interrupt ≤ current interrupt (*6)	yes	after current ISR exit
	1	0	requested interrupt > current interrupt	no	no
			requested interrupt ≤ current interrupt	no	no
	1	1	requested interrupt > current interrupt (*5)	yes	no
			requested interrupt ≤ current interrupt (*7)	yes	no

<Note-1> x: Don't care

<Note-2> PRIMASK: CPU interrupt mask register bit, 1= prevents the activation of all exceptions with configurable priority.

<Note-3> SEVONPEND: CPU SEVONPEND bit, 0=only enabled interrupts or events can wakeup CPU

1=only enabled events and all interrupt can wakeup CPU

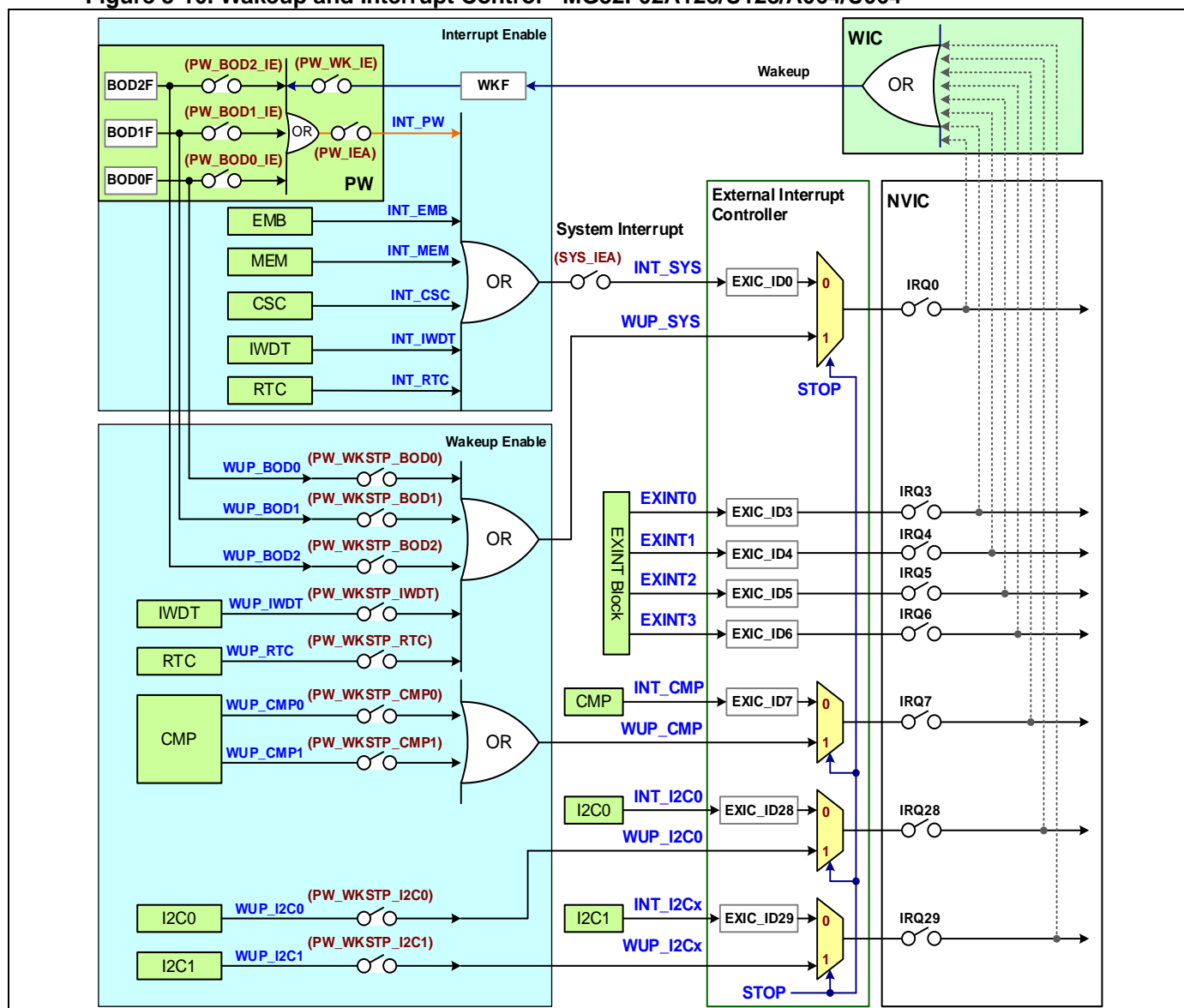
- <Note-4> wakeup and execute requested interrupt ISR
- <Note-5> wakeup and return thread mode to execute next instruction
- <Note-6> wakeup and return to next instruction of current interrupt ISR, then execute requested interrupt ISR
- <Note-7> wakeup and return to next instruction of current interrupt ISR, then return thread mode to execute next instruction

When the chip was waked up from power-down mode, the WKF flag will be asserted and user will be notified by interrupt if the **PW_WK_IE** bit is enabled. User can get the information which the system wakeup from which power-down mode (**SLEEP** or **STOP**) from **PW_WKMODE** register.

❖ MG32F02A128/U128/A064/U064

The following diagram is showing the system wakeup and interrupt control connection diagram for MG32F02A128/U128/A064/U064.

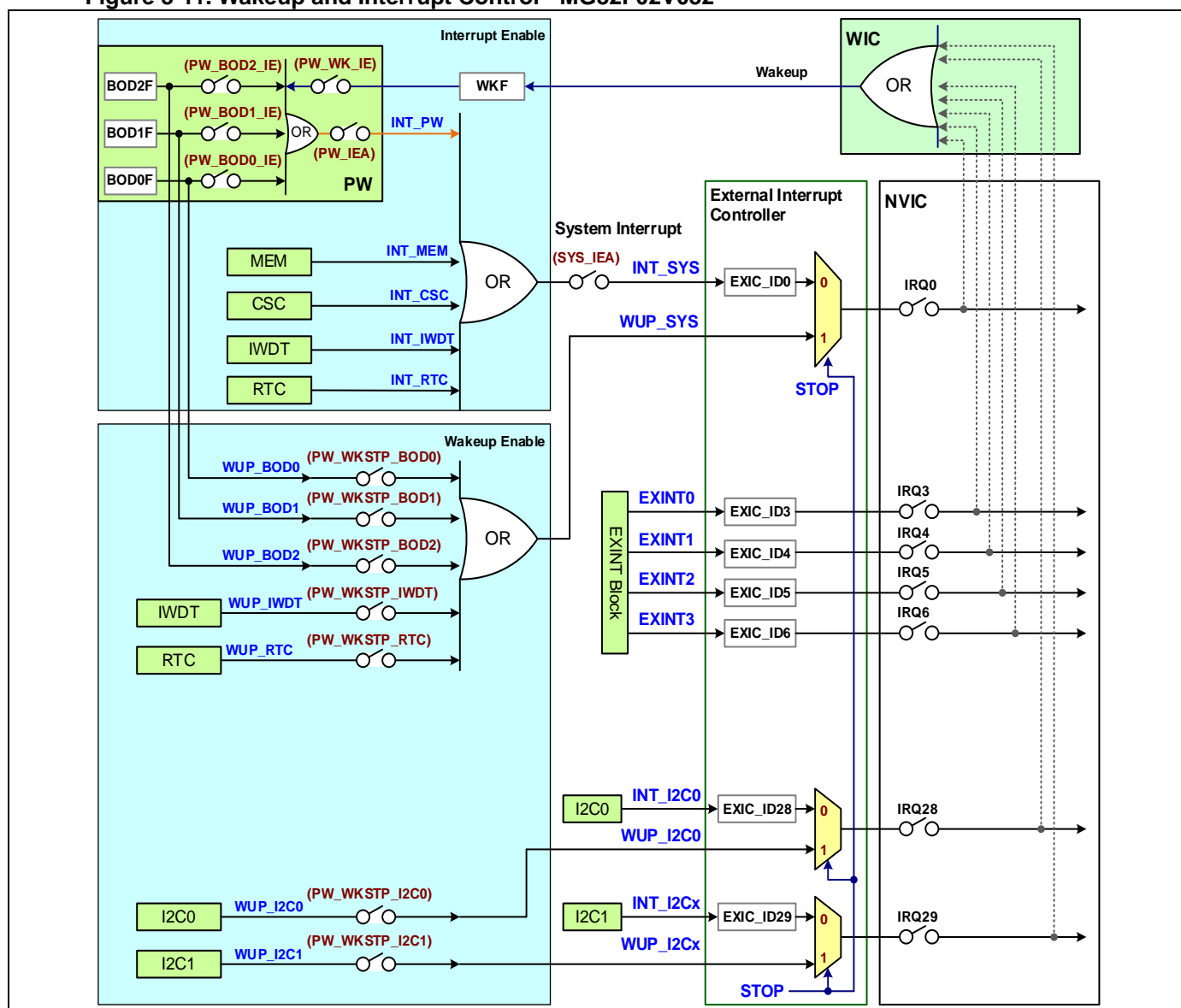
Figure 3-10. Wakeup and Interrupt Control - MG32F02A128/U128/A064/U064



❖ MG32F02V032

The following diagram is showing the system wakeup and interrupt control connection diagram for MG32F02V032.

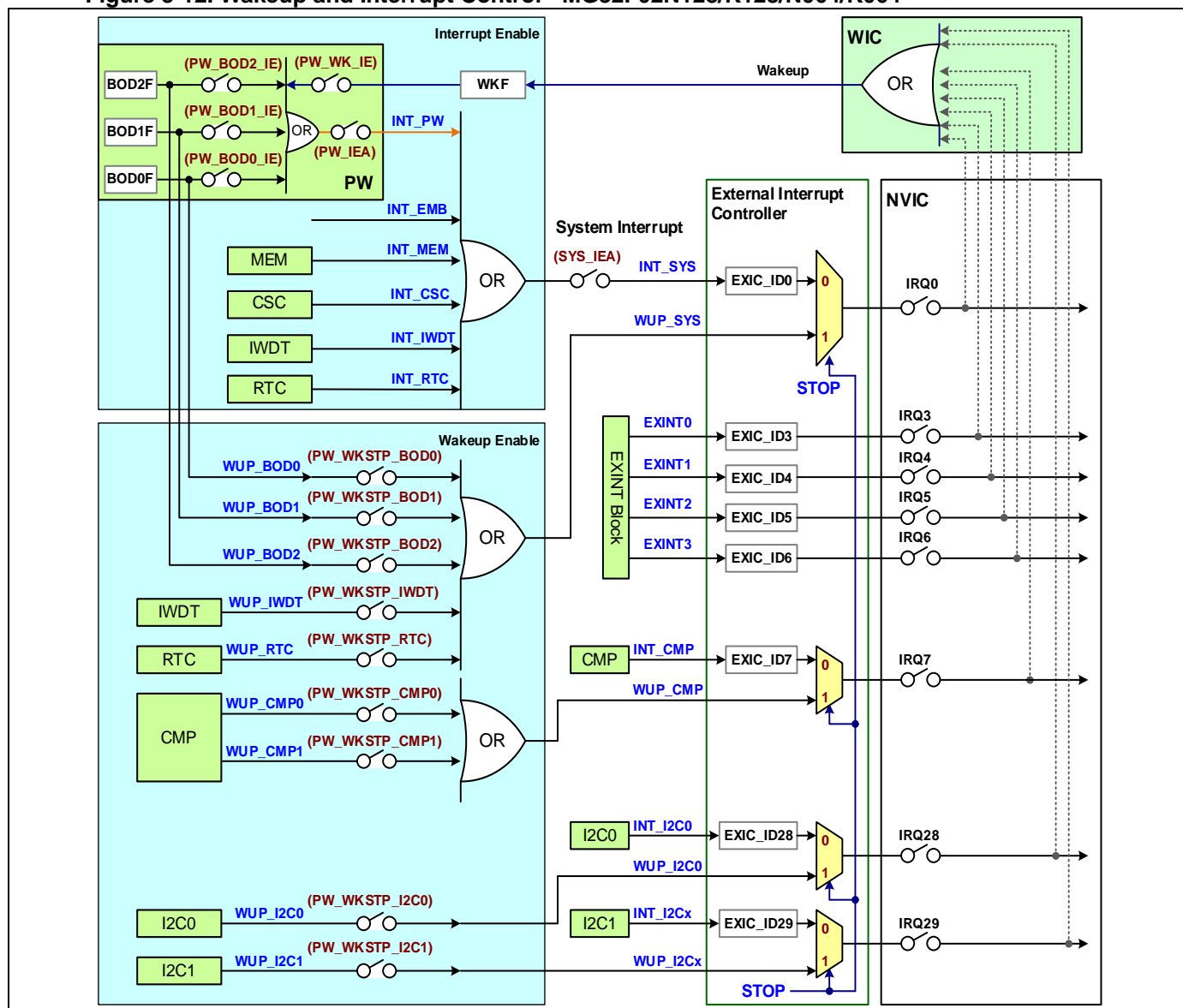
Figure 3-11. Wakeup and Interrupt Control - MG32F02V032



❖ MG32F02N128/K128/N064/K064

The following diagram is showing the system wakeup and interrupt control connection diagram for MG32F02N128/K128/N064/K064.

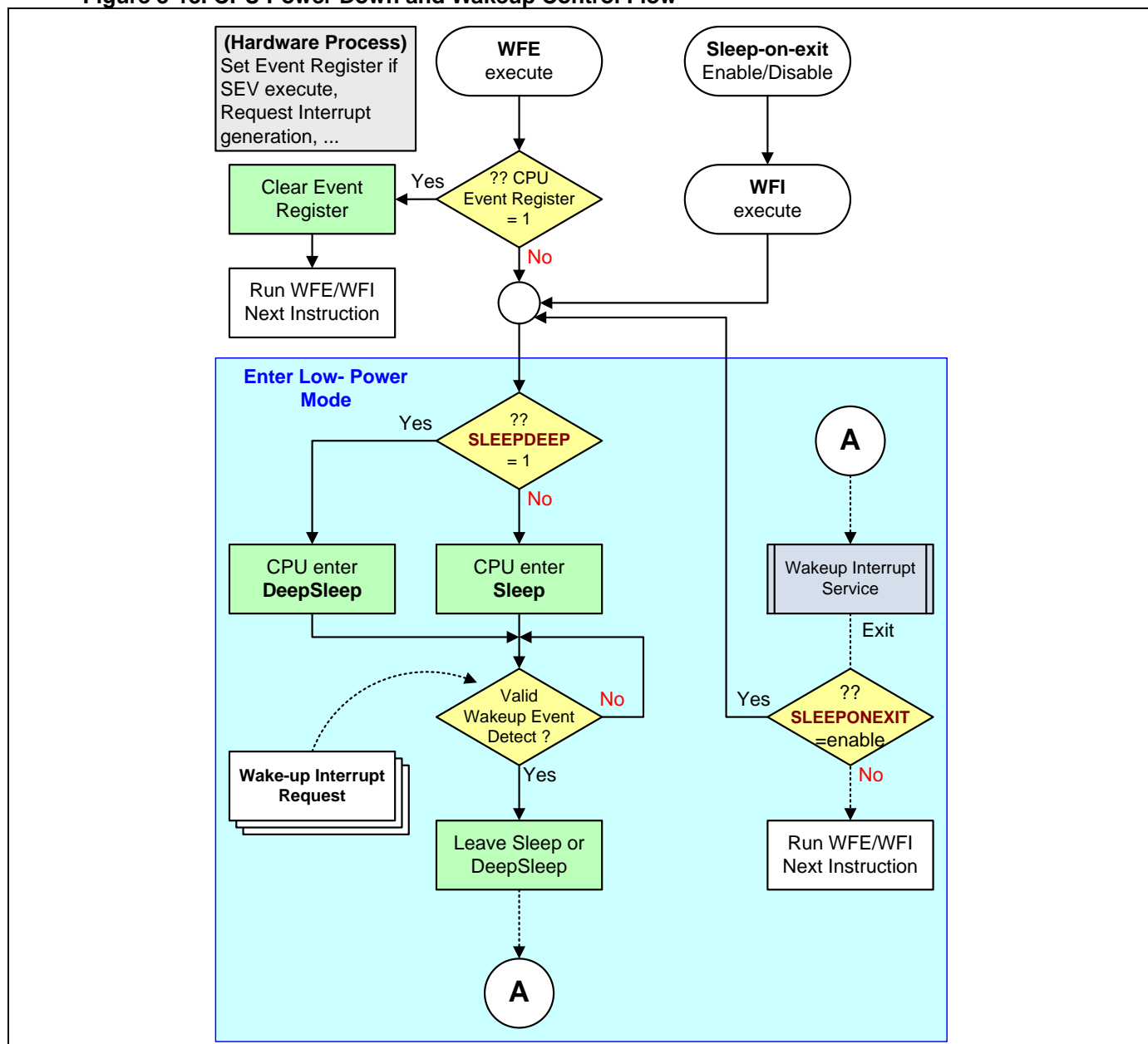
Figure 3-12. Wakeup and Interrupt Control - MG32F02N128/K128/N064/K064



3.11.3. CPU Power Down and Wakeup Control Flow

The following figure is showing the CPU power down and wakeup flow chart by **WFI**, **WFE** and **Sleep-on-exit** control. There is a CPU **Event register** to record the interrupt input state for **WFE** control flow. The CPU register **SLEEPDEEP** is set to control whether the CPU uses Sleep or Deepsleep as its low power mode. The CPU register **SLEEPONEXIT** is set to control whether the CPU enters Sleep/Deepsleep or not when CPU returns from an ISR to thread mode. Refer to the “Cortex-M0 Technical Reference Manual” for more information.

Figure 3-13. CPU Power Down and Wakeup Control Flow



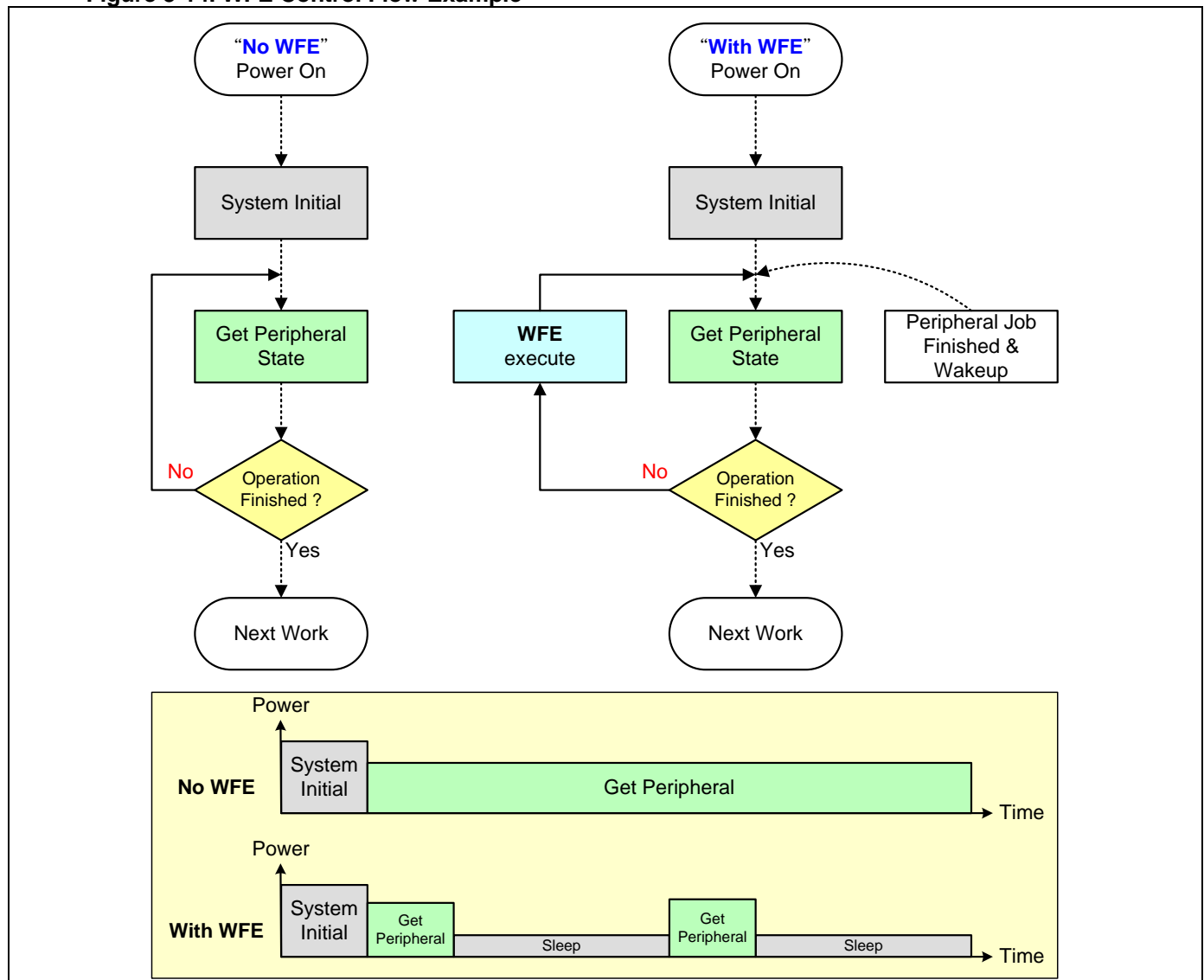
The following table is showing the WFE events for setting CPU **Event register**.

Table 3-11. WFE Events for Setting Event Register

Item	Event Description
1	The arrival of an interrupt request that needs servicing (enabled interrupts)
2	An exception(ISR) entrance and exception(ISR) exit
3	New pending interrupts (only when SEVONPEND=1), even if the interrupts are disabled
4	An external event signal from on-chip hardware (RXEV in)
5	Execution of a Send Event (SEV) instruction (TXEV out)
6	Debug event

The following figure is showing the **WFE** using control flow example. For the example, there are two control flows for the application system those 'With WFE' is using **WFE** control flow and 'No WFE' is not. The chip power consumption of 'With WFE' will be less than 'No WFE'.

Figure 3-14. WFE Control Flow Example



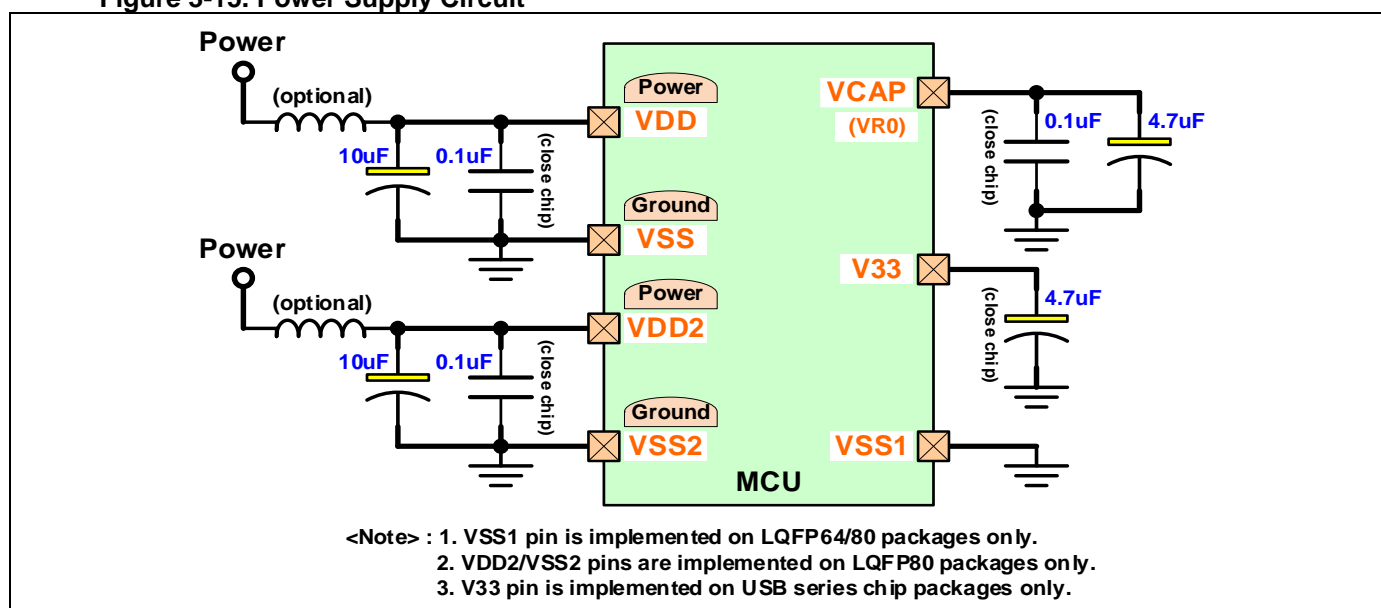
3.12. Chip Power Application Circuit

To have the chip work with power supply varying from 1.8V to 5.5V, adding some external decoupling and bypass capacitors is necessary on **VDD/VSS** power pins, as shown in following figure. Also the same application suggestion on **VDD2/VSS2** power pins for LQFP80 package. There is one extra ground pin **VSS1** for LQFP80 and LQFP64 packages. The **VCAP (VR0)** pin is the embedded LDO voltage output as internal core logic power supply. It needs to place one 0.1uF capacitor and one 4.7uF capacitor to be closed the pin. For USB series chip, one external decoupling and bypass 4.7uF capacitor is necessary on **V33** power pin.

Strongly suggests one bulk decoupling 10uF capacitor is built in the power supply source for VDD power. One bypass 0.1uF capacitor is built and placed close to each **VDD** pin. The bypass capacitor can store an electrical charge that is supplied stable voltage to the power line of **VDD**.

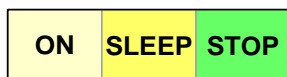
In order to improve the 'Power Integrity' and reduce the VDD power noise, the external power capacitors of 10uF and 0.1uF must placement and route by the '10uF-then-0.1uF' order from the VDD power source to MCU VDD pin.

Figure 3-15. Power Supply Circuit



4. System Reset

4.1. Introduction



The module can be running in all power operation modes.

During reset, all Registers are set to their initial values, and the program starts execution from the Reset Vector. The chip includes a reset source controller (RST) to manage multiple sources of reset and generates Warm reset and Cold reset signals to chip system and internal modules. This controller also provides the reset event flags for firmware, which are used to recognize the reset occurred source.

Notify: The sign of (**OB** = hardware configuration Option Byte flash memory) is using in the descriptions of this chapter.

4.2. Features

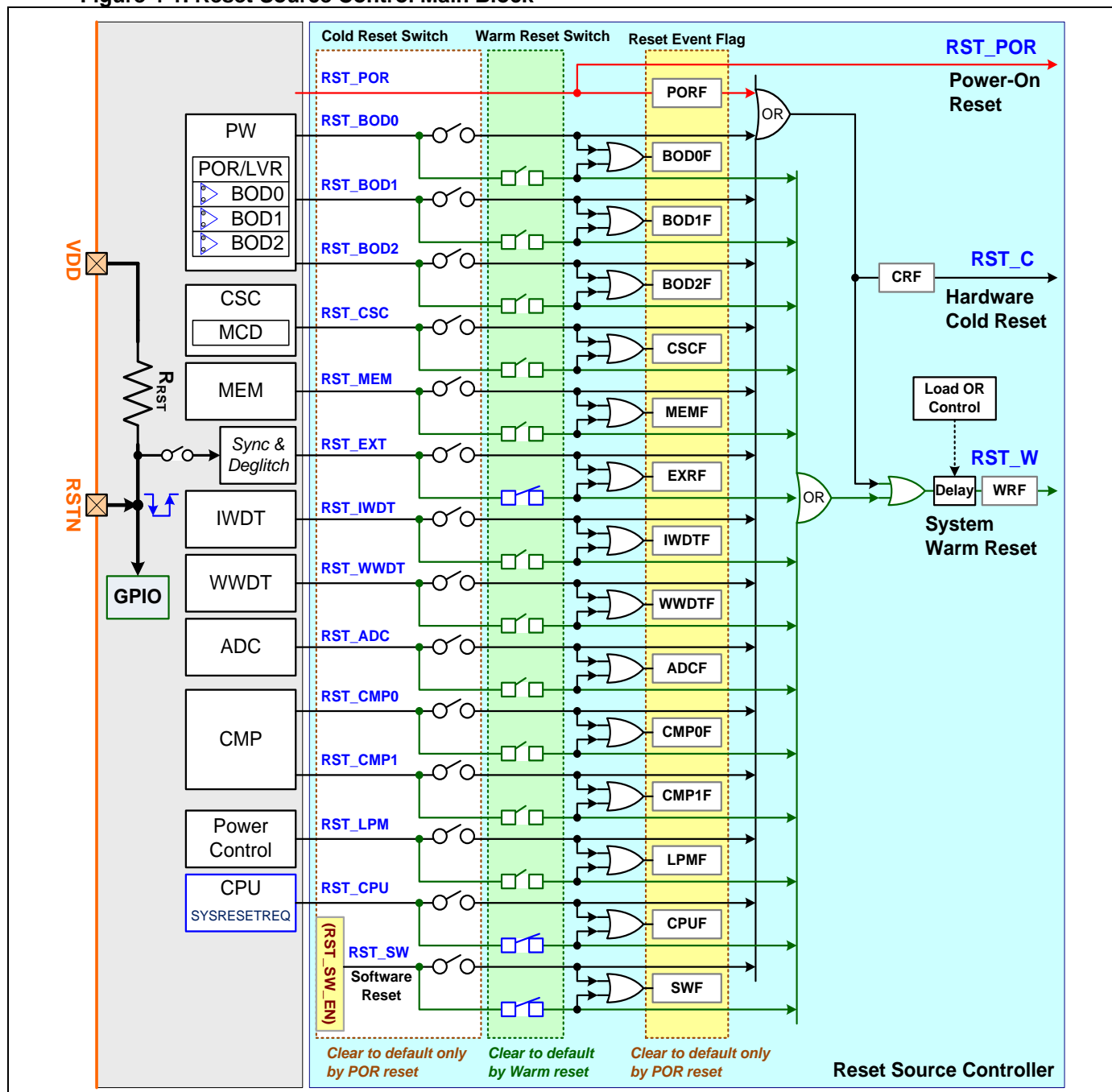
- Built-in embedded POR(power-on reset)/LVR(low-voltage reset) circuit
- Built-in one reset source controller
 - Programmable chip cold reset and warm reset for reset source
 - Independent software reset control for internal modules
- Provide multiple reset sources
 - POR/LVR/BOD /External reset pin input/Software force reset
 - IWDG/WWDT/ADC/Comparator
 - IAR(Illegal address error reset)/Flash access protect error reset
 - Missing clock detect (MCD) reset

4.3. Reset Source Controller

❖ MG32F02A128/U128/A064/U064

The following diagram is showing the MG32F02A128/U128/A064/U064 Reset source control block.

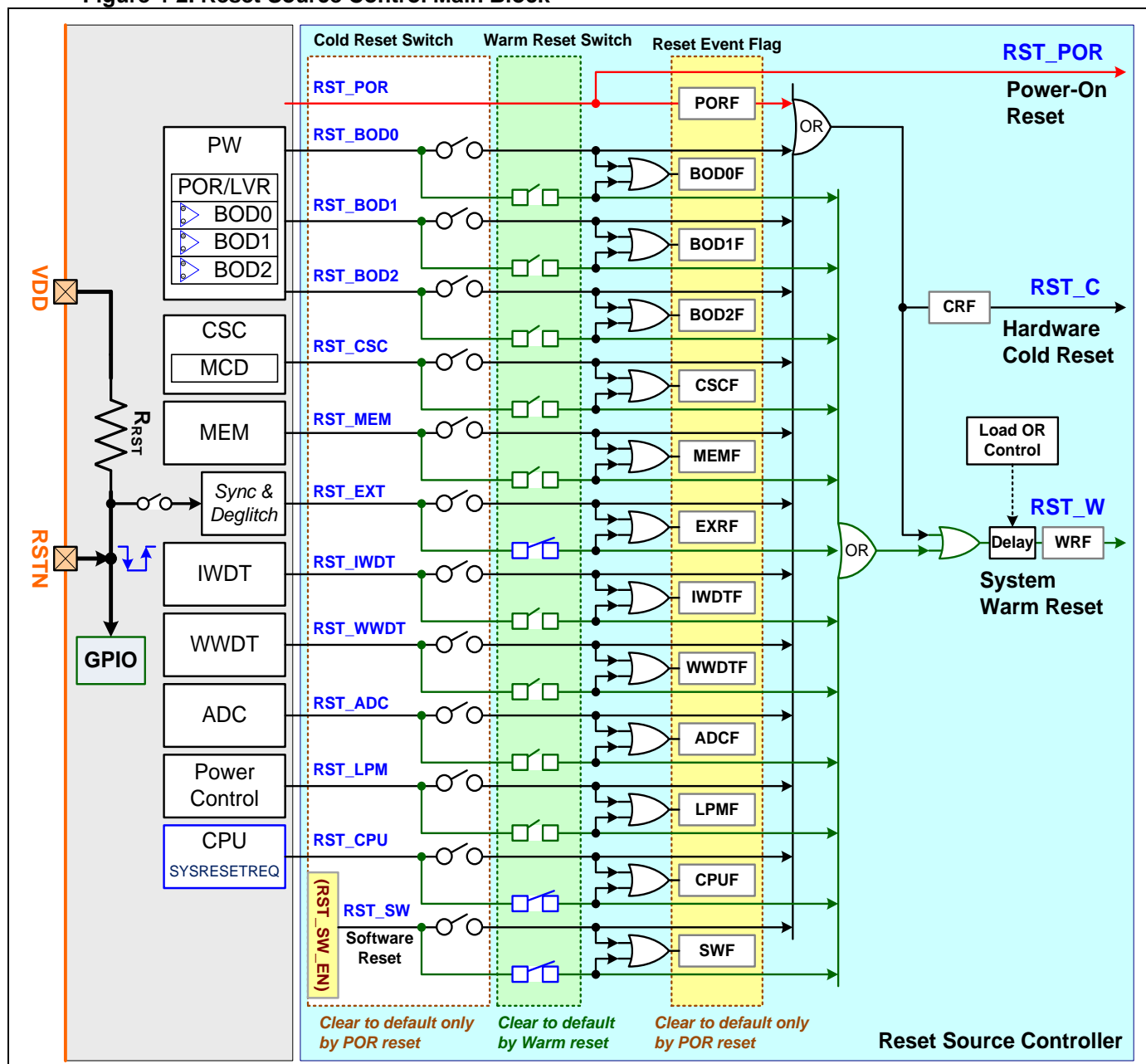
Figure 4-1. Reset Source Control Main Block



❖ MG32F02V032

The following diagram is showing the MG32F02V032 Reset source control block.

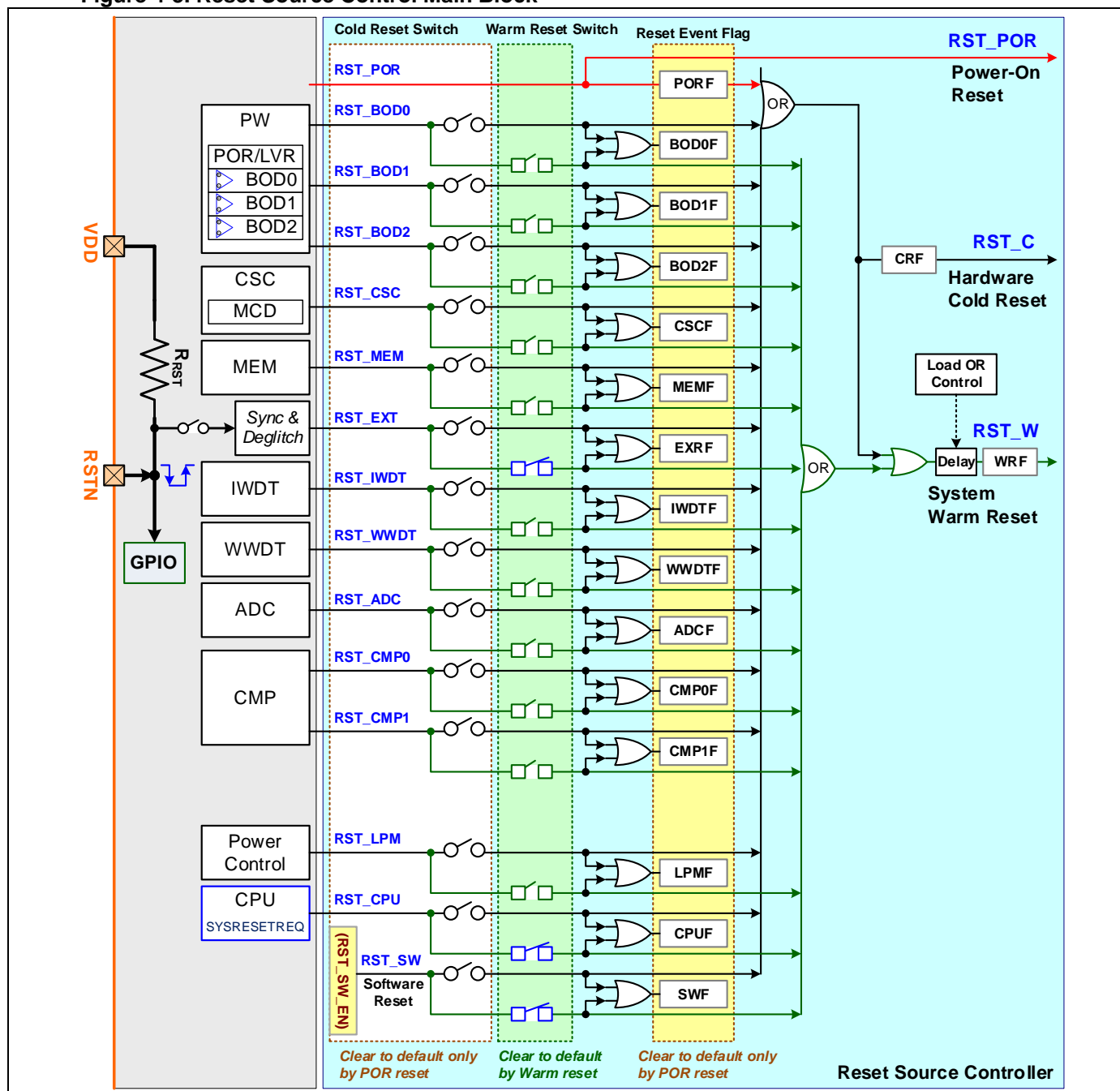
Figure 4-2. Reset Source Control Main Block



❖ MG32F02N128/K128/N064/K064

The following diagram is showing the MG32F02N128/K128/N064/K064 Reset source control block.

Figure 4-3. Reset Source Control Main Block



4.4. Chip Reset

4.4.1. Chip Reset Levels

The chip provides three reset levels – POR reset, Cold reset and Warm reset. POR reset is the highest priority reset and is generated by chip hardware. Cold reset is the 2nd priority and Warm reset is the lowest priority reset.

When POR reset occurred, it will cause to generate Cold reset to chip. Also when Cold reset occurred, it will cause to generate Warm reset to chip. User can read the status of CRF (**RST_CRF**) and WRF (**RST_WRF**) flags to get the previous reset level information. Refer the diagram of Reset Source Controller for the detail.

The following table is showing the reset control function for different reset level.

Table 4-1. Reset Control Function for Different Reset Level

Reset Level	Chip Function	
Warm Reset (Priority Normal)	(1) Reset all chip blocks and restart from the address of reset entry in vector table. (2) Reset flag and Cold Reset enable bits do not be cleared. (3) Some Option Byte flash (OB) data load into Hardware Option Byte Control registers (OR). (4) Some hardware OR data load into related Module Control registers and module other registers reset to default value. (5) Keep the IO setting of Non-Reset IO pins those are able to configure for Reset or Non-Reset. . (6) Lock function is kept.	
	Unlocked Condition	Locked Condition
	It clears Warm Reset enable bits. Key function is disabled.	It does not clear Warm Reset enable bits. Key function is kept.
Cold Reset (Priority High)	Same as "Warm Reset" except following difference: (1) All Option byte flash (OB) data load into Hardware Option Byte Control registers (OR). (2) Reset the IO setting of all IO pins. (3) Lock/Key function is disabled.	
POR/LVR Reset (Priority Highest)	Same as "Cold Reset" except following difference: (1) Reset flag and Cold/Warm Reset enable bits are cleared.	

● Power-On Reset

Power-on reset (**POR**) is used to internally reset the chip and also the CPU during power-up. The chip will keep in reset state and will not start to work until the VDD power rises above the voltage of Power-On Reset. And, the reset state is activated again whenever the VDD power falls below the POR threshold voltage. During a power off cycle, VDD must fall below the POR threshold voltage before power is reapplied in order to ensure a power-on reset.

The POR reset only sends to RST controller and is not using for other modules. It will clear the reset event source flags, Cold reset source enable bits and Warm reset source enable bits in RST controller.

Refer the section of "[Power Voltage Detect](#)" in System Power chapter for more information about the POR reset.

● Cold Reset

Cold reset is the 2nd priority reset. The Cold reset is also generated and caused by POR reset occurred. It sends to some modules like as IWDI, WWDT ... to do deep level module reset. It will cause to reload all hardware configurations **OB** and disable the register lock function for the modules which are support the register lock function.

● Warm Reset

Warm reset is the lowest priority reset. The Warm reset is also generated and caused by Cold reset occurred. It sends to all modules to clear flags and hardware circuit. It will cause to reload some hardware configuration **OB** and reset the registers of module to default value if the module is unlocked or not supported lock function. It will clear Warm reset source enable bits in RST controller if the RST controller is unlocked.

4.4.2. Power-On Reset and Chip Reset Timing

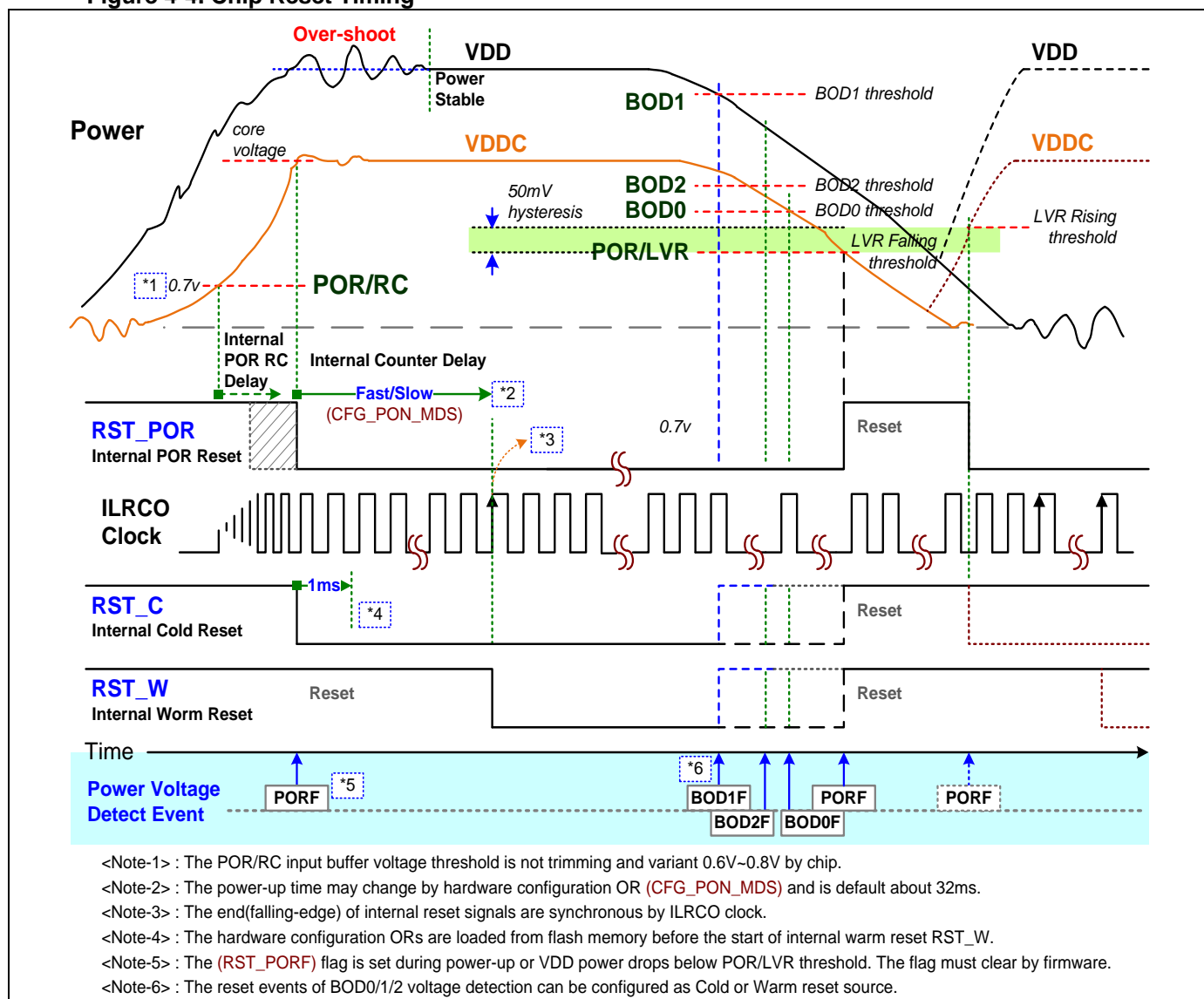
During power-up, the internal POR circuit will generate the POR reset to start up the ILRCO clock circuit and low voltage detector. The POR reset will release at the time which is controlled by the discharging of internal RC and voltage detection of power voltage detector for different power-up slop rate. Also the PORF (**RST_PORF**) reset flag will asserted at the end of POR reset.

At the same time, the POR reset causes the Cold reset to reset other internal devices. It will force to reload the hardware configuration setting from hardware configuration **OB**. For waiting stable ILRCO clock, which will be started after about 1ms time delay of Cold reset released in the following Warm reset.

The Warm reset will reset CPU and all the internal modules. It will be released after Cold reset end by a time delay about 32ms or 4ms. The time delay is set by the hardware configuration **OB**.

The chip is built-in BOD0/BOD1/BOD2 and POR/LVR voltage detectors to monitor the chip power voltage in order to protect the chip flash memory content and insure the operation correction for the chip product. Refer the section of "[Power Voltage Detect](#)" in System Power chapter for more information.

Figure 4-4. Chip Reset Timing



4.4.3. External Reset Timing

The chip provides an external hardware reset input from **RSTN** pin, which is accomplished by holding low level for the **RSTN** pin. The **RSTN** pin is configured to as external reset pin or others (GPIO ...) by hardware configuration **OB**. To ensure a reliable power-up reset, then the hardware reset from **RSTN** pin is necessary.

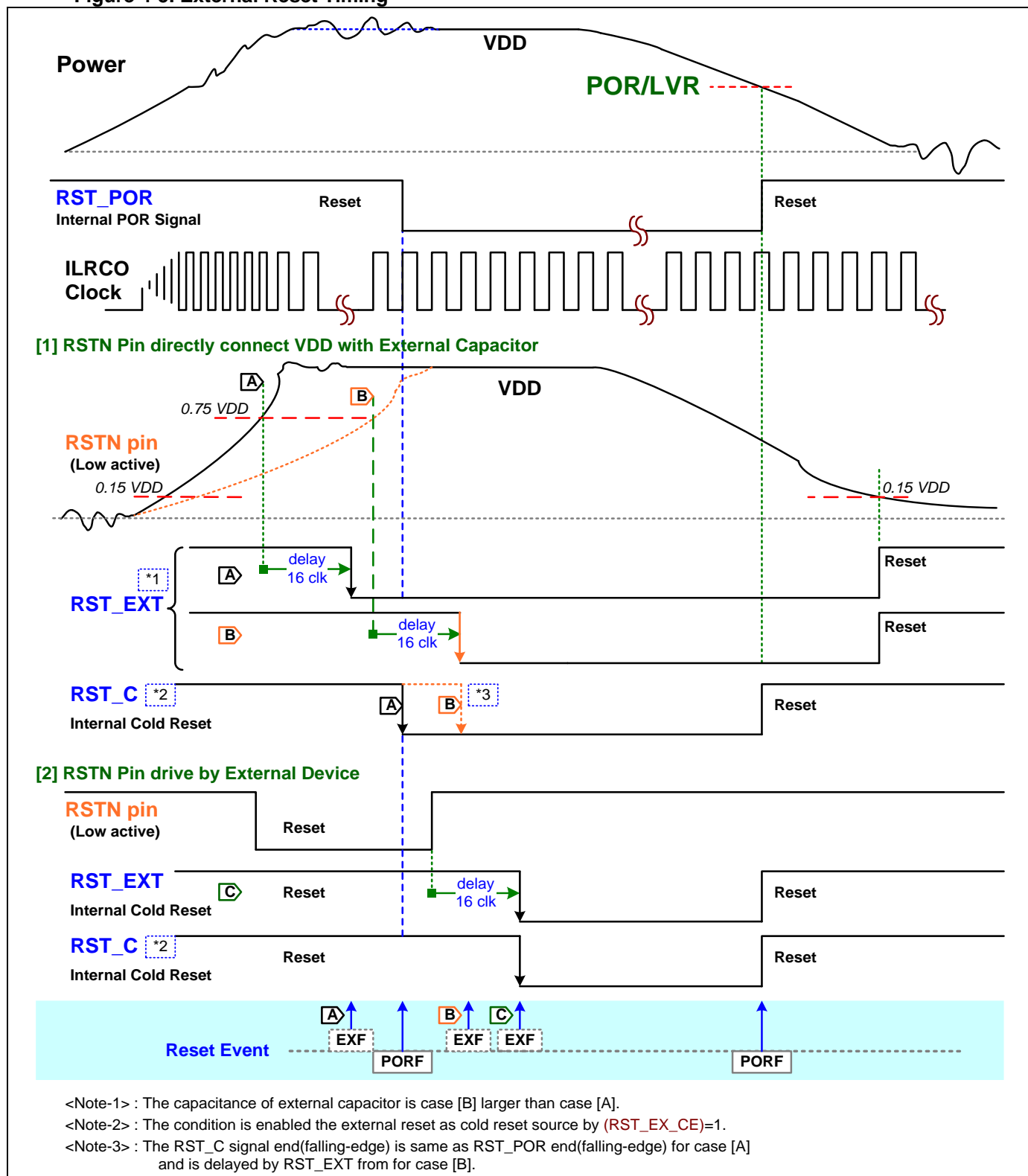
When the **RSTN** pin is connected to **VDD** pin directly and the input voltage of external reset signal is below 0.15 VDD (operation power voltage of **VDD** pin), the external reset is happened. When the input voltage of external reset signal is above 0.75 VDD, the external reset will release after 16 ILRCO clock delay. The EXF

(**RST_EXF**) reset flag is asserted at the end of external reset.

For chip external reset detection, the active low pulse width of external reset signal must be kept ≥ 48 AHB clock (**CK_AHB**) cycles to guarantee that the chip can detect the external reset event. When chip detects the external reset event, also the external reset causes the chip internal reset to reset other internal devices. It will force to reload the hardware configuration setting from hardware configuration **OB**.

Refer the section of "[External Reset Application Circuit](#)" for more information.

Figure 4-5. External Reset Timing



4.5. Reset Event

4.5.1. Reset Event Source

The chip support multiple reset source those include power-on reset, external reset, software reset, memory illegal address reset, memory access error reset, WDT reset, MCD reset, ADC window detect reset, Analog comparator reset and brown-out reset. User can read the reset event flags to recognize the reset occurred source.

Usually these reset events are also used to do as interrupts source in the related source module and can be configured as the system interrupts. Refer the chapters of related source module for more information.

The following table is showing the event source of warm/cold reset and the interrupt capability.

Table 4-2. Reset Event Source

Reset Source	Function	Reset Flag	Cold Reset Gating	Warm Reset Gating	Interrupt Capability
POR	Power-On Reset	yes	mandatory	mandatory	
BOD0	Brown-Out Reset	yes	yes	yes	yes
BOD1/2	Brown-Out Reset	yes	yes	yes	yes
CSC MCD Error	Missing clock detect Reset for XOSC	yes	yes	yes	yes
MEM IA Error	Flash/SRAM illegal address error Reset	yes	yes	yes	yes
MEM RP Error	Flash/SRAM read protect error Reset	yes	yes	yes	yes
MEM WP Error	Flash/SRAM write protect error Reset	yes	yes	yes	yes
IWDT Reset	Independent Watch-Dog Time-out Reset	yes	yes	yes	yes
WWDT Reset	Window Watch-Dog Time-out Reset	yes	yes	yes	yes
ADC Reset	ADC voltage window detect Reset	yes	yes	yes	yes
CMP0/1 Reset	Analog Comparator voltage detect Reset	yes	yes	yes	yes
EXT Reset	External input Reset from RSTN pin	yes	yes	yes	
SW Reset	Chip Software Reset by setting RST_SW_EN bit	yes	yes	yes	
Low Power Mode Reset	The chip is reset instead of entering STOP mode when processes into STOP.	yes	yes	yes	
CPU Reset	CPU Reset event by setting SYSRESETREQ bit	yes	yes	yes	

4.5.2. Reset Event Control

The RST controller can configure the reset events to decide which reset source can trigger Cold reset or Warm reset. When one reset event is occurred and the related Cold reset or Warm reset enable bit is enabled, the RST controller will asserted the related reset event flag to recognize the reset occurred source.

Generally these reset flags are set by hardware and clear by software writing 1. Refer the register descriptions for more information about the related reset flags and reset enable bits.

- **Power-On Reset**

There is one PORF (**RST_PORF**) flag to indicate this reset event source. Power-on reset (**POR**) is generated by internally hardware RC and voltage detection circuits and is no software enable control bit.

- **External Reset**

The chip provides an external hardware reset input from **RSTN** pin to ensure a reliable power-up reset.

There is one EXF (**RST_EXF**) flag to indicate this reset event source. Use can enable the reset event to trigger Cold reset or Warm reset by setting **RST_EX_CE** or **RST_EX_WE** registers.

- **Software Reset**

User can trigger the chip to restart by the software reset function, which is writing "1" on **RST_SW_EN** register bit.

There is one SWF (**RST_SWF**) flag to indicate this reset event source. Use can enable the reset event to trigger Cold reset or Warm reset by setting **RST_SW_CE** or **RST_SW_WE** registers.

- **Memory Read/Write Protect Error and Illegal Address Reset**

When occurs memory access protection error or software program runs to illegal address on internal Flash or SRAM memory such as over program memory limitation, which may triggers a Reset.

There is one MEMF (**RST_MEMF**) flag to indicate this reset event source. Use can enable the reset event to trigger Cold reset or Warm reset by setting **RST_MEM_CE** or **RST_MEM_WE** registers.

- **IWDT/WWDT Reset**

When IWDT or WWDT is enabled to start the counter, the overflow flag will be set by IWDT or WWDT overflow. If **RST_IWDT_WE** or **RST_WWDT_WE** is enabled, the IWDT/WWDT overflow will trigger a system Warm reset. If **RST_IWDT_CE** or **RST_WWDT_CE** is enabled, the WDT overflow may trigger a hardware Cold reset. There are one IWDTF (**RST_IWDTF**) flag and one WWDTF (**RST_WWDTF**) flag to indicate these two reset event sources.

- **MCD Reset**

When missing clock detection on **XOSC** clock or external input clock is occurred, it may trigger a Reset.

There is one CSCF (**RST_CSCF**) flag to indicate this reset event source. Use can enable the reset event to trigger Cold reset or Warm reset by setting **RST_CSC_CE** or **RST_CSC_WE** registers.

- **ADC Voltage Window Detection Reset**

When the ADC voltage windowed detection event is occurred, it may trigger a Reset.

There is one ADCF (**RST_ADCF**) flag to indicate this reset event source. Use can enable the reset event to trigger Cold reset or Warm reset by setting **RST_ADC_CE** or **RST_ADC_WE** registers.

- **Analog Comparator Reset**

When occurs any of analog comparator comparison event, which may trigger a Reset.

There are four CMPnF (**RST_CMPnF**) flags independently to indicate the reset event source for analog comparator CMPn (n= {0, 1, 2, 3}). Use can enable the reset event to trigger Cold reset or Warm reset by setting **RST_CMPn_CE** or **RST_CMPn_WE** registers.

- **Brown-Out Reset**

There are three Brown-Out Detectors (BOD0/BOD1/BOD2) to monitor chip power. If core power drops below BOD0 monitor level or VDD power drops below BOD1/BOD2 monitor level, which may trigger a Reset.

There are one BOD0F (**RST_BOD0F**) flag, one BOD1TF (**RST_BOD1F**) flag and one BOD2TF (**RST_BOD2F**) flag to indicate these three reset event sources. Use can enable the reset event to trigger Cold reset or Warm reset by setting **RST_BOD0_CE**, **RST_BOD0_WE**, **RST_BOD1_CE**, **RST_BOD1_WE** and **RST_BOD2_CE**, **RST_BOD2_WE** registers.

- **CPU System Reset**

When the CPU SYSRESETREQ system reset bit is set by software, which may trigger a Reset. This bit is only can write 1 to set and is no effect by writing 0.

There is one CPUF (**RST_CPUF**) flag to indicate this reset event source. Use can enable the reset event to trigger Cold reset or Warm reset by setting **RST_CPU_CE** or **RST_CPU_WE** registers.

- **Low Power Mode Reset**

When the chip enters the low power STOP mode and the Cold reset or Warm reset enable bits is set in **RST_LPM_CE** or **RST_LPM_WE** registers, the chip will trigger a Reset to replace entering STOP mode.

There is one LPMF (**RST_LPMF**) flag to indicate this reset event source.

4.6. Register Protect and Lock

All RST registers are write-access protected except **RST_STA**, **RST_KEY** registers. Write **0xA217** value to the **RST_KEY** register to unprotect the registers and process the control continuously. Oppositely write other value except **0xA217** value to protect the registers. Read the **RST_KEY** register to get the register value is Protected (=1) or Unprotected (=0).

It also provides the register lock function after Warm reset condition. Write value **0x712A** to the **RST_LOCK** register to lock the register write access except **RST_STA**, **RST_KEY** registers. When registers lock, the registers cannot change event through the system warm reset until the hardware cold reset. Write other value except **0x712A** is no effect. Read the **RST_LOCK** register to get the register value is Locked (=1) or Unlocked (=0).

The locked function of register write-access is supported for RST, RTC and IWDT modules after Warm reset. Other modules of CSC, CFG, PW, MEM and WWDT are only supported write-access protected function.

● Cold/Warm Reset VS Lock Function for Modules

Table 4-3. Reset VS Lock Function for Modules' Registers

Module	Functions	POR Reset	Cold Reset	Warm Reset		Comment
				Unlocked	Locked	
RST	Reset Status flags clear	V				Only clear to default by POR reset
	Lock register clear	V	V			
	Cold reset enable registers reset to default	V				Only clear to default by POR reset
	Warm reset enable registers reset to default	V	V	V		
	Key register / Protected registers reset	V	V	V		
CSC	Status flags clear	V	V	V		No Lock function
	Key register / Protected registers reset	V	V	V		
CFG	Status flags clear	V	V	V		No Lock function
	Key register / Protected registers reset	V	V	V		
PW	Status flags clear	V	V	V		No Lock function
	Key register / Protected registers reset	V	V	V		
MEM	Status flags clear	V	V	V		No Lock function
	Key register / Protected registers reset	V	V	V		
RTC	Lock register / Status flags clear	V	V			
	Key register / Protected registers reset	V	V	V		
IWDT	Lock register / Status flags clear	V	V			
	Key register / Protected registers reset	V	V	V		
WWDT	Status flags clear	V	V	V		No Lock function
	Key register / Protected registers reset	V	V	V		

<Sign> V: Indicates the function is allowed. Blank: Indicates the function is not allowed.

4.7. Module Reset

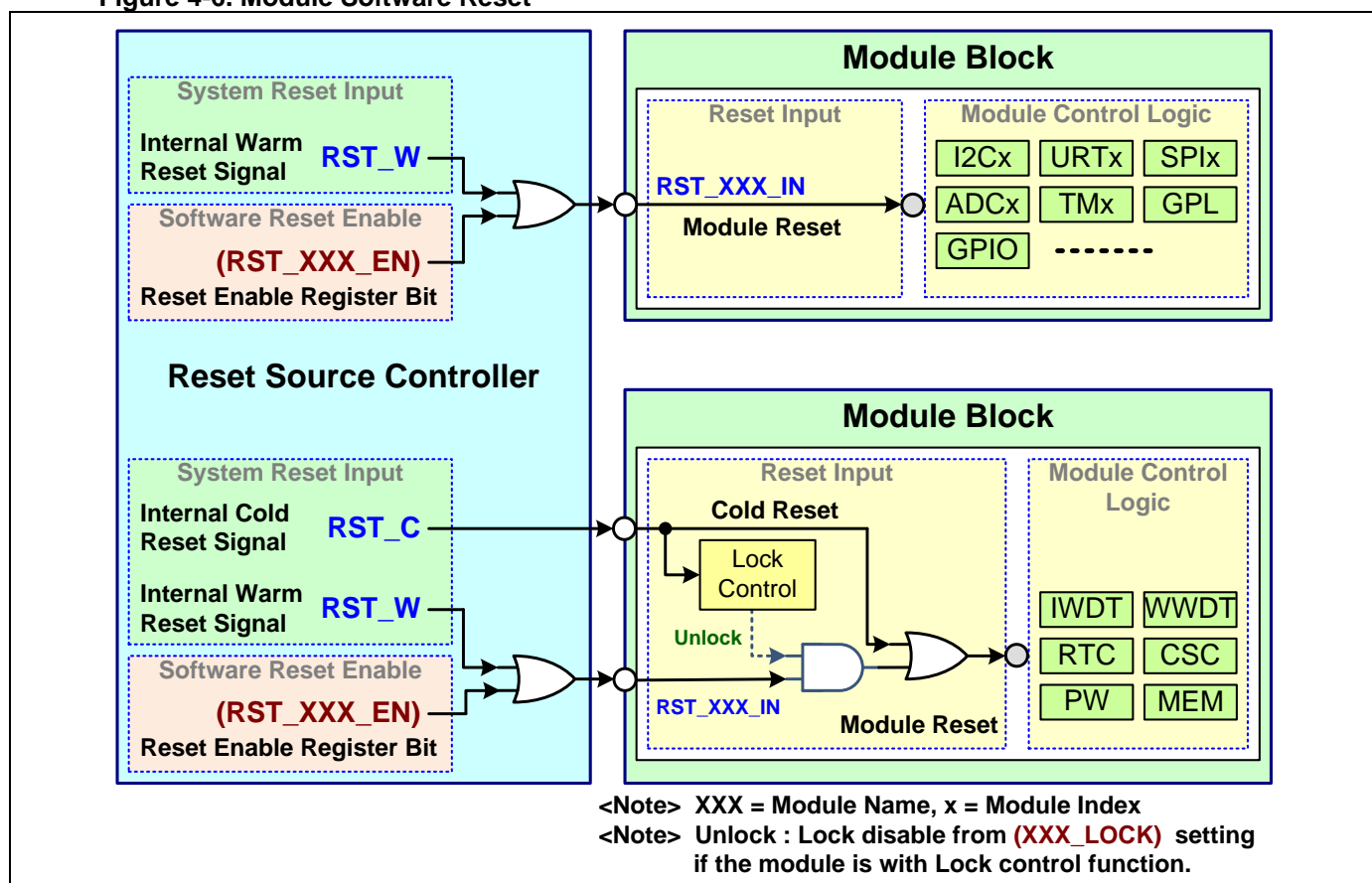
For each AHB or APB control module, it can receive the system Warm reset signal to reset the module's control flags, registers and logical circuit. For some modules of IWDT, WWDT, RTC, PW, CSC and MEM, they can receive the Cold reset to unlock the register locked function and reset the module.

4.7.1. Module Software Reset

There is one independent software reset enable bit for each AHB or APB control module. User can directly set the **RST_XXX_EN** register bit to reset the related module independently ('XXX': indicates a module name). It is like as which the module is receiving the Warm reset signal but only resets the related module. Refer the register descriptions for more information about the software reset of modules.

The following diagram is showing the software reset enable control of modules.

Figure 4-6. Module Software Reset



4.7.2. GPIO Reset Control

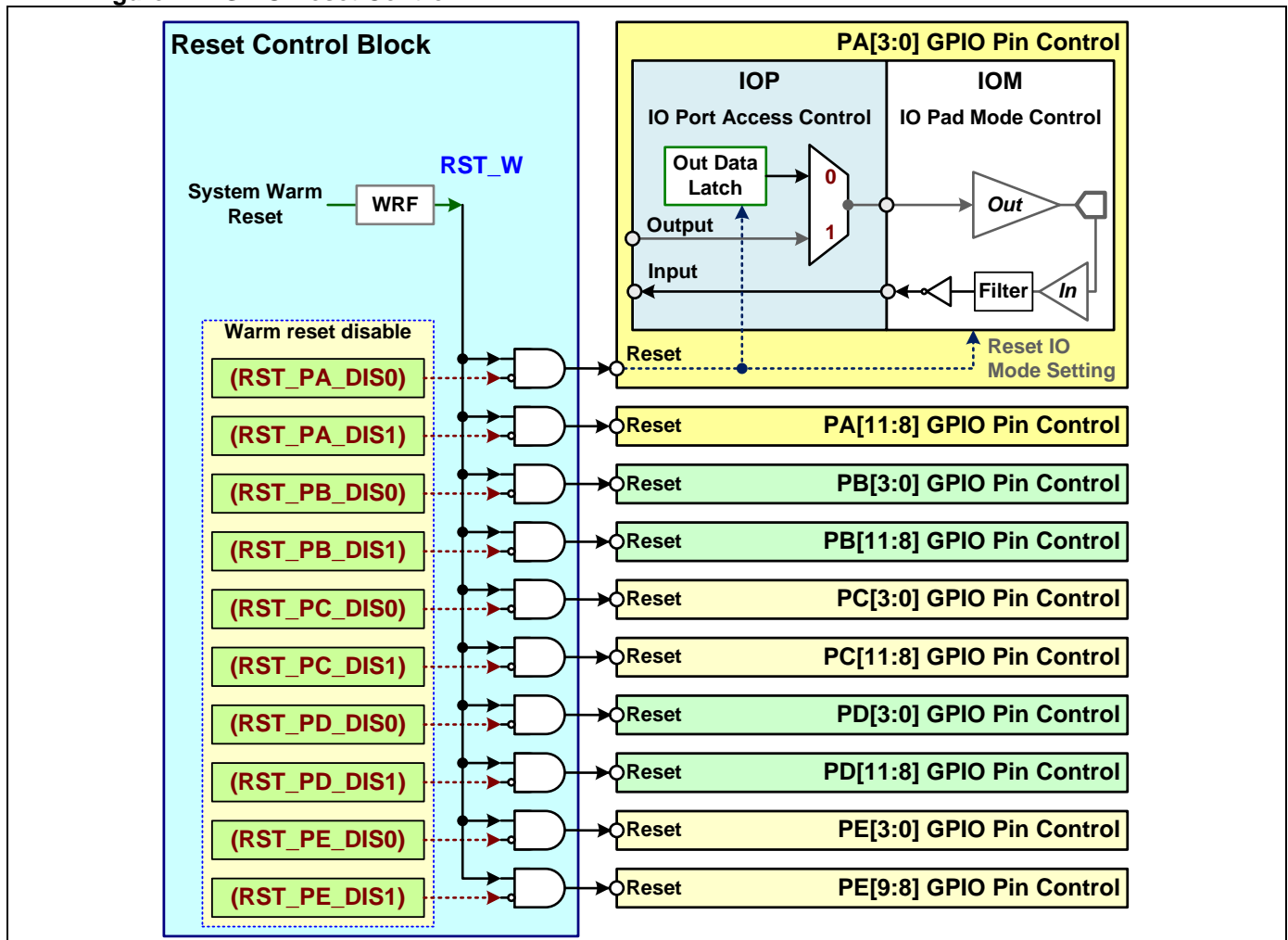
The chip provides that the GPIO output data bit and IO mode configuration can be disabled to be reset by system Warm reset for some GPIO pins. These GPIO pins are including of **PA[3:0]**, **PA[11:8]**, **PB[3:0]**, **PB[11:8]**, **PC[3:0]**, **PC[11:8]**, **PD[3:0]**, **PD[11:8]**, **PE[3:0]** and **PE[9:8]**.

They are separated to 10 groups which are 4 pins and have one independent control register bit for each group. User can configure the Warm reset disabled function for each group independently by setting the **RST_PA_DIS0**, **RST_PA_DIS1**, **RST_PB_DIS0**, **RST_PB_DIS1**, **RST_PC_DIS0**, **RST_PC_DIS1**, **RST_PD_DIS0**, **RST_PD_DIS1**, **RST_PE_DIS0** or **RST_PE_DIS1** registers.

[Notify]: The PE reset control functions is not supported for MG32F02V032.

The following diagram is showing the GPIO reset control for data output bit and IO mode configuration.

Figure 4-7. GPIO Reset Control



4.7.3. USB Software Reset Control

As same as other modules, there is one independent software reset enable bit **RST_USB_EN** to reset the USB module independently. For the software reset specially, the USB module have two reset control levels to reset the USB module by setting **RST_USB_RCTL** register. When this register is selected 'All', chip will auto reset USB all blocks and registers. When this register is selected 'LV1', chip will reset all blocks and registers except the registers' control of **USB_EN**, **USB_XTR_EN**, **USB_V33_EN** and **USB_DPU_EN**.

Refer the section of "[USB Reset Control](#)" in USB chapter for more information.

4.8. External Reset Application Circuit

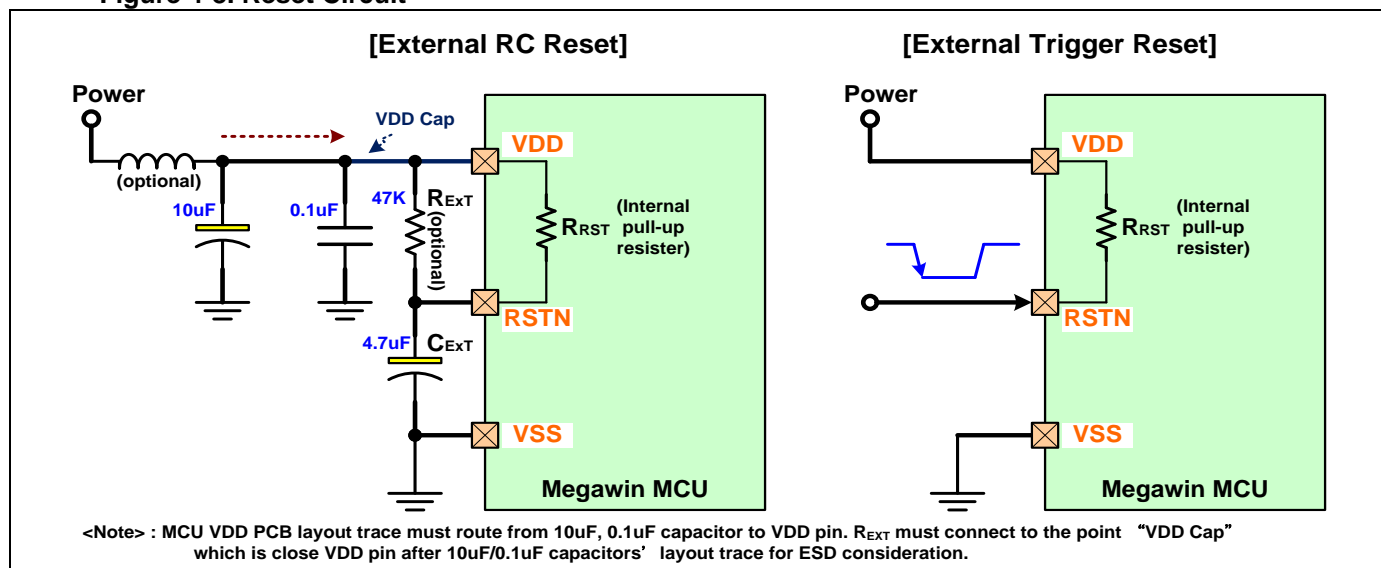
Normally, the power-on reset can be successfully generated during power-up. However, to further ensure the MCU a reliable reset during power-up, the external reset is necessary. The following diagram shows the external reset circuit, which consists of a capacitor C_{EXT} connected to VSS (ground) and a resistor R_{EXT} connected to VDD (power supply).

In general, R_{EXT} is optional because the $RSTN$ pin has an internal pull-high resistor (R_{RST}). This internal diffused resistor to VDD permits a power-up reset using only an external capacitor C_{EXT} to VSS .

Strongly suggestion, the $RSTN$ pin must set to output mode if it is used to do as both chip reset and GPIO functions in application. In this condition, the pin input low may make chip reset locked error if it set to GPIO input mode.

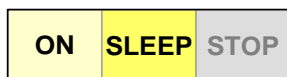
For ESD issue, it is very important that the R_{EXT} must connect from the “ $VDD\ Cap$ ” point. The “ $VDD\ Cap$ ” point is at the PCB trace behind the 10uF and 0.1uF capacitors from the VDD power source to MCU VDD pin.

Figure 4-8. Reset Circuit



5. System Clock

5.1. Introduction



The module can be running in ON and SLEEP modes only.

The chip builds in a clock source controller (CSC) for system clock source management. There are four clock sources for the system application: Internal High-frequency RC Oscillator (**IHRCO**), Internal crystal oscillator (**XOSC**), Internal Low-frequency RC Oscillator (**ILRCO**) and External Input Clock (**EXTCK**).

One **XOSC** oscillator is embedded for external Xtal circuit. One PLL is embedded to multiply the frequency of clock source and output clock for CPU and other peripheral modules. One missing clock detector (**MCD**) is built-in to monitor the clock of external Xtal or external clock source.

Notify: The sign of (**OB** = hardware configuration Option Byte flash memory) is using in the descriptions of this chapter.

5.2. Features

- Built-in embedded ILRCO (internal low frequency RC oscillator) by 32KHz
- Built-in embedded IHRCO (internal high frequency RC oscillator)
 - Trimmed to 11.059 or 12MHz $\pm 1\%$ at $+25^{\circ}\text{C}$
- Built-in embedded PLL clock output for system clock
- Built-in embedded XOSC oscillator with MCD for external 32KHz or 4 to 25MHz Xtal
- Support external clock input up to 36MHz
- Built-in a clock source controller with clock enable control for modules
- Support internal XOSC oscillator and internal ILRCO/IHRCO clock output

5.3. Implementation

5.3.1. Embedded Clock PLL Implementation

The following table is showing the implemented embedded PLL of chips.

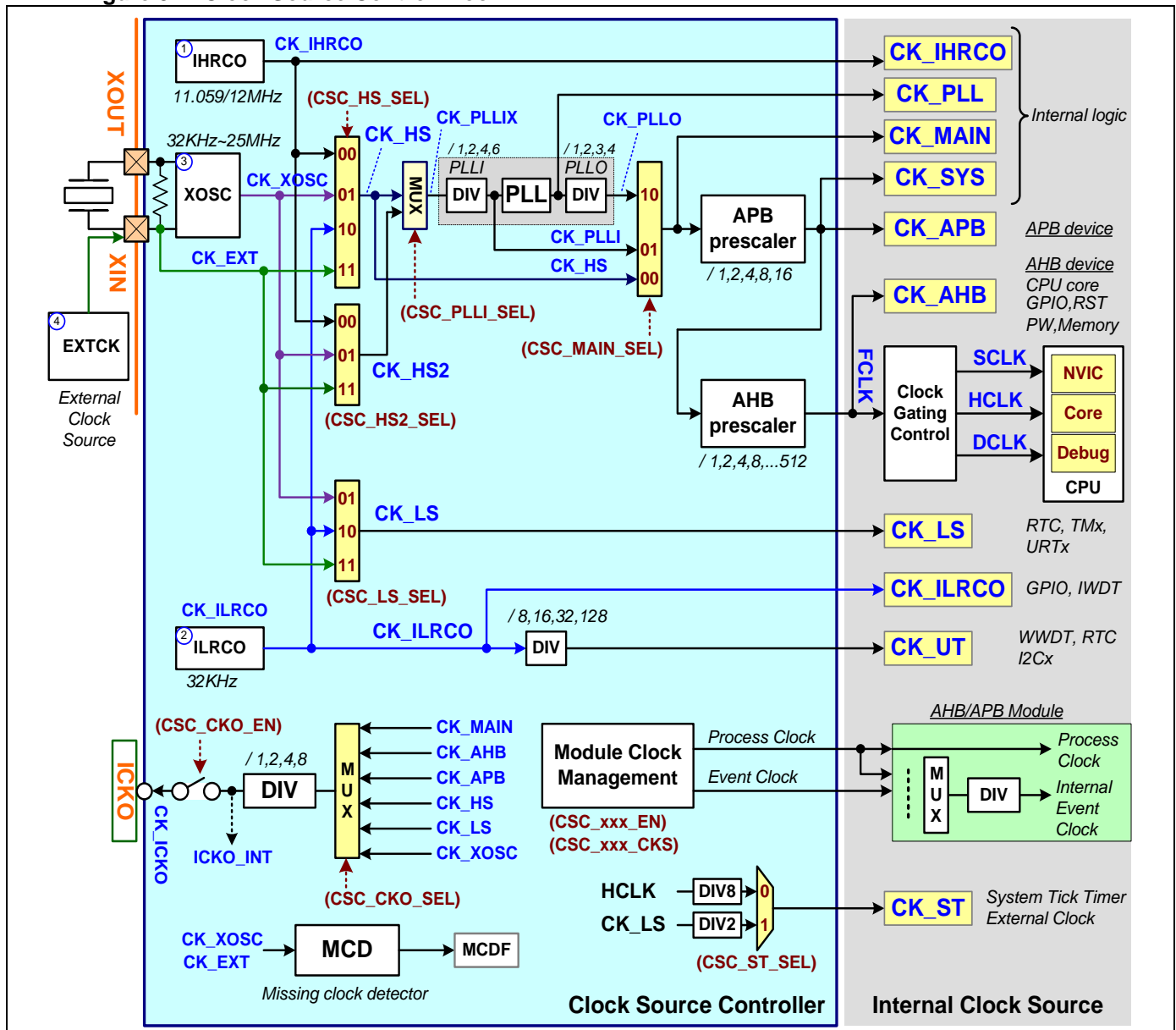
Table 5-1. Embedded PLL Implementation

Chip	Embedded PLL				Embedded Clock Device			External Input
	Input Range	Multiplier	VCO Max.	Lock Status	ILRCO	IHRCO	XOSC	EXTCK
MG32F02A128/A064 MG32F02U128/U064	4 ~ 8.5 MHz	x4 ~ x32	180MHz	-	32KHz	11.059 or 12MHz	32KHz and 4 ~ 25MHz	Max. 36MHz
MG32F02V032	4.2 ~ 8.5 MHz	x4 ~ x32	180MHz	-				
MG32F02N128/N064 MG32F02K128/K064	4 ~ 9 MHz	x4 ~ x32	144MHz	V				

5.4. Clock Source Controller

The following diagram is showing the Clock source control block.

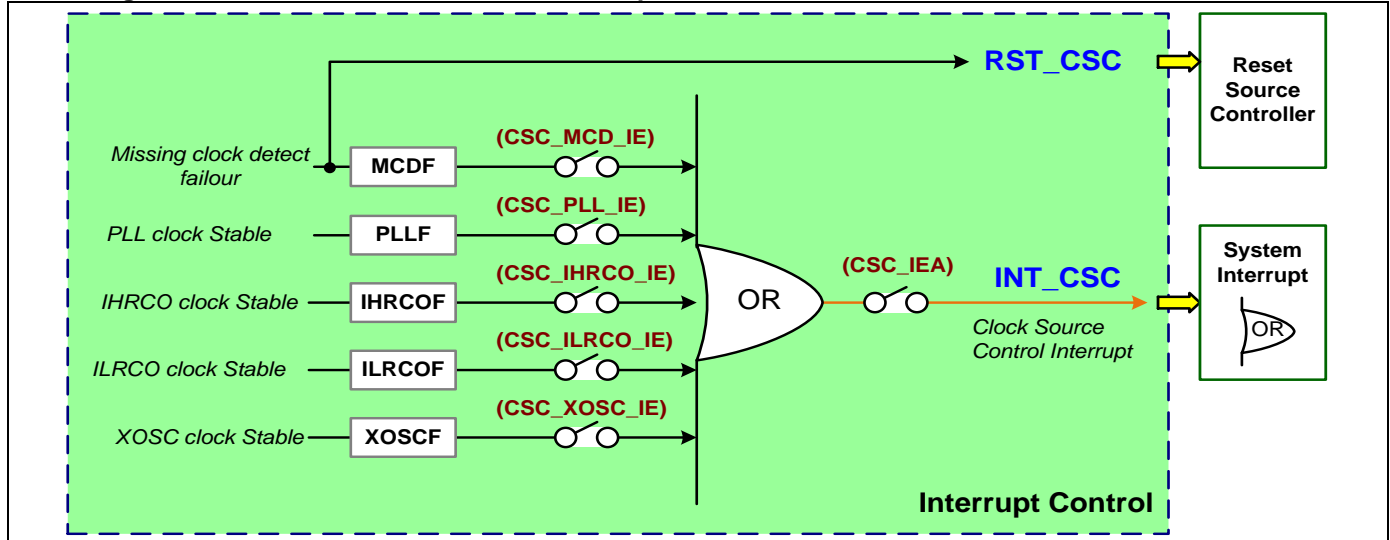
Figure 5-1. Clock Source Control Block



5.5. Interrupt and Reset

There are two signals of **INT_CSC** and **RST_CSC** to be generated in this PW control module. **INT_CSC** sends to External Interrupt Controller (EXIC) to do as an interrupt event. **RST_CSC** sends to Reset Source Controller to do as a reset event.

Figure 5-2. Clock Source Controller Interrupt



5.5.1. CSC Interrupt Flags

The interrupt of **INT_CSC** is OR with the clock stable status of **XOSC**, **IHRCO**, **ILRCO**, PLL and miss clock detection event flag. It is outputted as one of the interrupt events of the system interrupt of **INT_SYS**. Refer the section of “[Interrupt and Event](#)” in the chapter of System Common Control for more information about system interrupt.

These interrupt flags are using for the interrupt service routine (ISR) flow control. Generally these flags are set by hardware and software clears them when the serviced job is completed in the related ISR. Each interrupt flag has one interrupt enable bit. User can enable or disable it. There is one interrupt all enable bit of **CSC_IEA** to enable or disable all the interrupt sources for this module.

Refer the register descriptions for more information about the related interrupt flags and interrupt enable bits.

- **XOSCF**

XOSC clock stable and ready detect flag (**CSC_XOSCF**). There is a related interrupt enable register bit of **CSC_XOSC_IE**. The flag is active when the XOSC oscillator output clock is stable.

- **ILRCOF**

ILRCO clock stable and ready detect flag (**CSC_ILRCOF**). There is a related interrupt enable register bit of **CSC_ILRCO_IE**. The flag is active when the ILRCO RC oscillator output clock is stable.

- **IHRCOF**

IHRCO clock stable and ready detect flag (**CSC_IHRCOF**). There is a related interrupt enable register bit of **CSC_IHRCO_IE**. The flag is active when the IHRCO RC oscillator output clock is stable.

- **PLL_F**

PLL clock stable and ready detect flag (**CSC_PLL_F**). There is a related interrupt enable register bit of **CSC_PLL_IE**. The flag is active when the internal PLL output clock is stable.

- **MCDF**

XOSC missing clock detect failure event flag (**CSC_MCDF**). There is a related interrupt enable register bit of **CSC_MCD_IE**. The flag is active when the missing clock detector has detected the clock missing on XOSC oscillator output or external clock input.

5.5.2. CSC Clock Status

There are some read only status flags to indicate the clock source status or the clock switching status. Generally these flags are set and cleared by hardware control.

- **XOSC_STA**

XOSC clock stable and ready status after XOSC enabled (**CSC_XOSC_STA**). The flag is active when the

XOSC oscillator output clock is stable after XOSC enabled. The flag is inactive when the XOSC oscillator is disabled.

- **ILRCO_STA**

ILRCO clock stable and ready real time status after ILRCO enabled (**CSC_ILRCO_STA**). The flag is active when the ILRCO RC oscillator output clock is stable after ILRCO enabled. The flag is inactive when the ILRCO is disabled.

- **IHRCO_STA**

IHRCO clock stable and ready real time status after IHRCO enabled (**CSC_IHRCO_STA**). The flag is active when the IHRCO RC oscillator output clock is stable after IHRCO enabled. The flag is inactive when the IHRCO is disabled.

- **PLL_STA**

PLL clock stable and ready real time status after PLL enabled (**CSC_PLL_STA**). The flag is active when the internal PLL output clock is stable after PLL enabled. The flag is inactive when the PLL is disabled.

- **PLLI_STA**

Input PLL clock source select MUX (CK_PLLIX) switching status (**CSC_PLLI_STA**). It will report the current switched clock source or in switching for the clock source MUX.

- **LS_STA**

Input low speed clock source select MUX (CK_LS) switching status (**CSC_LS_STA**). It will report the current switched clock source or in switching for the clock source MUX.

- **HS_STA**

Input high speed clock source select MUX (CK_HS) switching status (**CSC_HS_STA**). It will report the current switched clock source or in switching for the clock source MUX.

- **HS2_STA**

Input high speed clock source select MUX (CK_HS2) switching status (**CSC_HS2_STA**). It will report the current switched clock source or in switching for the clock source MUX.

- **MAIN_STA**

System main clock source select MUX (CK_MAIN) switching status (**CSC_MAIN_STA**). It will report the current switched clock source or in switching for the clock source MUX.

5.5.3. CSC Reset Events

RST_CSC signal sends to Reset Source Controller (RST) to do as the warm reset event or cold reset event. This reset events can be enabled to reset the chip by setting the registers in RST.

- **MCD Detect Event**

When the missing clock failure event has detected, the **RST_CSC** signal will be active and the CSC will asserted the MCDF flag.

5.6. Register Protect and Lock

After clock source controller domain reset, all CSC registers are write-access protected except **CSC_STA**, **CSC_KEY** registers. Write **0xA217** value to the **CSC_KEY** register to unprotect the registers and process the control continuously. Oppositely write other value except **0xA217** value to protect the registers. Read the **CSC_KEY** register to get the register value is Protected (=1) or Unprotected (=0).

Refer the table and descriptions of "[Register Protect and Lock](#)" in System Reset chapter for more information.

Special for hardware function control, the following registers of **CSC_IWDT_EN**, **CSC_SLP_IWDT** and **CSC_STP_IWDT** are locked control by the **IWDT_LOCK** register. Another, the following registers of **CSC_RTC_EN**, **CSC_SLP_RTC** and **CSC_STP_RTC** are locked control by the **RTC_LOCK** register. Refer the "Register Protect and Lock" in IWDT and RTC chapter for more information.

5.7. System Clock Control

5.7.1. System Clock Source

There are four clock sources for the system application: Internal High-frequency RC Oscillator (**IHRCO**), Internal crystal oscillator (**XOSC**), Internal Low-frequency RC Oscillator (**ILRCO**) and External Clock Input (**EXTCK**). Software can select the one of the four clock sources by application required and switches them on the fly. But software needs to settle the clock source stably before clock switching.

- **IHRCO and ILRCO Clock**

The chip is embedded one Internal High-frequency RC Oscillator (**IHRCO**) and one Internal Low-frequency RC Oscillator (**ILRCO**). It always boots from **IHRCO** on 12-MHz or **ILRCO** on 32-KHz by configurable hardware option byte (**OB**).

The built-in **IHRCO** provides two kinds of frequency for software selected by setting **CSC_IHRCO_SEL** register. One is 12MHz and another frequency is 11.059MHz. Both two kinds of frequency in **IHRCO** provide high precision frequency for system clock source. User can enable or disable the **IHRCO** circuit by setting **CSC_IHRCO_EN** register.

One IHRCOF (**CSC_IHRCOF**) flag and One ILRCOF (**CSC_ILRCOF**) flag will be set by hardware to indicate the clock output of IHRCO and ILRCO Oscillators are stable.

- **XOSC Clock**

The chip is embedded one internal crystal oscillator (**XOSC**) for external Xtal circuit. The **XOSC** oscillator is support the external Xtal frequency for 32K-Hz or 4 to 25-MHz. In the **XOSC** crystal mode, user needs to configure the **XOSC** pins of **XIN** and **XOUT** as external crystal input and output by setting GPIO AFS registers. Also user can reserve the **XOSC** crystal pins as GPIO function by IO AFS setting for application.

When both **XIN** and **XOUT** pins are configured as external crystal input and output, the internal **XOSC** oscillator will be enabled by hardware and the external crystal is oscillating. One XOSCF (**CSC_XOSCF**) flag will be set by hardware to indicate the external crystal oscillating is stable for software to switch the clock source. Software must poll this bit before switching the crystal oscillator as system clock source.

User can adjust the **XOSC** oscillator control gain for different external Xtal by setting **CSC_XOSC_GN** register.

The chip can boot to enable the **XOSC** crystal and configure **XIN** and **XOUT** pins as external crystal input and output by configurable hardware option byte (**OB**).

Refer the section of "[Xtal Oscillating Circuit](#)" for more information.

- **EXTCK Clock**

The chip supports to input external clock frequency up to 36MHz. In the external clock input **EXTCK** mode, the clock source comes from **XIN** pin. User needs to configure it as external crystal input pin by IO AFS setting.

- **Missing Clock Detector**

One missing clock detector (**MCD**) is built-in to monitor the clock of internal **XOSC** oscillator with external Xtal. The **MCD** is default enabled and user can disable it by setting **CSC_MCD_DIS** register. When it is enabled and user selects the clock source from **XOSC** for CK_HS or CK_LS clock select MUX, the **MCD** will start and monitor the clock source status. When **MCD** has detected the missing clock event, the CSC will automatically assert the MCDF flag and switch to IHRCO clock for CK_HS clock select MUX or to ILRCO clock for CK_LS clock select MUX.

User can configure the missing clock detection duration cycle time in **CSC_MCD_SEL** register.

5.7.2. High Speed Clock and Low Speed Clock

The high speed clock **CK_HS** is obtained from one of these four clock sources of **IHRCO**, **ILRCO**, **XOSC** and **EXTCK** through the high speed clock selector by setting **CSC_HS_SEL** register. The high speed clock selector is default set to **CK_IHRCO** or **CK_ILRCO** by hardware option byte (**OB**) setting. The **CK_HS** can send to the PLL as the reference clock input or directly output as one of the system main clock **CK_MAIN** by the selection of main clock selector.

There is another high speed clock **CK_HS2** which is obtained from one of these three clock sources of **IHRCO**, **XOSC** and **EXTCK** through the high speed clock selector by setting **CSC_HS2_SEL** register. Usually this clock is using for USB application, user can select the clock **CK_HS** as system main clock **CK_MAIN** for CPU system clock and select the clock **CK_HS2** as PLL input clock **CK_PLLIX** for generation USB request 48MHz clock.

The low speed clock **CK_LS** is obtained from one of these three clock sources of **ILRCO**, **XOSC** and **EXTCK** through the low speed clock selector by setting **CSC_LS_SEL** register. The low speed clock selector is default set to **CK_ILRCO**. The **CK_LS** can send to the RTC and TMx modules as the timer clock for low speed operation.

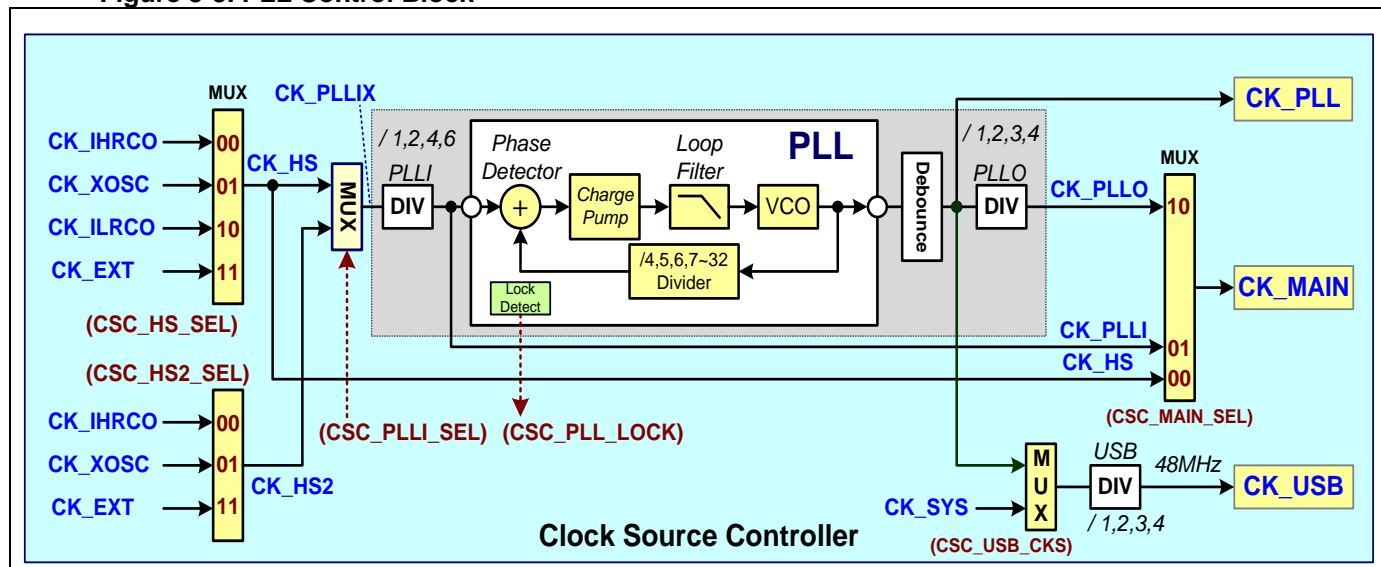
5.7.3. PLL Clock

One PLL is embedded to multiply the frequency of high speed clock source **CK_HS** or **CK_HS2**. It can be enabled by setting **CSC_PLL_EN** register. The clock frequency of **CK_HS** or **CK_HS2** will be divided by the PLL input divider and outputs as the PLL reference clock input **CK_PLLI**. User must set the valid divider value of 1, 2, 4, and 6 to fit the PLL input frequency range by setting **CSC_PLLI_DIV** register.

As the chip operation speed limitation, the PLL output clock can be down-converted by divided of 1, 2, 3 and 4 through the PLL output divider in **CSC_PLLO_DIV** register. It outputs the clock of **CK_PLLO** as one of the system main clock **CK_MAIN** by the selection of main clock selector.

The following diagram is showing the PLL control block. The CK_USB clock path is only supported for MG32F02U series chips. The Lock Detect block function is not supported for MG32F02A128/U128/A064/U064/V032.

Figure 5-3. PLL Control Block



There is a clock input multiplex to select the clock source **CK_HS** or **CK_HS2** for PLL input clock **CK_PLLI**. User can select the clock source in **CSC_PLLI_SEL** register. The PLL input frequency range is about 4~8 MHz, so user must set the valid divider value to fit this frequency range by setting **CSC_PLLI_DIV** register. Also the PLL output clock frequency of **CK_PLL** can be up over 144-MHz. Refer the “PLL Characteristics” of related chip Data Sheet for more information about the PLL VCO frequency information.

The chip supports two types of PLL frequency multiplication by setting **CSC_PLL_MDS** register. When the register is selected ‘MUL’, the PLL can multiply the PLL input clock frequency by 16 or 24 in **CSC_PLL_MUL** register that it is as same as the descriptions of previous paragraphs. When the register is selected ‘MULX’, the PLL can multiply the PLL input clock frequency from 4 to 32 in **CSC_PLL_MULX** and **CSC_PLL_MULS** registers under the PLL output frequency limitation.

The PLL VCO output frequency is as following formula.

- **MG32F02A128/U128/A064/U064/V032**

$$\text{PLL VCO Output Frequency} = \text{CK_PLLI} * (N+1) \quad N : \text{CSC_PLL_MULX register value}$$

- **MG32F02N128/K128/N064/K064**

$$\text{PLL VCO Output Frequency} = \text{CK_PLLI} * (P*2+S) \quad \begin{array}{l} P : \text{CSC_PLL_MULX register value} \\ S : \text{CSC_PLL_MULS register value} \end{array}$$

The previous formula must fit the PLL input clock frequency range, the PLL VCO output clock frequency range and PLL multiplication value range. Please refer the related chip Data Sheet about PLL input clock frequency range and PLL VCO output clock frequency range.

For MG32F02U series, the USB SIE process clock can come from the PLL. The PLL clock output can be divided by the USB divider by setting **CSC_USB_DIV** register.

5.7.4. Internal System Clocks

- **System Main Clock**

The system main clock **CK_MAIN** is obtained from **CK_HS** through the PLL clock dividers, PLL and system main clock selector. The system main clock **CK_MAIN** can select from one of these three clock sources of **CK_HS**, **CK_PLLI** and **CK_PLLO** through the main clock selector by setting **CSC_MAIN_SEL** register. The clock is used for internal control logic and the clock source of AHB and APB for peripheral modules.

The user can program the PLL input divider and output divider control bits to get the desired system main clock. The system main clock selector is default set to **CK_HS** and is bypassing the PLL dividers.

- **AHB Clock**

The clock of **CK_AHB** provides the operation clock for AHB devices and divides by 1, 2, 4, 8, 16, 32, 64, 128, 256 and 512 from the APB clock of **CK_APB** through the AHB prescaler by setting **CSC_AHB_DIV** register.

- **APB Clock**

The clock of **CK_APB** provides the operation clock for APB devices and divides by 1, 2, 4, 8 and 16 from the system main clock **CK_MAIN** through the APB prescaler by setting **CSC_APB_DIV** register. This clock is also to do as the system clock of **CK_SYS** for internal logic.

- **Unit Clock**

The clock of **CK_UT** provides the unit clock time about 1ms for internal modules of WWDT, RTC and I2C. It can divide from the **CK_ILRCO** clock by 8, 16, 32 and 128 through the unit clock divider by setting **CSC_UT_DIV** register.

- **System Tick Clock**

The ARM® CPU is built-in a 24-bit down-count system tick timer (**SysTick**). The clock of **CK_ST** provides the external reference clock input for the system tick timer if the control bit of **CLKSOURCE** is set in the CPU register of **SYST_CSR**. It can select the clock source from the CPU clock **HCLK** divided by 8 or **CK_LS** divided by 2 by setting **CSC_ST_SEL** register.

5.8. Module Process Clock Control

The CSC module is able to do the process clock enable setting and select the process clock source for internal modules. User must select the module process clock and enable the module process clock before configure the module for operation normal.

5.8.1. Module Process Clock Select

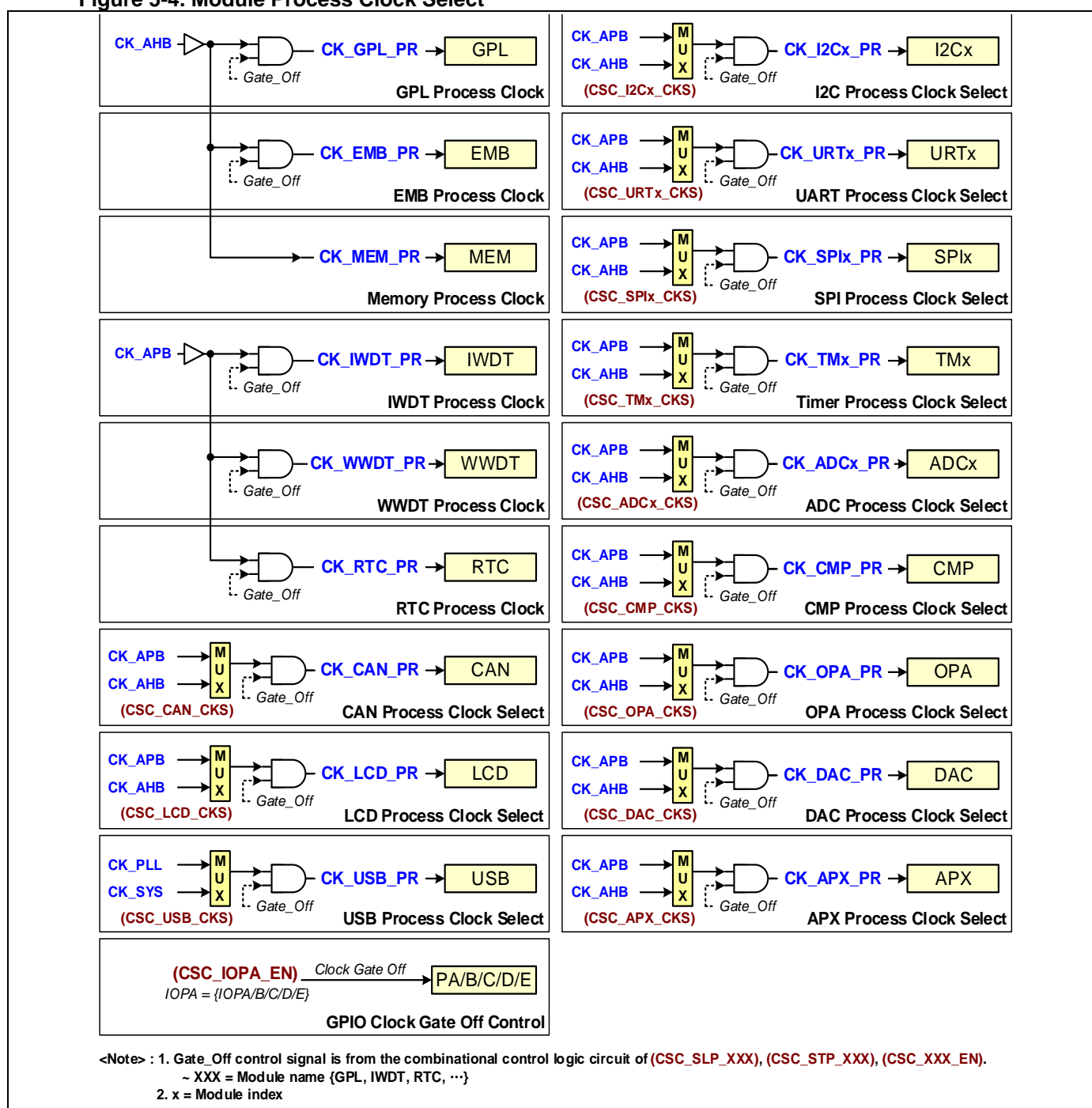
User can select the process clock source of a module which is coming from AHB clock (**CK_AHB**) or APB clock (**CK_APB**). User can select the module process clock independently for each module by setting the independent register of **CSC_XXX_CKS**. ('xxx': internal module name like as IWDT, RTC, and CMP ...)

Refer the register descriptions for more information about the module process clock select bits.

The following diagram is showing the process clock selection of modules.

[Notify]: Please refer the table of "[Chip Implementation Table](#)" in the section of "Applicable Chips" about MCU chip supported peripheral modules.

Figure 5-4. Module Process Clock Select



5.8.2. Module Process Clock Enable

User can enable the module process clock independently for each module by setting the independent register of **CSC_xxx_EN**.

The chip also support the clock enable preset function for **SLEEP** and **STOP** modes. User can plan to enable the process clock for keeping operation in **SLEEP** or **STOP** operation mode beforehand entering **SLEEP** or **STOP** mode by setting **CSC_SLP_xxx** and **CSC_STP_xxx** registers. Only the IWDG and RTC modules support the clock enable preset function for **STOP** mode. ('xxx': internal module name like as IWDG, RTC, and CMP ...)

Refer the register descriptions for more information about the module process clock enable and **SLEEP** / **STOP** modes' clock enable bits. The following table is showing module running state by different power mode and related clock control registers' setting.

Table 5-2. Module Running and Clock Enable Control

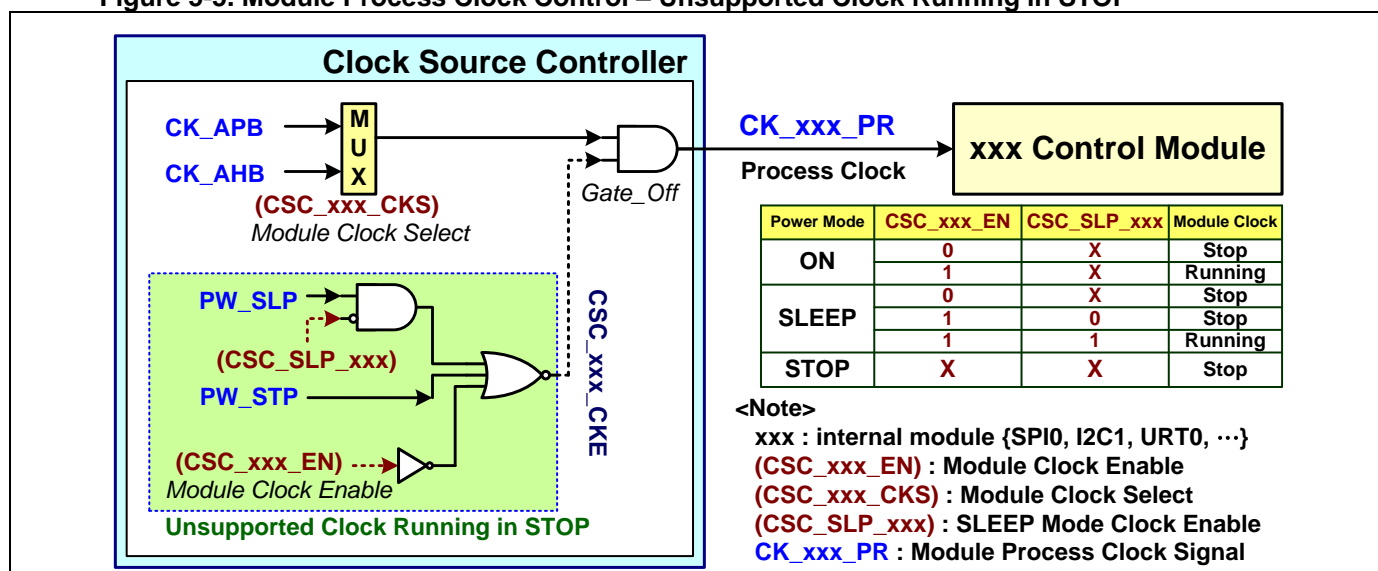
Power Mode	Signal		Module Support Running in STOP	Module Clock Enable Register	Clock Enable Register in SLEEP/STOP Mode		Module State
	PW_SLP	PW_STP		CSC_xxx_EN	CSC_SLP_xxx	CSC_STP_xxx	
ON	0	0	x	0	x	x	Stop
			x	1	x	x	Running
SLEEP	1	0	x	0	x	x	Stop
			x	1	0	x	Stop
			x	1	1	x	Running
STOP	0	1	No	x	x	x	Stop
			Yes	0	x	x	Stop
			Yes	1	x	0	Stop
			Yes	1	x	1	Running

<Sign> x: Don't care, xxx: Module Name {ADC,I2C0,SPI0,TM36, ...}

● Unsupported Clock Running in STOP

The following diagram is showing the process clock enable control of modules which are unsupported clock running in STOP mode.

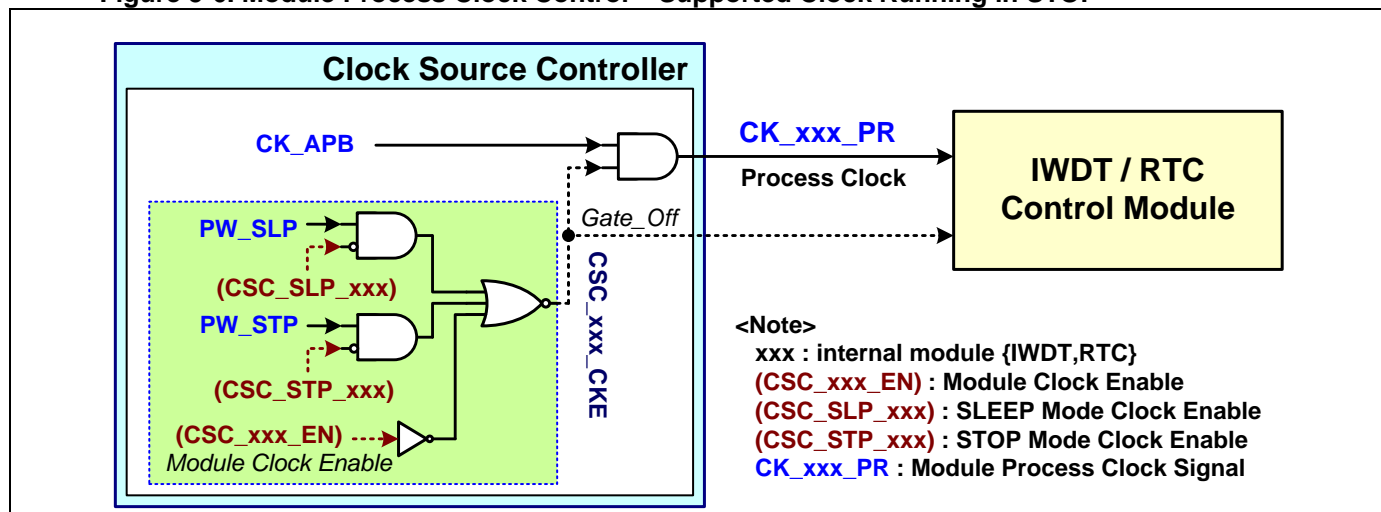
Figure 5-5. Module Process Clock Control – Unsupported Clock Running in STOP



- Supported Clock Running in STOP

The following diagram is showing the process clock enable control of IWDT and RTC modules which are supported clock running in STOP mode.

Figure 5-6. Module Process Clock Control – Supported Clock Running in STOP



The following table is the available of clock enable control of modules by register setting for **ON/SLEEP/STOP** operation modes.

Table 5-3. Modules' Available Clock Enable Register

Module Name	Register		
	CSC_XXX_EN	CSC_SLP_XXX	CSC_STP_XXX
EMB	V	V	
Px	V		
GPL	V		
DMA	V		
I2C0/1	V	V	
SPI0	V	V	
URT _x	V	V	
USB	V	V	
CAN	V	V	V
TM0 _x	V	V	
TM1 _x	V	V	
TM2 _x	V	V	
TM3 _x	V	V	
ADC0	V	V	
CMP	V	V	
DAC	V	V	
LCD	V	V	V
OPA	V	V	
IWDT	V	V	V
WWDT	V	V	
RTC	V	V	V
APX	V	V	

<Note> Px: GPIO port modules = {PA, PB, PC, PD, PE}

5.8.3. Device Clock Enable Control in Power-Down Modes

The CSC controller supports the power control planning ability for the internal devices in **SLEEP** or **STOP** modes. User can plan the internal devices are independently power-on or power-off beforehand entering **SLEEP** or **STOP** modes.

The following table is showing the power control of internal devices in different operation modes.

Table 5-4. Internal Device Clock Control

Internal Device	ON	SLEEP	STOP
ARM32 Cortex-M0	on	off	off
System Core Clock (AHB,APB)	on	on	off
SWD/Debug	CSC Reg setting	CSC Reg presetting	CSC Reg presetting
Ext INT pins	CSC Reg setting	interrupt enable setting	interrupt enable setting
Analog Comparator	CSC Reg setting	CSC Reg presetting	CSC Reg presetting
RTC	CSC Reg setting	CSC Reg presetting	CSC Reg presetting
IWDT	CSC Reg setting	CSC Reg presetting	CSC Reg presetting
LCD	CSC Reg setting	CSC Reg presetting	CSC Reg presetting
I2Cx	CSC Reg setting	CSC Reg presetting	off (slave address detect using external SCL clock)
Other Modules	CSC Reg setting	CSC Reg presetting	off

5.9. Module Event Clock

The peripheral modules' event clock is used as the clock source of module internal timers, timing generator and related control logic. The following table is the available input event clock sources table of modules.

[Notify]: Please refer the table of “[Chip Implementation Table](#)” in the section of “Applicable Chips” about MCU chip supported peripheral modules.

Table 5-5. Modules' Available Input Event Clock Sources Table

Module Name	High Speed Clock		Low Speed Clock			Timer Output as Clock		NCO
	CK_AHB	CK_APB	CK_LS	CK_ILRCO	CK_UT	TM00_TRGO	TM01_TRG1	NCO_P0
SYS	V							
CFG	V							
PW	V							
RST	V							
MEM	V							
EMB	V							
IOP	V							
Px	V			V		V		
AFS	V							
GPL	V							
DMA		V						
EXIC		V						
I2C0/1	V	V			TMO	V		
URTx	V	V	V			V		V
SPI0	V	V				V		
USB		V						
CAN	V	V	V					V
TM0x	V	V	V					V
TM1x	V	V	V					V
TM2x	V	V	V					V
TM3x	V	V	V					V
ADC0	V	V				V	V	
CMP	V	V						
DAC	V	V						
LCD	V	V	V				V	V
OPA	V	V						
IWDT				V				
WWDT		V			V			
RTC		V	V		V		V	
APB		V						
APX	V	V						

<Note>

Px: GPIO port modules = {PA,PB,PC,PD,PE} ; the selected clock is using for filter only

CK_LS: clock source from {CK_XOSC, CK_ILRCO or CK_EXT}

TMO: use CK_UT as one of I2C time-out timer clock source

5.10. Internal Clock Output Control

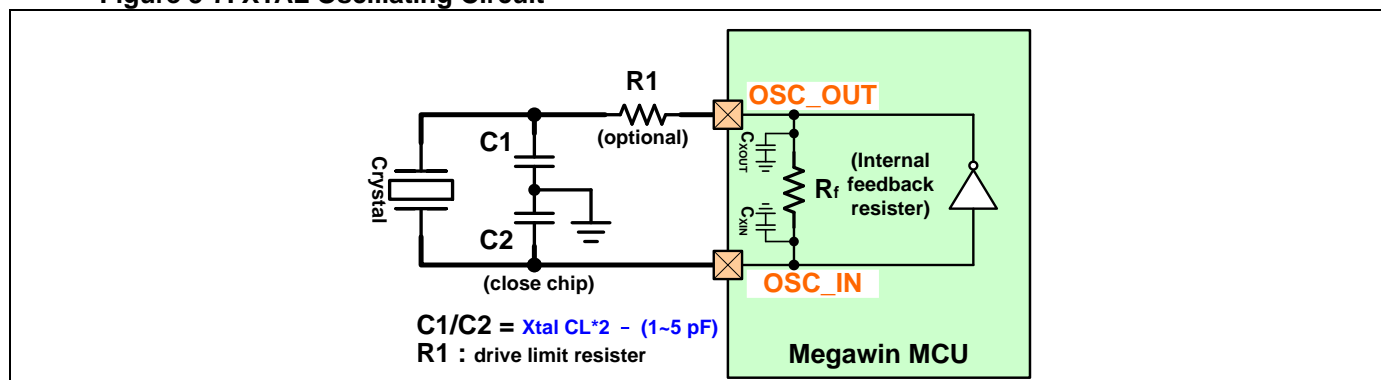
The CSC module is able to output the internal system clock of **CK_MAIN**, **CK_AHB**, **CK_APB**, **CK_HS**, **CK_LS** and **CK_XOSC** to external from **ICKO** pin. User can enable the output function in **CSC_CKO_EN** register and select the output clock source by setting **CSC_CKO_SEL** register.

For the speed limitation of **ICKO** pin, user can divide the frequency of selected clock source by 1, 2, 4 and 8 through the internal clock output divider by setting **CSC_CKO_DIV** register.

5.11. Crystal Oscillating Circuit

To achieve successful and exact oscillating (up to 24MHz), the capacitors **C1** and **C2** are necessary, as shown in following diagram. Normally, **C1** and **C2** have the same value.

Figure 5-7. XTAL Oscillating Circuit



The following table lists the chip internal total equivalent capacitance (C_{XIN} / C_{XOUT}) of internal oscillator circuit, bounding pad, bounding wire, lead frame. Please refer the “Xtal Oscillating Circuit” section of related chip Data Sheet about the capacitance (C_{XIN} / C_{XOUT}).

Table 5-6. Internal Total Equivalent Capacitance Example for XOSC circuit

Component	Capacitance Value
C_{XOUT}	1.5pF (0.9~2.0pF)
C_{XIN}	2.3pF (2.2~2.4pF)

The following table lists the suggested **C1** & **C2** value for the different capacitor load crystal application. Refer the capacitor load (**CL**) value in crystal manufacture specification for the final matching capacitor of **C1** & **C2**.

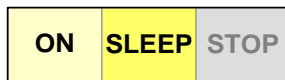
Table 5-7. Reference Capacitance of C1 & C2 for crystal oscillating circuit

Crystal C Load	C1, C2 Capacitance
12.5pF	20pF (18~22pF)
20pF	36pF (33~39pF)
32pF	62pF (56~62pF)

About the crystal placement and routing, the component placement needs to place optional feedback resister at the first, crystal capacitors (C1/C2) at the second and crystal component at the last on the path from MCU to crystal component. Others, suggests an independent ground plane with guard ring for the crystal circuits and avoids any other signal is routing through this area on top or bottom side.

6. System Common Control

6.1. Introduction



The module can be running in ON and SLEEP modes only.

The chip embeds one system control (SYS) module for system common control. It is including of one system event interrupt global enable control, chip manufacture identification code.

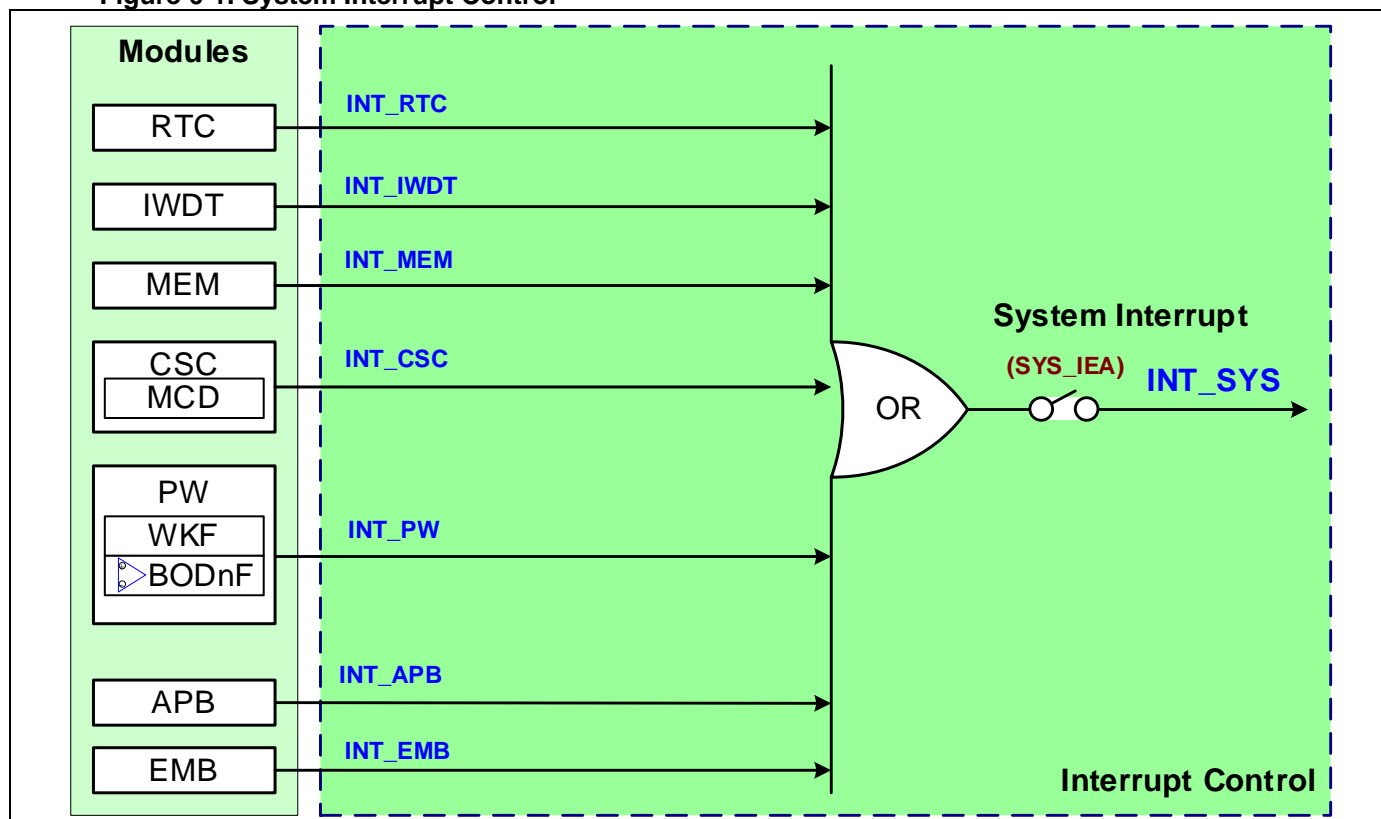
6.2. Features

- System interrupt global enable control for system interrupt source
- Chip manufacture identification code - Device ID, Product ID, User ID, Module Options
- 32-bit non-reset backup register

6.3. Interrupt and Event

The system interrupt global enable control from the interrupt events of RTC, MEM, CSC, PW, IWDT, APB, EMB modules. The system interrupt signal **INT_SYS** is sent to External Interrupt Controller (EXIC) to do as an interrupt event. **SYS_IEA** is the system interrupt global enable register.

Figure 6-1. System Interrupt Control



The following table is showing the system interrupt control source.

Table 6-1. System Interrupt Control Source

Chip	System Interrupt Control Source						
	IWDT	PW	RTC	CSC	APB	MEM	EMB
MG32F02A128/A064 MG32F02U128/U064	V	V	V	V	V	V	V
MG32F02V032 MG32F02N128/N064 MG32F02K128/K064	V	V	V	V	V	V	

<Note> V: Implemented

6.4. Chip Manufacture ID

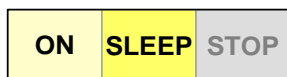
User can read the **SYS_MID** register to get the chip manufacture identification code about Device ID, Product ID, User ID and Module Options. The manufacture ID code is only for MG internal using.

6.5. System Backup Register

This 32-bit register is used for application firmware without any hardware control. It can be written or read but not reset by POR or other cold/warm reset.

7. System Memory

7.1. Introduction



The module can be running in ON and SLEEP modes only.

The chip has separate address spaces for program and data memory. The logical separation of program and data memory allows the memory to be accessed by 32bit addresses, which can be quickly stored and manipulated by the CPU. The chip supports one memory controller (MEM) to manage the internal flash memory and SRAM access operation.

7.2. Features

7.2.1. Embedded Memory

- Built-in embedded max. 128K bytes main flash memory for application code
 - Main flash is shared for AP, IAP and ISP flash
 - Provide fixed 512 bytes ISPD flash memory as ISP private data
- Built-in embedded max. 16K bytes main SRAM
 - Support private 2K bytes option for DMA to improve CPU access performance
- Built-in embedded independent 512 bytes SRAM as USB packet buffer for MG32F02U series

7.2.2. Memory Controller

- Support ICP (In-circuit program) for ISP boot code update through SWD interface
- Support ISP (In-system program) for application code update
 - Support programmable ISP flash memory size for ISP boot code
- Support IAP (In-application program) for application data update
 - Support programmable IAP flash memory size
- Support flash memory page erase in 512 bytes

7.3. Implementation

7.3.1. Embedded Memory Implementation

The following table is showing the implemented embedded memory of chips.

Table 7-1. Embedded Memory Implementation

Chip	Embedded Flash (total = Main + ISPD Flash)			Embedded SRAM (Main + DMA + USB SRAM)		
	Main Flash	ISPD Flash	Flash Page Size	Main SRAM	DMA SRAM	USB SRAM
MG32F02A128 MG32F02N128 MG32F02K128	128KB	512B	512B	14KB	2KB	-
MG32F02U128	128KB	512B	512B	14KB	2KB	512B
MG32F02A064 MG32F02N064 MG32F02K064	64KB	512B	512B	8KB	2KB	-
MG32F02U064	64KB	512B	512B	14KB	2KB	512B
MG32F02V032	32KB	512B	512B	4KB	-	-

<Note>

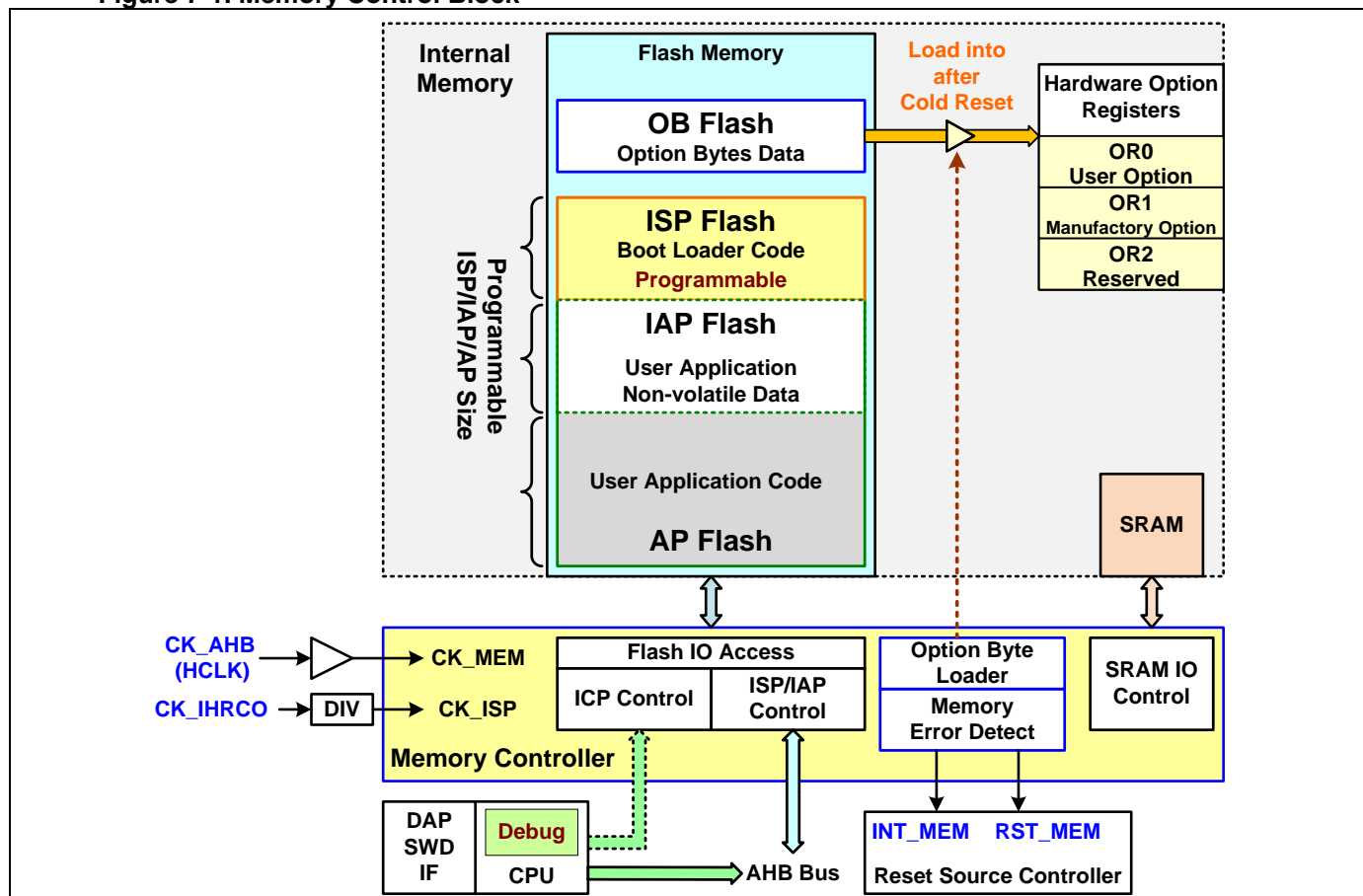
Main Flash is shared for AP, IAP and ISP. ISPD Flash is the independent flash for ISP.
Main SRAM is used for application software. DMA can access all of the Main, DMA and USB SRAM.
DMA SRAM is the private DMA access buffer to improve the DMA data transaction performance.
USB SRAM is the private and independent buffer for USB packet buffer.
Both DMA SRAM and USB SRAM can also use as general SRAM for software using.

7.4. Memory Controller

A memory controller is supported to access on chip flash memory, SRAM on AHB bus. It includes **ICP** (In-Circuit Programming)/ **ISP** (In-System Programming)/ **IAP** (In-Application Programming) circuits for flash memory accessing, option byte loader for hardware option registers loading and an external memory bus EMB interface with the capability of accessing external program memory.

The following diagram is showing the Memory controller block.

Figure 7-1. Memory Control Block



7.5. Enabling and Clock

There is a global enable bit of **MEM_EN** for all functions of this module. When this bit is disabled, all the MEM functions are not working.

The module process clock is using for the interface control logic between AHB bus and memory controller. It is always coming from CSC (Clock Source Controller) module and is no any gating off control register.

7.6. Interrupt and Event

There are two signals of **INT_MEM** and **RST_MEM** to be generated in this MEM control module. **INT_MEM** sends to External Interrupt Controller (EXIC) to do as an interrupt event. **RST_MEM** sends to Reset Source Controller to do as a reset event.

7.6.1. Memory Controller Interrupt Control and Reset

● Interrupt Events

INT_MEM signal sends to External Interrupt Controller (EXIC) to do as an interrupt event. These interrupt flags are using for the interrupt service routine (ISR) flow control. Generally these flags are set by hardware and software clears them when the serviced job is completed in the related ISR. Each interrupt flag has one interrupt enable bit. User can enable or disable it. There is one interrupt all enable bit of **MEM_IEA** to enable or disable all the interrupt sources for this module.

There are some status bits those are reading only to provide internal control status. One busy flag of **MEM_FBUSYF** is used to indicate the flash memory access busy status. One error flag of **MEM_IAPSEF** is used to indicate IAP flash memory size setting error. Refer the register descriptions of the related status bits for more information.

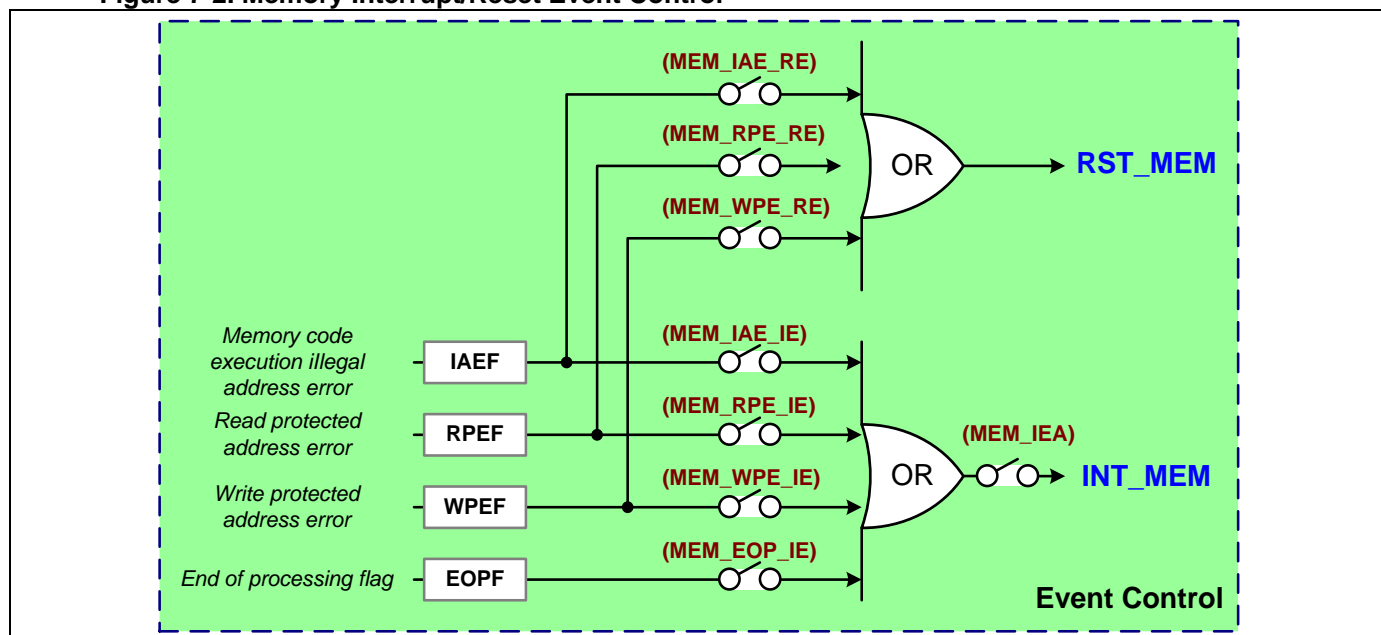
● Reset Events

RST_MEM signal sends to Reset Source Controller (RST) to do as the warm reset events or cold reset events. These reset events can be enabled to reset the chip by setting the registers in RST.

The MEM can detect the memory code execution illegal address error, flash memory write protect error and flash memory read protect error. There are independent reset event enable bits of **MEM_IAE_RE**, **MEM_WPE_RE** and **MEM_RPE_RE** for these reset events.

Refer the descriptions of System Reset chapter for more information about the reset events and control.

Figure 7-2. Memory Interrupt/Reset Event Control



7.6.2. MEM Interrupt Flags

Generally these interrupt flags are set by hardware and clear by software writing 1. Refer the register descriptions for more information about the related interrupt flags and interrupt enable bit.

● EOPF

Flash memory end of processing flag (**MEM_EOPF**). There is a related interrupt enable register bit of **MEM_EOP_IE**. This flag indicates the previous flash memory access command is finished.

- **IAEF**

Memory code execution illegal address error detection flag (**MEM_IAEF**). There is a related interrupt enable register bit of **MEM_IAE_IE**.

- **WPEF**

Flash memory write protect error detection flag (**MEM_WPEF**). There is a related interrupt enable register bit of **MEM_WPE_IE**. When write or erase the flash memory, this flag will be asserted if the operated command setting, address area is error or IHRCO device is disabled.

- **RPEF**

Flash memory read protect error detection flag (**MEM_RPEF**). There is a related interrupt enable register bit of **MEM_RPE_IE**. When read the flash memory, this flag will be asserted if the operated command setting or address area is error.

7.7. Register Protect and Lock

After memory controller domain reset, all MEM registers are write-access protected except **MEM_STA**, **MEM_KEY** registers. Write **0xA217** value to the **MEM_KEY** register to unprotect the registers and process the control continuously. Oppositely write other value except **0xA217** value to protect the registers. Read the **MEM_KEY** register to get the register value is Protected (=1) or Unprotected (=0).

Specially, the registers of **MEM_ISP_WEN** and **MEM_ISP_REN** are write-access protected by **MEM_KEY2** register. Write **0xA217** value to the **MEM_KEY2** register to unprotect the registers and process the control continuously. Oppositely write other value except **0xA217** value to protect the registers. Read the **MEM_KEY2** register to get the register value is Protected (=1) or Unprotected (=0).

Refer the table and descriptions of “[Register Protect and Lock](#)” in System Reset chapter for more information.

7.8. Boot and On-Chip Memory

7.8.1. Memory Boot Modes

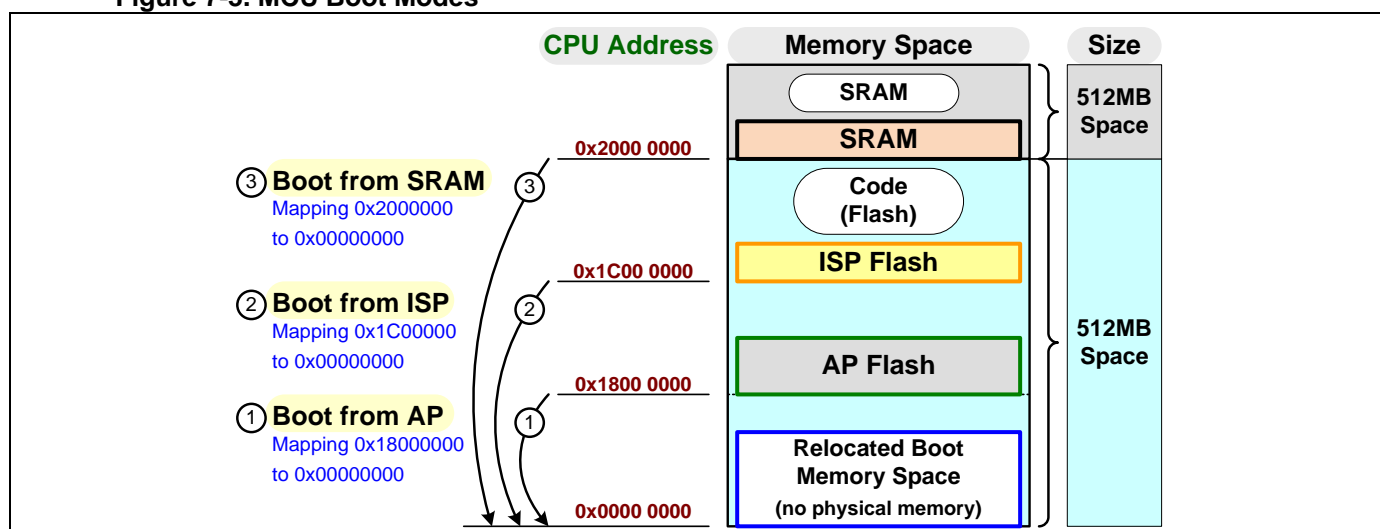
The chip can select the boot memory to boot from User Application Program (AP) Flash, In-System-Program (ISP) and embedded SRAM. During chip startup, the hardware configuration option-byte (**OB**) is loaded to **MEM_BOOT_MS** register and uses to select one of the three boot options as following table.

Table 7-2. Memory Boot Mode Select

Boot Mode	Memory	Register
		BOOT_MS
AP	Application Flash	0
ISP	ISP Boot Flash	1
SRAM	Embedded SRAM	2

<Note> The selected boot memory is re-mapped at 0x0000 0000 by hardware.

Figure 7-3. MCU Boot Modes



7.8.2. On-Chip Flash Memory

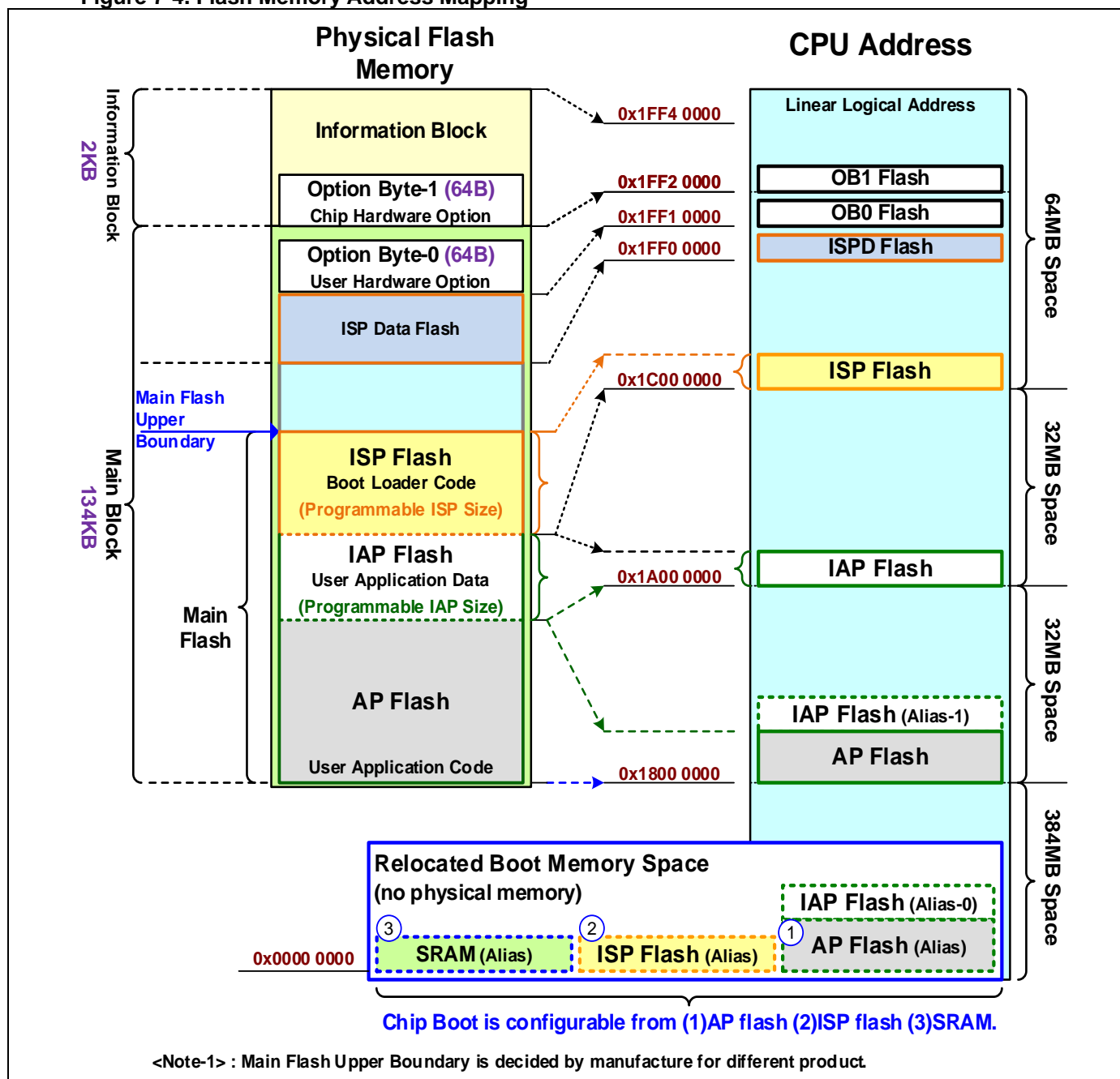
The main block is including of application program flash memory (**AP**-memory and **IAP**-memory), boot flash memory (**ISP**-memory), user configurable hardware option byte (**OB**) and ISP program private data flash memory (**ISPD**-memory). The **OB** flash memory and **ISPD** flash memory are located at the fixed upper 2K-byte space of main block. The information block is storing the internal chip and manufacture information.

There can be up to **128K** bytes of user program on-chip flash memory which is including of **AP**-memory, **IAP**-memory and **ISP**-memory. The extra flash memory of **ISPD**-memory is used as the data flash memory for ISP program privately.

These memory blocks are able to map to independent CPU address area. The relocated boot memory space (base address at 0x00000000) is remapping the AP flash, ISP flash or SRAM data content by boot configuration setting. It is no physical flash memory actually.

The following diagram is showing the flash memory address mapping for different flash memory block. The physical flash is embedded in chip.

Figure 7-4. Flash Memory Address Mapping



- **Boot Flash Memory**

ISP-memory is used to store the boot loader program for **ISP** (In-System Programming). When MCU is running in **ISP** region, MCU could modify the **AP** and **IAP** memory for software upgraded.

- **ISP Memory**

The flash of **ISP** memory is located at the base address of 0x1C000000. There can be programmable size and up to whole flash memory size and the **IAP** memory size is defined by the value of **CFG_ISP_SIZE** register. The **CFG_ISP_SIZE** register value is configured by hardware option byte (**OB**) flash.

When the chip configures to boot-from-ISP, the **ISP**-memory will be relocate at the base address of 0x00000000 and the content is the duplicated of the physical flash memory at the base address of 0x1C000000.

- **ISPD Memory**

The fixed 1K bytes ISPD flash memory is a private data flash memory for **ISP** function (boot from **ISP** mode). Also the ISPD is able using for boot-from-AP or boot-from-SRAM mode by setting **MEM_ISPD_WEN** and **MEM_ISPD_REN** registers. The **MEM_ISPD_WEN** is only able to change when boots from ISP mode. The **MEM_ISPD_REN** is only able to set when boots from **ISP** mode. And it can clear or disable for all boot modes.

- **Program Flash Memory**

The program flash memory is partitioned into **AP**-memory and **IAP**-memory. **AP**-memory is used to store user's application program and **IAP**-memory is used to store the non-volatile application data. If MCU is running in **AP** region, software could only modify the **IAP** memory for storage data updated.

- **AP Memory**

Application program memory is the memory which stores the program codes for the CPU to execute. The flash of **AP** memory is located at the base address of 0x18000000.

When the chip configures to boot-from-AP, the **AP**-memory will be relocate at the base address of 0x00000000 and the content is the duplicated of the physical flash memory at the base address of 0x18000000.

After reset and the chip is boot-from-AP, the CPU begins execution from the location of reset interrupt vector (0x00000004) addressing, where should be the starting of the user's application code. To service the interrupts, the interrupt service locations (called interrupt vectors) should be located in the address 0x000000C0~0x00000000.

- **IAP Memory**

The flash of **IAP** memory is located at the base address of 0x1A000000. The **IAP** memory size is defined by the value of **MEM_IAP_SIZE** register. User can program this register in unit of 512-byte. Value 0 indicates the IAP memory size 0K-byte and value 1 indicates the IAP memory size 512-byte. This register write access is no effect when **MEM_IAP_AEN**=0. The **MEM_IAP_AEN** is used to do the **MEM_IAP_SIZE** register access enable bit. Refer the section of "[Embedded Memory Implementation](#)" about flash memory page size and related chip Register Definition Guide for **MEM_IAP_AEN** register setting.

The chip supports a code execution option for the IAP flash memory by setting **MEM_IAP_EXEC** register.

The **MEM_IAP_SIZE** register value is able to initialize by hardware option byte (**OB**) flash after power-on or cold reset or software programming.

- **Option Bytes Flash Memory**

There can be up to 64 bytes of on-chip Option Bytes Flash memory.

Option Bytes is used to store the hardware option configuration setting. For example, **CFG_BOOT_MS** for system cold reset program starting memory selection and **CFG_HS_SEL** for **CK_HS** default source selection.

7.8.3. On-Chip Data RAM

These is embedded SRAM up to 16K bytes as data memory. Also user can run code on the SRAM memory. The SRAM memory is located at the base address of 0x20000000. Refer the section of "[Embedded Memory Implementation](#)" or related chip Data Sheet for the detail information about embedded SRAM memory.

When the chip configures to boot-from-SRAM, the SRAM will be relocate at the base address of 0x00000000 and the content is the duplicated of the physical flash memory at the base address of 0x20000000.

In order to improve the performance of DMA data transaction, the SRAM memory space is designed to separate to two block – upper 2K-bytes SRAM and lower SRAM. Strongly suggests user to arrange the upper 2K-bytes SRAM for DMA transaction and lower SRAM for software using. The upper 2K-bytes SRAM is located at the fixed base address of 0x20003800. Refer the section of "[DMA SRAM Using](#)" for more information.

Certainly, user can plan any memory space of SRAM for software using or DMA transaction and no hardware limit.

7.9. Memory Controller Function

7.9.1. Flash Memory Access

The chip has up to 128K bytes of embedded main flash memory for code and data, programmable memory size of embedded system flash memory for boot load code and 64 bytes of embedded option-byte flash memory for chip configuration.

The memory controller (MEM) supports to Read/ Program (Write)/ Erase the flash memory by setting **MEM_MDS** register. User can directly read the data from flash memory by CPU read instruction commands and do not need through any register. For “Program” mode, MEM provides the 32-bit data write operation into flash memory for new data updated. For “Erase” mode, the Erase address is only valid at low 9-bit CPU address=0 (X...XX0 0000 0000B) and is addressing 512-byte alignment. If the low 9-bit address erased start address is not all '0', this erased process is no effect.

The following table is showing of the flash memory access control command and sequence key setting value.

Table 7-3. Flash Memory Access Control and Key

Flash	Access	Mode	Register			
			MDS	Access Enable	SKEY 1st	SKEY 2nd
AP Flash	Write	Single	0x1	AP_WEN=1	0x46	0xB9
		Multiple				0xBE
	Erase	Single	0x2			0xB9
		Multiple				0xBE
	Read	Both	directly read by memory access instructions			
IAP Flash	Write	Single	0x1	IAP_WEN=1	0x46	0xB9
		Multiple				0xBE
	Erase	Single	0x2			0xB9
		Multiple				0xBE
	Read	Both	directly read by memory access instructions			
ISPD Flash	Write	Single	0x1	ISPD_WEN=1	0x46	0xB9
		Multiple				0xBE
	Erase	Single	0x2			0xB9
		Multiple				0xBE
	Read	Both	-			ISPD_REN=1
Flash	Access	Mode	Register			
			MDS	Access Enable	SKEY2 1st	SKEY2 2nd
ISP Flash	Write	Single	0x5	ISP_WEN=1	0x9867	0xB955
		Multiple				0xBEAA
	Erase	Single	0x6			0xB955
		Multiple				0xBEAA
	Read	Both	-			ISP_REN=1

<Sign> Both: Single or Multiple data access

x: no protected key sequence request ; - : no necessary

-: directly access by memory instructions and has no register necessary

<Note> Write (**MEM_SKEY**) by any value to lock function for multiple write.

SRAM Program belongs to AP Program level.

If (**ISP_REN=0**), ISP flash only enable to read data when chip boot from ISP mode.

When user wants to program or erase the on-chip flash memory, the first needs to unlock the MEM register protection by setting **MEM_KEY** register and enable the MEM controller by setting **MEM_EN**. The second needs to set the flash access mode by setting **MEM_MDS** register and enable the flash write enable bit by setting **MEM_AP_WEN**, **MEM_IAP_WEN**, **MEM_ISP_WEN** or **MEM_ISPD_WEN** register. The third needs to unlock the flash access sequence key by setting **MEM_SKEY** or **MEM_SKEY2** register. Following user can do flash program or erase process.

The access sequence key of **MEM_SKEY** is used for AP/IAP/ISPD flashes and the access sequence key of **MEM_SKEY2** is used for ISP/OB flashes.

The following sections are describing the examples of AP/IAP/ISP/ISPD/OB flash program or erase sequence.

- **AP/IAP Flash Program/Erase Sequence**

- (1) Unlock Register Protection ~ (**MEM_KEY**) = 0xA217
- (2) Enable Memory Controller ~ (**MEM_EN**) = 1
- (3) Set Flash Write Enable ~ (**MEM_AP_WEN/MEM_IAP_WEN**) = 1
Set Flash Access Mode ~ (**MEM_MDS**) (=1: Write Flash, =2: Erase Flash)
- (4) Unlock Flash Access Sequence Key ~ (**MEM_SKEY**) = 0x46
- (5) Single Write Access or Multiple Write Access
 - (a) Single Write Enable ~ (**MEM_SKEY**) = 0xB9
 - (b) Multiple Write Enable ~ (**MEM_SKEY**) = 0xBE
- (6) Flash Access ~ Write Flash or Erase Flash commands
(Directly CPU write data access at CPU address for trigger Flash Program/Erase)
- (7) Repeat (6) if Multiple Write
- (8) Lock Flash Access ~ (a) Single Write ~ Hardware auto locked after Single Write Access
(b) Multiple Write ~ (**MEM_SKEY**) = 0x64 (any value)

- **ISP Flash Program/Erase Sequence**

- (1) Unlock Register Protection ~ (**MEM_KEY**) = 0xA217, (**MEM_KEY2**) = 0xA217
- (2) Enable Memory Controller ~ (**MEM_EN**) = 1
- (3) Set Flash Write Enable ~ (**MEM_ISP_WEN/MEM_ISPD_WEN**) = 1
Set Flash Access Mode ~ (**MEM_MDS**) (=5: Write Flash, =6: Erase Flash)
- (4) Unlock Flash Access Sequence Key ~ (**MEM_SKEY2**) = 0x9867
- (5) Single Write Access or Multiple Write Access
 - (a) Single Write Enable ~ (**MEM_SKEY2**) = 0xB955
 - (b) Multiple Write Enable ~ (**MEM_SKEY2**) = 0xBEAA
- (6) Flash Access ~ Write Flash or Erase Flash commands
(Directly CPU write data access at CPU address for trigger Flash Program/Erase)
- (7) Repeat (6) if Multiple Write
- (8) Lock Flash Access ~ (a) Single Write ~ Hardware auto locked after Single Write Access
(b) Multiple Write ~ (**MEM_SKEY2**) = 0x1234 (any value)

For flash memory read access, user can directly read the data from flash memory by CPU read instruction commands. User needs to set the read access wait state control in the register of **MEM_FWAIT** by different AHB operation clock **CK_AHB** frequency for the flash memory. This register is selected the latency timer of the AHB clock period to the flash access time as following list.

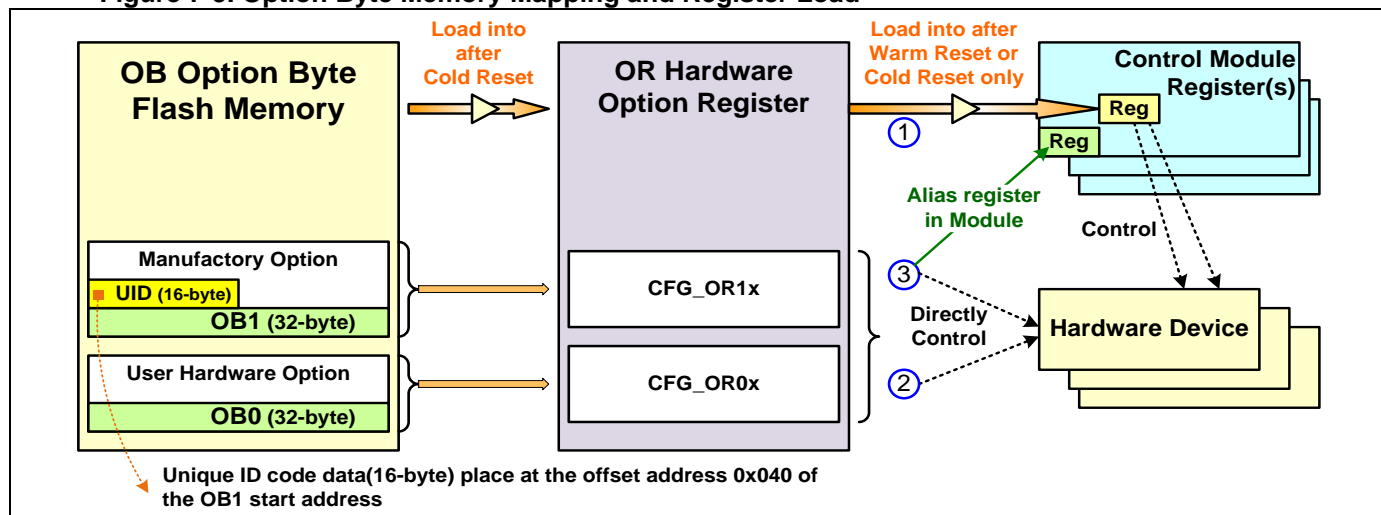
- Zero : Zero wait state if CK_AHB < 25 MHz
- One : One wait state if 25MHz <CK_AHB< 50 MHz

7.9.2. Hardware Option Byte Flash Memory

There can be up to 64 bytes of on-chip Option Bytes Flash memory. It is used to store the hardware option configuration setting.

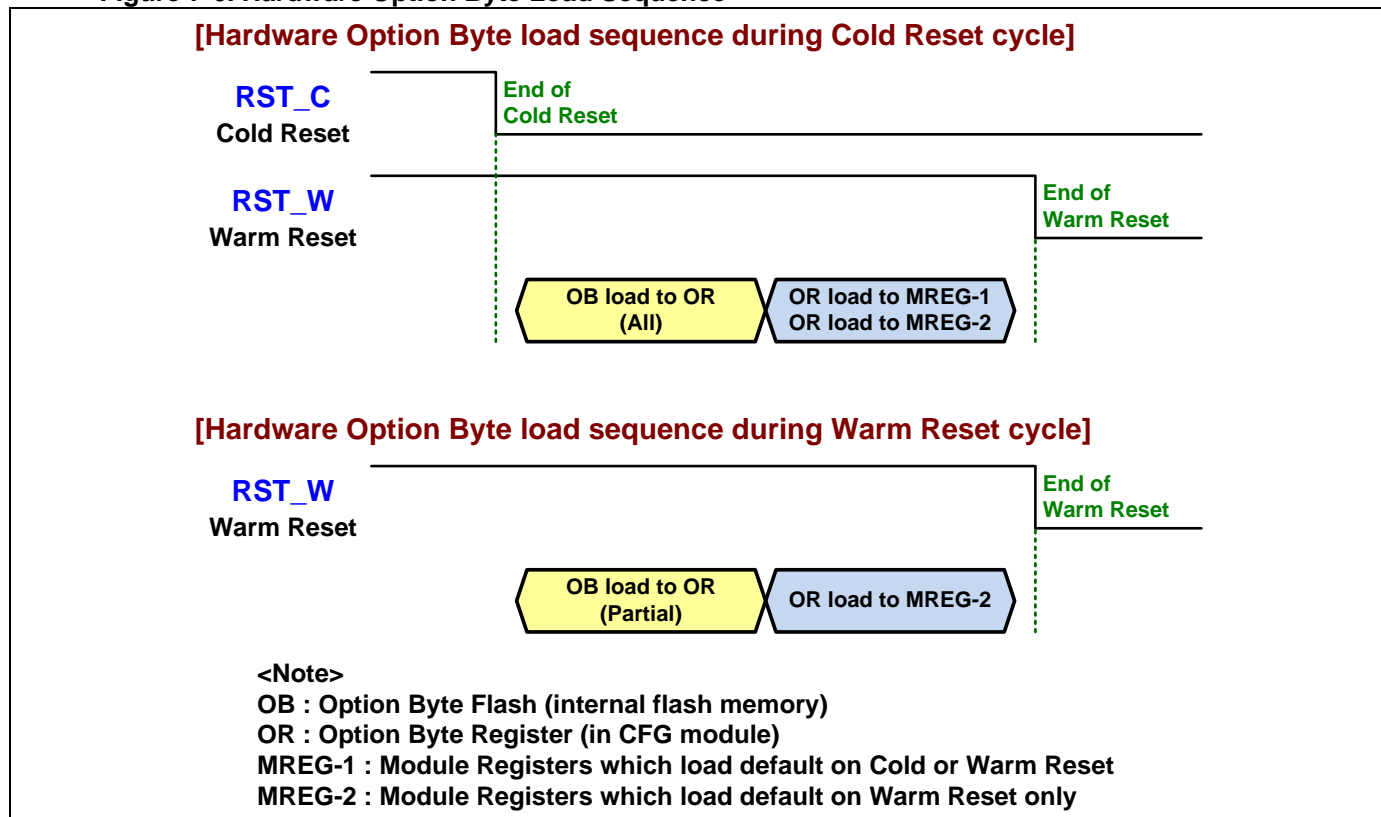
The following diagram is showing the flow and sequence that the Option Byte flash memory (**OB**) load into chip configuration Option Register (**OR**) and load into the module register during warm reset or cold reset cycle.

Figure 7-5. Option Byte Memory Mapping and Register Load



The following diagram is showing the Option Byte flash memory (**OB**) load timing during warm reset or cold reset cycle.

Figure 7-6. Hardware Option Byte Load Sequence

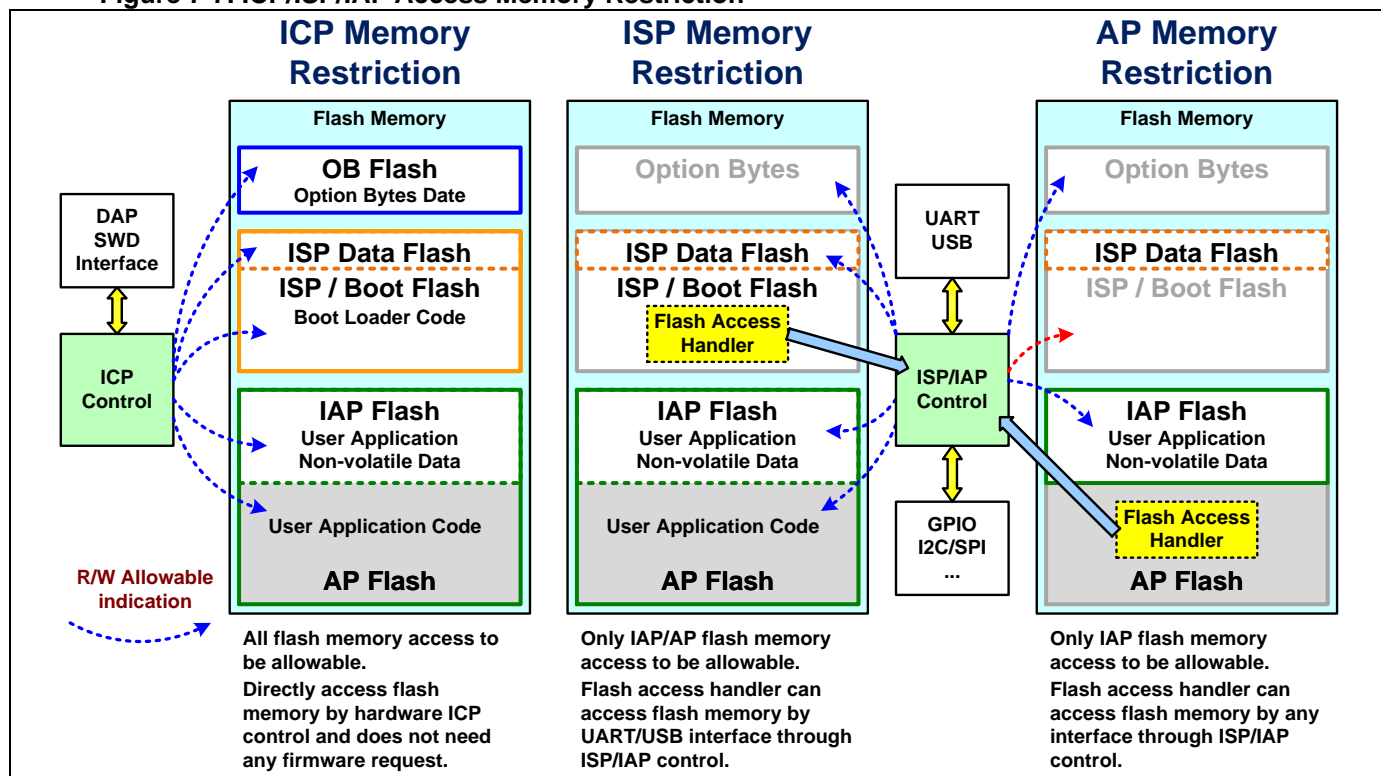


7.9.3. ICP/ISP/IAP for Flash Memory

There are 3 flash access modes are provided in chip for ICP, ISP and IAP application. ICP is allowed to update the entire contents of the flash memory by using the hardware SWD interface and no any firmware request. Others, User can use these two modes of ISP and IAP to update new data into flash storage and get flash content by a firmware flash memory access handler.

The following diagram is showing the ICP/ISP/IAP flash memory access and memory access restriction.

Figure 7-7. ICP/ISP/IAP Access Memory Restriction



7.9.4. Memory Access Restriction

When the chip boots from AP, ISP or SRAM memory, there are different memory access restrictions by hardware or user control. User can protect the write access of AP or IAP flash memory by setting **MEM_AP_WEN** or **MEM_IAP_WEN** register.

For ISP flash memory, user can protect the write or read access by setting **MEM_ISP_REN** or **MEM_ISP_WEN** register. Also user can protect the write or read access of ISP/DP flash memory by setting **MEM_ISPD_REN** or **MEM_ISPD_WEN** register. These registers of **MEM_ISP_REN**, **MEM_ISP_WEN** and **MEM_ISPD_REN** are only able to set when boots from ISP mode. And it can clear or disable for all boot modes. The register of **MEM_ISPD_WEN** is only able to change when boots from ISP mode.

The following table is showing the flash memory read/write access restriction with boot mode setting.

Table 7-4. Memory Access Restriction vs Boot Mode

Memory	CPU Start Address	Boot from			Access Enable Register
		AP Flash	ISP Flash	SRAM	
Relocated Memory Space	0x0000 0000	Alias of AP Read only	Alias of ISP Read only	Alias of SRAM Read only	
Alias-0 of IAP	joining with Boot Code	Read only			
AP Flash	0x1800 0000	Read only	Read only Read/Write	Read only Read/Write	AP_WEN=0 AP_WEN=1
Alias-1 of IAP	joining with AP Flash	Read only	Read only Read/Write	Read only Read/Write	IAP_WEN=0 IAP_WEN=1
IAP Flash	0x1A00 0000	Read only Read/Write	Read only Read/Write	Read only Read/Write	IAP_WEN=0 IAP_WEN=1

ISP Flash	0x1C00 0000	X	Read only	X	ISP_WEN=0 ISP_REN=0
		Write only		Write only	ISP_WEN=1 ISP_REN=0
		Read only		Read only	ISP_WEN=0 ISP_REN=1
		Read/Write		Read/Write	ISP_WEN=1 ISP_REN=1
ISPD Flash	0x1FF0 0000	X	Read only	X	ISPD_WEN=0 ISPD_REN=0
			Read/Write		ISPD_WEN=1 ISPD_REN=0
		Read only	Read only	Read only	ISPD_WEN=0 ISPD_REN=1
			Read/Write		ISPD_WEN=1 ISPD_REN=1
SRAM	0x2000 0000	Read/Write	Read/Write	Read/Write	

<Sign> X: access not allowed

<Note> ISPD_WEN/ISPD_REN register bits are only able to change when boots from ISP mode.

7.9.5. CPU Code Execution and Hold

The CPU can run code on AP/ISP flash memory or embedded SRAM. Also the CPU supports to run code on the IAP flash memory for an option for by setting **MEM_IAP_EXEC** register.

When chip boots from AP or ISP flash memory, the **MEM_HOLD** register bit is used to enable or disable CPU holds control under flash memory write access. Default is enable hold control and CPU will hold until the end of flash memory write access. When CPU runs code on flash memory, the CPU must hold if the flash memory is execution program or erase access at the same time. When CPU runs code on SRAM, the CPU does not need to hold and the firmware can disable the CPU hold control during the flash memory is execution program or erase access.

Table 7-5. CPU Hold Control under Flash Memory Access

Boot Mode	CPU Hold	Register
		HOLD
AP/ISP	Hold	0
	Normal	1
SRAM	Normal	x (don't care)

<Note> HOLD ~ CPU will hold until the end of flash memory access.

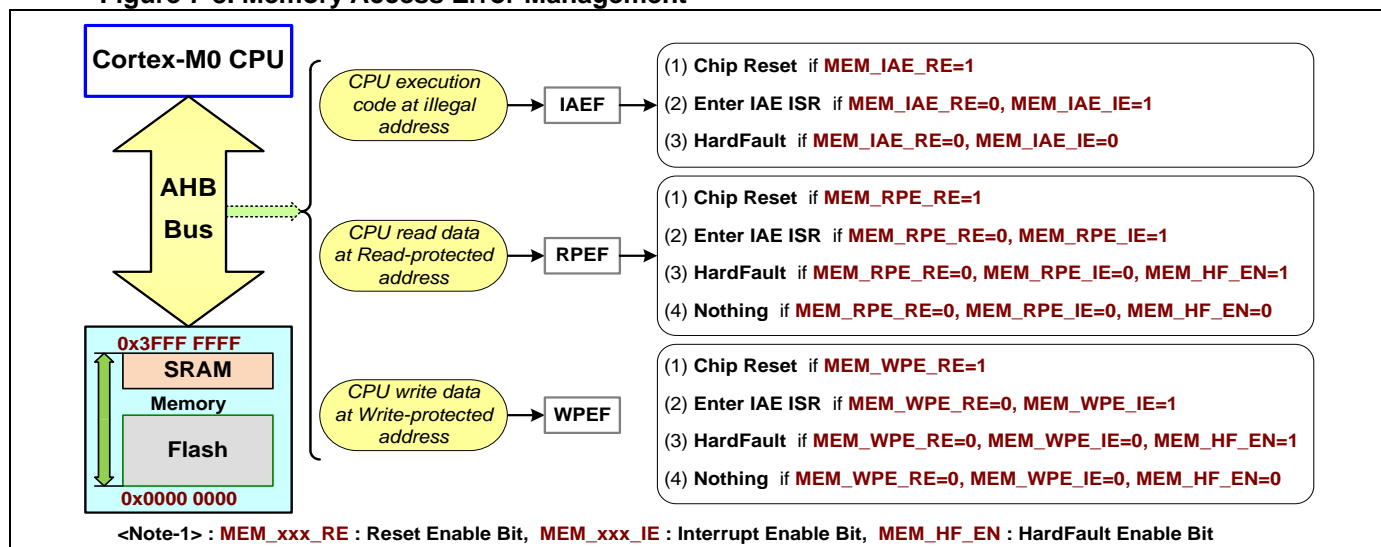
7.9.6. Memory Access Error Management

The memory controller is designed an error management control block to manage the memory access error of CPU execution code at illegal address, CPU read data at read-protected address and CPU write data at write-protected address.

User can enable to generate CPU **HardFault** exception for flash memory write or read protection error in **MEM_HF_EN** register. When memory data read error has happened and **MEM_RPE_IE** / **MEM_RPE_RE** registers are disabled, it will induce **HardFault** if this bit is enabled. When memory data write error has happened and **MEM_WPE_IE** / **MEM_WPE_RE** registers are disabled, it will induce **HardFault** if this bit is enabled.

The following diagram is showing the management of memory access error. User can select to do "Chip Reset", Generate "Interrupt", Generate "HardFault" or do "Nothing".

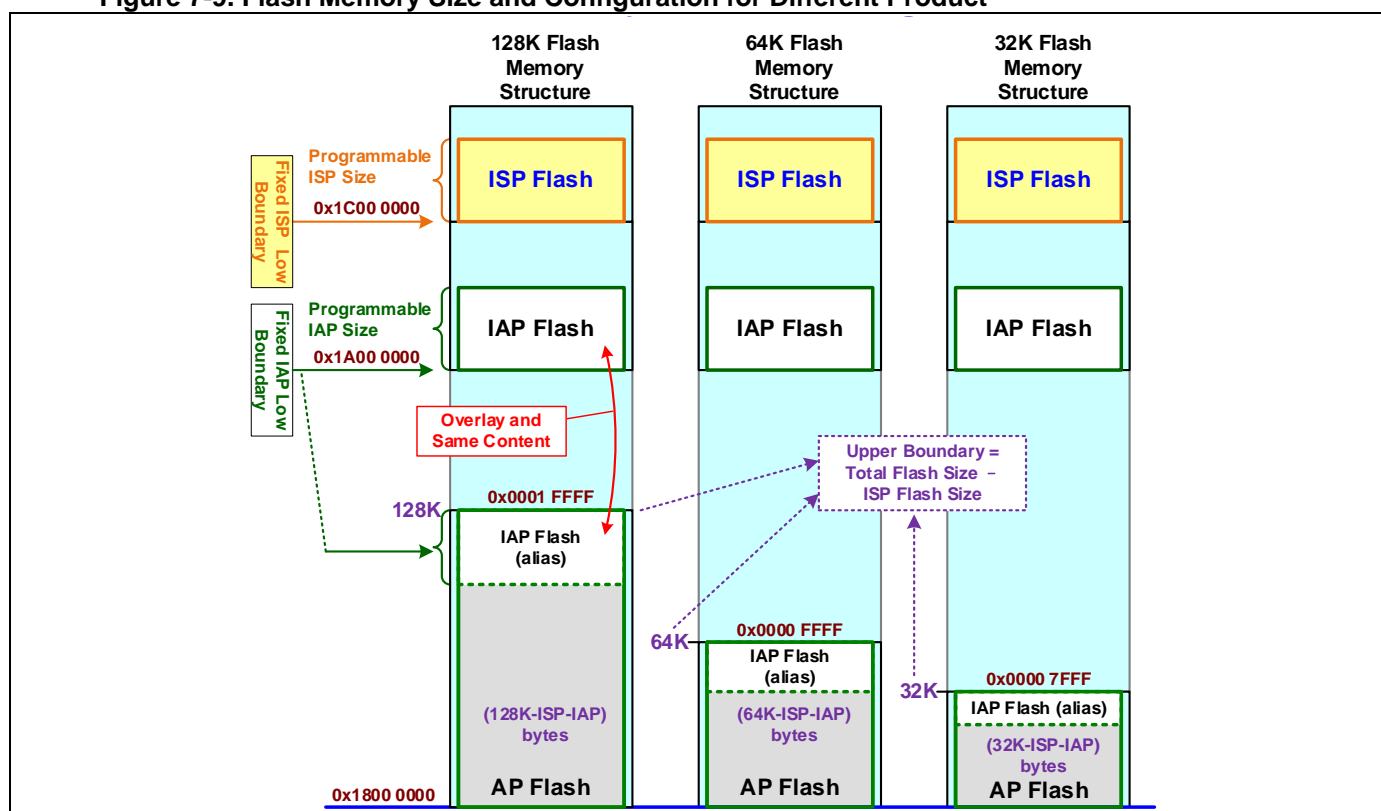
Figure 7-8. Memory Access Error Management



7.10. Flash Memory Configuration for Different Product

There are optional memory size flashes of 32K, 64K or 128K bytes by different product. The following diagram is showing the flash memory size and configuration for different product.

Figure 7-9. Flash Memory Size and Configuration for Different Product



8. Hardware Option

8.1. Introduction

The hardware option defines the chip default behavior those are not volatile after power off. There are up to 64 bytes of on-chip option bytes (**OB**) flash memory which are used to configure the hardware option and stored into embedded flash memory. The hardware option byte (**OB**) can only be configurable by a universal programmer, the “megawin ARM Writer” or the “megawin ARM ICE Adapter” (The ICE adapter also supports ICP programming function).

After whole-chip erased, all the hardware options are cleared to default state and there is no ISP-memory and IAP-memory configured. The option byte flash memory (**OB**) will load into chip configuration option registers (**OR**) for hardware configuration control during warm reset or cold reset cycle. Refer the section of “[Hardware Option Byte Flash Memory](#)” in System Memory chapter for more information about option byte flash memory (**OB**) and chip configuration option registers (**OR**).

8.2. Hardware Option Byte

The user configurable hardware option bytes are listed as following: (☑ ~ suggestion)

- **BOOT_MS**

System cold reset boot memory select and memory is mapped at 0x0000 0000.

☐: Application Flash

☒: Boot Flash

☐: Embedded SRAM

- **LOCK_DIS**

☐: Enable. Code dumped on a universal writer or programmer is locked to 0xFFFFFFFF for security.

☒: Disable. Not locked.

- **IAP_SIZE**

The IAP_SIZE specifies the user defined IAP memory size. Value 0 indicates the IAP memory size 0K-byte. Value 1 indicates the IAP memory size 0.5K-byte.

- **ISP_SIZE**

The ISP_SIZE specifies the ISP memory size which contains the boot loader code. Value 0 indicates no ISP flash memory and value 1 indicates ISP total memory size 0.5K-byte.

- **BOD1_TH**

BOD1 detect voltage threshold select.

☐: Select BOD1 to detect 2.0V.

☐: Select BOD1 to detect 2.4V.

☐: Select BOD1 to detect 3.6V. (Notify)

☐: Select BOD1 to detect 4.2V.

[Notify]: This voltage threshold is 3.7V for MG32F02A128/U128/A064/U064/V032.

- **BOD0_WE**

BOD0 trigger warm reset enable. When enables, BOD0 will trigger a reset to CPU if the voltage threshold detect event happened.

☐: Enable. BOD0 will trigger a reset event to CPU.

☐: Disable. BOD0 cannot trigger a reset to CPU.

- **BOD1_WE**

BOD1 trigger warm reset enable. When enables, BOD1 will trigger a reset to CPU if the voltage threshold detect event happened.

☐: Enable. BOD1 will trigger a reset event to CPU.

☐: Disable. BOD1 cannot trigger a reset to CPU.

- **BOD2_WE**

BOD2 trigger warm reset enable. When enables, BOD2 will trigger a reset to CPU if the voltage threshold detect event happened.

☐: Enable. BOD2 will trigger a reset event to CPU.

☐: Disable. BOD2 cannot trigger a reset to CPU.

- **IWDT_EN**

☐: Enable. IWDT will enable after power-on.

☐: Disable. IWDT is not enabled automatically after power-on.

- **IWDT_WP**

IWDT registers write protected enable.

☐: Enable. The IWDT registers will be write-protected.

☐: Disable. The IWDT registers are free for software writing.

- **IWDT_WE**

IWDT reset generation enable option.

☐: Enable. Enable to generate a system reset for IWDT events.

☐: Disable. Disable to generate a system reset for IWDT events.

- **IWDT_SLP**

IWDT counting control when chip in SLEEP mode.

☐: Stop. Stop counting and disable IWDT running.

☐: Keep. Keep counting and enable IWDT running.

- **IWDT_STP**

IWDT counting control when chip in STOP mode.

☐: Stop. Stop counting and disable IWDT running.

☐: Keep. Keep counting and enable IWDT running.

- **IWDT_DIV**

IWDT internal clock input divider select. When IWDT_EN is enabled, these bits will be loaded to IWDT input divider control register.

- **PC_IOM**

Port C default IO mode select after power-on. All the port-C PCn pins are default AIO mode or QB mode by this setting except PC4/5/6/13/14 pins. The IO modes of PC4/5/6 pins are always default QB mode. The IO modes of PC13/14 pins are directly control by chip if XOSC_EN is enabled. When XOSC_EN is disabled, the IO modes of PC13/14 pins are control by this register setting.

☒: AIO. Port-C pins' default is Analog IO.

☐: QB. Port-C pins' default is Quasi-Bidirectional output.

8.3. CFG Option Register

The option registers (**OR**) are used for chip hardware configuration control. They are loaded from the option byte flash memory during warm reset or cold reset cycle. Refer the section of "[Hardware Option Byte Flash Memory](#)" in System Memory chapter and the CFG register descriptions in related chip Register Definition Guide for more information.

8.3.1. CFG Register Protect and Lock

After chip reset, all CFG registers are write-access protected except **CFG_KEY** registers. Write **0xA217** value to the **CFG_KEY** register to unprotect the write access of registers. Oppositely write other value except **0xA217** value to protect the registers. Read the **CFG_KEY** register to get the register value is Protected (=1) or Unprotected (=0).

Refer the table and descriptions of "[Register Protect and Lock](#)" in System Reset chapter for more information.

8.3.2. Manufacturer ADC Calibration Value

The chip supports ADC and PGA input offset calibration function. These calibrated offset values will be recorded in option byte **OB** flash memory during manufacture.

For ADC offset, the calibrated offset value will be loaded to **CFG_ADC_OFFT** registers after power on or chip cold reset. The **CFG_ADC_OFFT** register provides the default offset calibration value and will be loaded to **ADC0_OFFT_ADC** register after chip cold reset. User can update this register after ADC calibration.

For ADC PGA offset, the calibrated offset value will be loaded to **CFG_PGA_OFFT** and **CFG_PGA2_OFFT** registers after power on or chip cold reset. These registers value will be also loaded to **ADCx_OFFT_PGA** and **ADCx_OFFT_PGA2** registers after chip cold reset. User can update these registers after ADC PGA calibration.

8.3.3. Manufacturer Temperature Sensor Calibration Value

The ADC is embedded one temperature sensor to measure the internal junction temperature of chip for product application.

There are two ADC codes those are recorded the voltages of temperature sensor at two temperatures in the chip configuration option registers (**OR**) of **CFG_TEMP_CAL0** and **CFG_TEMP_CAL1**. The two registers are loaded from the option byte (**OB**) flash memory after chip reset. These two **OR** registers are calibrated and stored in the **OB** flash memory during the chip is manufactured from megawin manufactory. User can update these registers after user on system temperature calibration. Please contact megawin about temperature sensor chip configuration for chip manufacture option.

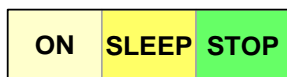
There are two read only registers of **ADCx_TCAL0** and **ADCx_TCAL1** in ADC module those are aliased to **CFG_TEMP_CAL0** and **CFG_TEMP_CAL1** registers. Refer the section of "[Temperature Sensor](#)" in ADC chapter for more information.

8.3.4. Manufacturer OPA Calibration Value

The chip supports OPA input offset calibration function. This calibrated offset value will be recorded in option byte **OB** flash memory during manufacture and will be loaded to **CFG_OP0_OFFT** register after power on or chip cold reset. The **CFG_OP0_OFFT** register provides the default offset calibration value and will be loaded to **OPA_OP0_OFFT** register after chip cold reset. User can update this register after ADC calibration.

9. GPIO (General Purpose IO)

9.1. Introduction



The module can be running in all power operation modes.

The **MG32x02z** series has **PA**, **PB**, **PC**, **PD** and **PE** I/O ports and is including the pins of **PA[15:0]**, **PB[15:0]**, **PC[14:0]**, **PD[15:0]**, **PE[3:0][9:8][15:12]**. Support maximum 73 GPIO pins for LQFP80 package. **PC4**, **PC5** and **PC6** pins are specially implemented with the alternated function of **SWCLK**, **SWDIO** and **RSTN**. If select external crystal oscillator as system clock input, **PC13** and **PC14** are configured to **XIN** and **XOUT**. The exact number of I/O pins available depends upon the package types.

The chip has built in several IO mode control (**PA/PB/PC/PD/PE**) modules for each GPIO port. These modules are used for GPIO pin IO mode control, alternated function selection, driver strength setting, input inverse selection, pull-high enable, deglitch filter setting and high speed enable. Also one IO Port access control (**IOP**) module is built-in to control the input and output state of GPIO mode for all GPIO ports.

Notify: The sign of (Px = module {PA, PB, PC, PD, PE}, n= input/output pin index number, **OB** = hardware configuration Option Byte flash memory) is using for Registers, Signals and Pins/Ports in the descriptions of this chapter.

9.2. Features

- Support general purpose IO pins for application
 - Maximum 73 GPIO pins for LQFP80 package
 - Maximum 59 GPIO pins for LQFP64 package
 - Maximum 44 GPIO pins for LQFP48 package
 - Maximum 29 GPIO pins for LQFP32/QFN32 package
 - Maximum 17 GPIO pins for TSSOP20 package
- Provide selectable IO modes by pin independent
 - Push-Pull output
 - Quasi bidirectional (PC pins only)
 - Open-drain output
 - Input only with high impedance
 - Analog IO
- Flexible pin alternate function selection
- Support programmable drive strength by pin independent
 - Programmable 2-level drive strength for all GPIO pins except RSTN, XIN, XOUT
 - Programmable 4-level drive strength for some special pins
 - Support high extended drive strength for some special pins (MG32F02V032 only)
- Support IO deglitch filter by pin independent
- Support input inverse selection by pin independent
- Support pull-high option by pin independent
- Support high speed option by pin independent except RSTN, XIN
- Support anti current leakage function for PB[3:0] (MG32F02V032 only)
- GPIO pin state and IO mode setting keep optional after reset

9.3. Implementation

9.3.1. GPIO Implementation

The following table is showing the implemented GPIO function of chips.

Table 9-1. GPIO Implementation

Chip	IO Pin			Drive Strength			Port C
	Package	IO Number	High Speed	4-Level	High Extended	Anti-Current Leakage	Power-on Mode
MG32F02A128 MG32F02N128/K128	LQFP80/64	73/59	All IO pins except RSTN, XIN	PB[5:0][9:8], PD[5:0][8:7], PE[3:0]	-	-	QB or PP
MG32F02A064 MG32F02N064/K064	LQFP64/48	59/44			-	-	QB or PP
MG32F02U128	LQFP80/64	70/56			-	-	QB or PP
MG32F02U064	LQFP64/48	56/41			-	-	QB or PP
MG32F02V032	LQFP32, QFN32, TSSOP20	29, 29, 17		PB[3:0][9:8], PD[2:0][7]	PA8, PA10	PB[3:0]	QB or PP

<Note> QB: Quasi bidirectional, PP: Push-Pull output

QB or PP: Port C pins can be set the default IO mode except PC4/5/6/13/14 by hardware option OB flash.

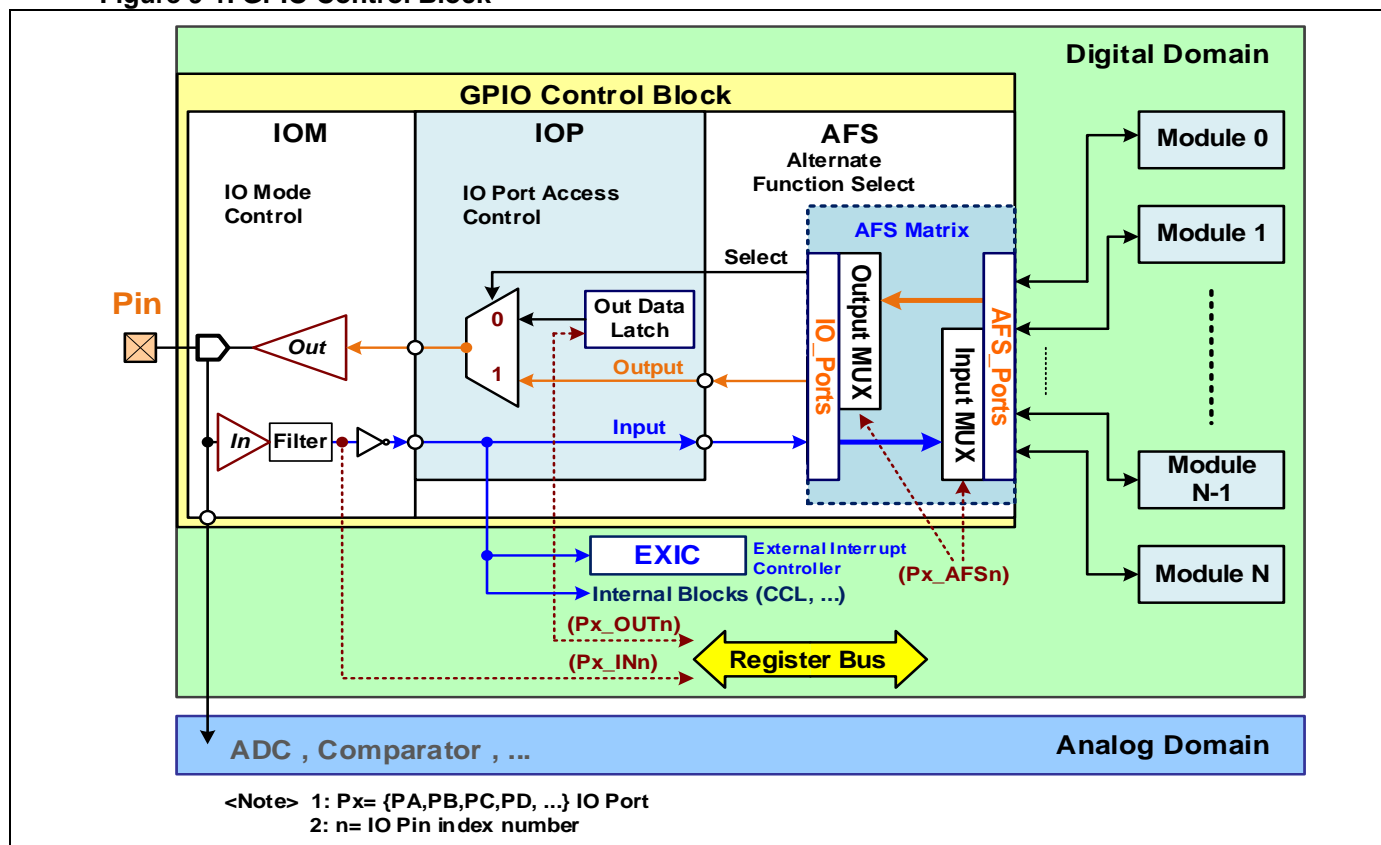
9.4. Control Block

The GPIO Control block includes IOM, IOP and AFS (Alternate Function Select) blocks. The IOM block is used to set the IO operation mode. The IOP block is used to control the GPIO port read and write access. The AFS block is used to select the IO alternate function. All GPIO pins can set the IOM, IOP and AFS functions by independent control blocks and registers.

The GPIO output data bit and IO mode configuration of some GPIO pins can be disabled to be reset by system Warm reset. Please refer the “[GPIO Reset Control](#)” section in System Reset chapter.

The following diagram is showing the GPIO Control block.

Figure 9-1. GPIO Control Block



9.5. IO Mode

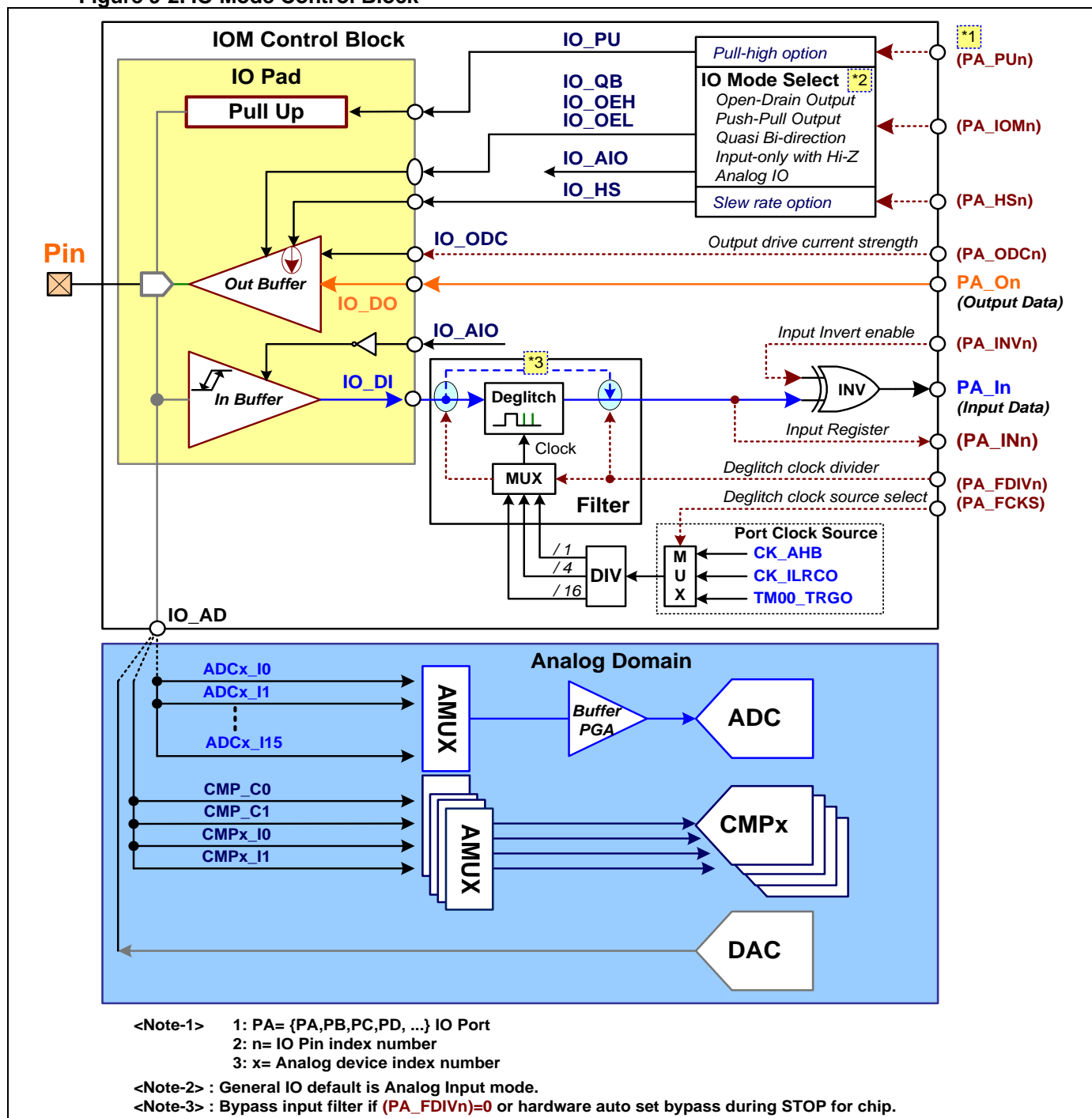
The IO operating modes are supported analog IO (AIO), digital input (DIN), push-pull output (PPO), and open-drain output (ODO), quasi-bidirectional (QB). Provide selectable IO modes by pin independent.

9.5.1. IO Mode Control Block

The IO mode control block supports programmable IO operation modes, output high speed option, pull-high option, output drive strength, IO deglitch filter and input inverse selection by pin independent.

The following diagram is showing the IO Mode Control block.

Figure 9-2. IO Mode Control Block



9.5.2. IO Configuration

The chip provides independent modules for IO mode configuration of PA, PB, PC, PD and PE ports. Refer the descriptions of pin definition for more detail information.

● IO Mode Control

All the GPIO pins can be configure the IO operation mode by pin independently. User can set the IO operating modes of analog IO, digital input, push-pull output and open-drain output by setting **Px_IOMn** register for each GPIO pin. The quasi-bidirectional operation mode is only supported on port C (PC) pins.

The following table is the register setting for IO mode, pull-high and high speed control.

Table 9-2. IO Operation Mode Control

IO Mode	Pull High	High Speed	IO Configuration	
Px_IOMn	Px_PUn	Px_HSn		
0x0	-	-	Analog IO	
0x1	0	0	Open Drain Output	
	0	1		High Speed
	1	0		Pull High
	1	1		Pull High + High Speed
0x2	0	0	Push Pull Output	
	0	1		High Speed
	1	0		Pull High
	1	1		Pull High + High Speed
0x3	0	-	Digital Input	
	1	-		Pull High
0x4	0	0	Quasi-Bidirectional Output (drive high one clock)	
	0	1		High Speed
	1	0		Pull High
	1	1		Pull High + High Speed

<Note> "Px" = {PA,PB,PC,PD,PE}, "n" = Pin index number, "-" = Don't care

The power-on IO mode of PA, PB, PD and PE are analog IO (AIO). The power-on IO mode of port C (PC) pins may be configured to AIO (default) or QB mode except **PC4/5/6/13/14** pins in **CFG_PC_IOM** register by hardware option byte (**OB**) flash.

Please refer the section of "[Alternate Function Select of Special Pins](#)" for more information about the power-on default setting of PC special pins.

● Output High Speed Control

All the GPIO pins can support high speed option except **RSTN**, **XIN** pins. User can enable the IO output high speed option by setting **Px_HSn** register for these GPIO pin independently.

● Pull-high Control

All the GPIO pins can be configured the IO pull-high option by pin independently. User can enable the IO pull-high option by setting **Px_PUn** register for each GPIO pin independently.

● Input Inverse Control

All the GPIO pins can be enabled the input inverse function by pin independently. User can enable the IO input inverse function by setting **Px_INVn** register for each GPIO pin independently.

● Output Drive Strength Control

All the GPIO pins can be selected the output drive strength level by pin independently. There are three types of IO pins with programmable output drive strength. The one is programmable two levels of output with 1 and 1/4 of full level drive strength. The two is programmable four levels of output with 1, 1/2, 1/4 and 1/8 of full level drive strength. The three is programmable five levels of output with 1, 1/2, 1/4, 1/8 and high extended of full level drive strength. The high extended drive strength is that the output sink current strength is 2 times of full level drive strength and the output drive current strength is as same as full level drive strength. User can enable the IO output drive strength by setting **Px_ODCn** register for each GPIO pin independently.

Please refer the related chip Data Sheet about the IO output high and low current information.

● Input Filter Control

All the GPIO pins are built in an input digital deglitch by pin independently. The filter provides configurable filter clock source and filter clock divider to adapt the input signal operation frequency and probable application

noise.

User can set the filter clock source from AHB clock, AHB clock divided by 8, ILRCO clock, TM00 trigger output or UT (Unit Time) clock by setting **Px_FCKS** register and uses the setting for all pins of one GPIO port. The filter clock source can be configured for each GPIO port independently. Also user can set the filter clock divider by divided 1, 4 and 16 or bypass the filter in **Px_FDIVn** register for each GPIO pin independently.

- **Alternate Function Selection**

All the GPIO pins can be enabled the input inverse function by pin independently. User can enable the IO input inverse function by setting **Px_AFSn** register for each GPIO pin independently.

- **Anti-Current Leakage IO Pin**

The anti-current leakage IO pins can use to reduce the IO leakage current to under 1nA. For some application product, the IO pins can use like as I2C signals under MCU VDD no power condition to avoid that MCU power is raised by the I2C signal input and makes MCU in invalid operation.

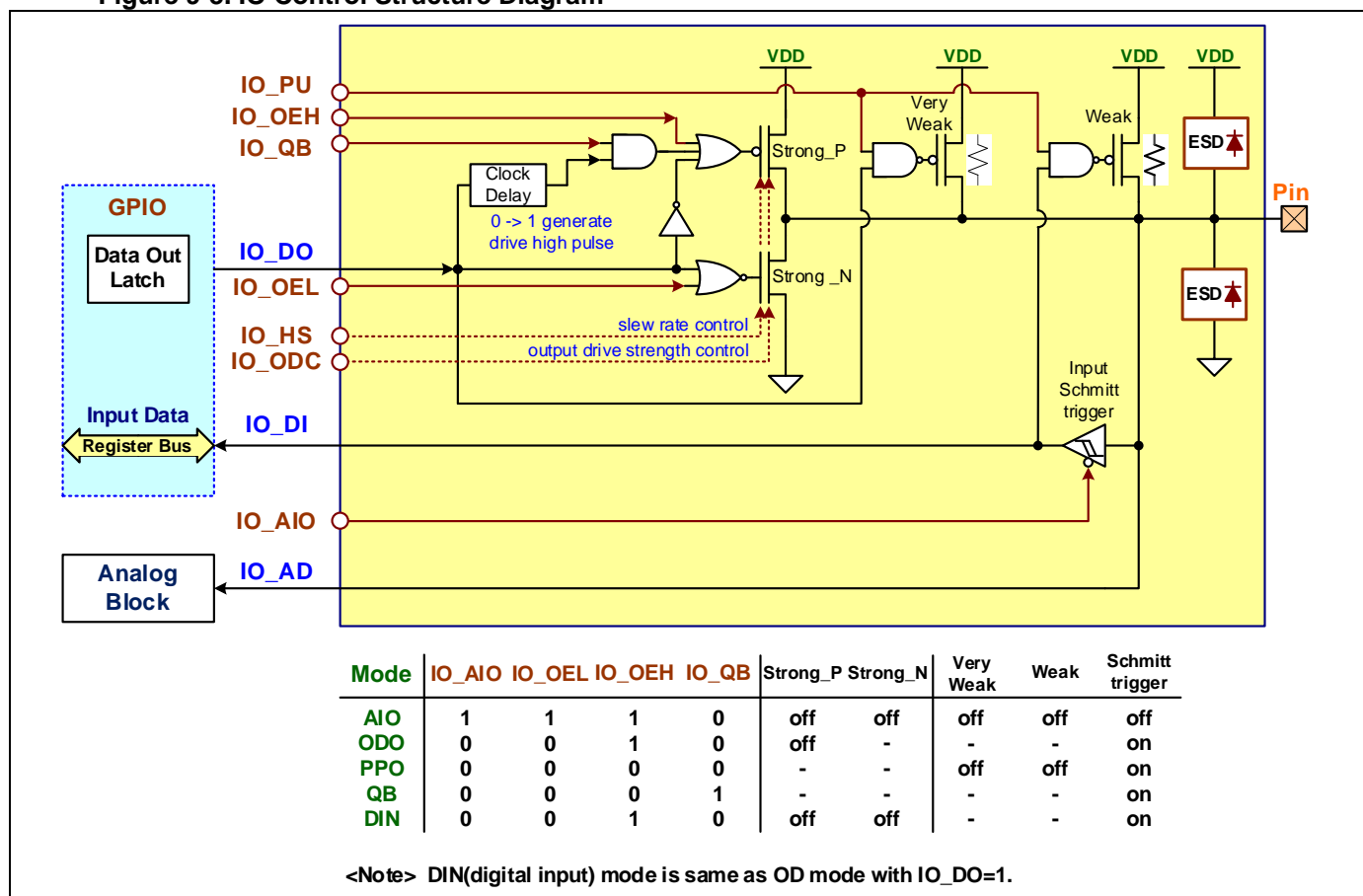
Please refer the GPIO implementation table in the section of “[GPIO Implementation](#)” or the related chip Data Sheet about this IO supported pins.

9.6. IO Structure

The I/O structures are supported for the operating modes of analog IO, digital input, push-pull output, and open-drain output, quasi-bidirectional.

The following diagram is showing the general IO control structure block.

Figure 9-3. IO Control Structure Diagram

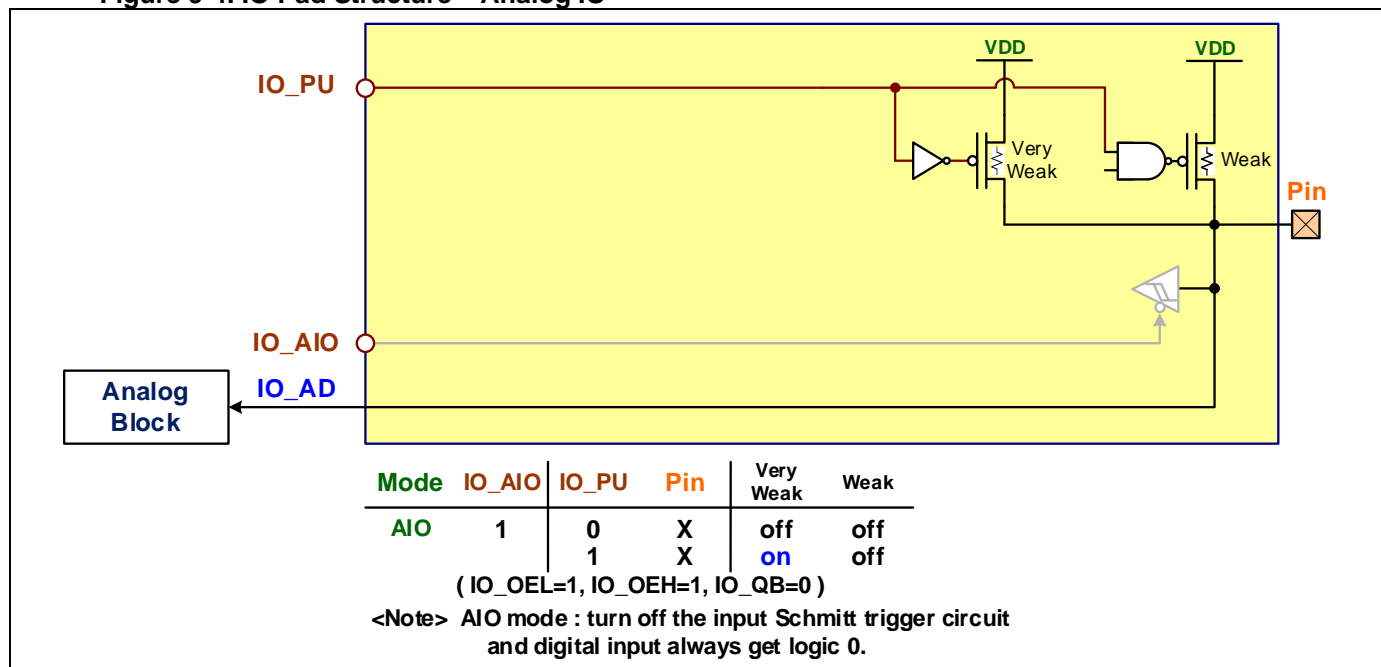


9.6.1. Analog IO Structure

The analog IO configuration is using for the input of ADC, analog comparator and the output of DAC, OPA. For application, usually the analog IO is without pull-up resistors on the pin. When selects analog IO mode (AIO), the Schmitt-trigger buffer will be forced to disabled.

This analog IO and digital input port configuration is showing in following diagram.

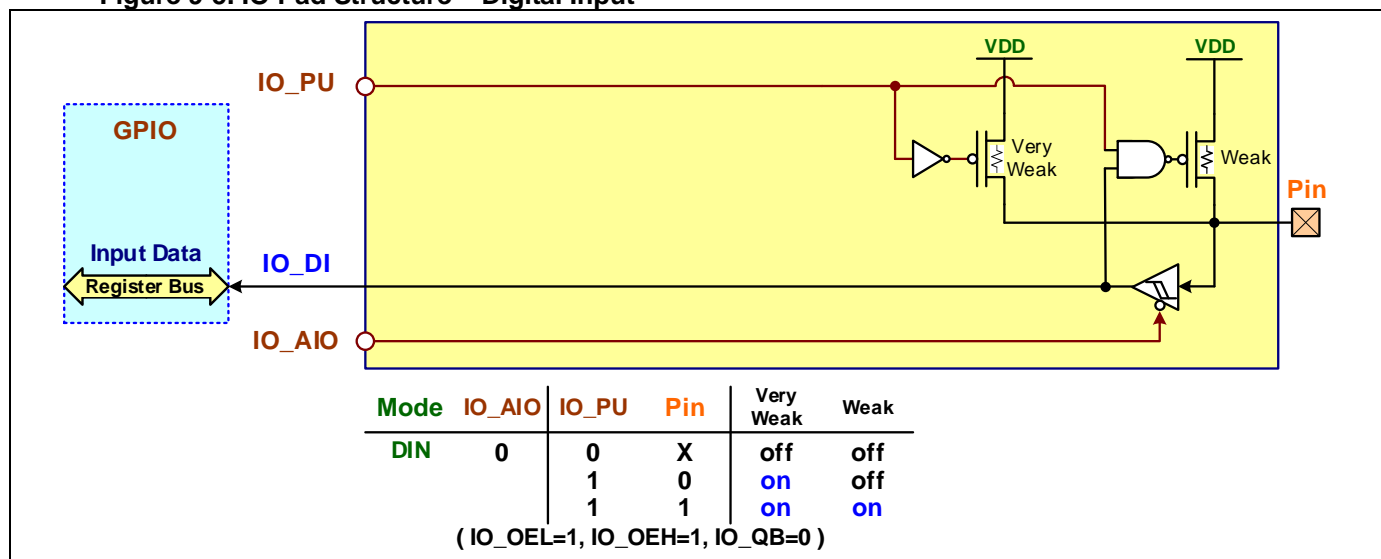
Figure 9-4. IO Pad Structure – Analog IO



9.6.2. Digital Input Structure

The digital input configuration is a high impedance input with a Schmitt-trigger buffer. A pull-up resistor option is controlled by pin independent. The analog input path is directly connected the IO pad to internal analog devices.

Figure 9-5. IO Pad Structure – Digital Input



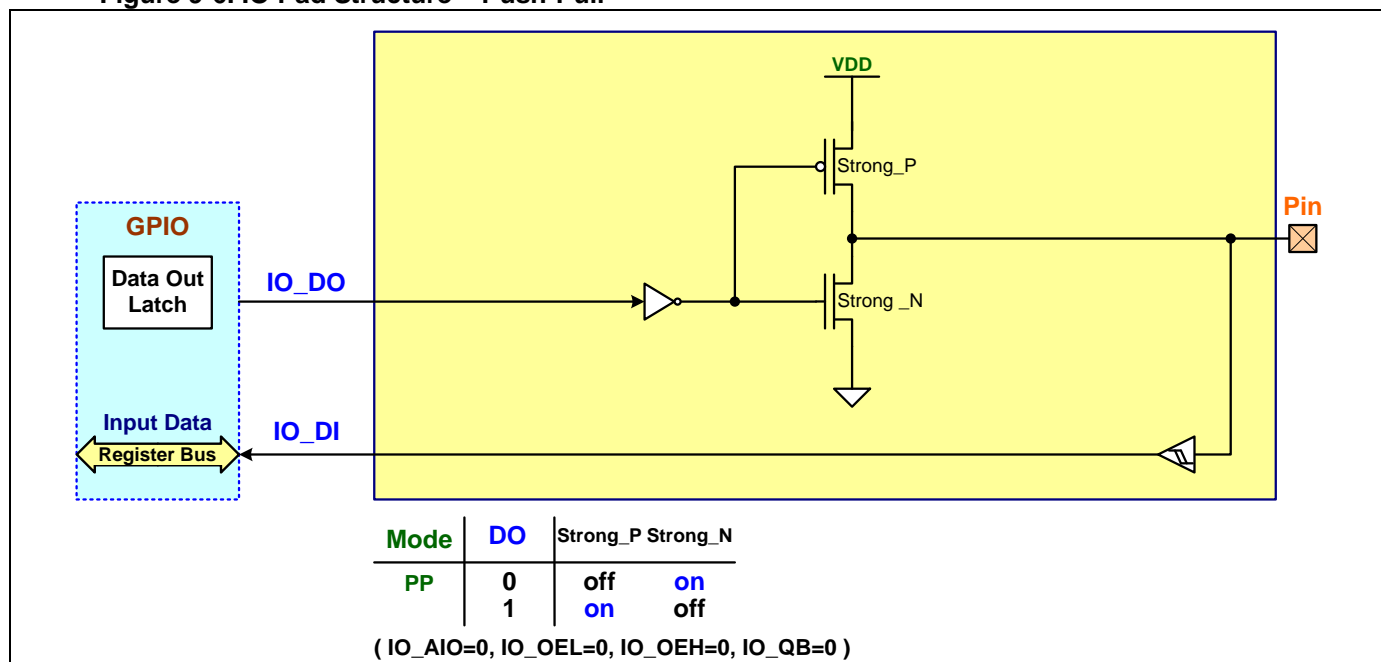
9.6.3. Push-Pull Output Structure

The push-pull output configuration has the same pull-down structure as both the open-drain and the quasi-bidirectional output modes, but provides a continuous strong pull-up when the port register contains a logical "1". The push-pull mode may be used when more source current is needed from a port output. In addition,

the input path of the port pin in this configuration is also the same as quasi-bidirectional mode.

The push-pull port configuration is shown in following diagram.

Figure 9-6. IO Pad Structure – Push-Pull

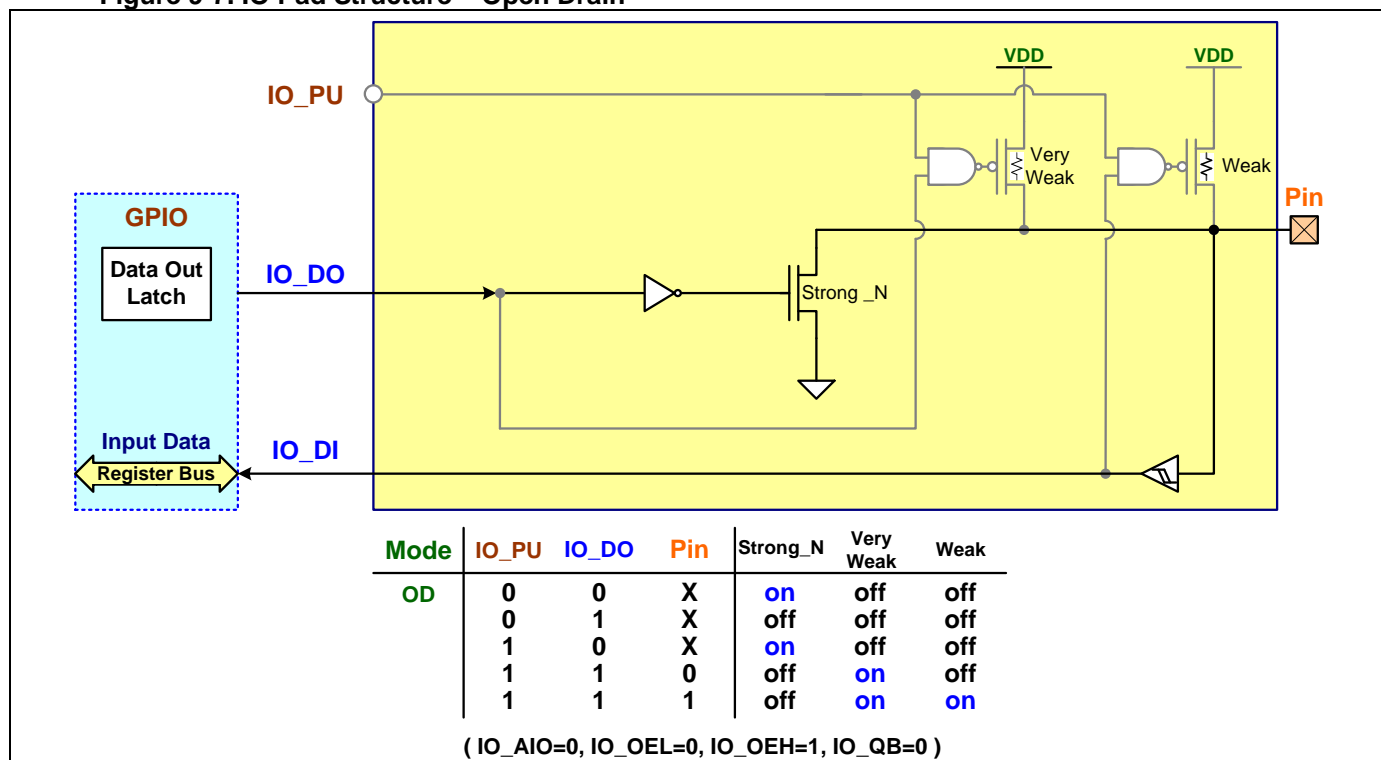


9.6.4. Open-Drain Output Structure

The open-drain output configuration turns off all pull-ups and only drives the pull-down transistor of the port pin when the port register contains logic “0”. To use this configuration in application, a port pin must have an external pull-up, typically a resistor tied to **VDD**. The pull-down for this mode is the same as for the quasi-bidirectional mode. In addition, the input path of the port pin in this configuration is also the same as quasi-bidirectional mode.

The open-drain port configuration is shown in following diagram.

Figure 9-7. IO Pad Structure – Open Drain



9.6.5. Quasi-Bidirectional IO Structure

The quasi-bidirectional operation mode is only supported on port C (PC) pins. A quasi-bidirectional port can be used as an input and output without the need to reconfigure the port. This is possible because when the port outputs a logical “high”, it is weakly driven, allowing an external device to pull the pin low. When the pin outputs low, it is driven strongly and able to sink a large current. There are three pull-up transistors in the quasi-bidirectional output that serve different purposes.

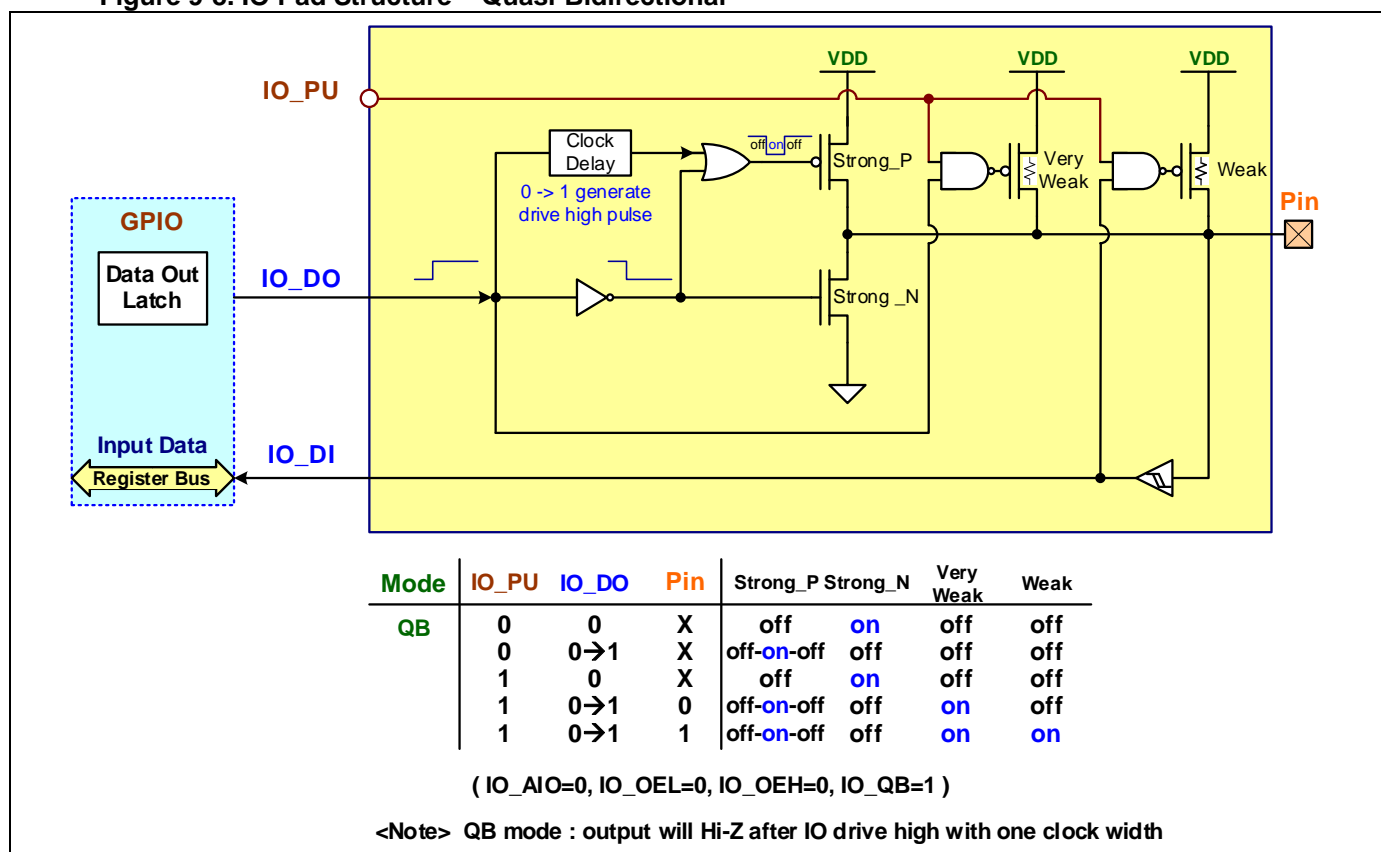
One of these pull-ups, called the “very weak” pull-up, is turned on whenever the port register for the pin contains a logical “1”. This very weak pull-up sources a very small current that will pull the pin high if it is left floating.

A second pull-up, called the “weak” pull-up, is turned on when the port register for the pin contains a logical “1” and the pin itself is also at a logic “1” level. This pull-up provides the primary source current for a quasi-bidirectional pin that is outputting a 1. If this pin is pulled low by the external device, this weak pull-up turns off, and only the very weak pull-up remains on. In order to pull the pin low under these conditions, the external device has to sink enough current to over-power the weak pull-up and pull the port pin below its input threshold voltage.

The third pull-up is referred to as the “strong” pull-up. This pull-up is used to speed up low-to-high transitions on a quasi-bidirectional port pin when the port register (**Px_OUTn**) changes from a logical “0” to a logical “1”. When this occurs, the strong pull-up turns on for one system clock (**CK_SYS**), quickly pulling the port pin high.

The quasi-bidirectional port configuration is shown in following diagram.

Figure 9-8. IO Pad Structure – Quasi-Bidirectional

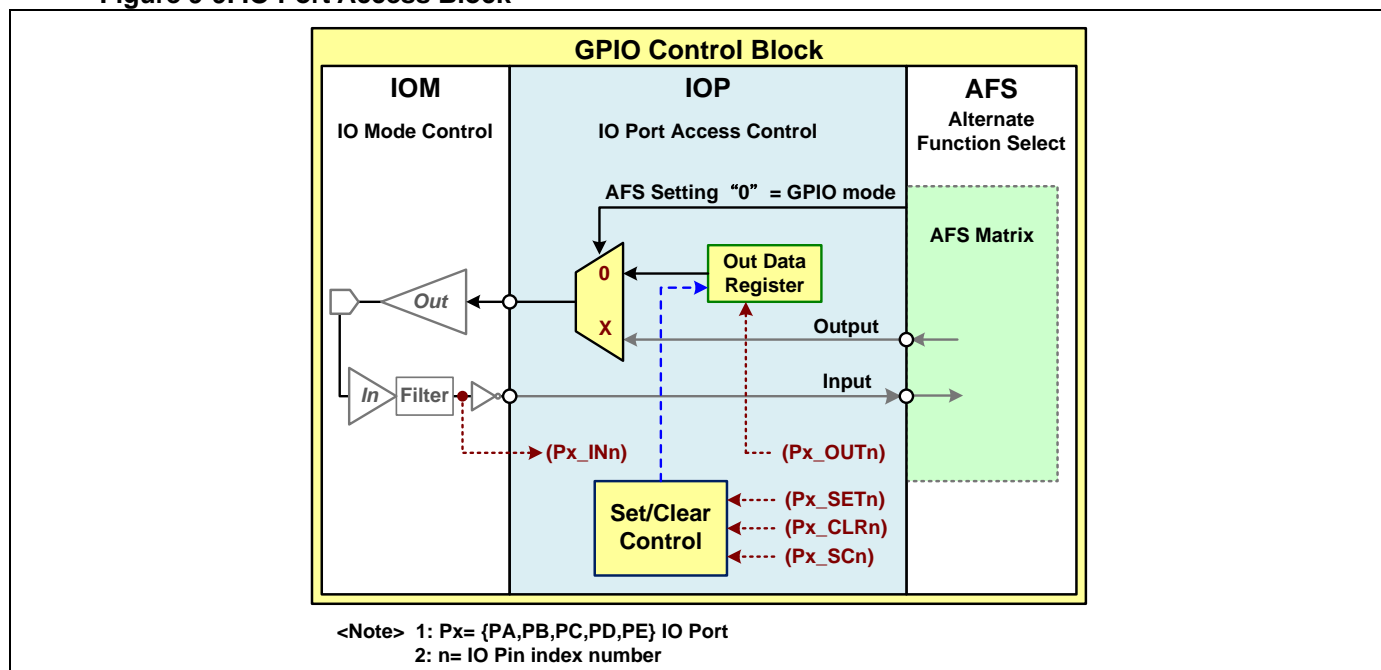


9.7. IO Port Access

9.7.1. GPIO IO Control

When the AFS setting is set GPIO function mode for any IO pin, user can directly set the logical output or get the logical input for the IO pin. There is one independent data out register bit to store the output logic value for each GPIO pin. User can read and write this data register in **Px_OUTn** register bit for each GPIO pin. Also user can directly read the **Px_INn** register bit to get the GPIO pin logical state for each GPIO pin.

Figure 9-9. IO Port Access Block



The **Px_OUT** register and **Px_IN** register are designed to write and read all pins' logical output and input through the **Px_OUTn** bits and the **Px_INn** bits for one GPIO port. It lets firmware easy to control one GPIO port simultaneously. (n = {0~15})

9.7.2. Set and Clear Control

For firmware control, there are one **Px_SETn** control bit to set the data out register bit and one **Px_CLRn** control bit to clear the data out register bit for each GPIO pin. There is a **Px_SC** register which is designed to set or clear the data out register bits of all pins through the **Px_SETn** bits and the **Px_CLRn** bits for one GPIO port.

These control register bits are designed to write 1 only and is no effect for writing 0. So it lets firmware easy to set or clear for multiple pins of one GPIO port simultaneously. When the related **Px_SETn** bit and **Px_CLRn** bit of a GPIO pin are both set to 1, the related data bit is set to 1 (n = {0~15})

9.7.3. Bit Like IO Control

The chip provides one **Px_SCn** register control bit to set, clear the data out register bit or read pin status for each GPIO pin. The register bit is written 1 to set data bit and written 0 to clear data. Read the register bit to get the GPIO pin status.

As the **Px_SCn** register bit is cost eight bit memory space, firmware is easy to control single GPIO pin by CPU byte-access instruction command. It is like the bit access IO control of 8051 MCU.

Refer the **Px_SCRn** register definitions for more information.

9.8. Alternate Function Select

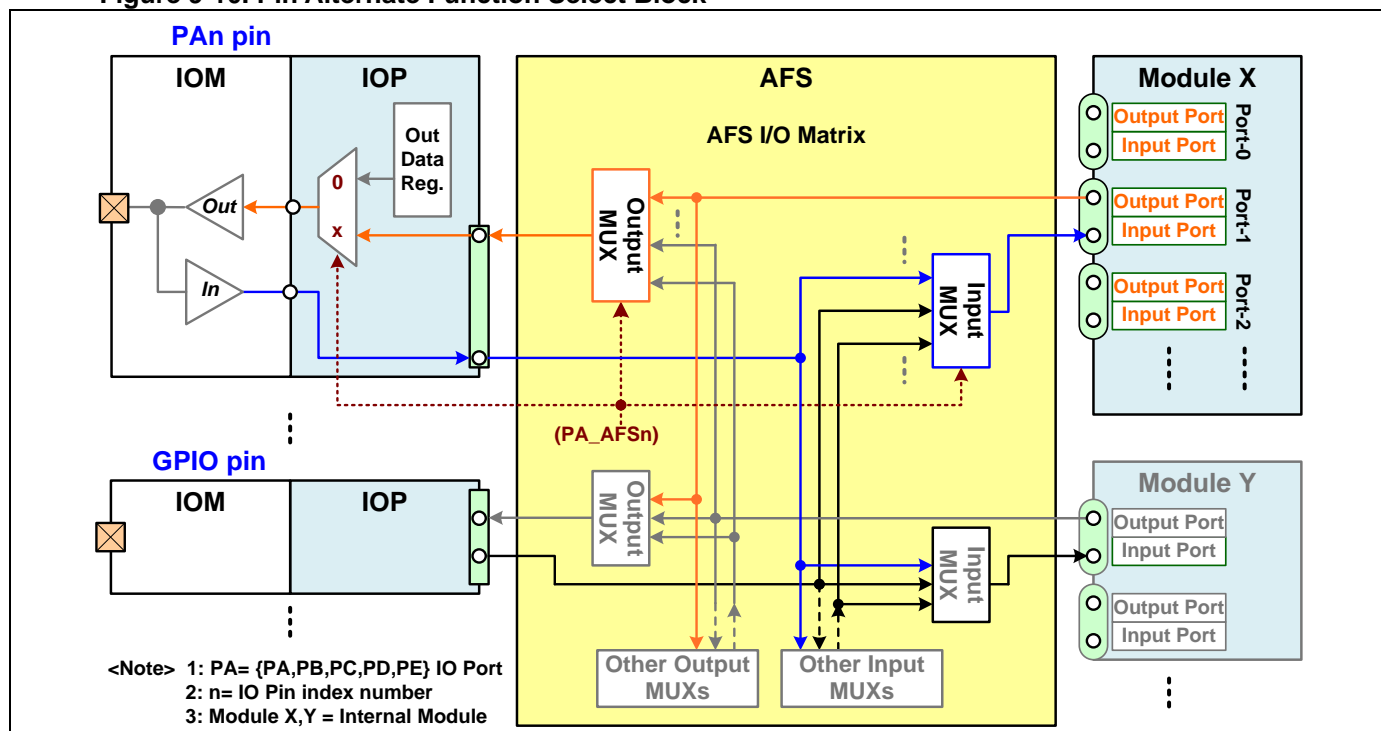
9.8.1. Alternate Function Select Control

User can configure the alternate function between module function IO and IO pins through the AFS matrix by setting **Px_AFSn** register for each GPIO pin independently. Refer the descriptions of pin definition and register definition for more detail information.

Usually the AFS default setting is GPIO function for each GPIO pin except the **XIN/XOUT**, **SWCLK/SWDIO** and **RSTN** function pins. These pins may be changed by hardware configuration **OB**. Refer the section of “[Alternate Function Select of Special Pins](#)” for more information.

The following diagram is showing the Pin Alternate Function Select block.

Figure 9-10. Pin Alternate Function Select Block



- **Pin AFS Table**

Refer to the “Pin Alternate Functions Selected Table” in the chip Data Sheet.

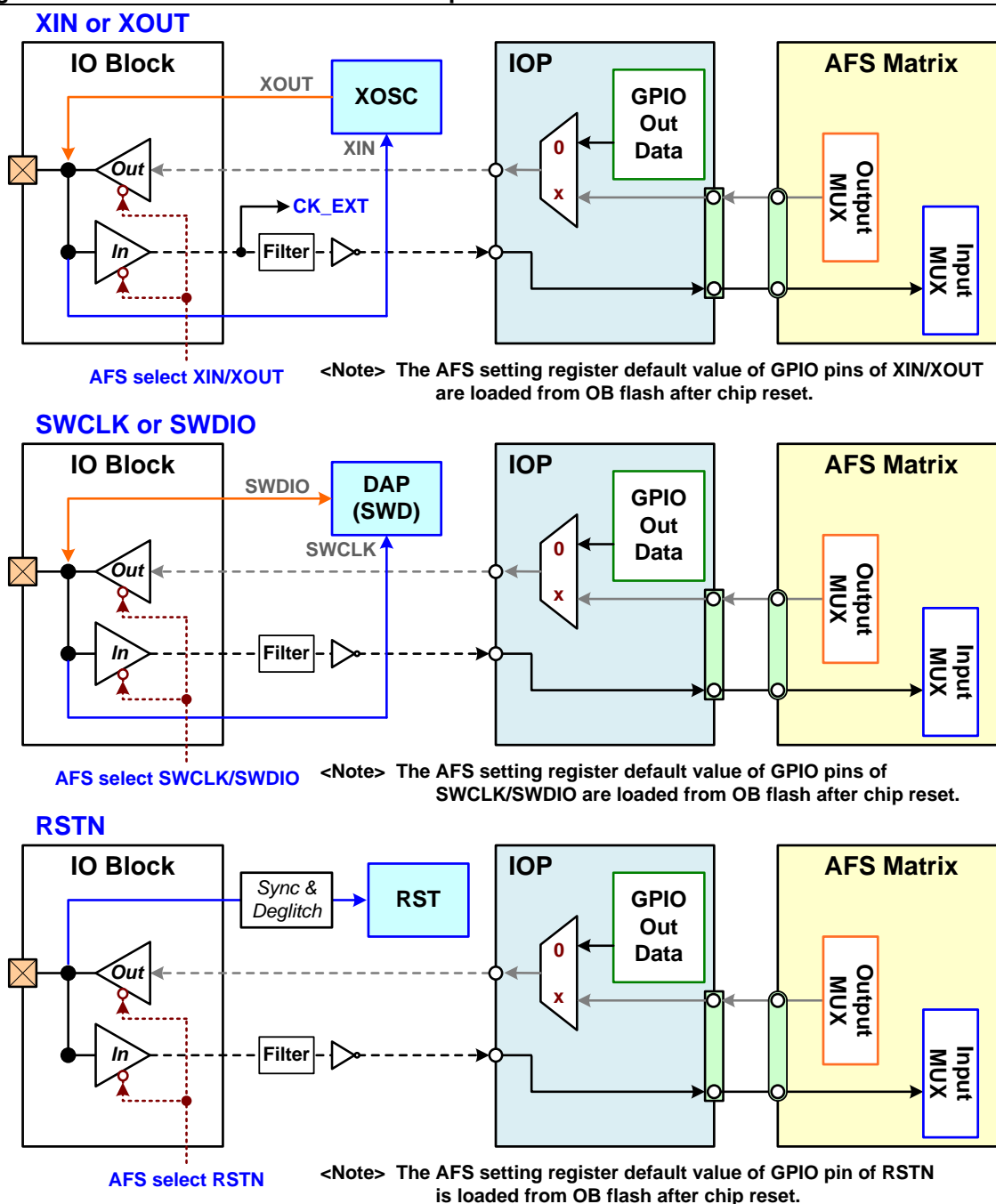
9.8.2. Alternate Function Select of Special Pins

Usually the AFS default setting is GPIO function for each GPIO pin except the **XIN/XOUT**, **SWCLK/SWDIO** and **RSTN** function pins. The **XIN/XOUT**, **SWCLK/SWDIO** and **RSTN** function pins can be enabled by hardware configuration **OB** flash after chip reset (Cold reset). Also these hardware configuration **OB** value are loaded to update the GPIO AFS setting for these pins. So user can change the GPIO AFS setting of these pins after power on sequence.

When the **XIN/XOUT**, **SWCLK/SWDIO** or **RSTN** AFS function is enabled, the IO mode register setting and pull-high register setting of related GPIO pin are no effect. The IO mode and pull-high control are directly control by hardware.

The following diagram is showing the Pin Alternate Function Select block of **XIN/XOUT**, **SWCLK/SWDIO** and **RSTN** function pins.

Figure 9-11. Alternate Function Select of Special Pins



● External Xtal Pins

The chip is embedded one internal crystal oscillator (**XOSC**) for external Xtal circuit. The chip can boot to enable the **XOSC** crystal and configure **XIN** and **XOUT** pins as external crystal input and output by configurable hardware option byte (**OB**). Also user can re-configure the AFS setting of **XIN** and **XOUT** pins by setting GPIO AFS registers.

When both **XIN** and **XOUT** pins are configured as external crystal input and output, the internal **XOSC** oscillator will be enabled by hardware and the external crystal is oscillating. When the **XOUT** pin does not be configured as external crystal output, the internal crystal oscillator (**XOSC**) will be powered off by chip. That is only the **XIN** pin to be configured as external crystal input and the **XIN** pin can input the clock from external clock source like as OSC. Refer the [System Clock](#) chapter for more information.

● SWD Pins

The chip is embedded one debug access port (**DAP**) and debugging control circuit for system debugging. The chip can boot to enable the **SWCLK** and **SWDIO** pins to connect external debugging device like as ICE by configurable hardware option byte (**OB**). Also user can re-configure the AFS setting of **SWCLK** and **SWDIO** pins by setting GPIO AFS registers.

- **External Reset Pin**

The chip provides an external hardware reset input from **RSTN** pin to ensure a reliable power-up reset. Refer the [System Reset](#) chapter for more information.

The chip can boot to enable the **RSTN** pin as external reset pin or others (GPIO ...) by hardware option byte (**OB**). Also user can re-configure the AFS setting of **RSTN** pin by setting GPIO AFS registers. Strongly suggests keeping the **RSTN** pin as external reset pin except all other GPIO pins are in use.

The power-on IO modes of **PC4/5/6** pins are always QB mode. The power-on IO modes of **PC13/14** pins are directly control by chip if **CFG_XOSC_EN** is enabled. When **CFG_XOSC_EN** is disabled, the power-on IO modes of **PC13/14** pins are control by this register setting.

The following table is showing the Port C IO mode power-on default setting.

Table 9-3. Port C IO Mode Default Setting

Port C Pin	Hardware OR Setting				IO Register Power-On Default		
	CFG_XOSC_EN	CFG_EXRST_PIN	CFG_SWD_PIN	CFG_PC_IOM	Px_IOMn	Px_PUn	Px_AFSn
PC[14:13]	0	x	x	0	0x0 (AIO)	0	GPIO
				1	0x4 (QB)	1	GPIO
	1	x	x	x	0x0 (AIO)	0	XIN, XOUT
PC6	x	0	x	x	0x4 (QB)	1	RSTN
		1					GPIO
PC[5:4]	x	x	0	x	0x4 (QB)	1	SWDIO
			1				SWCLK
PC[12:7,3:0]	x	x	x	0	0x0 (AIO)	0	GPIO
				1	0x4 (QB)	0	GPIO

<Note> x: don't care, AIO : analog IO, QB: Quasi-Bidirectional output

9.8.3. GPIO AFS Locking

The chip supports the GPIO AFS setting locking function only for the function pins of **SWCLK/SWDIO** and **RSTN**. User must set the AFS setting in **PC_AFSn** register and set **PC_LCKn** register bit to 1 simultaneously (n = {4, 5, 6}). The write access of **PC_AFSn** register is no effect if **PC_LCKn** register bit is 0. The chip will auto clear the **PC_LCKn** register bit after register write access.

Refer the register descriptions of **PC_CR4**, **PC_CR5** and **PC_CR6** registers for more information.

9.9. GPIO Application Circuit

9.9.1. GPIO Input and Output

During the product operation, the MCU IO signals are easy to be with 'noise' which is usually generated by original signal source, power noise, ground bouncing, signal crosstalk, thermal noise or others on application PCB.

As following diagrams, user can insert a low pass filter on the MCU input pin or a transition noise filter on the MCU output pin to remove the unexpected noises and glitches. The filter can guarantee the signal performance in order to operate correctly for the request IO function which is like as UART, I2C, SPI and others. The filter needs to design by depending on the IO signal request highest frequency and must place close to the MCU pin.

Also the serial termination resistor or/and the parallel termination resistor is/are necessary for the signal integrity issue as following diagrams.

Figure 9-12. GPIO Input Application Circuit

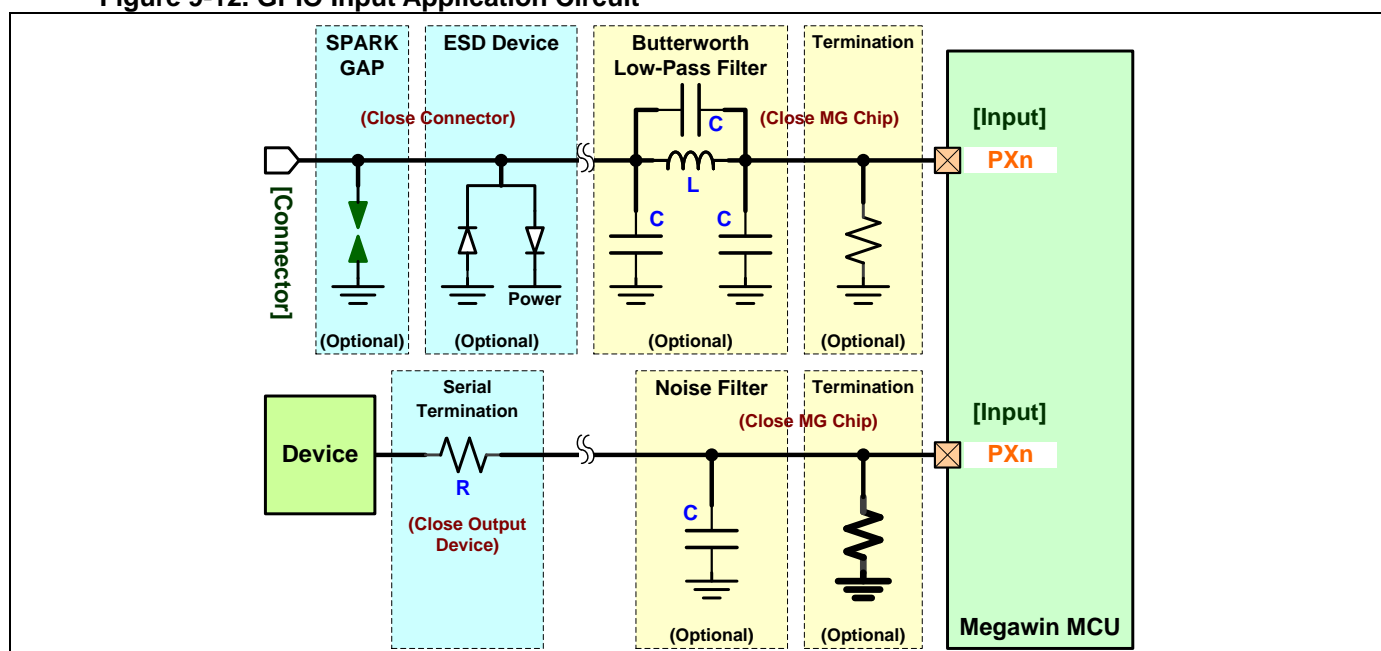
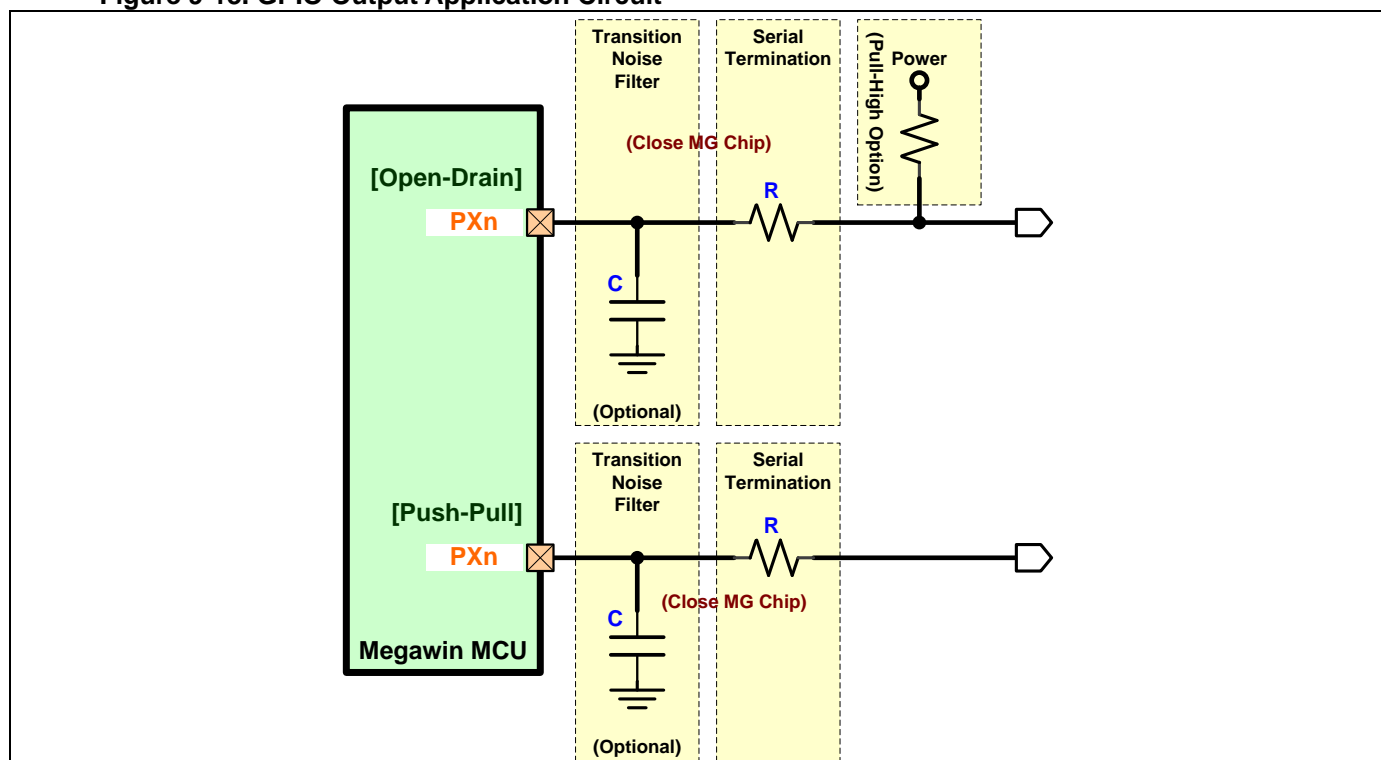


Figure 9-13. GPIO Output Application Circuit



9.9.2. GPIO Input Filter

When the MCU input signal is with noise and glitch on application PCB, user can place the noise bypass capacitor on the MCU input pin or/and enable the MCU input filter function to fix or reduce this issue. The noise and glitch may be generated by original signal source, power noise, ground bouncing, signal crosstalk, thermal noise or others during the product operation.

All the GPIO pins are built in an input digital deglitch filter by pin independently. The filter provides configurable filter clock source and filter clock divider to adapt the input signal operation frequency and probable application noise.

User can set the filter clock source from AHB clock, AHB clock divided by 8, ILRCO clock, TM00 trigger output or UT (Unit Time) clock by setting **Px_FCKS** register and uses the setting for all pins of one GPIO port.

The filter clock source can be configured for each GPIO port independently. Also user can set the filter clock divider by divided 1, 4 and 16 or bypass the filter in **Px_FDIVn** register for each GPIO pin independently.

Refer the IO mode control block about GPIO input filter in the section "[IO Mode Control Block](#)".

9.9.3. GPIO Output High Speed and Drive Strength

The MCU can support the IO pin programmable output high speed control and output drive current strength control by pin independent. Refer the figure of "[IO Mode Control Block](#)", user can set the **Px_HSn** and **Px_ODCn** registers for output high speed control and output drive current strength control.

◆ Output High Speed Control

The output high speed control can be enabled to disable the output slew rate control for high speed timing application. That is like as the clock and data signals for SPI (Serial Peripheral Interface) bus, EMB (External Memory Bus) bus or others. User can enable the IO output high speed option by setting **Px_HSn** register for these GPIO pin independently.

All the GPIO pins can support high speed option except **RSTn**, **XIN** pins.

◆ Output Drive Strength Control

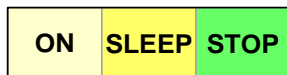
The output drive current strength control can be set to reduce the signal overshoot or undershoot damping conditions by PCB trace impedance unmatched and signal reflection problems. Also it can reduce some signal crosstalk conditions and improve SI (signal integrity) issue even reduce EMI.

All the GPIO pins can be selected the output drive strength level by pin independently. There are two types of IO pins with programmable output drive strength. Generally, the output drive current strength control be programmable two levels and can be set to full level and 1/4 level. Some GPIO pins be programmable four levels and can be set to full level, 1/2 level, 1/4 level and 1/8 level for the output drive current strength control. User can enable the IO output drive strength by setting **Px_ODCn** register for each GPIO pin independently.

The IO pins of **PB[5:0][9:8]**, **PD[5:0][8:7]** and **PE[3:0]** are supported four levels' output drive strength.

10. Interrupt

10.1. Introduction



The module can be running in all power operation modes.

After reset, the CPU begins execution from the location of reset interrupt vector (0x00000004) addressing, where should be the starting of the user's application code. To service the interrupts, the interrupt service locations (called interrupt vectors) should be located in the address 0x000000BF~0x00000000.

The chip is built-in ARM® cortex® M0 CPU and is embedded a **NVIC** (Nested Vectored Interrupt Controller) for 32 external interrupt inputs with 4-level priority. Also builds in an EXIC (External Interrupt Controller) module and connects to **NVIC**.

Notify: The sign of (Px = module {PA, PB, PC, PD, PE}, n= input/output pin index number, **OB** = hardware configuration Option Byte flash memory) is using for Registers, Signals and Pins/Ports in the descriptions of this chapter.

10.2. Features

- Built-in one NVIC for 32 external interrupt inputs with 4-level priority
- Built-in one EXIC (external interrupt controller) for NVIC connection
 - Independent high/low level and rising/falling edge trigger selection
- Built-in one WIC (wakeup interrupt controller) for wakeup event control
- All GPIO pins can be configured as interrupt source and key pad input
 - Support port OR logic for interrupt function
 - Support port AND logic for KBI function
- Support external pins for CPU NMI/RXEV/TXEV function
 - Configurable pin for CPU NMI input function
 - Configurable pin for CPU RXEV input function
 - Configurable pin for CPU TXEV output function

10.3. Interrupt Structure

Each interrupt is assigned a fixed location in the program memory. The interrupt causes the CPU to jump to that location, where it commences execution of the service routine. NMI interrupt, for example, is assigned to location 0x00000008. If NMI is going to be used, its service routine must begin at location 0x00000008.

The interrupt service locations are spaced at an interval of 4 bytes: 0x00000004 for Reset Interrupt, 0x00000008 for **NMI**, 0x0000000C for **Hard-Fault**, 0x0000002C for **SVC**, 0x00000038 for **PendSV**, 0x0000003C for **SysTick**, etc.

The NVIC has 7 exception types: **Reset**, **NMI**, **HardFault**, **SVC**, **PendSV**, **SysTick** and Interrupt (IRQ). The NVIC supports 32 external interrupt input. An interrupt is an exception signaled by a peripheral or generated by a software request. The four priority level interrupt structure allows great flexibility in handling these interrupt sources.

10.3.1. Interrupt Sources

The 'Pending Bits' are the interrupt flags that will generate an interrupt if it is enabled by setting the 'Set Enable Bit'. The 'Pending Bits' can be set or cleared by software, with the same result as though it had been set or cleared by hardware. That is, interrupts can be generated or pending interrupts can be cancelled in software. The 'Priority Bits' determine the priority level for each interrupt. The 'Priority within Level' is the polling sequence used to resolve simultaneous requests of the same priority level. The 'Vector Address' is the entry point of an interrupt service routine in the program memory.

The following table of interrupt sources lists all the interrupt sources.

Table 10-1. Interrupt Sources

NVIC			Registers			Priority	Vector Address	Comment
Exception No.	IRQ No.	Interrupt Name	Set/Clear Enable	Pending Bits	Priority Bits			
0	-	Initial				-	0x00000000	Initial MSP value
1	-	Reset				-3	0x00000004	Reset exception
2	-14	NMI		0XE000ED04[31]		-2	0x00000008	Non Maskable Interrupt
3	-13	HardFault				-1	0x0000000C	Cortex-M0 Hard Fault Interrupt
4~10	-	Reserved				-		
11	-5	SVC			0XE000ED1C [31:24]	Configurable	0x0000002C	Cortex-M0 SV Call Interrupt
12~13	-	Reserved				-		
14	-2	PendSV		0XE000ED04[28] 0XE000ED04[27]	0XE000ED20 [23:16]	Configurable	0x00000038	Cortex-M0 Pend SV Interrupt
15	-1	SysTick	0XE000E010 [1]	0XE000ED04[26] 0XE000ED04[25]	0XE000ED20 [31:24]	Configurable	0x0000003C	Cortex-M0 System Tick Interrupt
16	0	WWDT	0XE000E100 0XE000E180	0XE000E200 0XE000E280	0XE000E400	Configurable	0x00000040	Window Watchdog interrupt
17	1	SYS				Configurable	0x00000044	System global interrupt
18	2	Reserved				Configurable	0x00000048	Reserved
19	3	EXINT0				Configurable	0x0000004C	EXIC EXINT0 (PA) interrupt
20	4	EXINT1			0XE000E404	Configurable	0x00000050	EXIC EXINT1 (PB) interrupt
21	5	EXINT2				Configurable	0x00000054	EXIC EXINT2 (PC) interrupt
22	6	EXINT3				Configurable	0x00000058	EXIC EXINT3/4 (PD/PE) interrupt
23	7	COMP				Configurable	0x0000005C	Analog Comparators global interrupt
24	8	DMA			0XE000E408	Configurable	0x00000060	DMA all channel global interrupt
25	9	OPA				Configurable	0x00000064	OPA interrupt
26	10	ADC				Configurable	0x00000068	ADC interrupt
27	11	DAC				Configurable	0x0000006C	DAC interrupt
28	12	TM0x			0XE000E40C	Configurable	0x00000070	Timer TM0x global interrupt
29	13	TM10				Configurable	0x00000074	Timer TM10 interrupt
30	14	TM1x				Configurable	0x00000078	Timer TM16 TM1x global interrupt
31	15	TM20				Configurable	0x0000007C	Timer TM20 interrupt
32	16	TM2x			0XE000E410	Configurable	0x00000080	Timer TM26 TM2x global interrupt
33	17	TM3x				Configurable	0x00000084	Timer TM3x global interrupt
34	18	Reserved				Configurable	0x00000088	Reserved
35	19	LCD				Configurable	0x0000008C	LCD interrupt
36	20	URT0			0XE000E414	Configurable	0x00000090	UART URT0 interrupt
37	21	URT123				Configurable	0x00000094	UART URT1/2/3 global interrupt
38	22	URT4x				Configurable	0x00000098	UART URT4/5/6/7 global interrupt

39	23	Reserved				Configurable	0x0000009C	Reserved
40	24	SPI0			0XE000E418	Configurable	0x000000A0	SPI0 interrupt
41	25	Reserved				Configurable	0x000000A4	Reserved
42	26	Reserved				Configurable	0x000000A8	Reserved
43	27	CAN0				Configurable	0x000000AC	CAN0 interrupt
44	28	I2C0			0XE000E41C	Configurable	0x000000B0	I2C0 interrupt
45	29	I2Cx				Configurable	0x000000B4	I2C1 I2Cx global interrupt
46	30	USB				Configurable	0x000000B8	USB interrupt
47	31	APX				Configurable	0x000000BC	APX interrupt

<Note> Configurable : Programmable priority level 0~3

10.3.2. Exception types

There are 7 exception types: **Reset**, **NMI**, **HardFault**, **SVC**, **PendSV**, **SysTick** and Interrupt (IRQ).

● Reset

Reset is invoked on power up or a warm reset. The exception model treats reset as a special form of exception. When reset is asserted, the operation of the processor stops potentially at any point in an instruction. When reset is deasserted, execution restarts from the address provided by the reset entry in the vector table. Execution restarts in Thread mode.

● NMI

An NMI can be signaled by a peripheral or triggered by software.

This is the highest priority exception other than reset. It is permanently enabled and has a fixed priority of -2. NMIs cannot be:

- masked or prevented from activation by any other exception
- preempted by any exception other than Reset.

● HardFault

A **HardFault** is an exception that occurs because of an error during normal or exception processing. HardFault have a fixed priority of -1, meaning they have higher priority than any exception with configurable priority.

● SVC

A supervisor call (SVC) is an exception that is triggered by the SVC instruction. In an OS environment, applications can use SVC instructions to access OS kernel functions and device drivers.

● PendSV

PendSV is an interrupt-driven request for system-level service. In an OS environment, use PendSV for context switching when no other exception is active.

● SysTick

If the device implements the **SysTick** timer, a **SysTick** exception is an exception the system timer generates when it reaches zero.

Software can also generate a SysTick exception. In an OS environment, the device can use this exception as system tick.

● Interrupt (IRQ)

An interrupt, or IRQ, is an exception signaled by a peripheral, or generated by a software request. All interrupts are asynchronous to instruction execution. In the system, peripherals use interrupts to communicate with the processor.

10.3.3. Exception handlers

The processor handles exceptions using:

- **ISRs**

The IRQ interrupts are the exceptions handled by ISRs.

- **Fault handler**

HardFault is the only exception handled by the fault handler.

- **System handlers**

NMI, **PendSV**, **SVCall**, **SysTick**, and **HardFault** are all system exceptions handled by system handlers.

10.3.4. Interrupt Priority

The priority scheme for servicing the interrupts has four interrupt levels. The priority bits in CPU registers, **IPR0-7**, **SHPR2** and **SHPR3**, determine the priority level of each interrupt.

The interrupt priority registers provide an 8-bit priority field for each interrupt and each register holds four priority fields. The processor implements only bits [7:6] of each field, bits [5:0] read as zero and ignore writes.

Higher-priority interrupt will be not interrupted by lower-priority interrupt request. If two interrupt requests of different priority levels are received simultaneously, the request of higher priority is serviced. If interrupt requests of the same priority level are received simultaneously, an internal polling sequence determine which request is serviced. The table of "interrupt sources" shows the internal polling sequence in the same priority level and the interrupt vector address. The lower exception number gets the higher priority.

10.3.5. Lockup on Cortex-M0

The processor enters a lockup state if a fault occurs when executing the NMI or HardFault handlers, or if the system generates a bus error when unstacking the PSR on an exception return using the MSP. When the processor is in lockup state it does not execute any instructions. The processor remains in lockup state until one of the following occurs:

Table 10-2. CPU Lockup Exit Events

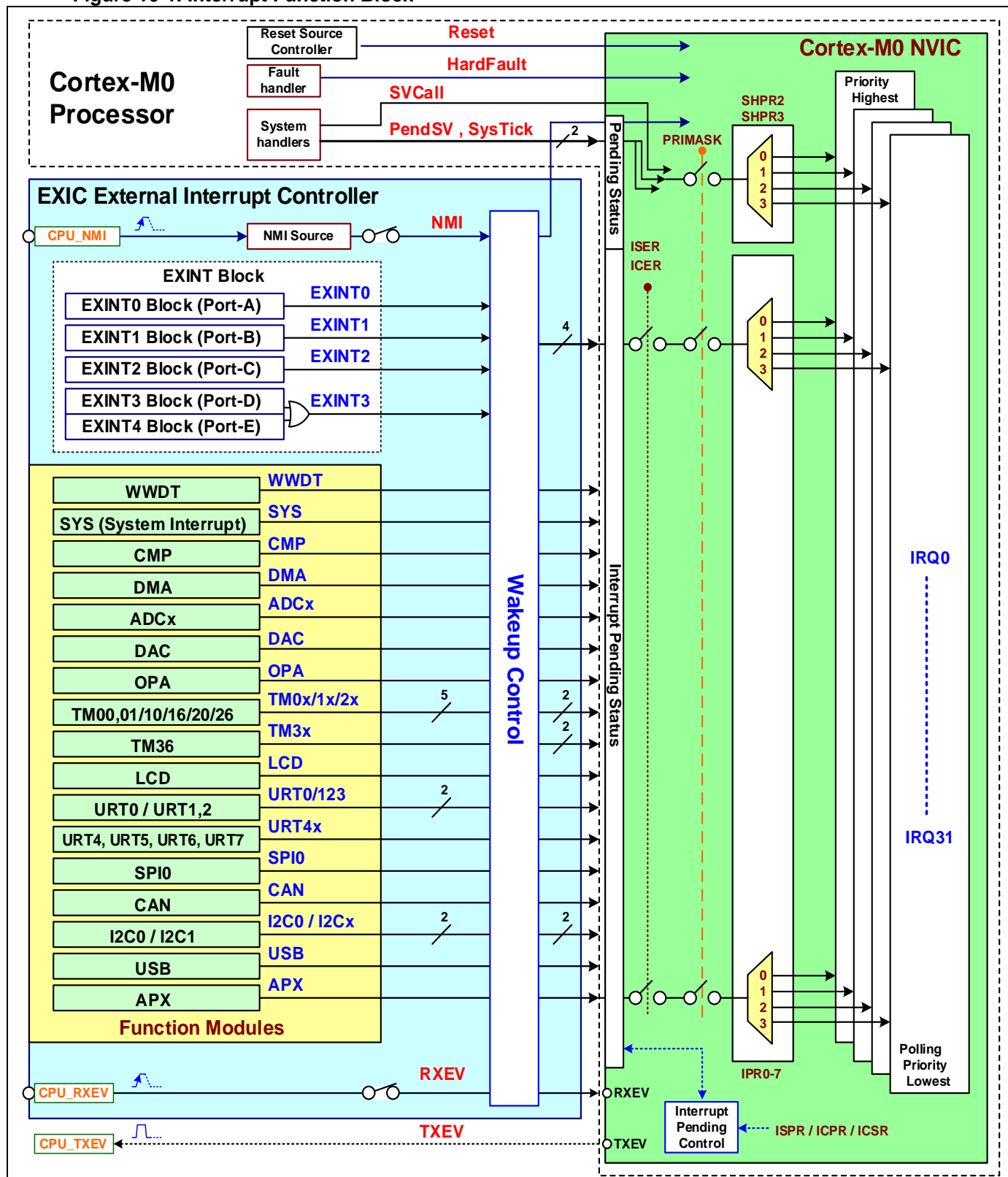
Item	Lockup Exit Events
1	it is reset
2	a debugger halts it
3	an NMI occurs and the current lockup is in the HardFault handler

<Note> If lockup state occurs in the NMI handler a subsequent NMI does not cause the processor to leave lockup state.

10.4. Interrupt Control Block

The interrupt control block is including of Cortex®-M0 NVIC and EXIC controller. The following diagram is showing the interrupt control block.

Figure 10-1. Interrupt Function Block



10.5. Nested Vectored Interrupt Controller

10.5.1. NVIC Function

The Cortex®-M0 processor integrates a configurable Nested Vectored Interrupt Controller (NVIC) that supports low latency interrupt processing and includes a non-mask interrupt (**NMI**). The NVIC provides a zero-jitter interrupt option and four interrupt priority levels. Refer the ARM® “Cortex-M0 Devices Generic User Guide” for more information about NVIC.

Interrupt handlers do not require any assembler wrapper code, removing any code overhead from the ISRs. Tail-chaining optimization also significantly reduces the overhead when switching from one ISR to another.

To optimize low-power designs, the NVIC integrates with sleep mode. Optionally, sleep mode support can include a deep sleep function that enables the entire device to be rapidly powered down.

- **NVIC supports:**

- An implementation-defined number of interrupts, in the range 1-32.
- Programmable priority levels of 0-192 in steps of 64 for each interrupt.

A higher level corresponds to a lower priority, so level 0 is the highest interrupt priority.

- Level and pulse detection of interrupt signals.
- Interrupt tail-chaining and Late-arriving.
- An external NMI.

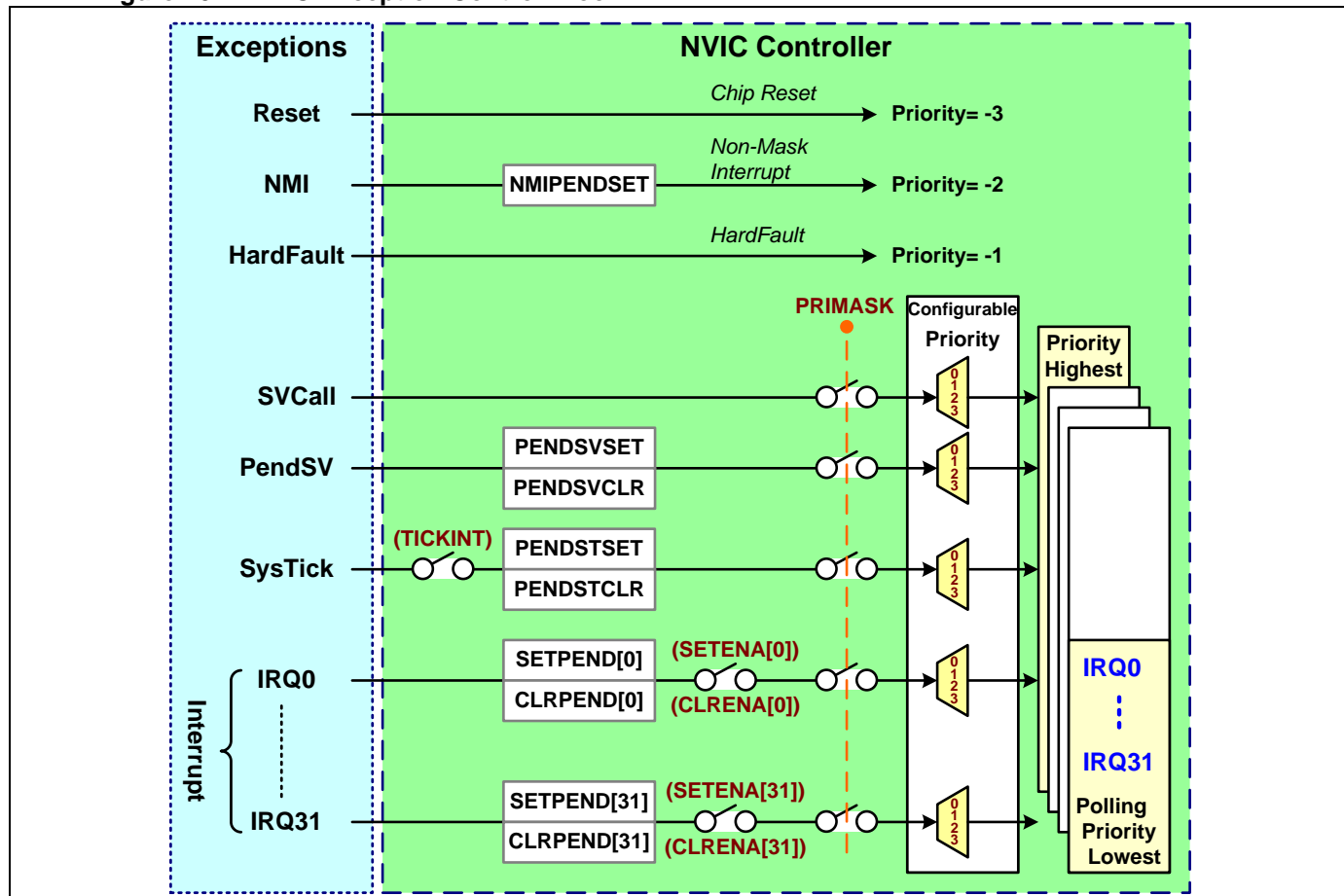
10.5.2. NVIC Exception Control

The NVIC supports 7 exception types: **Reset**, **NMI**, **HardFault**, **SVCall**, **PendSV**, **SysTick** and Interrupt (IRQ). Refer the section of “[Exception types](#)” for more information.

- **Exception Control**

The following diagram is showing the NVIC exception control block. When the **SysTick** is underflow and **TICKINT** bit is set 0, the **SysTick** pending bit will not be active and does not assert the exception request.

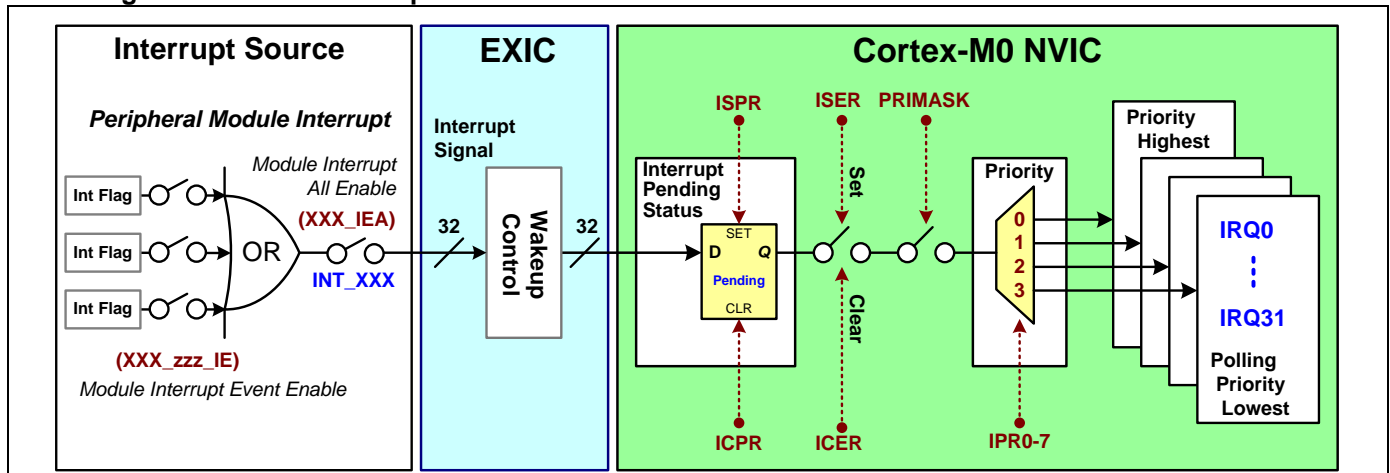
Figure 10-2. NVIC Exception Control Block



● Interrupt enable and pending

The NVIC provides the CPU registers of **ISER** and **ICER** to set and clear the interrupt enable bit independently for each interrupt exception. Also it provides the CPU registers of **ISPR** and **ICPR** to set and clear the interrupt pending bit independently for each interrupt exception. If a pending interrupt is enabled, the NVIC activates the interrupt based on its priority. If an interrupt is not enabled, asserting its interrupt signal changes the interrupt state to pending, but the NVIC never activates the interrupt. The **PRIMASK** register prevents activation of all exceptions with configurable priority.

Figure 10-3. NVIC Interrupt Control



● Interrupt block conditions

Each interrupt flag is sampled at every system clock cycle. If one of the flags was in a set condition, the interrupt system will generate a hardware exception and jump to the appropriate service routine as long as it is not blocked by any of the following conditions.

- The related interrupt enable bit(s) is/are not set.
- An interrupt of equal or higher priority level is already in progress.

[Notify]: The related interrupt enable bits include the interrupt event enable bit (xxx_zzz_IE), interrupt-all enable bit (xxx_IEA) of interrupt source module and the interrupt enable bit of NVIC.

● Interrupt priority

The NVIC provides the interrupt priority registers of **IPR0 ~ IPR7** which provide an 8-bit priority field for each interrupt, and each register holds four priority fields. This means the number of registers is implementation-defined, and corresponds to the number of implemented interrupts. These registers are only word-accessible.

If an interrupt is not enabled, the NVIC never activates the interrupt and is regardless of its priority.

10.5.3. Level-sensitive and pulse interrupts

The NVIC can support both level-sensitive and pulse interrupts. Pulse interrupts are also described as edge-triggered interrupts.

A level-sensitive interrupt is held asserted until the peripheral releases the interrupt signal. Typically this happens because the ISR accesses the peripheral, causing it to clear the interrupt request. A pulse interrupt is an interrupt signal sampled synchronously on the rising edge of the processor clock. To ensure the NVIC detects the interrupt, the peripheral must assert the interrupt signal for at least one clock cycle, during which the NVIC detects the pulse and latches the interrupt.

When the processor enters the ISR, it automatically removes the pending state from the interrupt, see “Hardware and software control of interrupts”. For a level-sensitive interrupt, if the signal is not deasserted before the processor returns from the ISR, the interrupt becomes pending again, and the processor must execute its ISR again. This means that the peripheral can hold the interrupt signal asserted until it no longer requires servicing.

● Hardware and software control of interrupts

The NVIC latches all interrupts. A peripheral interrupt becomes pending for one of the following reasons:

- the NVIC detects that the interrupt signal is active and the corresponding interrupt is not active
- the NVIC detects a rising edge on the interrupt signal

— software writes to the corresponding interrupt set-pending register bit

10.5.4. Hard Fault handling

Hard faults are a subset of exceptions. All faults result in the HardFault exception being taken or cause lockup if they occur in the NMI or HardFault handler. Only Reset and NMI can preempt the fixed priority Hard Fault handler. A HardFault can preempt any exception other than Reset, NMI, or another HardFault.

The following table lists hard fault handling events on Cortex®-M0.

Table 10-3. Hard Fault Handling Events on Cortex-M0

Item	Faults
1	execution of an SVC instruction at a priority equal or higher than SVCall
2	execution of a BKPT instruction without a debugger attached
3	a system-generated bus error on a load or store
4	execution of an instruction from an XN memory address
5	execution of an instruction from a location for which the system generates a bus fault
6	a system-generated bus error on a vector fetch
7	execution of an Undefined instruction
8	execution of an instruction when not in Thumb-State as a result of the T-bit being previously cleared to 0
9	an attempted load or store to an unaligned address

<Note-1> All faults result in the HardFault exception being taken or cause lockup if they occur in the NMI or HardFault handler.

<Note-2> A HardFault can preempt any exception other than Reset, NMI, or another HardFault.

<Note-3> The hard fault exception handler routine typically resets the system if a hard fault does occur.

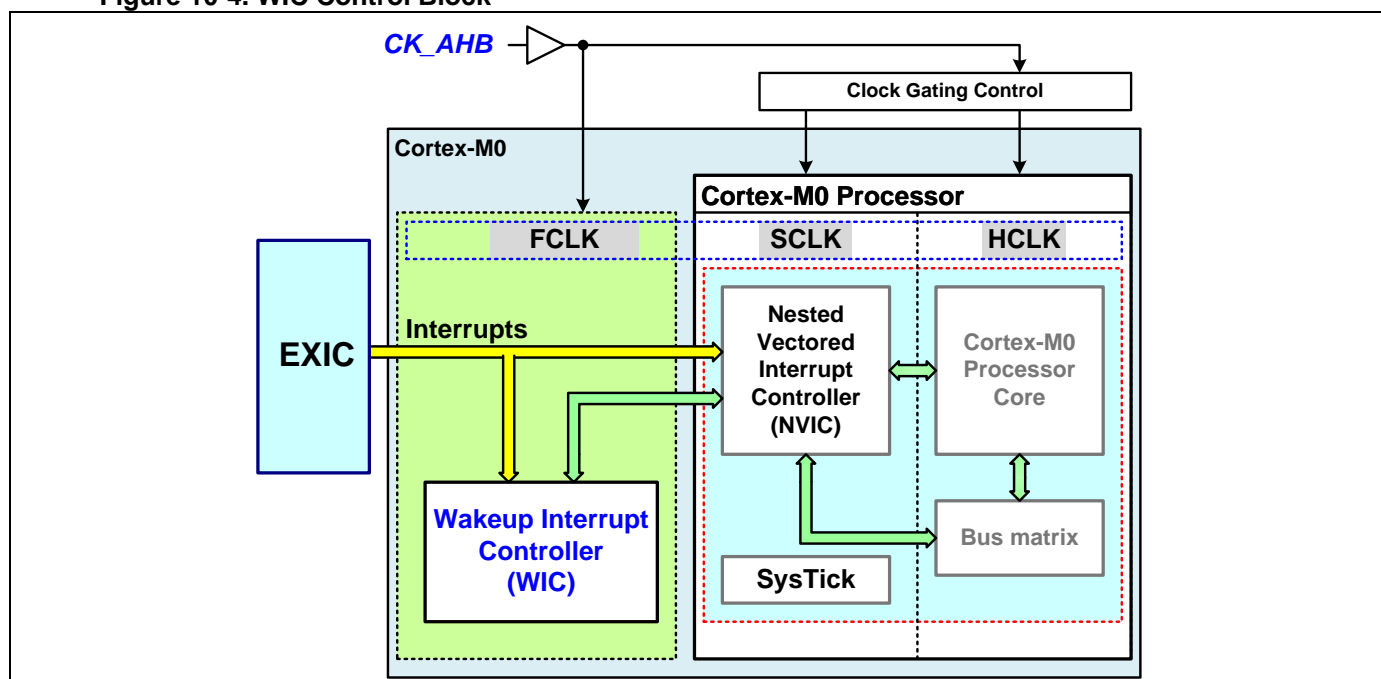
10.6. Wakeup Interrupt Controller

The chip includes a Wakeup Interrupt Controller (WIC) which can detect an interrupt or wakeup event from EXIC and wake the processor from deep sleep mode. The WIC is enabled only when the **DEEPSLEEP** bit in the CPU register of **SCR** is set to 1. The WIC is not programmable, and does not have any registers or user interface. It operates entirely from hardware signals.

When the WIC is enabled and the processor enters deep sleep mode, the PW controller will power down CPU. This has the side effect of stopping the SysTick timer. When the WIC receives an interrupt, it takes a number of clock cycles to wakeup the processor and restores its state before it can process the interrupt. This means interrupt latency is increased in deep sleep mode.

The following diagram is showing the WIC control block.

Figure 10-4. WIC Control Block

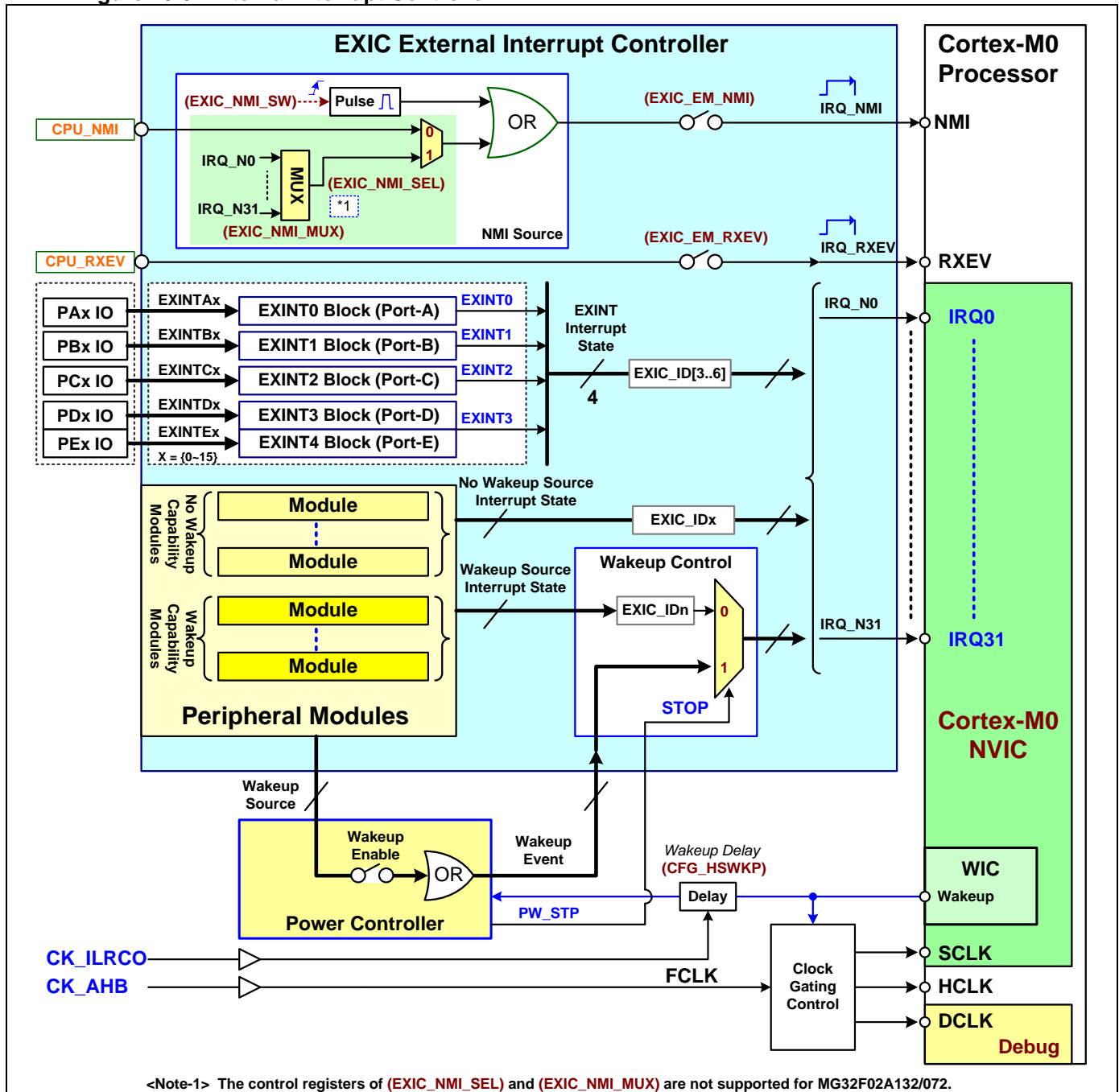


10.7. External Interrupt Controller

The External Interrupt Controller (EXIC) includes four external port interrupt blocks (EXINT) to manage the external pin input interrupt events, one wakeup control block for wakeup event control and control the NMI/RXEV events. The EXIC also do as the interface controller between internal modules and NVIC for the interrupt and wakeup events management.

The following diagram is showing the EXIC control block.

Figure 10-5. External Interrupt Controller



10.7.1. EXIC Interrupt Control

All of the bits that generate interrupts can be set or cleared by software, with the same result as though it had been set or cleared by hardware. In other words, interrupts can be generated by software and also the pending interrupts can be canceled by software.

- **Interrupt Source Identity**

When interrupt events have happened, user can read the **EXIC_ID0 ~ EXIC_ID31** registers to check the

interrupt event source and serve the related event by firmware. There is an independent identity bit for each interrupt source.

Refer the register descriptions of **EXIC_SRC0 ~ EXIC_SRC7** for the detail information.

● NMI Control

The chip supports a **NMI** pin input and sends the non-maskable interrupt (**NMI**) signal to NVIC. One masked control register bit of **EXIC_EM_NMI** is used to mask the **NMI** signal and one software NMI trigger bit of **EXIC_NMI_SW** is used to trigger the **NMI** for testing. User can select the NMI signal source from external **CPU_NMI** or internal interrupt events **IRQ_Nn** by setting **EXIC_NMI_SEL** register. When selects internal interrupt events, user also can select the **IRQ_Nn** signals by setting **EXIC_NMI_MUX** register.

● RXEV and TXEV Control

The chip supports a **RXEV** pin input and a **TXEV** pin output by GPIO AFS setting. When the CPU is entering sleep or deep sleep mode by CPU instruction “**WFE**” and **RXEV** input signal is active, the chip will be waked up after NVIC detects rising edge of the signal **RXEV**. One masked control register bit of **EXIC_EM_RXEV** is used to mask the **RXEV** signal.

When the CPU is executing the CPU instruction “**SEV**”, the chip will send an active pulse to **TXEV** output pin by GPIO AFS setting.

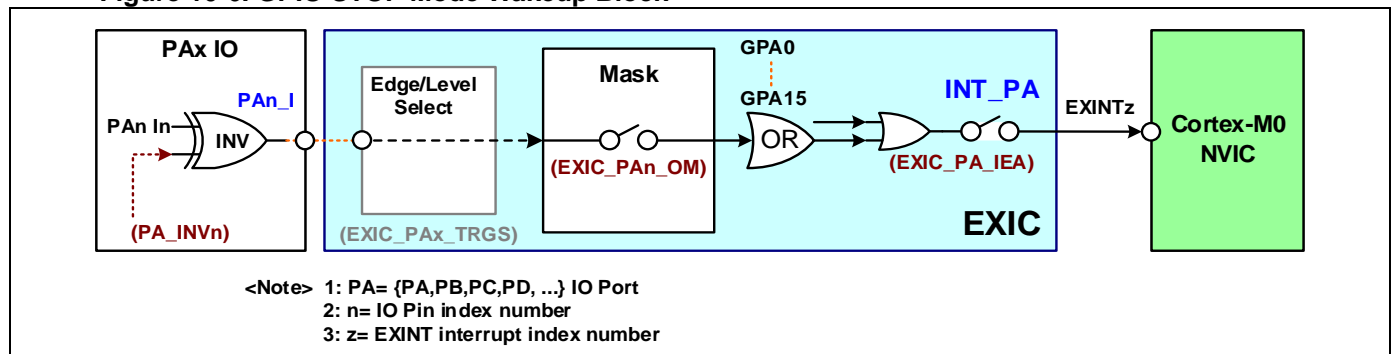
● Wakeup Control

The EXIC is built in a wakeup control block to receive the interrupt events from interrupt source and wakeup events from PW controller. When the chip operates in **ON** mode, the wakeup control block will pass the interrupt events to NVIC for interrupt function. When the chip operates in **SLEEP** mode, the wakeup control block will also pass the interrupt events to NVIC for **SLEEP** mode wakeup function. At the time, the interrupt events are using as wakeup events.

When the chip operates in **STOP** mode, the wakeup control block will send the wakeup events of PW to Wakeup Interrupt Controller (WIC) for **STOP** mode wakeup function. Refer the section of “[Wakeup Control](#)” in System Power chapter for more information about the wakeup function.

User can enable to wake up by only level trigger detection of external interrupt GPIO pins from **STOP** mode by setting **EXIC_Pxn_OM** and **EXIC_Px_IEA** registers. Refer the following figure of “GPIO STOP Mode Wakeup Block”, in **STOP** mode the register setting of **EXIC_Pxn_TRGS** is no effect and the external GPIO input signal is bypass the “Edge/Level Select” block through EXIC block to NVIC block. When **Px_INVn** register is disabled, the chip will be waked up if the external GPIO input signal(PAx In) is changed to low level. When **Px_INVn** register is enabled, the chip will be waked up if the external GPIO input signal(PAx In) is changed to high level.

Figure 10-6. GPIO STOP Mode Wakeup Block



10.7.2. External Port Input Interrupt

There are four external port interrupt blocks (EXINT) in the EXIC controller to manage the external pin input interrupt events for GPIO port A, B, C, D and E. Each one of the external port interrupt blocks support that all pins of one GPIO port can be configured as the interrupt or wakeup event sources and the key pad input. It provides independent high/low level and rising/falling edge trigger selection by pin independent.

The EXINT supports the port OR logic for interrupt function and the port AND logic for key board input (KBI) function. There is one interrupt-all enable bit of **EXIC_Px_IEA** for each GPIO port.

- **External Interrupt Flag**

There is one independent interrupt pending flag of **EXIC_Pxn_PF** for each GPIO pin. The flag is used to indicate that the event is detected by use configured Interrupt edge or level. It set by hardware and software write 1 to clear the interrupt pending flag.

- **External Interrupt Edge and Level Selection**

There is one independent **EXIC_Pxn_TRGS** register configures the pin change detection level on low / falling or high / rising event for each GPIO pin. When the register is set 0, it disables external interrupt pending flag of **EXIC_Pxn_PF** to be update.

Set the input signal inversion register bit of **Px_INVn** to select low/high level or rising/falling edge. When **Px_INVn=0**, select low level if **EXIC_Pxn_TRGS=0x01** and falling edge if **EXIC_Pxn_TRGS=0x02**. In **STOP** mode, this function is forced to level-sensitive operation by hardware however any of the falling edge or rising edge configurations. (x= {A, B, C, D, E}; n= {0~15})

- **External Interrupt Mask**

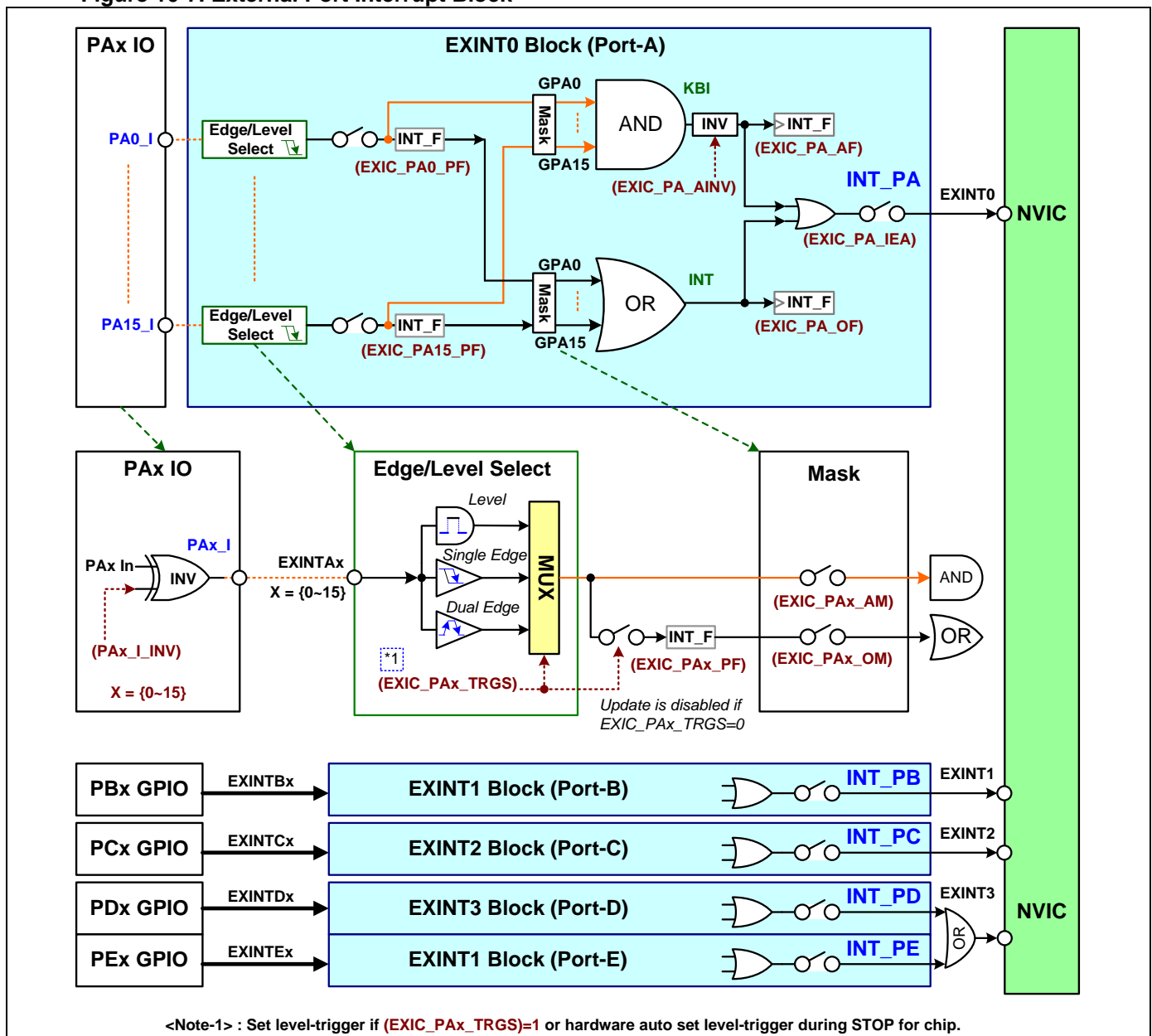
For interrupt function, there is a port OR logic to OR all the external interrupt input signals of one GPIO port. When an external interrupt is detected, the interrupt pending flag of **EXIC_Pxn_PF** is generated and asserted an interrupt if the interrupt-all enable bit of **EXIC_Px_IEA** is enabled. There is an external interrupt OR path interrupt flag of **EXIC_Px_OF** to indicate that any of the interrupt events of Px GPIO port is detected. User can mask any the interrupt input by setting the **EXIC_Pxn_OM** register for Px GPIO pin independently.

- **External KBI Mask**

For KBI function, there is a port AND logic to AND all the external KBI input signals of one GPIO port. User can mask any the KBI input by setting the **EXIC_Pxn_AM** register for Px GPIO pin independently. When all the non-masked external KBI input is detected, the AND path KBI detection flag of **EXIC_Px_AF** is generated and asserted an interrupt if the interrupt-all enable bit of **EXIC_Px_IEA** is enabled. There is an inverted control bit of **EXIC_Px_AINV** to invert the AND logic output for KBI function.

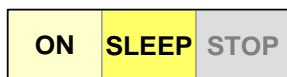
The following diagram is showing the external port interrupt block.

Figure 10-7. External Port Interrupt Block



11. GPL (General Purpose Logic)

11.1. Introduction



The module can be running in ON and SLEEP modes only.

The chip builds in one general purpose logic (GPL) module. It provides the combined functions of Data Order Change, Data Inverse, Parity Check and CRC.

11.2. Features

- Support data inverse, bit order change, byte order change and parity check
 - Data bit order change for 8/16/32-bit reverse
 - Data byte order change for Little endian to Big endian
 - Parity Check for 8/16/32 bit range
- Support CRC (Cyclic Redundancy Check) calculation
 - Programmable CRC initial value
 - CRC output bit order change
 - CRC computation done in 4/2/1 AHB clock cycles for 32/16/8-bit data
- CRC with fixed common polynomial
 - CRC8 polynomial 0x07
 - CRC16 polynomial 0x8005
 - CCITT16 polynomial 0x1021
 - CRC32(IEEE 802.3) polynomial 0x4C11DB7
- Input data are buffered with DMA capability
- Support signed/unsigned 32-bit divider
 - Operation in 8 AHB clock cycles
 - Division by zero error flag

11.3. Implementation

11.3.1. Chip Implementation

The following table is showing the implemented APB functions of chips.

Table 11-1. GPL Implementation

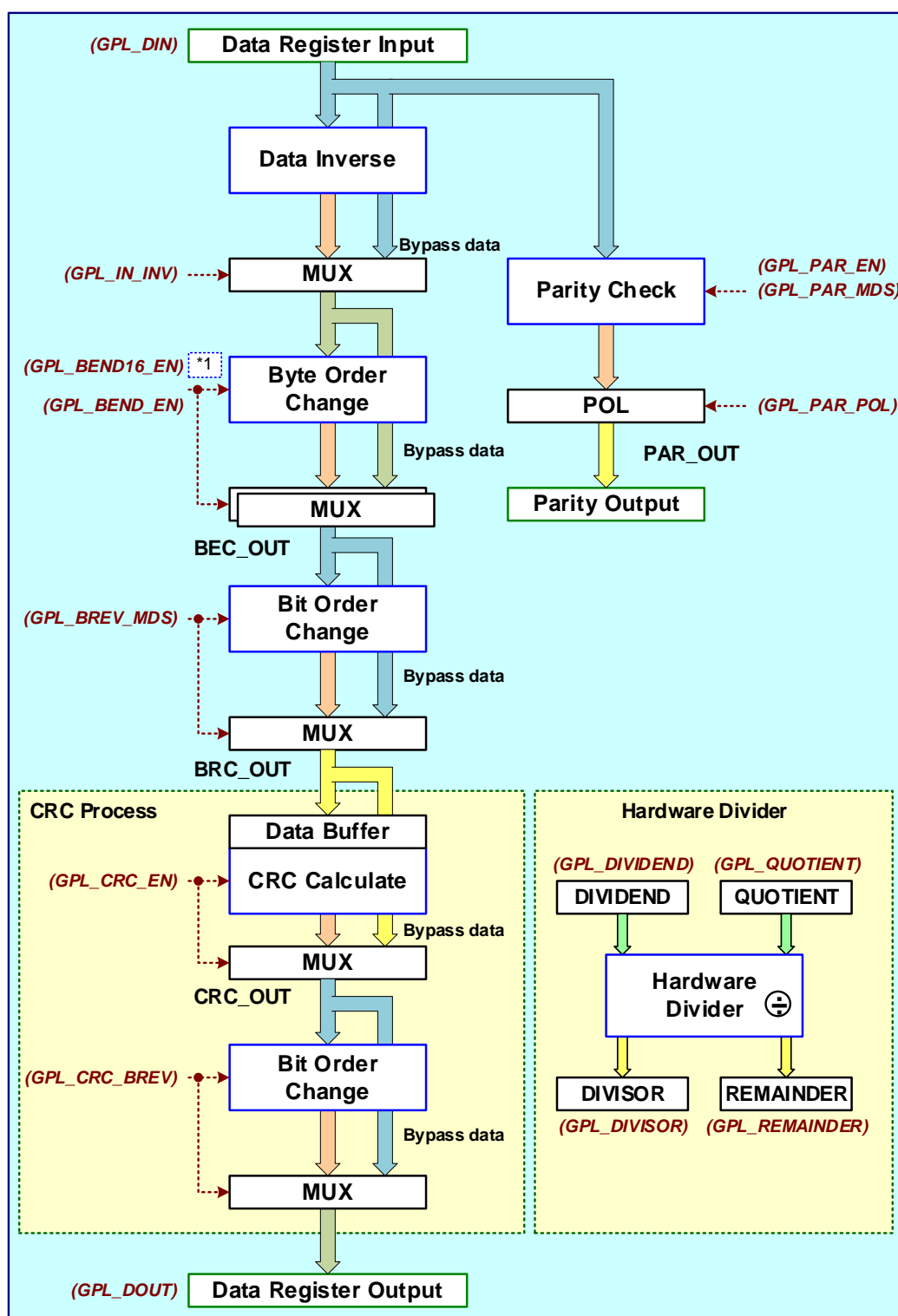
Chip	GPL Module Sub-Functions					
	Bit Reverse	Byte Swap	Parity Check	Data Inverse	CRC Mode	Hardware Divider
MG32F02V032 MG32F02N128/N064 MG32F02K128/K064	8/16/32-bit	16/32-bit	8/16/32-bit	V	32/16/8-bit	-
MG32F02A128/A064 MG32F02U128/U064	8/16/32-bit	16/32-bit	8/16/32-bit	V	32/16/8-bit	32-bit

<Note> V: Implemented

11.4. Control Block

The following diagram is showing the GPL control block.

Figure 11-1. General Purpose logic



<Note-1> The control register of **(GPL_BEND16_EN)** is not supported for MG32F02A132/A072.

<Note-2> The Hardware Divider is not supported for MG32F02A132/A072/A032.

11.5. Clock

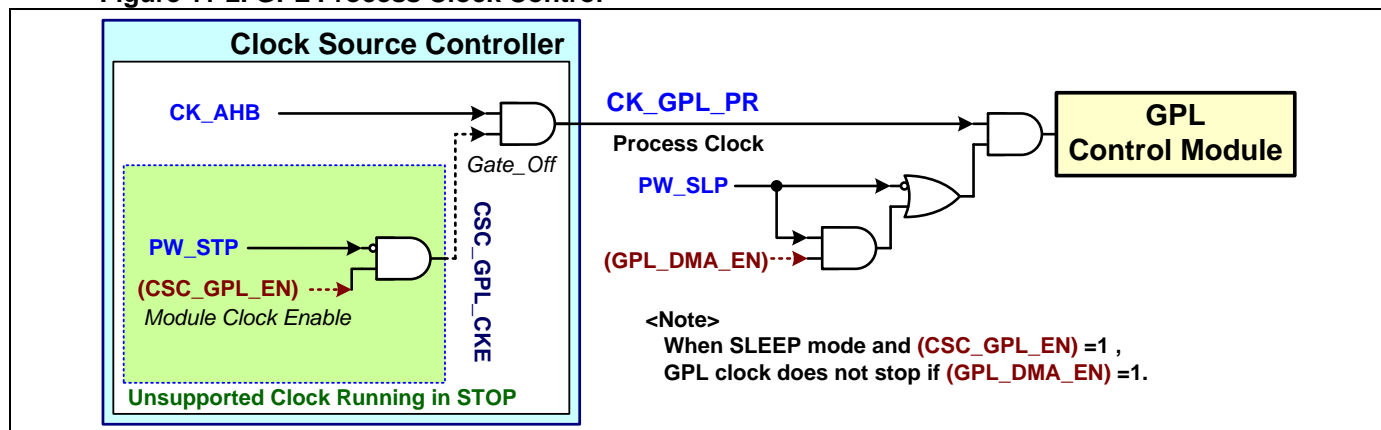
11.5.1. GPL Clock Control

- **Module Process Clock**

The module process clock of **CK_GPL_PR** is using for the interface control logic between AHB bus and the module. It is coming from CSC (Clock Source Controller) module. It can be enabled in **CSC_GPL_EN** register.

In **ON** mode, the GPL process clock is running only if **CSC_GPL_EN** register is enabled. In **SLEEP** mode, the GPL process clock is running if both **CSC_GPL_EN** and **GPL_DMA_EN** registers are enabled. Usually **GPL_DMA_EN** register is enabled for CRC data calculation through DMA, the process clock must not stop. Refer the System Clock chapter for more information. In **STOP** mode, the GPL process clock is always stopping.

Figure 11-2. GPL Process Clock Control



11.6. GPL Function Control

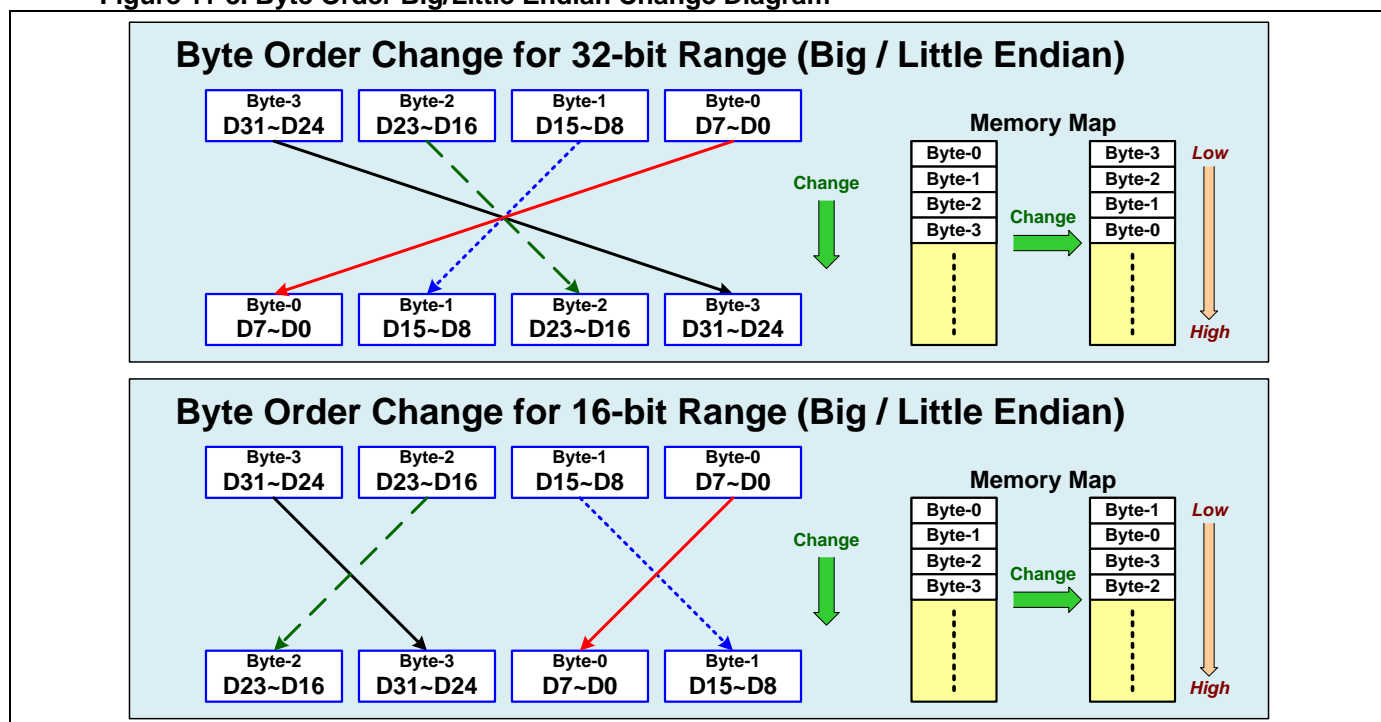
There are one input data register of **GPL_DIN** and one output data register of **GPL_DOUT** for the GPL process. For write operation, the **GPL_DIN** register is used to write new calculation data. For read operation, this register is used to read the previous CRC calculation result. The **GPL_DOUT** register is storing the calculated process result except parity check function.

11.6.1. Byte Order Change

The GPL can change the byte order of input data for big or little endian format for 32-bit range or 16-bit range. This process is enabled by setting **GPL_BEND_EN** and **GPL_BEND16_EN** register.

The following diagram is showing the GPL byte order change control block.

Figure 11-3. Byte Order Big/Little Endian Change Diagram

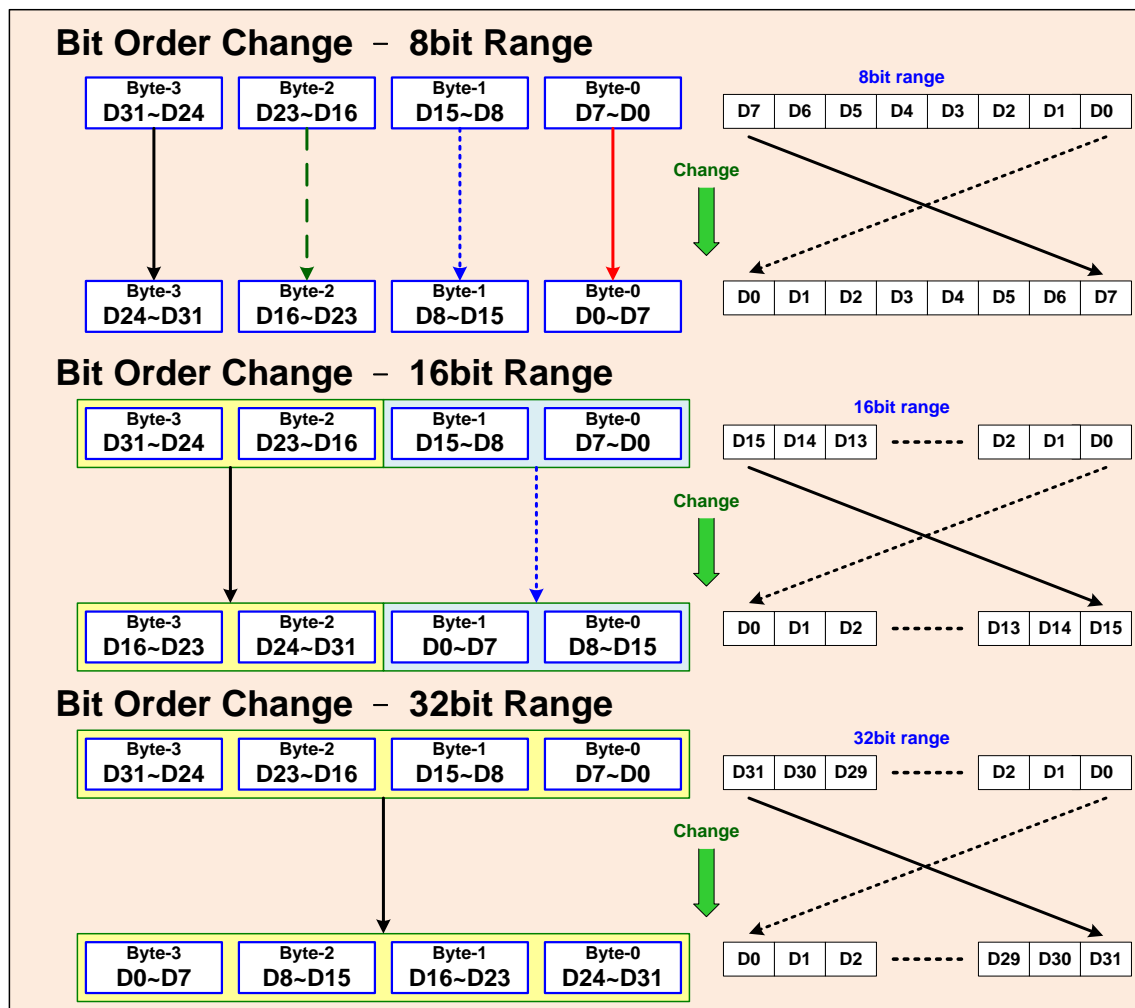


11.6.2. Bit Order Change

The GPL can change the bit order of input data for 8/16/32-bit reversion. This process is able to select data size of 8-bit, 16-bit and 32-bit by setting **GPL_BREV_MDS** register.

The following diagram is showing the GPL bit order reverse change control block.

Figure 11-4. Bit Order Reverse Change Diagram



11.6.3. Data Invert

The GPL can invert the data value from input data and enable by setting **GPL_IN_INV** register.

11.6.4. Parity Check

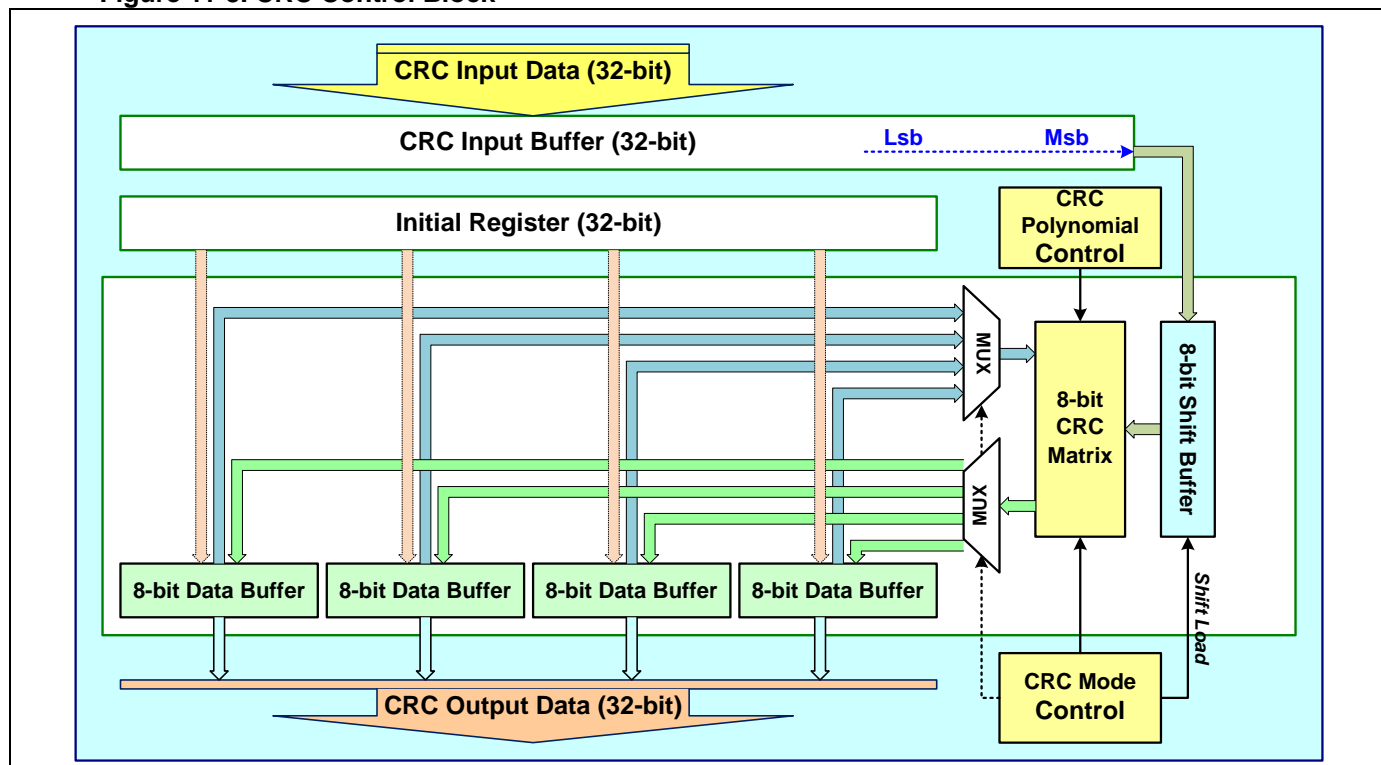
The GPL can set the parity check polarity by odd or even in **GPL_PAR_POL** register and calculate parity check value from input data. It can output the result simultaneous to **GPL_PAR8_OUT**, **GPL_PAR16_OUT** and **GPL_PAR32_OUT** bits for 8-bit, 16-bit and 32-bit input data.

11.6.5. Cyclic Redundancy Check

The CRC (cyclic redundancy check) block is used to get an 8/16/32-bit CRC data code and calculates the result with data buffer by through a CRC polynomial control block. The CRC block can process continuously with sequent CRC code and store the last result into data buffer always.

The following diagram is showing the CRC control block.

Figure 11-5. CRC Control Block



This CRC process is enabled by setting **GPL_CRC_EN** register and is set the CRC polynomial mode by setting **GPL_CRC_MDS** register. The CRC polynomial control block is support following fixed common polynomial.

Figure 11-6. CRC Polynomials

CRC-CCITT	$X^{16} + X^{12} + X^5 + 1$
CRC-8	$X^8 + X^2 + X + 1$
CRC-16	$X^{16} + X^{15} + X^2 + 1$
CRC-32	$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$

The CRC is able to select the input CRC data size of 8-bit, 16-bit and 32-bit by setting **GPL_CRC_DSIZE** register. Also user can set the CRC calculation initial value in **GPL_CRC_INIT** register.

There is one Bit Order Change block to process bit order reversion for the CRC output data. This process is able to select data size of 8-bit, 16-bit and 32-bit by setting **GPL_CRC_BREV** register.

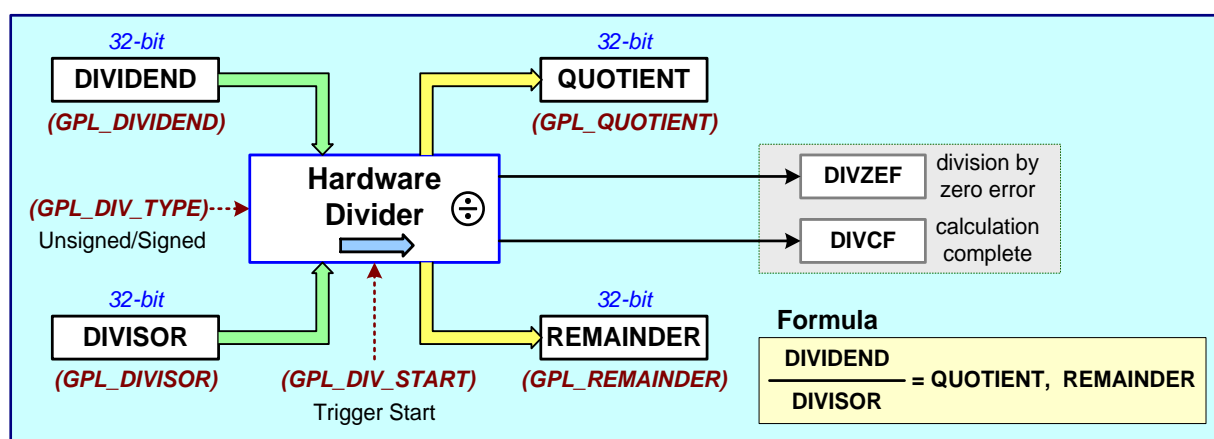
11.6.6. Hardware Divider

As the ARM® 32-bit Cortex®-M0 CPU is not built-in the hardware arithmetic divider. Generally user needs to cost code size to develop the software arithmetic divider function during firmware development for application. The GPL module is including a 32-bit hardware arithmetic divider. The build-in hardware divider can execute one 32-bit arithmetic division operation in 8 AHB clock cycles. It therefore can save the time and code size to replace software arithmetic divider function. The hardware divider is supported both unsigned and signed arithmetic calculation by setting **GPL_DIV_TYPE**.

User only needs to input the Dividend value and Divisor value to the **GPL_DIVIDEND** and **GPL_DIVISOR** registers. Then user starts the hardware divider operation by setting **GPL_DIV_START** bit. After eight CPU clock time, user can get the result of Quotient value and Remainder value in **GPL_QUOTIENT** and **GPL_REMAINDER** registers. Also the hardware divider provides one calculation complement flag **GPL_DIVCF** for user using.

Others, a “divider division by zero” error flag **GPL_DIVZEF** is use to indicate the Divisor value is zero. User must not input a zero value to the **GPL_DIVISOR** register.

Figure 11-7. Hardware Divider



11.7. GPL DMA Operation

11.7.1. DMA Module Configure

When the chip supports a DMA (direct memory access) controller, user can configure the DMA setting of transferred source/destination devices, channel request arbitration and others in the DMA module before a DMA data transaction. The DMA source and the destination can be memory or peripheral.

Refer the DMA chapter for more detail information about the DMA module configuration.

11.7.2. GPL DMA Control

After DMA configuration is finished, user needs to set the GPL module DMA enable bit of **GPL_DMA_EN**.

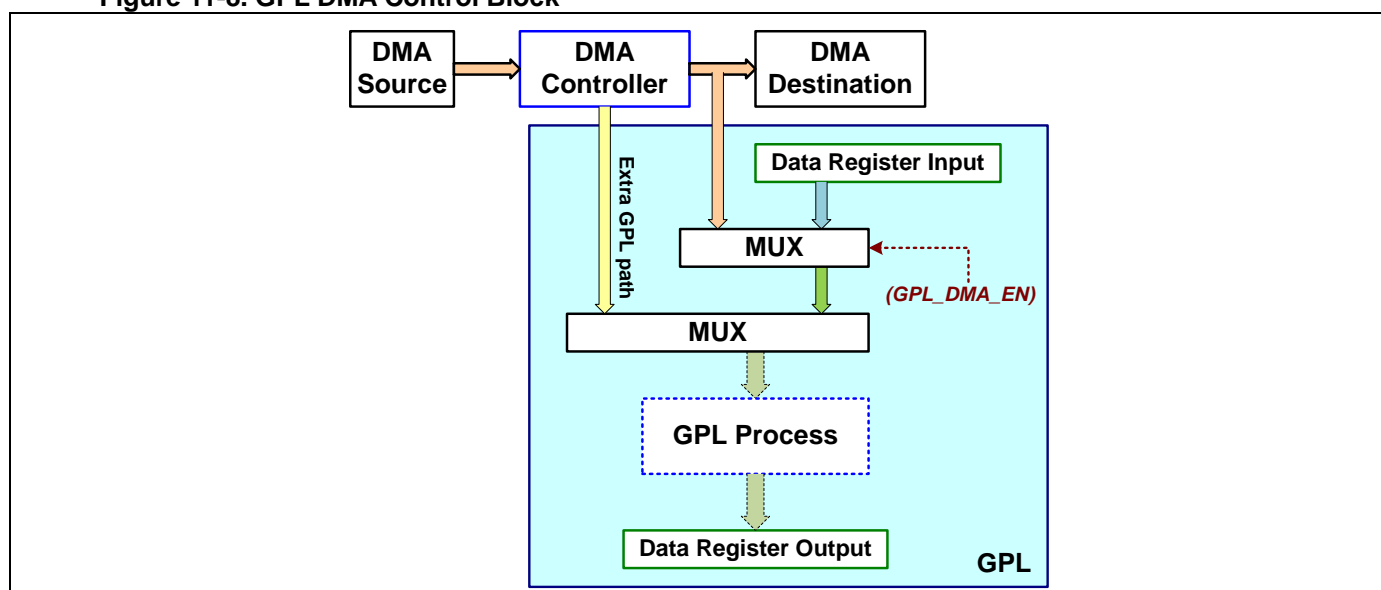
Finally, the related channel request start bit of **DMA_CHn_REQ** is necessary to be set to start the DMA transaction (n = DMA channel index). Then the transferred source/destination devices will assert the RX/TX request signal to DMA controller and the DMA controller will assert the acknowledge signal to the request source/destination devices. At the time, the data transferred connection is built for DMA transaction.

The **GPL_DMA_EN** register bit is used to enable DMA data transfer from DMA source to GPL module for GPL process. When **GPL_DMA_EN** bit is enabled, the input data path is switched to DMA destination data path and replaces the GPL data register input. After DMA operation finished, hardware will automatically disable the **GPL_DMA_EN** bit.

When a DMA source and a DMA destination are operating DMA process, the DMA supports an extra GPL path which data is copied from the destination data and directly sends to GPL to do GPL process. For this process, the GPL is hardware auto control and is not necessary to enable the **GPL_DMA_EN** bit. User can select the DMA channel source for the extra GPL data path by setting **DMA_GPL_CHS** register.

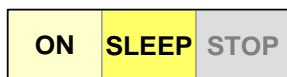
When the DMA source is the embedded flash and the DMA destination is GPL, user can configure the DMA transfer bus width 8-bit or 32-bit by setting **DMA_FGBUS_SEL** register. When the GPL DMA is enabled and **DMA_FGBUS_SEL**=0, the **GPL_CRC_DSIZE** register is fixed 8-bit setting by hardware. When the GPL DMA is enabled and **DMA_FGBUS_SEL**=1, the **GPL_CRC_DSIZE** register is fixed 32-bit setting by hardware.

Figure 11-8. GPL DMA Control Block



12. DMA (Direct Memory Access)

12.1. Introduction



The module can be running in ON and SLEEP modes only.

The chip is built-in a direct memory access controller (DMA) which is used to improve the performance of data transfer between peripheral and memory, memory to memory and peripheral to peripheral. Data can be quickly transfer by through DMA without costing any CPU resources.

Notify: The sign of (n= DMA channel index number) is using for Registers, Signals and Pins/Ports in the descriptions of this chapter. [EX]: **DMA_CHn_EN** ~ n indicates channel number.

12.2. Features

- DMA transfer management type
 - memory-to-memory, peripheral-to-memory, memory-to-peripheral, peripheral-to-peripheral
- 1~5 independently configurable channels with dedicated hardware DMA requests
 - Access to Memory, APB and AHB Peripherals as source and destination
 - Support SRAM/Flash/EMB as memory source and SRAM/EMB as memory destination
 - Peripherals are including of ADC0, DAC, I2Cx, URTx, SPIx, TM36, APX(ASB) and GPL modules
- Built-in two type priority control between channel requests
 - Channel request by Round Robin
 - Software configurable priority level
- Programmable transfer number of data and up to 65536 or 131072
- Programmable burst length 1,2,4
- Support transfer loop mode and start address auto reload control
- Provide single/block/demand mode for external pin trigger request

12.3. Implementation

12.3.1. Chip Implementation

The following table is showing the implemented DMA channels of chips.

Table 12-1. DMA Implementation

Chip	DMA Channel			DMA Memory Source (*1)			Transfer
	Channels	Channel Index	M-to-M Channel	SRAM	Flash	EMB	Max. Number
MG32F02A128/A064 MG32F02U128/U064	5	0,1,2,3,4	0,3	V	-	V	Max. 131072
MG32F02V032	4	0,1,2,3	0,3	V	V	-	Max. 65536
MG32F02N128/N064 MG32F02K128/K064	5	0,1,2,3,4	0,3	V	V	-	Max. 131072

<Note> V: Implemented; Only Channel-0, 3 support Memory-to-Memory operation.

*1: Flash cannot be DMA Destination for write access.

12.4. Control Block

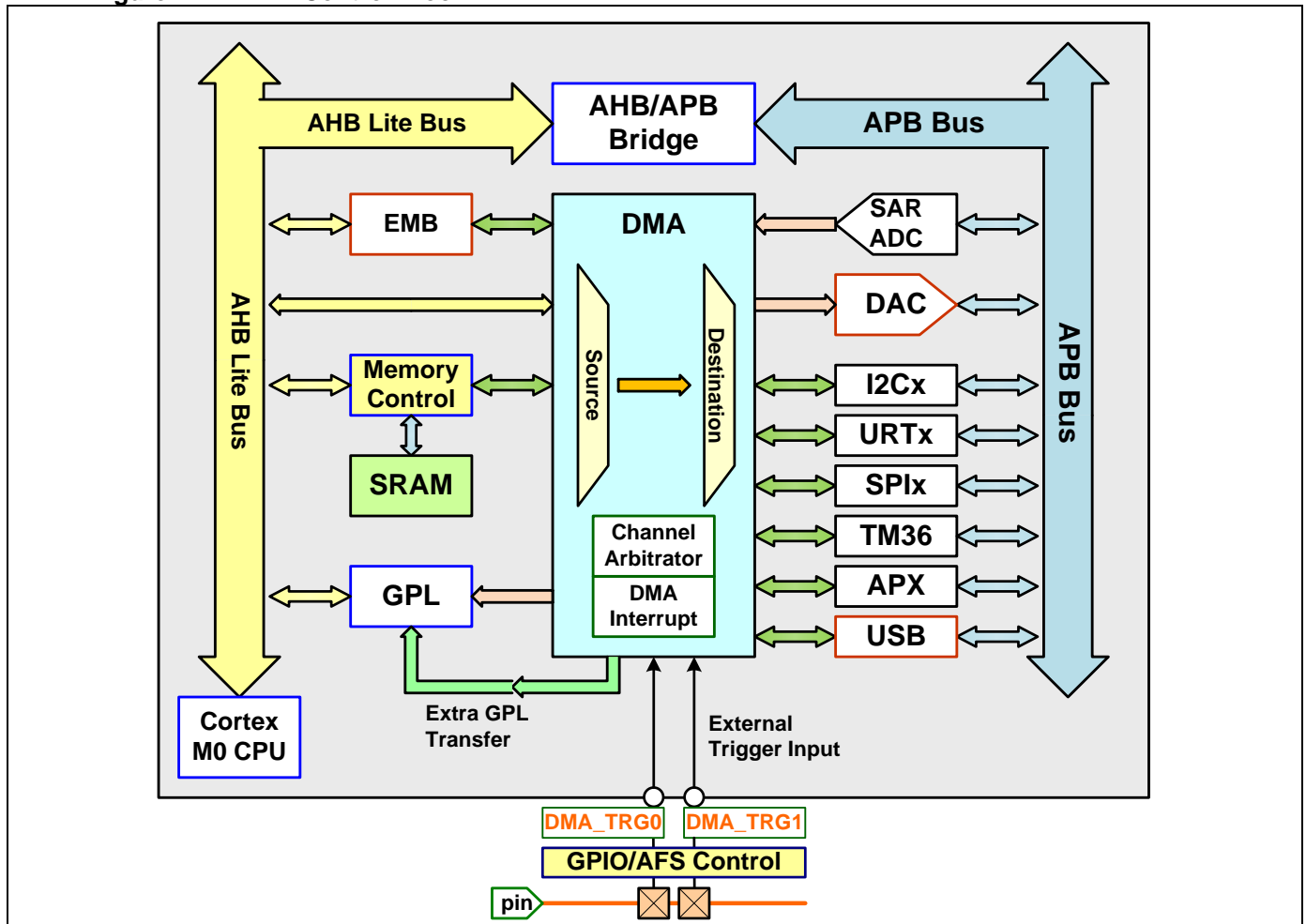
The DMA controller (DMA) is used to transfer data between these sources and destinations of AHB peripheral, APB peripheral, SRAM and external memory. A DMA interrupt block to detect and service for DMA events. Also the DMA has an arbiter for handling the priority of DMA requests.

Two external pins of **DMA_TRG0** and **DMA_TRG1** are able to input as the trigger signal for DMA data transaction.

The following diagram is showing the connection diagram of DMA control.

[Notify]: The USB modules is only supported for MG32F02U series.

Figure 12-1. DMA Control Block



12.5. IO Lines

12.5.1. IO Signals

- **DMA_TRGn**

It is the DMA external trigger signal input and uses to trigger DMA data transfer for Single mode, Block mode and Demand mode.

12.5.2. IO Configure

User must configure the related IO pins in order to use the IO lines of this module. The IO operation modes, output high speed option, pull-high option, output drive strength, IO deglitch filter and input inverse selection are programmable by pin independent. Please refer the section of “[IO Mode](#)” in GPIO chapter for more detail descriptions of IO mode configuration.

Each IO signal may be mapped and selected on several IO pins by configuring the IO AFS matrix. Please refer the section of “[Alternate Function Select](#)” in GPIO chapter for more detail descriptions of IO AFS configuration. About the actual IO pin AFS information, please refer the section of “Pin Alternate Functions Selected Table” in Pin Description chapter of the chip Data Sheet.

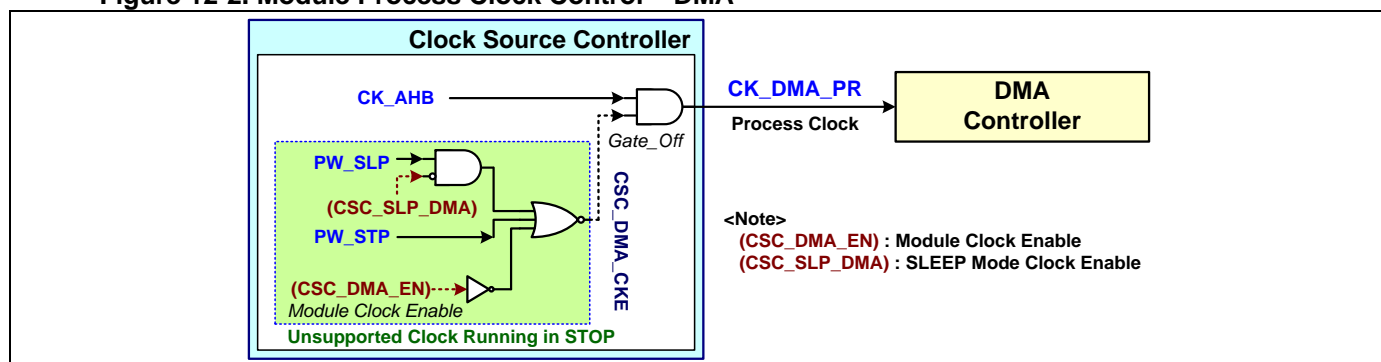
12.6. Enabling and Clock

There is a global enable bit of **DMA_EN** for all functions of this module. When this bit is disabled, all the DMA functions are not working.

The module process clock of **CK_DMA_PR** is using for the interface control logic between AHB bus and module. It is coming from CSC (Clock Source Controller) module. It can be enabled in **CSC_DMA_EN** register.

In **ON** mode, the process clock is running only if **CSC_DMA_EN** register is enabled. The DMA transaction can be operation and the process clock is running during **SLEEP** mode if both **CSC_DMA_EN** and **CSC_SLP_DMA** register is enabled. In **STOP** mode, the process clock is always stopping. Refer the System Clock chapter for more information.

Figure 12-2. Module Process Clock Control – DMA



12.7. Interrupt and Event

There is one signal of **INT_DMA** to be generated in this DMA module. **INT_DMA** sends to External Interrupt Controller (EXIC) to do as an interrupt event.

12.7.1. DMA Interrupt Control and Status

These interrupt flags are using for the interrupt service routine (ISR) flow control. Generally these flags are set by hardware and software clears them when the serviced job is completed in the related ISR. Each interrupt flag has one interrupt enable bit. User can enable or disable it. There is one interrupt all enable bit of **DMA_IEA** to enable or disable all the interrupt sources for this module.

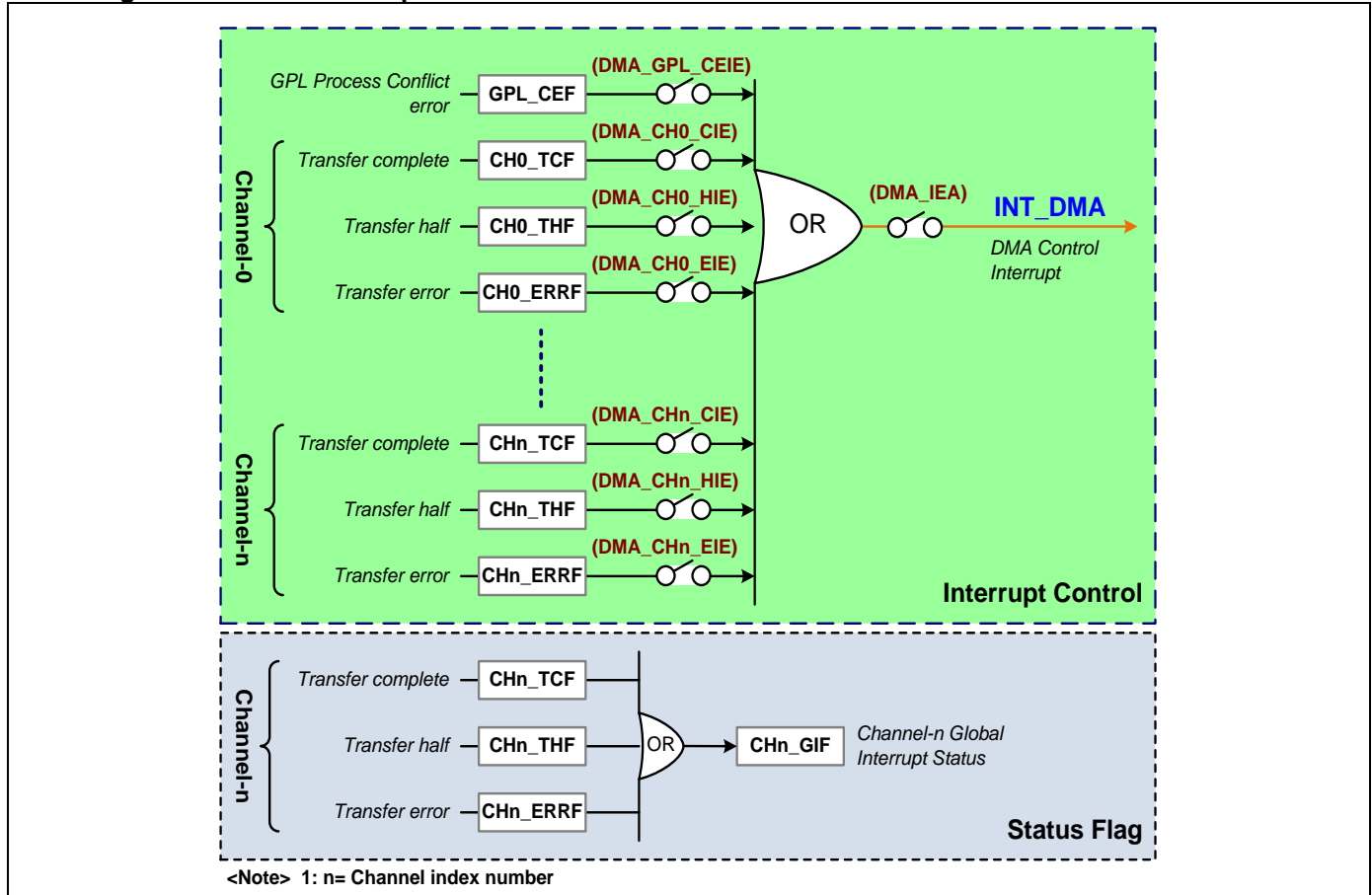
There is one global interrupt flag status (**DMA_CHn_GIF**) which is reading only for each DMA channel. This status bit is asserted to indicate any interrupt event is detected for this channel. Refer the register descriptions of the related status bits for more information.

12.7.2. DMA Interrupt Flags

Generally these interrupt flags are set by hardware and clear by software writing 1. Refer the register descriptions for more information about the related interrupt flags and interrupt enable bit.

The following diagram is showing the DMA status and interrupt control block. Refer the section of “[Chip Implement](#)” about the supported DMA channel number.

Figure 12-3. DMA Interrupt Control



● TCF / TC2F

DMA channel transfer complete flag (**DMA_CHn_TCF**). There is a related interrupt enable register bit of **DMA_CHn_CIE**. There is one independent flag and one independent interrupt enable bit for each channel. When all the DMA transaction data are transferred completely from source to destination, then set this flag. The TC2F flag (**DMA_CHn_TC2F**) is the identical flag to TCF flag for the choice by firmware using. All the TCF flags of every channel are grouped in the **DMA_STA** register. The TC2F flag is implemented and grouped with THF, ERRF flag and the related interrupt enable bits in the **DMA_CHnA** register for each channel independently.

● THF / TH2F

DMA channel transfer half flag (**DMA_CHn_THF**). There is a related interrupt enable register bit of **DMA_CHn_HIE**. There is one independent flag and one independent interrupt enable bit for each channel. When the DMA transaction data are transferred over the half data from source to destination, then set this flag. Use can prepare data beforehand for next DMA transfer by the flag asserting. The TH2F flag (**DMA_CHn_TH2F**) is the identical flag to THF flag for the choice by firmware using. All the THF flags of every channel are grouped in the **DMA_STA** register. The TH2F flag is implemented and grouped with TCF, ERRF flag and the related interrupt enable bits in the **DMA_CHnA** register for each channel independently.

● ERRF / ERR2F

DMA channel transfer error flag (**DMA_CHn_ERRF**). There is a related interrupt enable register bit of **DMA_CHn_EIE**. There is one independent flag and one independent interrupt enable bit for each channel. When all the DMA transaction data are transferred complete for a DMA channel (**DMA_CHn_CNT** =0) and the peripheral module is still asserted the DMA request, then the DMA channel transfer error is happened and the DMA controller sets this flag.

The ERR2F flag (**DMA_CHn_ERR2F**) is the identical flag to ERRF flag for the choice by firmware using. All the ERRF flags of every channel are grouped in the **DMA_STA** register. The ERR2F flag is implemented and grouped with TCF, THF flags and the related interrupt enable bits in the **DMA_CHnA** register for each channel independently.

12.8. DMA Control

12.8.1. DMA Source and Destination

The DMA controller has maximum 5 independent channels those each one can decide independently to manage data transactions between the source and the destination. The source and the destination can be memory or any peripheral. Also each channel has the one DMA source multiplex and one DMA destination multiplex to decide the DMA source and destination independently.

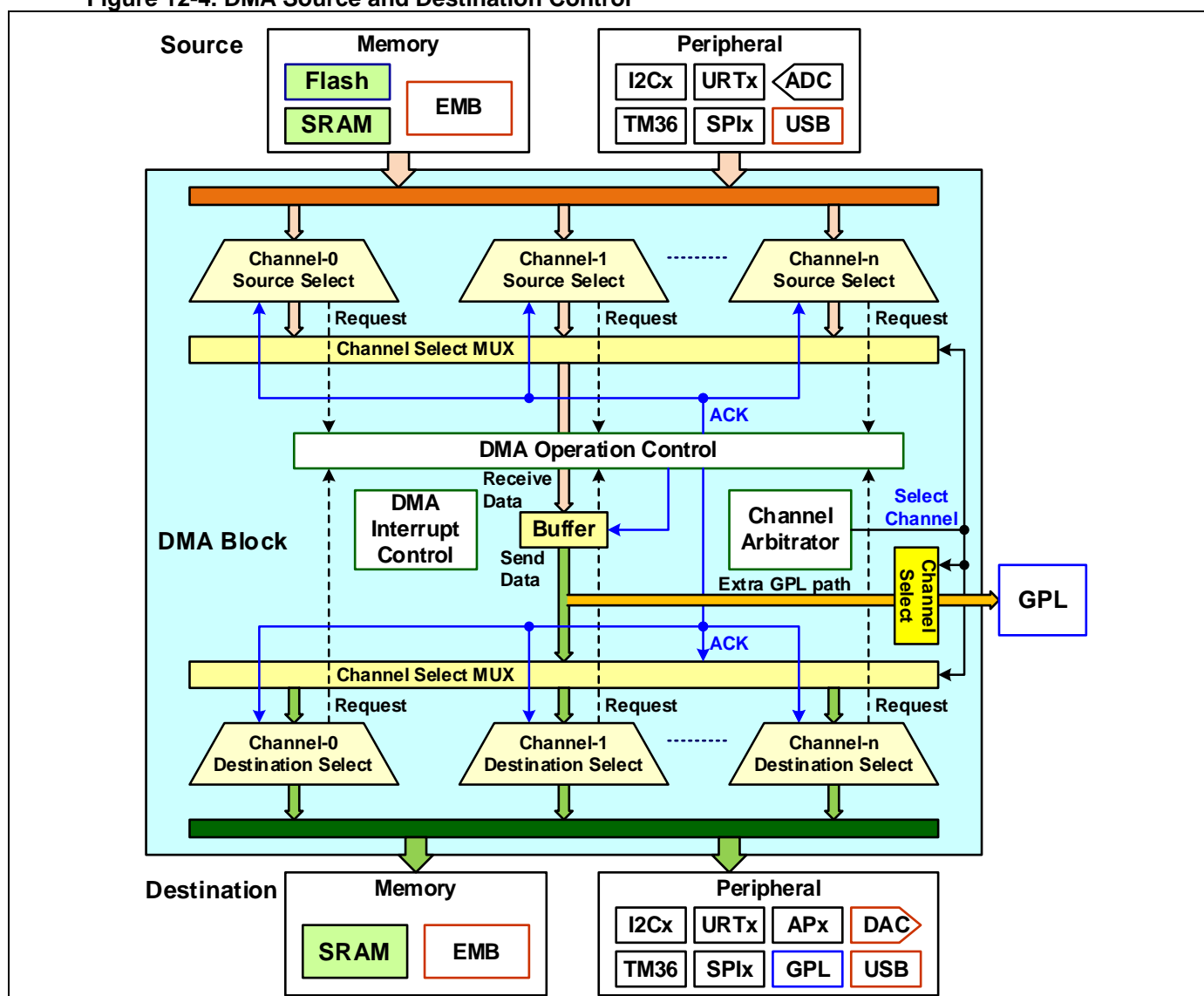
Specially, there is one extra GPL path which can copy the transferred data from a configured channel and send to GPL to operate the GPL process (like as CRC).

The following diagram is showing the DMA data transacted connections between source and destination.

[Notify]: Please refer the table of “[Chip Implementation Table](#)” in the section of “Applicable Chips” about MCU chip supported peripheral modules.

[Notify]: The USB modules is only supported for MG32F02U series.

Figure 12-4. DMA Source and Destination Control



Memory are including of embedded Flash, embedded SRAM and EMB access memory space. Peripherals are including of ADC0, DAC, I2Cx, URTx, SPIx, TM36 and GPL modules. The ADC module can only to be as DMA source. The DAC and GPL modules can only to be as DMA destination.

When the DMA source is selected embedded Flash, the AHB clock divider is limited to set divided only by 1, 2 or 4 by setting **CSC_AHB_DIV** register for access speed issue.

The following table is showing the memory source support table for DMA channel source and destination.

Table 12-2. DMA Memory Source Support

Memory Source Type	Memory as DMA Source		Memory as DMA Destination	
	MG32F02U128 MG32F02U064 MG32F02A128 MG32F02A064	MG32F02V032 MG32F02K128 MG32F02K064 MG32F02N128 MG32F02N064	MG32F02U128 MG32F02U064 MG32F02A128 MG32F02A064	MG32F02V032 MG32F02K128 MG32F02K064 MG32F02N128 MG32F02N064
SRAM	V	V	V	V
Flash	-	V	-	-
EMB	V	-	V	-

<Sign> "V" = not support , "-" = not support

● DMA Channel Source and Destination Selection

The DMA channel source and destination selection table are identical for all DMA channels. User can configure the DMA source by setting **DMA_CHn_SRC** register and DMA destination by setting **DMA_CHn_DET** register.

The following tables are showing the selection table and register setting for DMA channel source and destination.

The suffix of "RX" is meaning that the source data is coming from the module receiving interface. The suffix of "TX" is meaning that the destination data will be sent to the module transmission interface. Also the suffix of "Read" is meaning that the source data is coming by reading. The suffix of "Write" is meaning that the destination data will be used to write. The "ADC0_IN" is meaning that the source data is inputted from ADC conversion output.

The "DAC_OUT" is meaning that the destination data will be outputted to DAC conversion input. The TM36 can send the DMA source data from the timer input capture data and receive the DMA destination data to one of three timer output compare reload registers.

Table 12-3. DMA Channel Source Request Selection

Type	Channel Register Setting	Source Request Select		
	CHn_SRC	MG32F02U128 MG32F02U064 MG32F02A128 MG32F02A064	MG32F02V032	MG32F02K128 MG32F02K064 MG32F02N128 MG32F02N064
Memory (*1)	0	MEM_Read	MEM_Read	MEM_Read
Peripheral	1	ADC0_IN	ADC0_IN	ADC0_IN
	2	I2C0_RX	I2C0_RX	I2C0_RX
	3	I2C1_RX	I2C1_RX	I2C1_RX
	4	URT0_RX	URT0_RX	URT0_RX
	5	URT1_RX	URT1_RX	URT1_RX
	6	URT2_RX	-	URT2_RX
	7	-	-	-
	8	SPI0_RX	SPI0_RX	SPI0_RX
	9	-	-	-
	10	USB_RX	-	-
	11	-	-	-
	12	-	-	-
	13	-	-	-
	14	-	TM36_IC2	TM36_IC2
	15	TM36_IC3	TM36_IC3	TM36_IC3

<Sign> "-" = reserved , CHn: n = channel number, MEM: Memory source ~ Flash/SRAM/EMB

<Note> *1: Only Channel-0,3 support Memory-to-Memory operation

Table 12-4. DMA Channel Destination Request Selection

		Destination Request Select		
Type	Channel Register Setting	MG32F02U128 MG32F02U064 MG32F02A128 MG32F02A064	MG32F02V032	MG32F02K128 MG32F02K064 MG32F02N128 MG32F02N064
	CHn_DET			
Memory (*1)	0	MEM_Write	MEM_Write	MEM_Write
Peripheral	1	DAC_OUT	-	-
	2	I2C0_TX	I2C0_TX	I2C0_TX
	3	I2C1_TX	I2C1_TX	I2C1_TX
	4	URT0_TX	URT0_TX	URT0_TX
	5	URT1_TX	URT1_TX	URT1_TX
	6	URT2_TX	-	URT2_TX
	7	-	-	-
	8	SPI0_TX	SPI0_TX	SPI0_TX
	9	-	-	-
	10	USB_TX	-	-
	11	GPL_Write	GPL_Write	GPL_Write
	12	TM36_CC0B	TM36_CC0B	TM36_CC0B
	13	TM36_CC1B	TM36_CC1B	TM36_CC1B
	14	TM36_CC2B	TM36_CC2B	TM36_CC2B
	15	-	-	
	16	-	ASB0_TX	ASB0_TX
	17	-	ASB1_TX	ASB1_TX
	18	-	ASB2_TX	ASB2_TX
	19	-	ASB3_TX	ASB3_TX

<Sign> "-" = reserved , CHn: n = channel number, MEM: Memory destination ~ SRAM/EMB

<Note> *1: Only Channel-0,3 support Memory-to-Memory operation

● DMA Channel Operation Type Support

The DMA controller supports the channel operation types of memory-to-memory, peripheral-to-memory, memory-to-peripheral and peripheral-to-peripheral. User can configure the channel operation type by directly setting channel source and destination. The memory-to-memory DMA transferred operation is only supported for DMA channel-0, 3.

The following table is showing the supported DMA channel operation types for each channel.

Table 12-5. DMA Channel Operation Type Support

DMA Operation Type		Channel 0	Channel 1	Channel 2	Channel 3	Channel 4
Source	Destination					
Memory	Memory	V	-	-	V	-
Memory	Peripheral	V	V	V	V	V
Peripheral	Memory	V	V	V	V	V
Peripheral	Peripheral	V	V	V	V	V

<Sign> "V" = not support , "-" = not support

● DMA ADC Transfer

When the DMA source is ADC data output, user can configure the DMA transfer data packet size 16-bit or 32-bit by setting **ADCx_DMA_DSIZE** register. Please refer the section of "[ADC DMA Control](#)" for more information.

● DMA Flash to GPL

When the DMA source is the embedded flash and the DMA destination is GPL, user can configure the DMA transfer bus width 8-bit or 32-bit by setting **DMA_FGBUS_SEL** register. When it selects 1BYTE, the byte number is 1-byte for each transferred data cycle. When it selects 4BYTE, the byte number is 4-byte for each transferred data cycle. User can set 4BYTE only for flash-to-GPL DMA data transfer. It must set 1BYTE for other DMA data transfer conditions. Please refer the section of "[GPL DMA Control](#)" for more information.

● DMA Extra GPL Data Path

There is one extra GPL data path which can copy the transferred data from a configured channel and send to GPL to operate the GPL process (like as CRC). User can enable the extra GPL function and select the DMA

channel by setting **DMA_GPL_CHS** register. The choice channel processes the DMA operation which one request source transfers to another destination.

12.8.2. DMA Channel Arbitrator

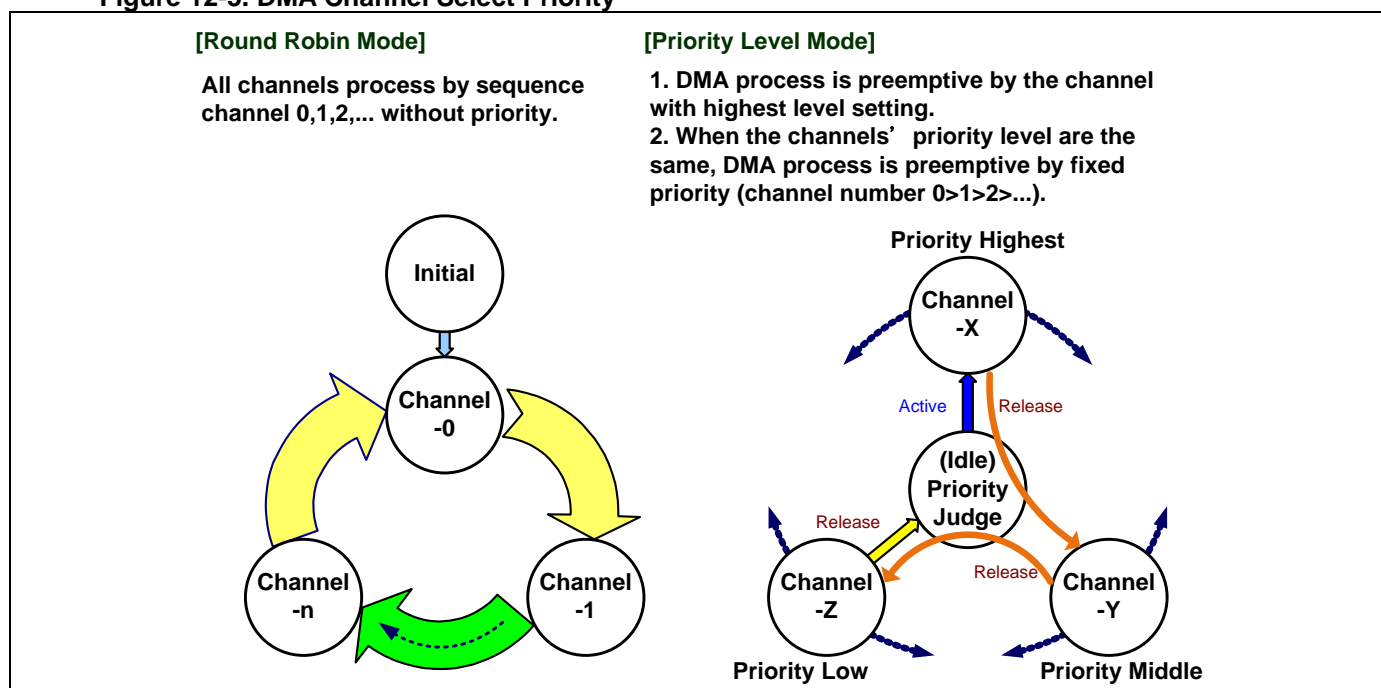
The DMA controller is built in a channel arbitrator. It provides two type of priority control between channel requests and user can set by the register of **DMA_PRI_MDS**. One is the “Round Robin” arbitration and another is using software configurable “Priority Level”.

When selects “Round Robin”, the first operation channel is channel-0. The next operation channel will be changed to channel-1 after channel-0 has finished one burst data transfer. Continuously, the channel will be changed to channel-2 and then channel-0 in loop.

When selects “Priority Level”, the act is very like as interrupt priority control and the priority level can be set in **DMA_CHn_PLS** register for each channel independently.

The following diagram is showing the two types of channel priority control.

Figure 12-5. DMA Channel Select Priority



12.8.3. DMA Channel Operation

• DMA Channel Enable

Each DMA channel has a channel enable bit of **DMA_CHn_EN** for operation control. When this bit is disabled, this DMA channel operation is not working and goes to channel reset state. There is a pseudo control bit **DMA_CHn_ENB** for each channel and it is identical to **DMA_CHn_EN** register bit. All the pseudo control bits of all channels are implemented in a same register of **DMA_CR0** for firmware easy using.

• DMA Channel Transferred Burst Size

Each DMA channel can be configured the transferred burst data size by setting independent **DMA_CHn_BSIZE** register. The burst data size is the sequential data byte number which cannot be interrupted by channel arbitration during transferred cycle.

The value of burst data size can be 1/2/4-byte that is configured by peripheral buffer size or EMB bus width. Refer the table of “DMA Transfer Number / Start Address and Burst Size Setting Note” for more information. For example, the burst size cannot set to one-byte if the EMB bus width is 16-bit.

• DMA Channel Transferred Total Byte Number

Each DMA channel can be configured the transferred total data byte number in every time of DMA data transferred process by setting independent **DMA_CHn_NUM** register. Value 0 is meaning that 65536 or 131072 data to be transferred and 0xFFFF is transferred 65535 or 131071 data. This register value must equal the integer multiples of **DMA_CHn_BSIZE** setting size. By chip supported functions, *the total byte number of DMA channel-0 and channel-3 can be supported up to 131072 if these channels are support embedded flash memory*

as *DMA source*. Refer the section of “[Chip Implement](#)” about the supported DMA channel functions.

The DMA controller provides the channel independent total counter value register of **DMA_CHn_CNT** which can be reading to indicate the remaining bytes to be transmitted for user. This register is decreased by the burst size number in **DMA_CHn_BSIZE** register after each DMA burst transfer.

● DMA Channel Source and Destination Start Address

When the channel operation type is memory-to-memory, peripheral-to-memory, memory-to-peripheral, each DMA channel can be configured the source or/and destination memory start address in every time of DMA data transferred process by setting independent **DMA_CHn_SSA** and **DMA_CHn_DSA** registers. When the DMA source or destination is peripheral, the related memory start address register of source or destination is no effect.

Refer the table of “DMA Transfer Number/Start Address and Burst Size Setting Note”, the source and destination memory start address must be Word alignment if the burst size is four-byte. The same, they must be Half-Word alignment if the burst size is two-byte.

The DMA controller provides the channel independent source and destination memory address registers of **DMA_CHn_SCA** and **DMA_CHn_DCA** which can be reading to indicate the memory transfer current address for user. The address operation range is limited in a 64K aligned address space. When the address is operating over the 64K boundary, the address is rolling up to 0x0000 of the 64K aligned address space. By chip supported functions, the address operation range of DMA channel-0 and channel-3 can be supported up to 128K if these channels are support embedded flash memory as DMA source. Refer the section of “[Chip Implement](#)” about the supported DMA channel functions.

The transferred address of source and destination can be automatically increased or not change after each burst transfer complete by setting **DMA_CHn_SINC** and **DMA_CHn_DINC** for each channel independently.

When enables the address auto increment, user can select the increased mode of “Normal” or “SKIP3” mode by setting **DMA_CHn_ADSEL** register. When selects “Normal”, the address will be sequential increment 1. When selects “SKIP3”, the Lsb two-bit word address is increased from 0-to-1, 1-to-2 and 2-to-0 but skip address 3.

Table 12-6. DMA Transfer Number/Start Address and Burst Size Setting Note

Burst Size (Byte)	Transfer Number	Memory Start Address		Peripheral	EMB
	CHn_NUM	CHn_SSA	CHn_DSA	Buffer Size (Byte)	Bus Width (Bit)
CHn_BSIZE	Integer Multiples of	Address alignment	Address alignment	(suggestion)	(suggestion)
One	1	Byte	Byte	1,2,3,4	8
Two	2	Half-Word	Half-Word	2,4	8,16
Four	4	Word	Word	4	8,16

<Sign> CHn : n = channel number

● DMA Channel Synchronous Operation

Each DMA channel can be enabled the synchronous operation of data transfer for DMA source-to-DMA controller and DMA controller-to-DMA destination by setting independent **DMA_CHn_SSYNC** and **DMA_CHn_DSYNC** registers. When the source process clock frequency equals to DMA process clock frequency, suggests enabling this bit to improve DMA performance. Also it is the same for destination.

● DMA Channel Loop Mode

Each DMA channel can be enabled the DMA data transaction loop mode by setting independent **DMA_CHn_LOOP** register. When enables, the number of transaction data is automatically reloaded with the initial value after previous DMA data transaction is completed and the DMA requests will be continuous. In the condition, the DMA data counter in **DMA_CHn_CNT** register will be reloaded automatically by **DMA_CHn_NUM** register. When disables, there is no DMA request to be served after the last DMA data transaction.

For firmware flow control, user can set the last loop command in the DMA TCF flag interrupt service routine at the previous TCF active before last loop DMA transaction by setting channel independent **DMA_CHn_LAST** register. Then the DMA controller will disable the loop function and stop DMA transaction after last data transferred is complete and TCF flag is asserted. User needs clear this register bit of **DMA_CHn_LAST** for next time using.

The DMA loop mode is available for a sequential data transaction and loop data transferred operation which is like as ADC channel scan operation. For the ADC channel scan operation, the DMA can transfer the ADC conversion data to SRAM from channel-0 to channel-15. Continuously, the channel-0 ADC data of next loop will write to the same address as previous channel-0 data writing. And it is the same for channel-1, 2, 3 ... 15.

12.8.4. DMA SRAM Using

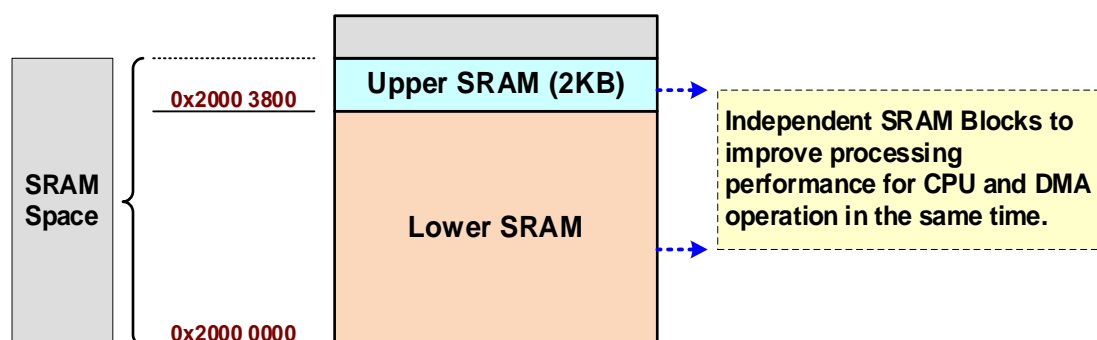
DMA can access any embedded SRAM memory space which is including of all of the Main SRAM, DMA SRAM and USB SRAM. The SRAM memory is located at the base address of 0x20000000. The USB SRAM is addressed start at 0x3000 0000 and is only implemented for **MG32F02U** series.

When both the CPU and the DMA controller access the same memory address of internal embedded SRAM in a same time, the CPU is higher priority than the DMA controller for SRAM memory access. When the CPU gets the SRAM memory access control, the DMA controller will hold until the CPU releases the SRAM memory access control. And the DMA controller can go on to access SRAM for DMA data transaction until next time interrupt by CPU access command.

In order to improve the performance of DMA data transaction, the SRAM memory space is designed to separate to two block – upper 2K-bytes and lower SRAM. Strongly suggests user to arrange the upper 2K-bytes SRAM for DMA transaction and lower SRAM for software using. The upper 2K-bytes SRAM is located at the fixed base address of 0x20003800.

Certainly, user can plan any memory space of SRAM for software using or DMA transaction and no hardware limit. For example, one MCU with total 16K-bytes SRAM can be planned all bytes of SRAM for application firmware using or upper 2K-bytes SRAM as DMA buffer and lower 14K-bytes SRAM as application firmware using.

Figure 12-6. DMA SRAM Using Suggestion



[Upper 2KB]

1. Strongly suggest to use upper 2KB SRAM as DMA transfer 1st data buffer.
2. Use for firmware parameter and data buffer if does not request DMA.

[Lower SRAM]

1. Suggest to use lower SRAM for firmware parameter and data buffer.
 2. If it is necessary that use partial of lower SRAM as DMA transfer 2nd data buffer.
- Additionally the DMA transfer performance is reduced by CPU operation for this SRAM.

12.9. DMA Transaction

12.9.1. DMA Transaction Configuration and Sequence

- **Prepare DMA Transaction**

Prior to starting the DMA transaction, the user should:

- Configure the processing Peripheral module if the channel operation type is peripheral-to-memory, memory-to-peripheral or peripheral-to-peripheral.
- Enable the DMA process clock by setting **CSC_DMA_EN** register.
- Turn on the DMA hardware by setting the **DMA_EN** bit.
- Select the channel priority mode “Round Robin” or “Priority Level” if the chip is supported in **DMA_PRI_MDS** register

If the DMA source or destination is peripheral or EMB module, the peripheral module must be configuration finished. Also the DMA controller and the operation channel must configure those registers are described in “DMA Control” section. And user needs to set the module’s DMA enable bit(s) (Like as **XXX_DMA_RXEN**, **XXX_DMA_TXEN** or **XXX_DMA_EN**). (‘XXX’ = peripheral module name)

- **DMA Channel Selection**

Refer the section of “[DMA Source and Destination](#)” for more information about the DMA channel.

- Enable the DMA transaction channel by setting the **DMA_CHn_EN** bit.
- Set the DMA transfer complete interrupt enable in the **DMA_CHn_CIE** and **DMA_IEA** bits.
- Select the DMA channel source and destination in **DMA_CHn_SRC** and **DMA_CHn_DET** registers.
- Configure the DMA transfer burst size by setting **DMA_CHn_BSIZE** register.
- Configure the DMA transfer data count initial number by setting **DMA_CHn_NUM** bit.
- Configure the DMA source and/or destination start address by setting **DMA_CHn_SSA** and **DMA_CHn_DSA** bits.
- Enable DMA Channel loop mode if request it by setting **DMA_CHn_LOOP** bit

- **DMA Transaction Start**

When the DMA controller and the processing peripheral are configuration complete, user needs to set the related channel request start bit of **DMA_CHn_REQ** is necessary to be set to start the DMA transaction. This **DMA_CHn_REQ** bit is auto clear by hardware after DMA data transaction is complete.

After a data transfer event, the transferred source/destination device will assert the RX/TX request signal to DMA controller. By channel arbitrator control, the DMA controller will service the request depending on the channel arbitration priority and assert the acknowledge signal to the request source/destination device. At the time, the data transferred connection is built for DMA transaction. The request source/destination device will release the request when it gets the acknowledge signal from the DMA controller and transfers the transaction data. When all transaction data are transferred complete by repeat the same process, the DMA transfer data counter in **DMA_CHn_CNT** is equal to 0 and the DMA controller will be active the TCF flag.

12.9.2. DMA Transaction under SLEEP

Normally, the DMA transaction is configured and started processing on the **ON** mode. The DMA controller can support the DMA transaction to be continuously processing during **SLEEP** mode for the channel operation types of memory-to-memory if **CSC_DMA_EN** register is enabled beforehand chip entering **SLEEP** mode.

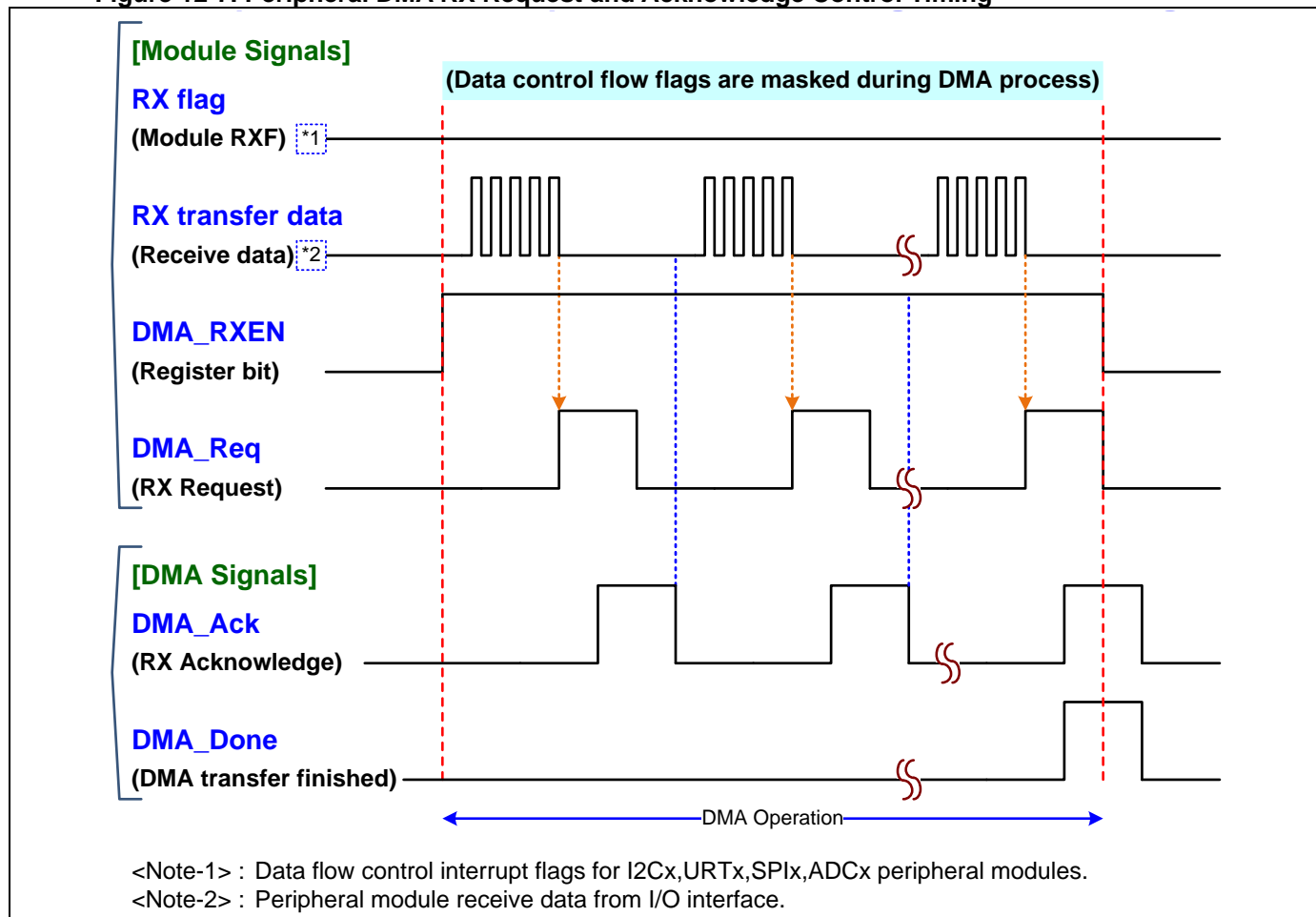
For the channel operation types of peripheral-to-memory, memory-to-peripheral and peripheral-to-peripheral, also user can plan the DMA transaction to be processing during **SLEEP** mode if the module clock of the peripheral has planned running on **SLEEP** mode by setting the **SLEEP** mode clock enable **CSC_SLP_xxx** register beforehand chip entering **SLEEP** mode. When the chip is entering **SLEEP** mode, the DMA transaction can be kept going to transfer the data until the DMA transaction complete.

12.9.3. Peripheral DMA RX Request and Acknowledge

After the DMA controller and the peripheral module are configuration complete for DMA data transaction. When the peripheral module receives data from external device by through the communication interface, the peripheral module will assert the DMA request to DMA controller. By DMA controller operation, it will send the DMA acknowledge to the peripheral module to notify the DMA request is receipted. And the DMA request will be released by the peripheral module. Then the DMA data transaction is starting from peripheral module to destination by through DMA controller. Repeat the processes until all data are transferred complete. At last, the DMA controller sends a DMA done signal to the peripheral module to end the DMA data transaction.

During the DMA transaction cycle, the received data flag of **XXX_RXF** of DMA source device is masked by hardware.

Figure 12-7. Peripheral DMA RX Request and Acknowledge Control Timing

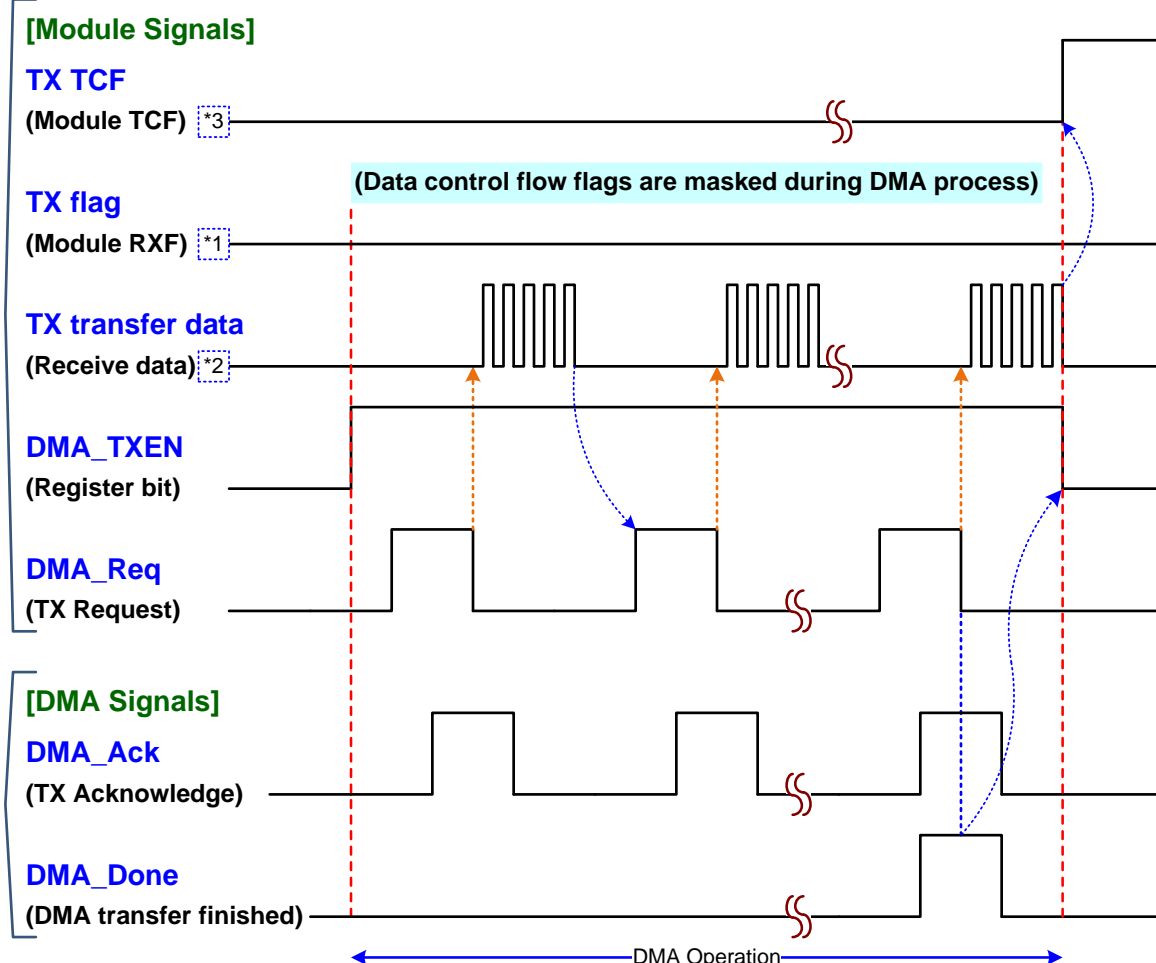


12.9.4. Peripheral DMA TX Request and Acknowledge

As same as “Peripheral DMA RX Request and Acknowledge”, the DMA data transaction is processing the same for peripheral module TX operation. When the DMA controller transfers the last data to the peripheral module, the DMA controller sends a DMA done signal to the peripheral module to end the DMA data transaction. And the peripheral module will transmit the last data to external device by through the communication interface.

During the DMA transaction cycle, the transmitted data flag of **XXX_TXF** of DMA destination device is masked by hardware. In general, the transmitted complete flag of **XXX_TCF** will be asserted after the finished of DMA transaction.

Figure 12-8. Peripheral DMA TX Request and Acknowledge Control Timing



<Note-1> : Data flow control interrupt flags for I2Cx, URTx, SPIx, DAC peripheral modules.

<Note-2> : Peripheral module transmit data to I/O interface.

<Note-3> : Transfer complete flag for URTx, SPIx peripheral modules.

12.9.5. Peripheral DMA Transaction Hold

For DMA data flow control or some unexpected conditions, the firmware can directly force to hold the DMA data transaction by setting channel independent register of **DMA_ChN_HOLD**.

When one channel hold is enabled, the DMA controller will transfer completely for current burst transaction data if the related DMA channel is data transfer processing now. Then this DMA channel data transfer operation is holding and the channel operation is changing to next channel by arbitration priority control. This DMA channel keeps holding for DMA data transaction until this control bit of **DMA_ChN_HOLD** is disabled by firmware.

12.9.6. Peripheral Interrupt Flag Control

There are three types control for the peripheral module interrupt flags during the DMA process.

- **Mask the Flag during DMA Process**
 - Usually for data flow control interrupt flags
- **DMA Disable after the Flag asserted**
 - Usually for error detection interrupt flags
- **Normal Control**
 - the flags are independent of DMA process

❖ MG32F02A128/U128/A064/U064

The following table is showing the peripheral module DMA interrupt flag control for MG32F02A128/U128/A064/U064.

Table 12-7. Peripheral Module Interrupt Flag Control for DMA – MG32F02A128/U128/A064/U064

Action	Mask the Flag during DMA Process	DMA Disable after the Flag asserted (*1)	Normal Control		
Peripheral	(Data flow flags)	(Error/Detect flags)	(Other flags)		
I2Cx	I2Cx_EVENTF I2Cx_BUFF	I2Cx_ERRF(*1) I2Cx_STPSTRF(*1)	I2Cx_TMOUTF I2Cx_WUPF	I2Cx_RXF I2Cx_TXF I2Cx_RSTRF I2Cx_STOPF I2Cx_CNTF I2Cx_ERRCF I2Cx_SADRF	I2Cx_TSCF I2Cx_ROVRF I2Cx_TOVRF I2Cx_NACKF I2Cx_ALOSF I2Cx_BERRF
URTx	URTx_TCF (*2) URTx_RXF URTx_TXF (*2)	URTx_ERRF URTx_BKF(*3) URTx_IDLF(*4) URTx_CTSF(*5)	URTx_UGF URTx_LSF	URTx_SADRF URTx_BRTF URTx_TMOF URTx_CALCF URTx_PEF URTx_FEF URTx_NCEF	URTx_ROVRF URTx_TXEF URTx_RXTMOF URTx_IDTMOF URTx_BKTMOF URTx_CALTMOF
SPIx	SPIx_TCF (*2) SPIx_RXF SPIx_TXF (*2)	SPIx_MODF SPIx_WEF SPIx_ROVRF SPIx_TUDRF	SPIx_IDLF		
ADCx	ADCx_ESMPF ADCx_E1CNVF(*6) ADCx_ESCNVF(*2)	ADCx_OVRF ADCx_WDLF ADCx_WDIF ADCx_WDHF	ADCx_SUMOF ADCx_SUMCF ADCx_SUMOVRF		
DAC	DAC_RDY0F (*2)	DAC_UDR0F			
TM36	TM36_CF0A/0B TM36_CF1A/1B TM36_CF2A/2B TM36_CF3A/3B	TM36_TOF TM36_TUF	TM36_EXF TM36_TOF2 TM36_TUF2 TM36_DIRCF	TM36_IDXF TM36_QPEF TM36_BKF	
EMB		EMB_WPEF EMB_BWEF EMB_IAEF			
USB	USB_RXDnF USB_TXDnF	USB_ERRF USB_BUSF USB_LPMF	USB_SOF USB_ESOF USB_EPnF USB_OVRF USB_SETUPF USB_NORSF USB_BTSTF USB_CRCF	USB_SUSF USB_RSMF USB_RSTF USB_RWKF USB_BSUSF USB_SE1F USB_LPMSTF USB_LPMNYF	USB_STOVWnF USB_EDOVWnF USB_SETUPnF USB_RXNAKnF USB_RXSTLnF USB_ISOOWnF USB_TXNAKnF USB_TXSTLnF USB_ISOTXEnF

<Note> *1: When the flag is asserted, it will not force peripheral DMA disabled if the related interrupt enable bit is not enabled. Specially, the flag asserted that will force peripheral DMA disabled for I2Cx_ERRF or I2Cx_STPSTRF however the related interrupt is enabled or not.

***2:** This flag will be asserted after DMA TX complete.

***3:** When the flag is asserted, it will force URTx DMA RX disabled if the related interrupt enable bit is enabled. At the same time, the URTx DMA TX is disabled or not by URTx_DDTX_EN setting.

***4:** When the flag is asserted, it will force URTx DMA RX disabled if the related interrupt enable bit is enabled. But URTx DMA TX will be not.

- *5: When the flag is asserted, it will force URTx DMA TX disabled if the related interrupt enable bit is enabled. But URTx DMA RX will be not. Generally user sets CTS hardware auto control mode (URTx_CTS_EN=1) and will not enable the URTx_CTS_IE, URTx will hold the transmission and not disable URTx DMA TX when detects the CTS active.
- *6: When ADCx_DMA_MDS is disabled, the ADCx_E1CNVF flag is masked during DMA process. When ADCx_DMA_MDS is "Keep", the ADCx_E1CNVF flag is normal and does not mask during DMA process.

❖ MG32F02V032

The following table is showing the peripheral module DMA interrupt flag control for MG32F02V032.

Table 12-8. Peripheral Module Interrupt Flag Control for DMA – MG32F02V032

Action	Mask the Flag during DMA Process	DMA Disable after the Flag asserted (*1)	Normal Control	
Peripheral	(Data flow flags)	(Error/Detect flags)	(Other flags)	
I2Cx	I2Cx_EVENTF I2Cx_BUFF	I2Cx_ERRF(*1) I2Cx_STPSTRF(*1)	I2Cx_TMOUTF I2Cx_WUPF	I2Cx_RXF I2Cx_TXF I2Cx_RSTRF I2Cx_STOPF I2Cx_CNTRF I2Cx_ERRCF I2Cx_SADRF I2Cx_TSCF I2Cx_ROVRF I2Cx_TOVRF I2Cx_NACKF I2Cx_ALOSF I2Cx_BERRF
URTx	URTx_TCF (*2) URTx_RXF URTx_TXF (*2)	URTx_ERRF URTx_BKF(*3) URTx_IDLF(*4) URTx_CTSF(*5)	URTx_UGF URTx_LSF	URTx_SADRF URTx_BRTF URTx_TMOF URTx_CALCF URTx_PEF URTx_FEF URTx_NCEF URTx_ROVRF URTx_TXEF URTx_RXTMOF URTx_IDTMOF URTx_BKTMOF URTx_CALTMOF
SPIx	SPIx_TCF (*2) SPIx_RXF SPIx_TXF	SPIx_MODF SPIx_WEF SPIx_ROVRF SPIx_TUDRF	SPIx_IDLF	
ADCx	ADCx_ESMPF ADCx_E1CNVF(*6) ADCx_ESCNVF(*2)	ADCx_OVRF ADCx_WDLF ADCx_WDIF ADCx_WDHF	ADCx_SUMOF ADCx_SUMCF ADCx_SUMOVRF	
TM36	TM36_CF0A/0B TM36_CF1A/1B TM36_CF2A/2B TM36_CF3A/3B	TM36_TOF TM36_TUF	TM36_EXF TM36_TOF2 TM36_TUF2 TM36_DIRCF	TM36_IDXF TM36_QPEF TM36_BKF
APX	APX_ASBN_TCF (*2) APX_ASBN_TXF (*2) (n=0,1,2,3)		APX_CCLnF (n=0,1)	

<Note> *1: When the flag is asserted, it will not force peripheral DMA disabled if the related interrupt enable bit is not enabled. Specially, the flag asserted that will force peripheral DMA disabled for I2Cx_ERRF or I2Cx_STPSTRF however the related interrupt is enabled or not.

*2: This flag will be asserted after DMA TX complete.

*3: When the flag is asserted, it will force URTx DMA RX disabled if the related interrupt enable bit is enabled. At the same time, the URTx DMA TX is disabled or not by URTx_DDTX_EN setting.

*4: When the flag is asserted, it will force URTx DMA RX disabled if the related interrupt enable bit is enabled. But URTx DMA TX will be not.

*5: When the flag is asserted, it will force URTx DMA TX disabled if the related interrupt enable bit is enabled. But URTx DMA RX will be not. Generally user sets CTS hardware auto control mode (URTx_CTS_EN=1) and will not enable the URTx_CTS_IE, URTx will hold the transmission and not disable URTx DMA TX when detects the CTS active.

*6: When ADCx_DMA_MDS is disabled, the ADCx_E1CNVF flag is masked during DMA process. When ADCx_DMA_MDS is "Keep", the ADCx_E1CNVF flag is normal and does not mask during DMA process.

❖ MG32F02N128/K128/N064/K064

The following table is showing the peripheral module DMA interrupt flag control for MG32F02N128/K128/N064/K064.

Table 12-9. Peripheral Module Interrupt Flag Control for DMA –MG32F02N128/K128/N064/K064

Action	Mask the Flag during DMA Process	DMA Disable after the Flag asserted (*1)	Normal Control	
Peripheral	(Data flow flags)	(Error/Detect flags)	(Other flags)	
I2Cx	I2Cx_EVENTF I2Cx_BUFF	I2Cx_ERRF(*1) I2Cx_STPSTRF(*1)	I2Cx_TMOUTF I2Cx_WUPF	I2Cx_RXF I2Cx_TXF I2Cx_RSTRF I2Cx_STOPF I2Cx_CNTF I2Cx_ERRCF I2Cx_SADRF I2Cx_TSCF I2Cx_ROVRF I2Cx_TOVRF I2Cx_NACKF I2Cx_ALOSF I2Cx_BERRF
URTx	URTx_TCF (*2) URTx_RXF URTx_TXF (*2)	URTx_ERRF URTx_BKF(*3) URTx_IDLF(*4) URTx_CTSF(*5)	URTx_UGF URTx_LSF	URTx_SADRF URTx_BRTF URTx_TMOF URTx_CALCF URTx_PEF URTx_FEF URTx_NCEF URTx_ROVRF URTx_TXEF URTx_RXTMOF URTx_IDTMOF URTx_BKTMOF URTx_CALTMOF
SPIx	SPIx_TCF (*2) SPIx_RXF SPIx_TXF (*2)	SPIx_MODF SPIx_WEF SPIx_ROVRF SPIx_TUDRF	SPIx_IDLF	
ADCx	ADCx_ESMPF ADCx_E1CNVF(*6) ADCx_ESCNVF(*2)	ADCx_OVRF ADCx_WDLF ADCx_WDIF ADCx_WDHF	ADCx_SUMOF ADCx_SUMCF ADCx_SUMOVRF	
TM36	TM36_CF0A/0B TM36_CF1A/1B TM36_CF2A/2B TM36_CF3A/3B	TM36_TOF TM36_TUF	TM36_EXF TM36_TOF2 TM36_TUF2 TM36_DIRCF	TM36_IDXF TM36_QPEF TM36_BKF
APX	APX_ASBN_TCF (*2) APX_ASBN_TXF (*2) (n=0,1,2,3)		APX_CCLnF (n=0,1)	APX_SDTFn (n=4,5)

<Note> *1: When the flag is asserted, it will not force peripheral DMA disabled if the related interrupt enable bit is not enabled. Specially, the flag asserted that will force peripheral DMA disabled for I2Cx_ERRF or I2Cx_STPSTRF however the related interrupt is enabled or not.

*2: This flag will be asserted after DMA TX complete.

*3: When the flag is asserted, it will force URTx DMA RX disabled if the related interrupt enable bit is enabled. At the same time, the URTx DMA TX is disabled or not by URTx_DDTX_EN setting.

*4: When the flag is asserted, it will force URTx DMA RX disabled if the related interrupt enable bit is enabled. But URTx DMA TX will be not.

*5: When the flag is asserted, it will force URTx DMA TX disabled if the related interrupt enable bit is enabled. But URTx DMA RX will be not. Generally user sets CTS hardware auto control mode (URTx_CTS_EN=1) and will not enable the URTx_CTS_IE, URTx will hold the transmission and not disable URTx DMA TX when detects the CTS active.

*6: When ADCx_DMA_MDS is disabled, the ADCx_E1CNVF flag is masked during DMA process. When ADCx_DMA_MDS is "Keep", the ADCx_E1CNVF flag is normal and does not mask during DMA process.

12.10. DMA External Request Trigger Input

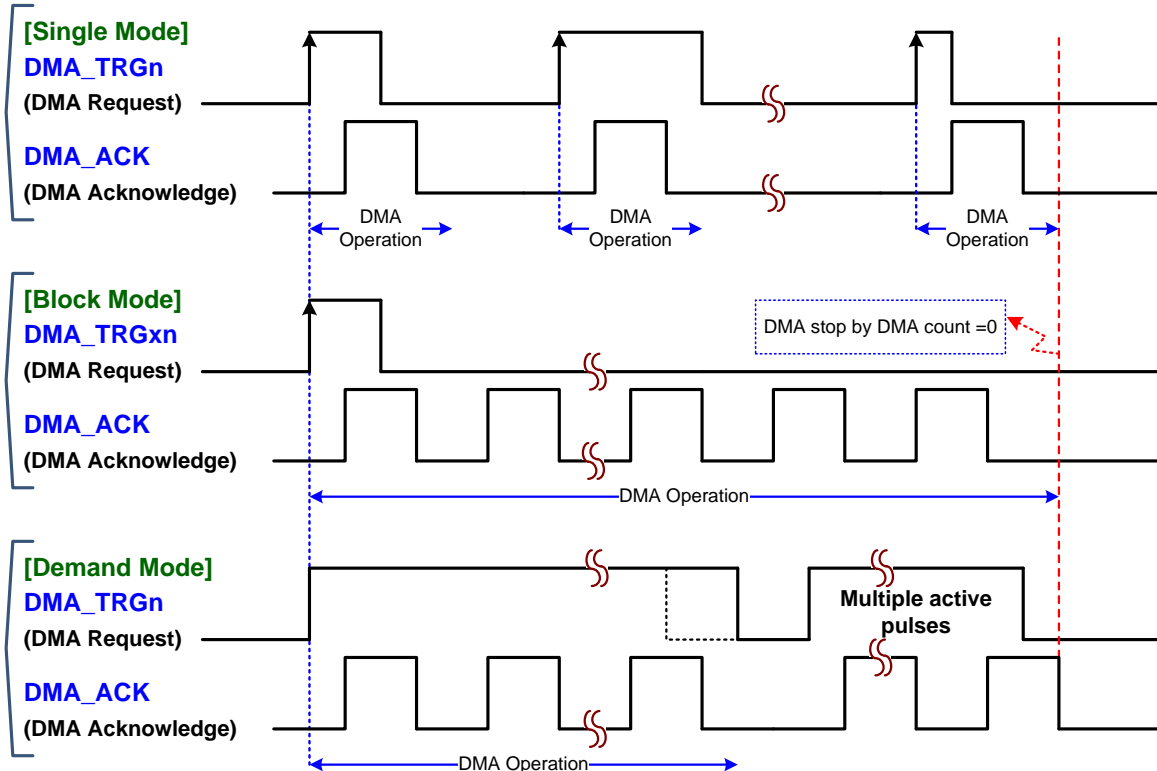
The DMA supports two external pins of **DMA_TRG0** and **DMA_TRG1** which can input as the DMA request trigger signal for DMA data transaction. There are three types of control mode for external request trigger input. User can enable and select Single mode, Block mode or Demand mode by setting **DMA_CH0_XMDS** register for each channel independently.

Notify: The Demand mode is not supported peripheral-to-peripheral DMA transfer.

When enables the DMA external trigger mode, each DMA channel can be configured the trigger signal input from **DMA_TRG0** or **DMA_TRG1** pin by setting independent **DMA_CHn_XPIN** register.

As following diagram, it is showing the control timing of Single mode, Block mode and Demand mode.

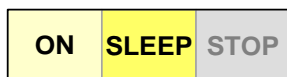
Figure 12-9. DMA External Request Trigger Input Timing



<Note> 1: DMA_TRGn ~ DMA request signal from pin
2: n= DMA request trigger pin number

13. EMB (External Memory Bus)

13.1. Introduction



The module can be running in ON and SLEEP modes only.

The chip has built in an external memory bus (EMB) controller to access the external devices of SRAM, NOR/NAND-flash and 8080 interface LCD. The EMB controller supports address bus and data bus multiplex mode. Also, it provides two address latch enable signals to support the multiple control of the address and data cycle.

13.2. Features

- Support SRAM, NOR/NAND-flash, LCD interface
- Support synchronous or asynchronous timing mode control
- Support 8/16-bit data width
- Support multiple types of address and data multiplex mode
- Provide optional 16/24/30-bit address mode
 - Memory space 2G/32M/128K-byte for 16-bit data width
 - Memory space 16M/64K-byte for 8-bit data width
- Support byte write in 16-bit data width mode
- Configurable time cycle for address latch time and data access time
- Received and transmitted data are buffered with DMA capability
 - External memory space acts as same as embedded memory for DMA
- Allow running CPU code on external SRAM

13.3. Implementation

13.3.1. Chip Implementation

The following table is showing the implementation of EMB module.

Table 13-1. EMB Implementation

Chip	Package	EMB Bus	EMB Interface Modes					
		Bus Width	16bit-MA 16bit-MD	16bit-MA 16bit-MAD	16bit-MAD	16bit-MA 8bit-MD	16bit-MA 8bit-MAD	8bit-MAD
MG32F02A128/U128	LQFP80/64	8/16-bit	V	V	V	V	V	V
MG32F02A064/U064	LQFP64	8/16-bit	V	V	V	V	V	V
MG32F02A064/U064	LQFP48	Not Implemented						
MG32F02V032 MG32F02N128/N064 MG32F02K128/K064	All	Not Implemented						

<Note>

V: Implemented

MA: dedicated Memory Address , MD: dedicated Memory Data ,

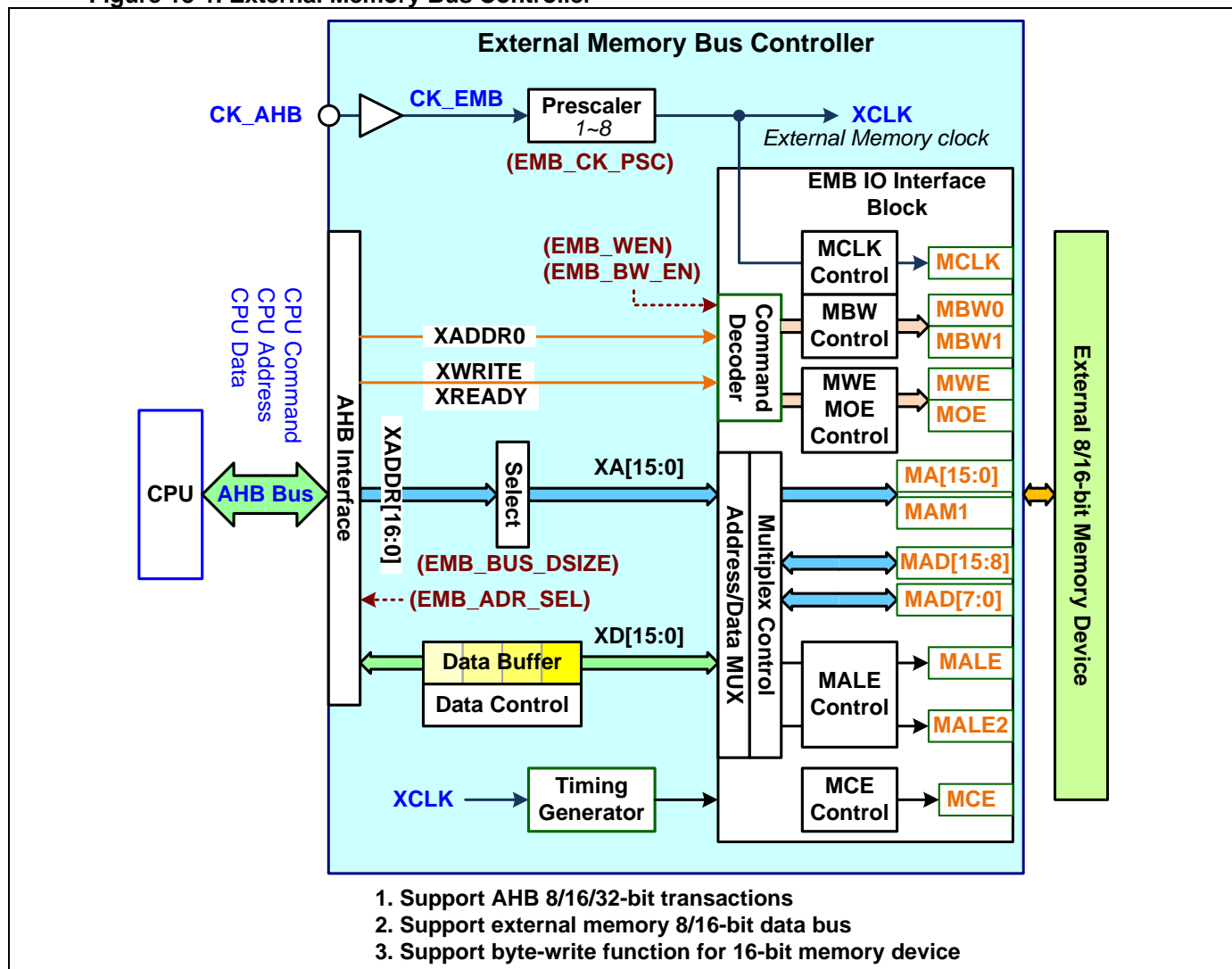
MAD: Memory Address and Data multiplexed

13.4. Control Block

The EMB controller is including of the AHB interface logic and the external memory bus IO interface block, 16-bit data buffer and one bus timing generator. The EMB IO interface block is used to control the address, data and command signals for external devices. The bus timing generator inputs the MCLK clock and output the programmable time sequence for EMB IO interface block.

The following diagram is showing the EMB controller block.

Figure 13-1. External Memory Bus Controller



13.5. IO Lines

13.5.1. IO Signals

- **MCLK**

It is the external memory bus clock signal and uses as output signal for external synchronous SRAM, NOR flash or other memory devices.

- **MCE**

It is the chips enable or select signal and uses as output for the external memory bus.

- **MOE**

It is the output enable (OE) or read strobe (RD) signal and uses as output for the external memory bus.

- **MWE**

It is the write enable (WE) or write strobe (WR) signal and uses as output for the external memory bus.

- **MALE**

It is the address latch enable (ALE) or data/command select (DC) signal and uses as output for the external memory bus.

- **MALE2**

It is the 2nd address latch enable (ALE2) or command latch enable (CLE) signal and uses as output for the external memory bus.

- **MBW0**

It is the byte write enable 0 or the least significant bit of the address signal and uses as output for the external memory bus.

- **MBW1**

It is the byte write enable 1 signal and uses as output for the external memory bus.

- **MAM1**

It is the least significant bit of the address signal and uses as output for the 8-bit external memory bus.

- **MA[0..15]**

They are the memory address output signals for the external memory bus.

- **MAD[0..15]**

They are the data bidirectional signals for the external memory bus. Also they can use as the multiplex signals of address and data those can use MALE/MALE2 control signal to distinguish the address and data.

13.5.2. IO Configure

User must configure the related IO pins in order to use the IO lines of this module. The IO operation modes, output high speed option, pull-high option, output drive strength, IO deglitch filter and input inverse selection are programmable by pin independent. Please refer the section of “[IO Mode](#)” in GPIO chapter for more detail descriptions of IO mode configuration.

Each IO signal may be mapped and selected on several IO pins by configuring the IO AFS matrix. Please refer the section of “[Alternate Function Select](#)” in GPIO chapter for more detail descriptions of IO AFS configuration. About the actual IO pin AFS information, please refer the section of “[Pin Alternate Functions Selected Table](#)” in Pin Description chapter of the chip Data Sheet.

13.6. Enabling and Clock

13.6.1. EMB Global Enable

There is a global enable bit of **EMB_EN** for all functions of this module. When this bit is disabled, all the EMB functions are not working.

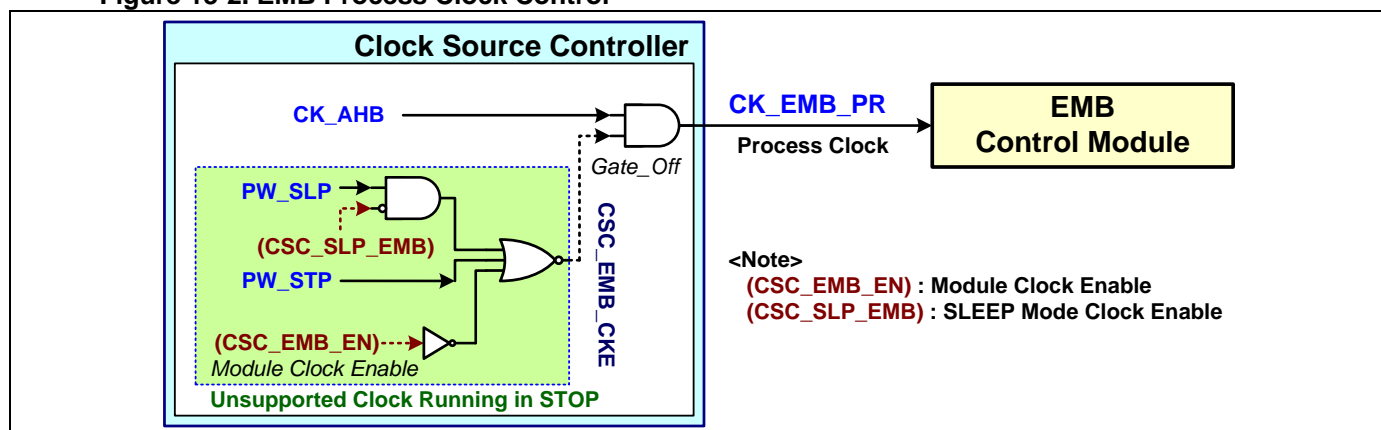
13.6.2. EMB Clock Control

- **Module Process Clock**

The module process clock of **CK_EMB_PR** is using for the interface control logic between AHB bus and the module. It is coming from CSC (Clock Source Controller) module. It can be enabled in **CSC_EMB_EN** register.

In **ON** mode, the process clock is running only if **CSC_EMB_EN** register is enabled. User can plan the module clock is running or not beforehand for chip entering **SLEEP** mode by setting **CSC_SLP_EMB** register. In **STOP** mode, the process clock is always stopping. Refer the System Clock chapter for more information.

Figure 13-2. EMB Process Clock Control



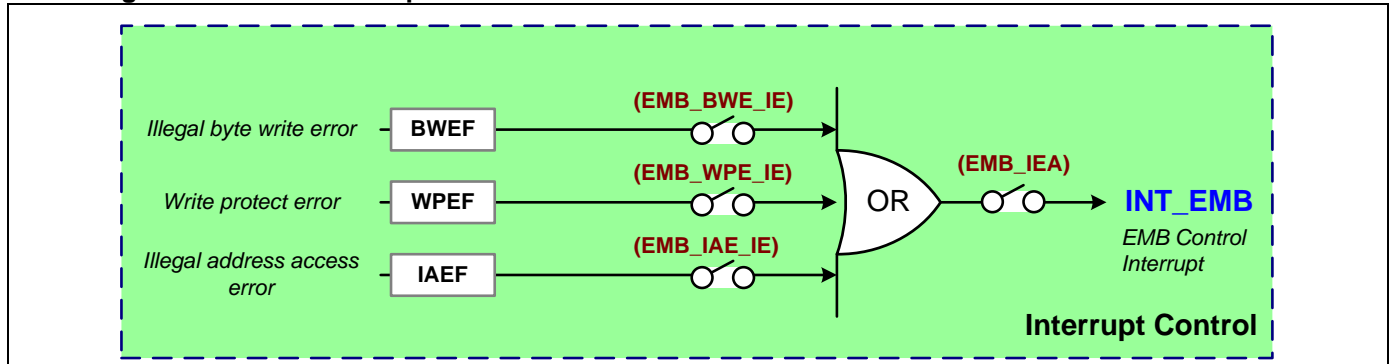
- **Module Internal Clock**

The EMB internal clock is coming from AHB clock of **CK_AHB**. The clock frequency can be divided by 1~8 through a clock prescaler to do as the external memory bus output clock **MCLK**. The **MCLK** clock is also used as the time base clock for EMB timing generator. User can set the clock prescaler by setting **EMB_CK_PSC** register.

13.7. Interrupt and Event

There is one signal of **INT_EMB** to be generated in this EMB control module. **INT_EMB** sends to External Interrupt Controller (EXIC) to do as an interrupt event.

Figure 13-3. EMB Interrupt Control



13.7.1. EMB Interrupt Control and Status

These interrupt flags are using for the interrupt service routine (ISR) flow control. Generally these flags are set by hardware and software clears them when the serviced job is completed in the related ISR. Each interrupt flag has one interrupt enable bit. User can enable or disable it. There is one interrupt all enable bit of **EMB_IEA** to enable or disable all the interrupt sources for this module.

One busy flag of **EMB_BUSYF** is used to indicate the data transfer busy status. Refer the register descriptions of the status bit for more information.

13.7.2. EMB Interrupt Flags

Generally these interrupt flags are set by hardware and clear by software writing 1. Refer the register descriptions for more information about the related interrupt flags and interrupt enable bit.

- **WPEF**

EMB bus write-protect error detect flag (**EMB_WPEF**). There is a related interrupt enable register bit of **EMB_WPE_IE**. When EMB write access is disabled (**EMB_WEN**=0) and operates an EMB write process, this flag is asserted.

- **BWEF**

EMB bus byte-write error flag (**EMB_BWEF**). There is a related interrupt enable register bit of **EMB_BWE_IE**. When user configure 16-bit EMB data bus (**EMB_BUS_DSIZE**=16-bit) and EMB write access is enabled (**EMB_WEN**=1) but byte-write access is disabled (**EMB_BW_EN**=0), this flag is asserted if operates an EMB byte-write process.

- **IAEF**

EMB bus access illegal address error detection flag (**EMB_IAEF**). There is a related interrupt enable register bit of **EMB_IAE_IE**. The flag is asserted when CPU access out of the **EMB_ADR_SEL** setting address in the EMB 2GB memory space.

13.8. EMB IO Control

The EMB provides 8-bit/16-bit data signals – **MAD[15:0]**, maximum 16-line address signals – **MA[15:0]**, chip select signal – **MCE**, data output enable signal – **MOE**, data write enable signal – **MWE**, two address latch enable signals – **MALE/MALE2**, byte write enable signals – **MBW0/MBW1** and output clock signal – **MCLK**. The **MAD[15:0]** signals are able to output address or data signals for address and data multiplex mode.

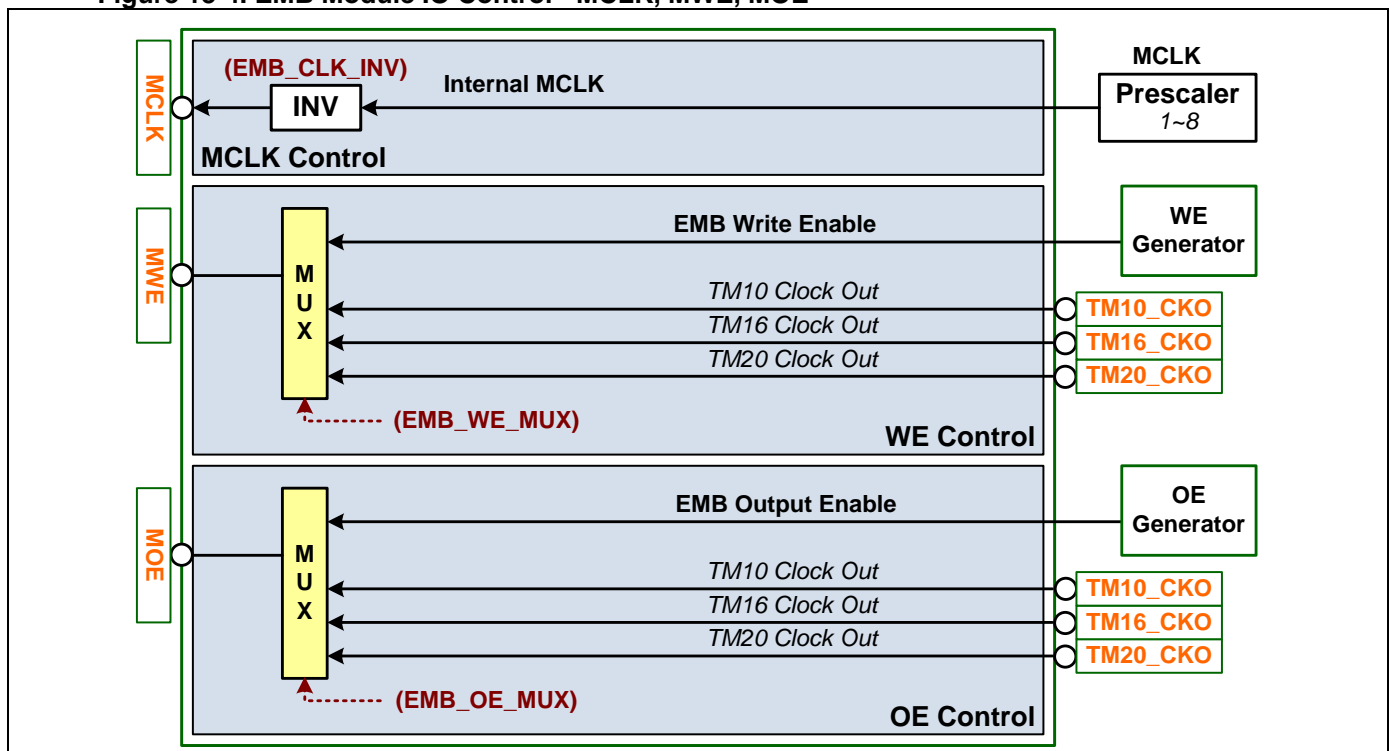
The EMB supports IO signal inversion, swapping and source selection for user easy design. The EMB interface IO can force to software control mode for **MCE**, **MOE**, **MWE**, **MBW0**, **MBW1**, **MALE**, **MALE2** outputs.

13.8.1. EMB Clock and Command Signals Control

The **MCLK** clock can be inverted by setting **EMB_CLK_INV** register. For application, user can select the **MWE** output signal source from WE generator or timer TM10/16/20 clock output by setting **EMB_WE_MUX** register. Also user can select the **MOE** output signal source from WE generator or timer TM10/16/20 clock output by setting **EMB_OE_MUX** register. All the **TMx_CKO** signals are come from TMx timer modules.

The following diagram is showing the EMB module IO signals control of **MCLK**, **MWE** and **MOE**.

Figure 13-4. EMB Module IO Control - MCLK, MWE, MOE



The **MCE**, **MALE** and **MALE2** outputs can be inverted by setting **EMB_CE_INV**, **EMB_CLK_ALE** and **EMB_ALE2_INV** registers. The **EMB_CE_SWEN** register is used to enable **EMB_MCE** output by software control. When enables, user can directly control the output by setting **EMB_CE_SWO** register. For **MALE** and **MALE2** outputs, user can enable software control by setting **EMB_ALE_SWEN** and **EMB_ALE2_SWEN** registers. When enables, also user can directly control the output by setting **EMB_ALE_SWO** and **EMB_ALE2_SWO** registers.

For PCB routing or external device request, user can select the **MCE** output signal source from CE generator, ALE generator or ALE2 generator by setting **EMB_CE_MDS** register. Also user can select the **MALE2** output signal source from ALE generator or ALE2 generator by setting **EMB_ALE2_MDS** register.

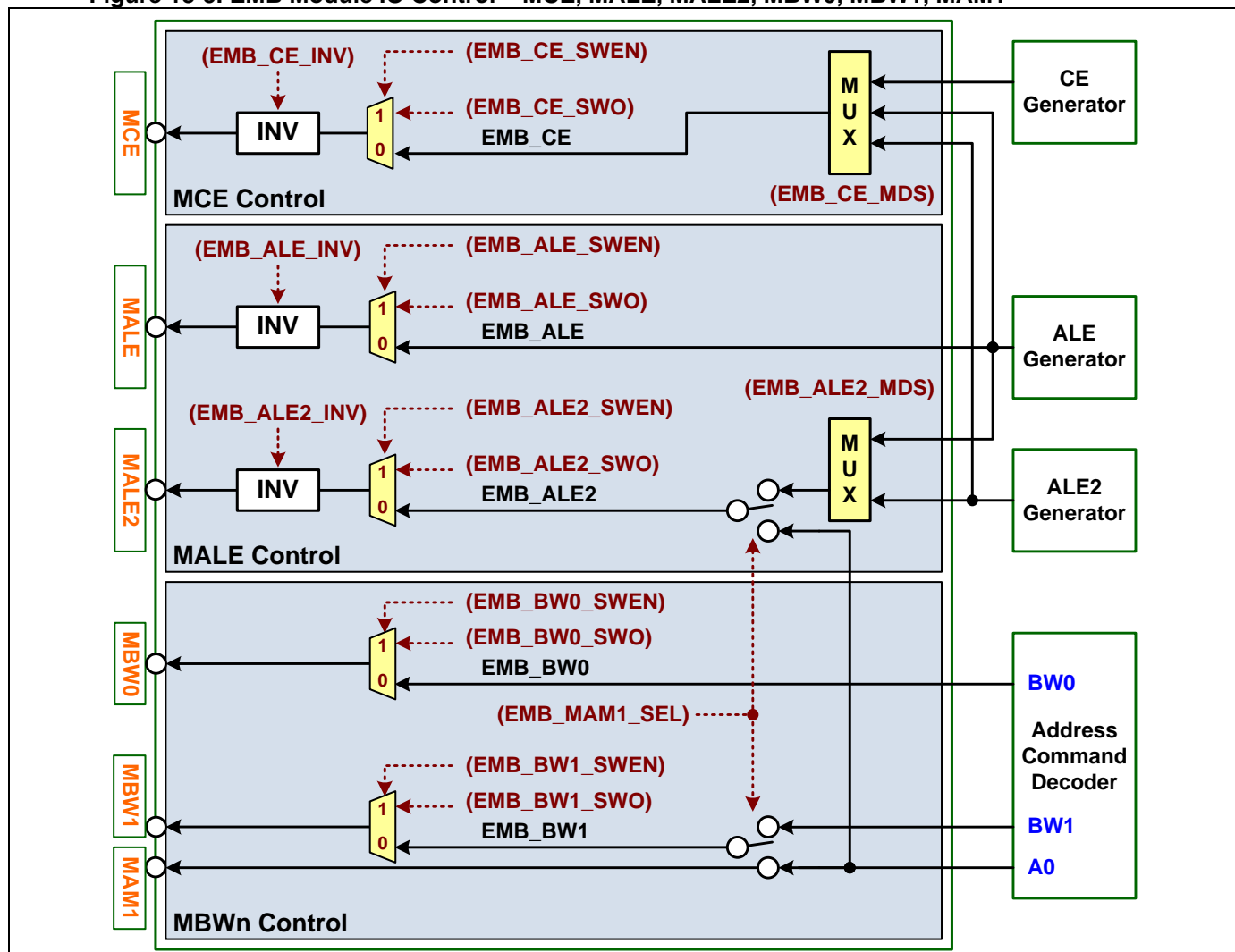
For **MBW0** and **MBW1** outputs, user can enable software control by setting **EMB_BW0_SWEN** and **EMB_BW1_SWEN** registers. When enables, also user can directly control the output by setting **EMB_BW0_SWO** and **EMB_BW1_SWO** registers.

For 8-bit memory bus, user can enable EMB internal memory address **A-1** signal output by setting The **EMB_MAM1_EN** register. The bit is only effective if EMB bus is 8-bit (**EMD_BUS_DSIZE=0**). When **EMB_MAM1_EN** is enabled, the address LSB is MAM1 pin. When **EMB_MAM1_EN** is disabled, the address LSB is MA0.

The **EMB_MAM1_SEL** register is used to select output pin of the internal memory lowest address bit of **A-1** signal. The signal can be no output or output to **MBW1** and **MALE2**. Refer the register descriptions of **EMB_MAM1_SEL** for more information.

The following diagram is showing the EMB module IO signals control of **MCE**, **MALE**, **MALE2**, **MBW0**, **MBW1** and **MAM1**.

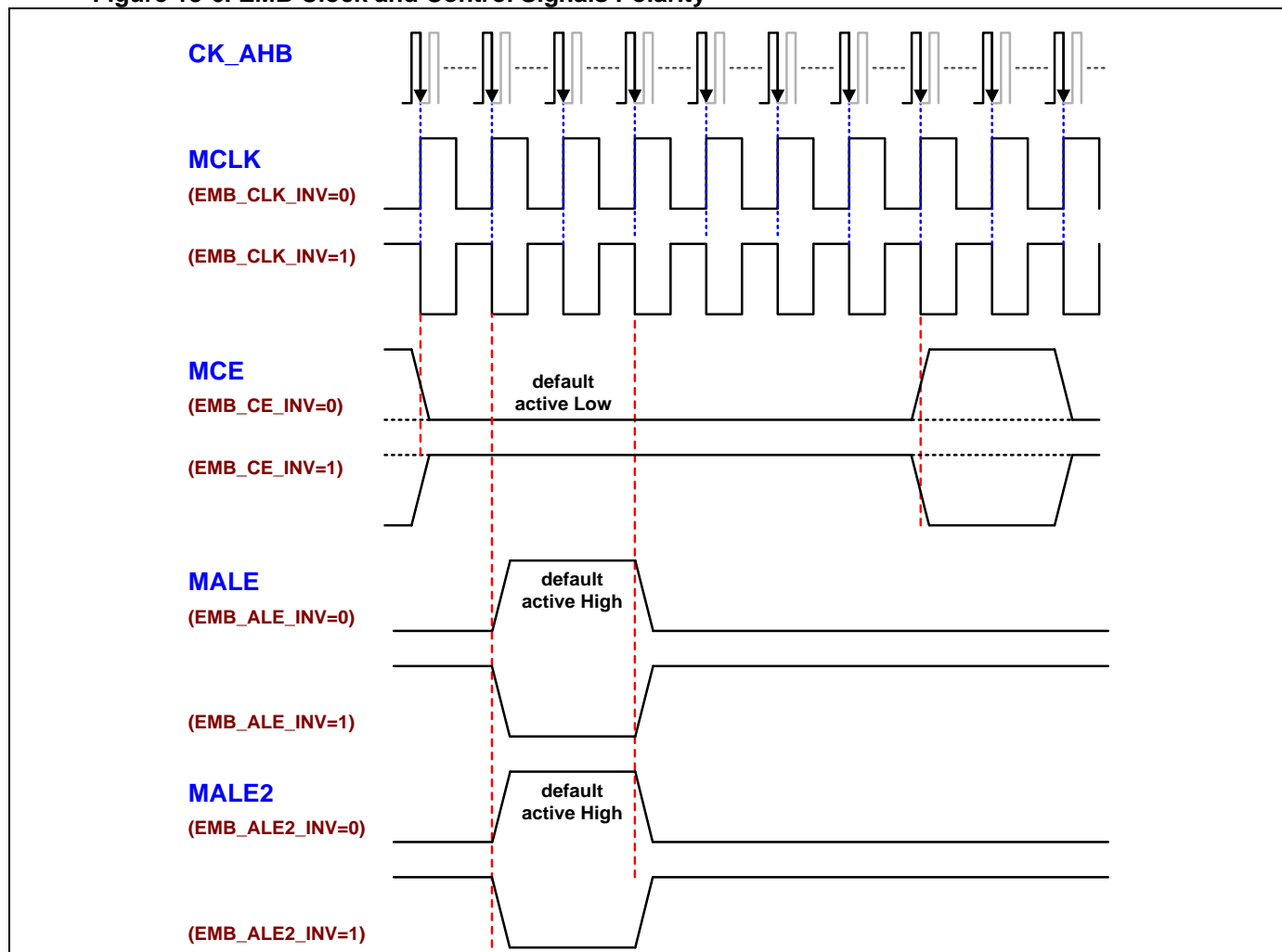
Figure 13-5. EMB Module IO Control – MCE, MALE, MALE2, MBW0, MBW1, MAM1



The EMB supports IO signal inversion for **MCLK**, **MCE**, **MALE** and **MALE2** by setting **EMB_CLK_INV**, **EMB_CE_INV**, **EMB_ALE_INV** and **EMB_ALE2_INV** register.

The following diagram is showing the programmable polarity timing of EMB clock and control signals.

Figure 13-6. EMB Clock and Control Signals Polarity



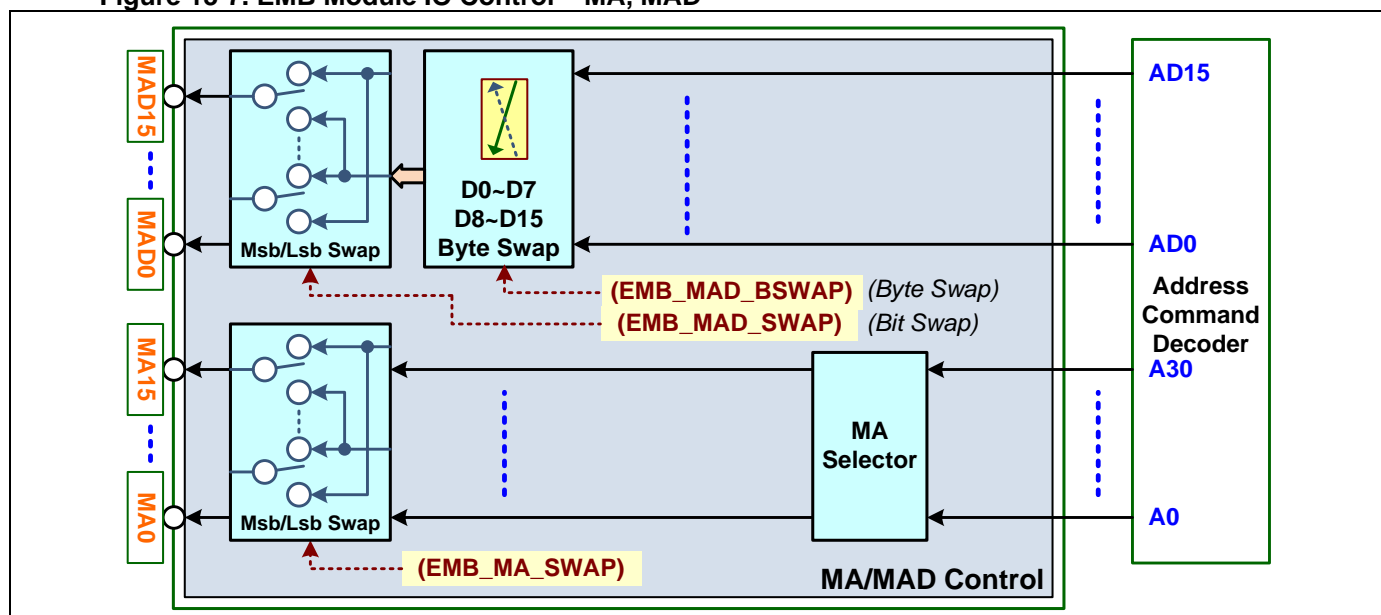
13.8.2. EMB Address and Data Signals Control

For **MBW0** and **MBW1** outputs, user can enable software control by setting **EMB_BW0_SWEN** and **EMB_BW1_SWEN** registers. When enables, also user can directly control the output by setting **EMB_BW0_SWO** and **EMB_BW1_SWO** registers.

The **EMB_MA_SWP** register is used to enable to do Msb/Lsb signal swap for the signals of **MA[15:0]**. The **EMB_MAD_SWP** register is used to do Msb/Lsb signal swap for the signals of **MAD[15:0]**. The **EMB_MAD_BSWP** register is used to do byte-swap for the **MAD[15:8]** and **MAD[7:0]** signals.

The following diagram is showing the EMB module IO signals control of **MA** and **MAD**.

Figure 13-7. EMB Module IO Control – MA, MAD



13.8.3. Signal Mapping Suggestion for External Devices

The EMB module is able to access the external devices of SRAM, NOR/NAND-flash and 8080 interface LCD. The following table is showing the suggested signal mapping of EMB interface and external device.

Table 13-2. EMB Interface and Device Pin Mapping

Device	SRAM		NOR		NAND		LCD RAM	
	16-bit data	8-bit data	16-bit data	8-bit data	16-bit data	8-bit data	16-bit data	8-bit data
MCLK	CLK		CLK (*3)		-		-	
MA[15:0]	A[15:0]	A[15:0]	A[15:0]	A[15:0]	-		-	
MAD[15]	DQ[15]	-	DQ[15]/A-1	-	DQ[15]	-	DB[15]	-
MAD[14:8]	DQ[14:8]	-	DQ[14:8]	-	DQ[14:8]	-	DB[14:8]	-
MAD[7:0]	DQ[7:0]	DQ[7:0]	DQ[7:0]	DQ[7:0]	DQ[7:0]	DQ[7:0]	DB[7:0]	DB[7:0]
MCE	CE#		CE#		CE#		/CS (*1)	
MWE	WE#		WE#		WE#		/WR	
MOE	OE#		OE#		RE#		/RD	
MALE	ADSP#		ADV# (*3)		ALE (*1)		RS (*1)	
MALE2	-		RST# (*1)		CLE (*1)		RESET (*1)	
MBW0	BW0		BYTE#		-		-	
MBW1	BW1		WP# (*1)		WP# (*1)		-	
GPIO	MODE =0		RY/BY# (*4)		RY/BY#		-	

<Note> *1: Output control directly by software mode register setting

*2: Any unused GPIO pin (input)

*3: Only for synchronous NOR

*4: Only for asynchronous NOR

13.9. EMB Memory Control

13.9.1. EMB Memory Space

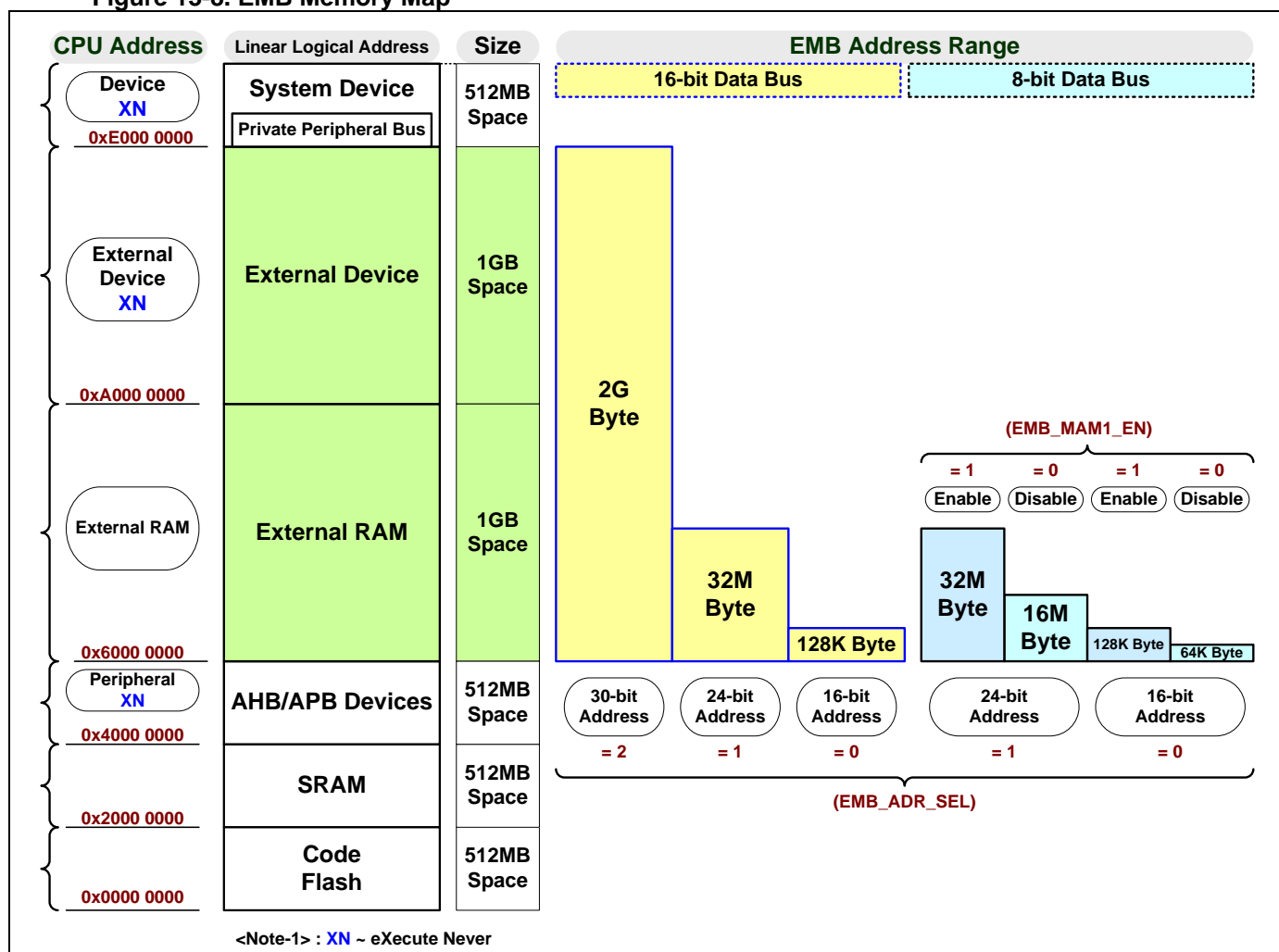
The maximum memory space size of EMB is 2G-byte. The EMB memory space is located at the CPU base address 0x6000 0000.

The EMB supports optional data bus 8/16-bit data width and optional 16/24/30-bit address mode. User can set data width by setting **EMB_BUS_DSIZE** register and address range by setting **EMB_ADR_SEL** register.

The maximum memory space size of external device is 2G/32M/128K-byte for 16-bit data width or 16M/64K-byte for 8-bit data width. Except the configuration of 16-bit data width and 30-bit address mode which costs all 2G-byte space, others are filled the repeat pseudo memory space with identical content of 1st EMB memory space in the EMB 2G-byte space.

The following diagram is showing the CPU memory map for external device.

Figure 13-8. EMB Memory Map



13.9.2. AHB to External Memory Transactions

AHB bus supports 8/16/32-bit data width and EMB supports 8/16-bit data width. When CPU executes the instruction command and the access memory space is located at the EMB space, the EMB will automatically process the data transactions between CPU and external memory/device by user configuration. User can acts the external memory or device as same as internal embedded memory only if the EMB has been initialized to valid configuration.

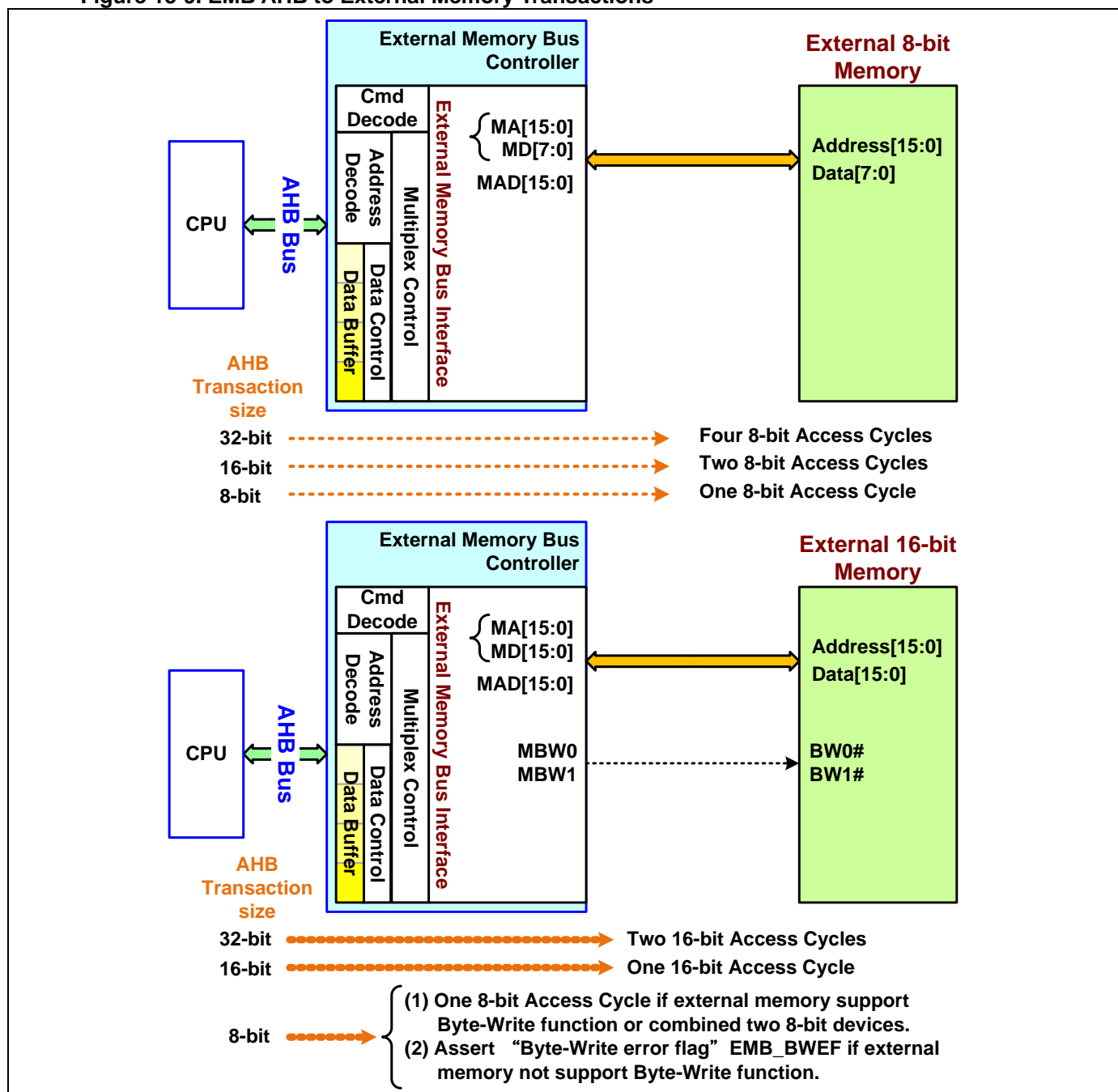
Before the EMB data transactions, user needs to initialize the fundamental configurations about the EMB data bus width, EMB address mode, EMB address and data multiplex mode, EMB write protection and EMB timing setting.

The EMB write access error is necessary to avoid which CPU executes the instruction with 8-bit data write and writes to 16-bit data bus of external device but the device does not support 8-bit write access. Refer the

section of “[EMB Write Protection](#)” for more information.

The following diagram is showing the data transaction between AHB bus and external device.

Figure 13-9. EMB AHB to External Memory Transactions



The following table is showing the capability of data transaction between AHB bus and external device. The EMB supports to access the external memory of SRAM, NOR/NAND-flash with 8-bit or 16-bit data bus. The SRAM and NOR-flash are supported both synchronous and asynchronous timing. But the NAND-flash is only supported asynchronous timing.

Table 13-3. EMB AHB to External Memory Supported Transactions

Device	Mode	Memory data size	AHB data size	Access RW	Access Allowed	Access Cycles	Note
SRAM	Sync/Async	16	32	R/W	Yes	2	
			16	R/W	Yes	1	
			8	R	Yes	1	
				W	No	1	not support Byte-write function

	Sync/Async	8 (*1)	32	R/W	Yes	4	
			16	R/W	Yes	2	
			8	R/W	Yes	1	
NOR	Sync/Async	16	32	R/W	Yes	2	
			16	R/W	Yes	1	
			8	R	Yes	1	
				W	No	1	not support BYTE# function
	Sync/Async	8 (*1)	32	R/W	Yes	4	
			16	R/W	Yes	2	
			8	R/W	Yes	1	
NAND	Async	16	32	R/W	Yes	2	
			16	R/W	Yes	1	
			8	R	Yes	1	
				W	No	1	not support Byte-write function
	Async	8 (*1)	32	R/W	Yes	4	
			16	R/W	Yes	2	
			8	R/W	Yes	1	

13.9.3. EMB Write Protection

The EMB supports the write protection function for external memory or device. User can enable or disable the write access by setting **EMB_WEN** register. When EMB write access is disabled (**EMB_WEN**=0) and operates an EMB write process, one write-protect error detect flag of WPEF (**EMB_WPEF**) is asserted.

The EMB also support the byte-write protection to avoid to writes 8-bit data to an unsupported 8-bit write access device with 16-bit data bus. User can enable or disable the byte-write access by setting **EMB_BW_EN** register. When user configure 16-bit EMB data bus (**EMB_BUS_DSIZE**=16-bit) and EMB write access is enabled (**EMB_WEN**=1) but byte-write access is disabled (**EMB_BW_EN**=0), this byte-write error detect flag of BWEF (**EMB_BWEF**) is asserted if operates an EMB byte-write process.

Refer the table of “EMB AHB to External Memory Supported Transactions”. The 16-bit data bus NAND-flash is not supported the byte-write function, so user must set byte-write access control to disable (**EMB_BW_EN**=0).

13.9.4. EMB Timing Control

The EMB is built in a timing generator to generate the EMB output signals. It provides multiple timing states and programmable timing cycle for external device flexible design.

User can program the time cycles of **tALES**, **tALEW**, **tALEH**, **tACCS**, **tACCW**, **tACCH**, **tIDLE** in the unit of MCLK clock time by register setting of **EMB_ALES**, **EMB_ALEW**, **EMB_ALEH**, **EMB_ACCS**, **EMB_ACCW**, **EMB_ACCH**, **EMB_IDLE**. The time cycles of **tACCH** can be set in **EMB_ACCS** register for write access and is forced to zero time by hardware for read access. The time cycles of **tIDLE** is forced to zero time between two 16-bit EMB data access for a 32-bit data CPU instruction. It is forced to one MCLK clock time after last 16-bit EMB data access for a 16-bit or 32-bit data CPU instruction.

The EMB can support both synchronous and asynchronous data transactions for external device. User needs to set synchronous timing for synchronous type device in **EMB_SYNC_EN** register.

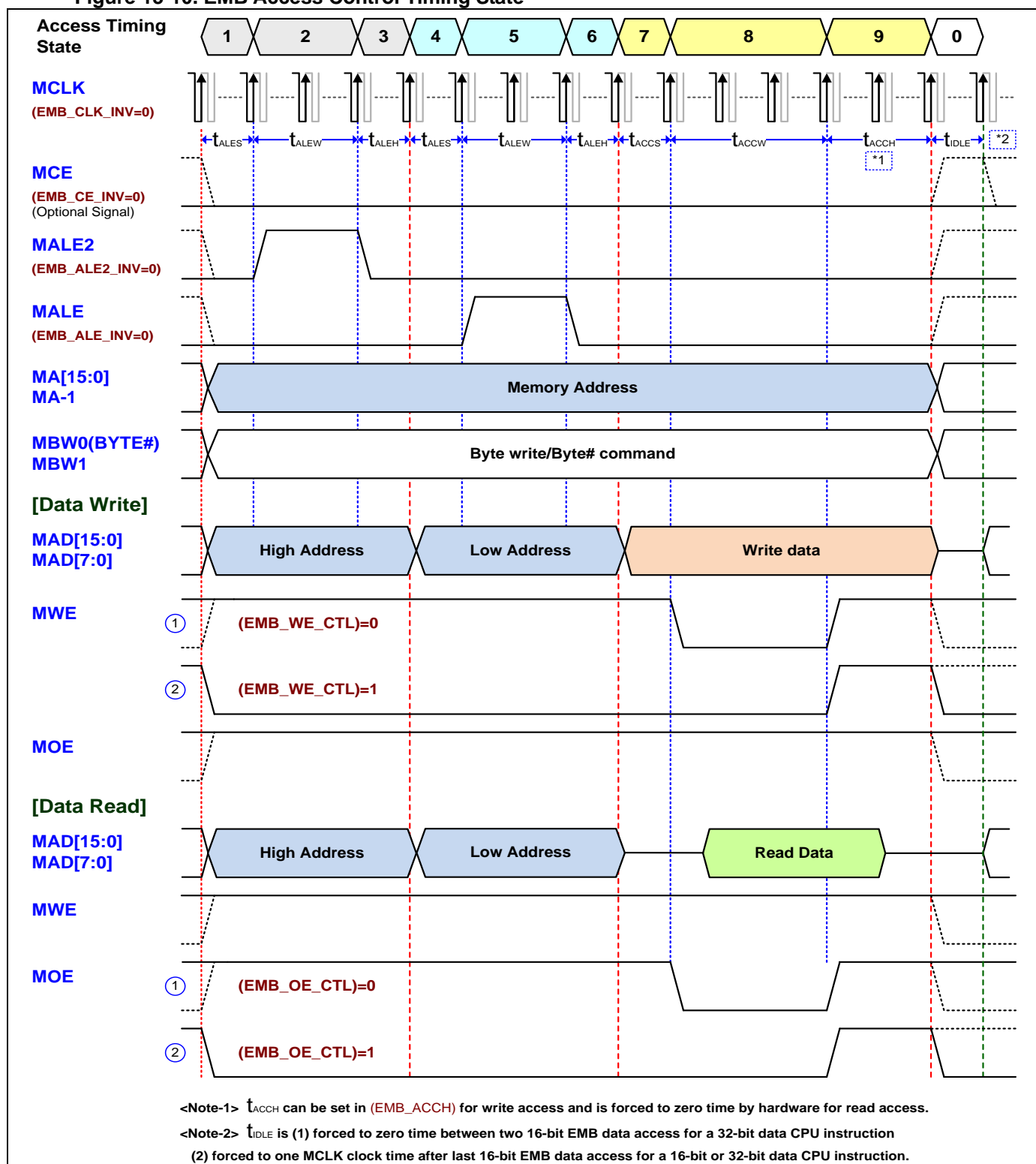
The **MWE** and **MOE** output timing are able to configure two control modes of “TOGGLE” and “LOW”. When sets “TOGGLE” mode, the signal will active at the data access cycle (State-8 in following diagram). When sets “Low” mode, the signal will active at beginning (State-1 in following diagram).

The following diagram is showing the EMB access control timing state. State-0 is idle state. State-1, 4, 7 are setup time state. State-2, 5 are pulse width time state. State-8 is access data time state. State-3, 6, 9 are hold time state.

User can configure the multiplex address phase number by setting **EMB_BUS_MDS** and **EMB_ADR_TWO** registers. Refer the section of “[EMB Address and Data Interface Mode](#)” for more information. When selects one multiplex address phase, State-1, 2, 3 are disappeared. When selects no multiplex address phase, State-1, 2, 3, 4, 5, 6 are all disappeared.

The following diagram is showing the EMB access control timing state.

Figure 13-10. EMB Access Control Timing State



13.10. EMB Address and Data Interface Mode

The EMB supports six types of address and data interface modes. They include four types of address and data multiplex modes. These modes are including of “16bit-MA + 16bit-MD”, “16bit-MA + 8bit-MD”, “16bit-MA + 8bit-MAD”, “8bit-MAD with 2 Adr”, “16bit-MA + 16bit-MAD”, “16bit-MAD with 2 Adr”. (Adr: address phase)

User needs to select one mode by setting **EMB_ADR_TWO**, **EMB_BUS_DSIZE** and **EMB_BUS_MDS** register. The **EMB_BUS_MDS** register is used to select the EMB bus address and data which is “Multiplex” or “Separated”.

The **EMB_ADR_TWO** register is used to enable two multiplex address phase timing mode. When disables, it will be no multiplex address phase if **EMB_BUS_MDS** sets “Separated” and one multiplex address phase if **EMB_BUS_MDS** sets “Multiplex”. When enables, it will be two address phase and **EMB_BUS_MDS** must be “Multiplex”.

The **EMB_BUS_DSIZE** register is used to set the EMB data width of 8-bit or 16-bit. The **EMB_ADR_SEL** register is used to select address range of 16-bit, 24-bit and 30-bit.

The following table is showing the EMB Address/Data Interface mode register setting.

Table 13-4. EMB Address/Data Interface Mode Setting

Interface Mode	EMB Setting Register					Memory Space (Bytes)
	BUS_DSIZE (*1)	ADR_TWO	BUS_MDS	ADR_SEL	MAM1_EN	
16bit-MA + 16bit-MD	1 16-bit data	0 0 phase Adr	1 separated	0 16-bit Adr	x	128K
16bit-MA(HAdr) + 16bit-MAD(LAdr+Data)	1 16-bit data	0 1 phase Adr	0 multiplexed	0,1,2 16,24,30-bit Adr	x	128K, 32M, 2G
16bit-MAD with 2 Adr (HAdr+LAdr+Data)	1 16-bit data	1 2 phase Adr	0 multiplexed	1,2 24,30-bit Adr	x	32M, 2G
16bit-MA + 8bit-MD	0 8-bit data	0 0 phase Adr	1 separated	0 16-bit Adr	0 (Disable)	64K
					1 (Enable)	128K
16bit-MA(HAdr) + 8bit-MAD(LAdr+Data)	0 8-bit data	0 1 phase Adr	0 multiplexed	0,1 16,24-bit Adr	0 (Disable)	64K, 16M
					1 (Enable)	128K, 32M
8bit-MAD with 2 Adr (HAdr+LAdr+Data)	0 8-bit data	1 2 phase Adr	0 multiplexed	0 16-bit Adr	0 (Disable)	64K
					1 (Enable)	128K

<Sign> x: don't care

MA: Memory address, MAD: Memory address and data multiplexed,
MD: Memory data, HAdr: High address, LAdr: Low address

The following table is showing the EMB internal signal and external signal mapping by difference EMB interface modes and memory size for user reference. User can refer the EMB control block for more information.

Table 13-5. EMB Internal and External Signal Mapping

Interface Mode	Data Bus	Memory Space	Register	EMB Interface													
			MAM1_EN	EMB Pin	MA[15:0]	MAM1	MAD[15:8]	MAD[7:0]	MALE	MALE2	MBW0	MBW1	MCLK	MCE	MWE	MOE	
16bit-MA + 16bit-MD	16-bit	128KB	x	Internal	XA[16:1]		XD[15:8]	XD[7:0]			XBW0	XBW1	XCLK	XCE	XWE	XOE	
				Device Pin	A[15:0]		DQ[15:8]	DQ[7:0]		-	BW0	BW1	MCLK	CE#, /CS	WE#, /WR	OE#, /RD	
16bit-MA(HAdr) + 16bit-MAD (LAdr+Data)		128KB	x	Internal			XA[16:9] XD[15:8]	XA[8:1] XD[7:0]	XALE		XBW0	XBW1	XCLK	XCE	XWE	XOE	
				Device Pin			A[15:8] DQ[15:8]	A[7:0] DQ[7:0]	ALE	-	BW0	BW1	MCLK	CE#, /CS	WE#, /WR	OE#, /RD	
		32MB		Internal	0,XA[24:17]		XA[16:9] XD[15:8]	XA[8:1] XD[7:0]	XALE		XBW0	XBW1	XCLK	XCE	XWE	XOE	
				Device Pin	0,A[23:16]		A[15:8] DQ[15:8]	A[7:0] DQ[7:0]	ALE	-	BW0	BW1	MCLK	CE#, /CS	WE#, /WR	OE#, /RD	
		2GB		Internal	0,XA[30:17]		XA[16:9] XD[15:8]	XA[8:1] XD[7:0]	XALE		XBW0	XBW1	XCLK	XCE	XWE	XOE	
				Device Pin	0,A[29:16]		A[15:8] DQ[15:8]	A[7:0] DQ[7:0]	ALE	-	BW0	BW1	MCLK	CE#, /CS	WE#, /WR	OE#, /RD	
16bit-MAD with 2 Adr (HAdr+ LAdr+Data)		32MB		Internal			0x00 XA[16:9] XD[15:8]	XA[24:17] XA[8:1] XD[7:0]	XALE	XALE2	XBW0	XBW1	XCLK	XCE	XWE	XOE	
				Device Pin			- A[15:8] DQ[15:8]	A[23:16] A[7:0] DQ[7:0]	ALE	ALE2	BW0	BW1	MCLK	CE#, /CS	WE#, /WR	OE#, /RD	
		2GB		Internal			0,XA[30:25] XA[16:9] XD[15:8]	XA[24:17] XA[8:1] XD[7:0]	XALE	XALE2	XBW0	XBW1	XCLK	XCE	XWE	XOE	
				Device Pin			0,A[29:24] A[15:8] DQ[15:8]	A[23:16] A[7:0] DQ[7:0]	ALE	ALE2	BW0	BW1	MCLK	CE#, /CS	WE#, /WR	OE#, /RD	
		16bit-MA + 8bit-MD	64KB	Disable	Internal	XA[15:0]			XD[7:0]			XBW0		XCLK	XCE	XWE	XOE
					Device Pin	A[15:0]			DQ[7:0]			BW0		MCLK	CE#, /CS	WE#, /WR	OE#, /RD
128KB			Enable	Internal	XA[16:1]	XA0		XD[7:0]			XBW0		XCLK	XCE	XWE	XOE	
				Device Pin	A[15:0]	MA-1 (*1)		DQ[7:0]			BW0		MCLK	CE#, /CS	WE#, /WR	OE#, /RD	
16bit-MA(HAdr) + 8bit-MAD (LAdr+Data)		8-bit	64KB	Disable	Internal	0,XA[15:8]			XA[7:0] XD[7:0]	XALE		XBW0		XCLK	XCE	XWE	XOE
					Device Pin	A[15:8]			A[7:0] DQ[7:0]	ALE		BW0		MCLK	CE#, /CS	WE#, /WR	OE#, /RD
	128KB		Enable	Internal	0,XA[16:9]	XA0		XA[8:1] XD[7:0]	XALE		XBW0		XCLK	XCE	XWE	XOE	
				Device Pin	0,A[15:8]	MA-1 (*1)		A[7:0] DQ[7:0]	ALE		BW0		MCLK	CE#, /CS	WE#, /WR	OE#, /RD	
	16MB		Disable	Internal	XA[23:8]			XA[7:0] XD[7:0]	XALE		XBW0		XCLK	XCE	XWE	XOE	
				Device Pin	A[23:8]			A[7:0] DQ[7:0]	ALE		BW0		MCLK	CE#, /CS	WE#, /WR	OE#, /RD	
	32MB		Enable	Internal	XA[24:9]	XA0		XA[8:1] XD[7:0]	XALE		XBW0		XCLK	XCE	XWE	XOE	
				Device Pin	A[23:8]	MA-1 (*1)		A[7:0] DQ[7:0]	ALE		BW0		MCLK	CE#, /CS	WE#, /WR	OE#, /RD	
8bit-MAD with 2 Adr (HAdr+ LAdr+Data)	64KB		Disable	Internal				XA[15:8] XA[7:0] XD[7:0]	XALE	XALE2	XBW0		XCLK	XCE	XWE	XOE	
				Device Pin				A[15:8] A[7:0] DQ[7:0]	ALE	ALE2	BW0		MCLK	CE#, /CS	WE#, /WR	OE#, /RD	
	128KB		Enable	Internal		XA0		XA[16:9] XA[8:1] XD[7:0]	XALE	XALE2	XBW0		XCLK	XCE	XWE	XOE	
				Device Pin		MA-1 (*1)		A[15:8] A[7:0] DQ[7:0]	ALE	ALE2	BW0		MCLK	CE#, /CS	WE#, /WR	OE#, /RD	

<Note> *1: MA-1 signal can output from MAM1, MBW1 and ALE2 pin.

13.10.1. EMB 16bit MA and 16bit MD

This mode is configured to separated 16-bit address and 16-bit data. The address mode is only 16-bit address space. There is no timing state-0, 1, 2, 3, 4, 5 and 6.

Figure 13-11. EMB Address/Data Interface – 16bit MA + 16bit MD

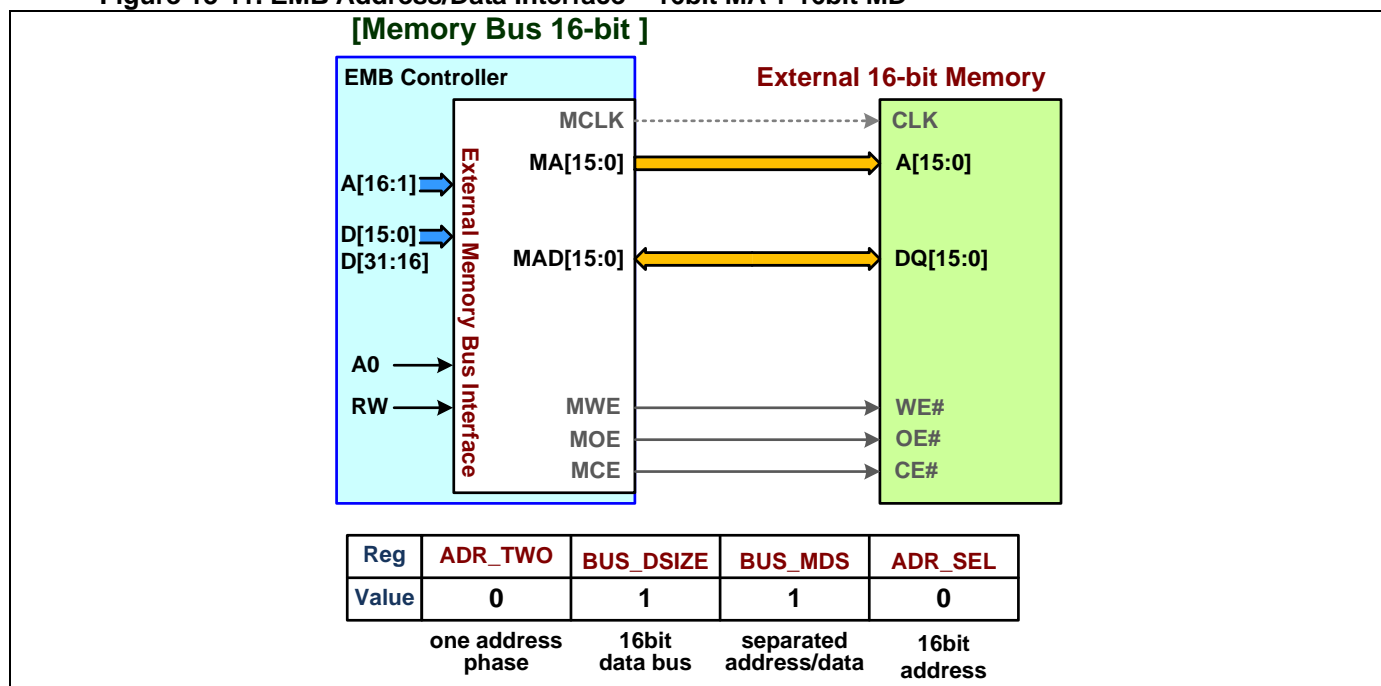
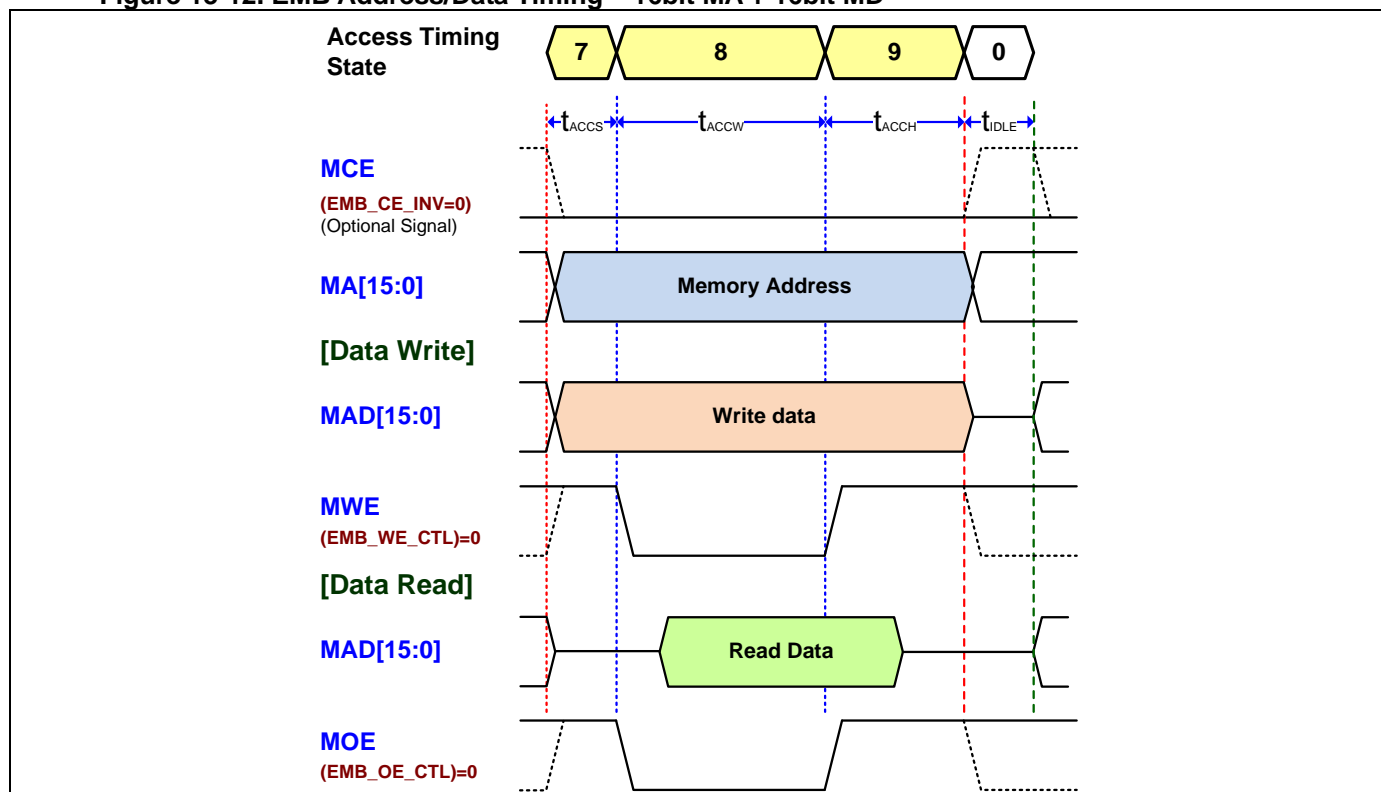


Figure 13-12. EMB Address/Data Timing – 16bit MA + 16bit MD



13.10.2. EMB 16bit MA and multiplexed 16bit MD

This mode is configured to independent 16-bit address and multiplexed 16-bit address/data with one address latch signal of **MALE**. The address mode can select 16-bit, 24-bit or 30-bit address space. There is no timing state-0, 1, 2 and 3.

Figure 13-13. EMB Address/Data Interface – 16bit MA + 16bit MAD

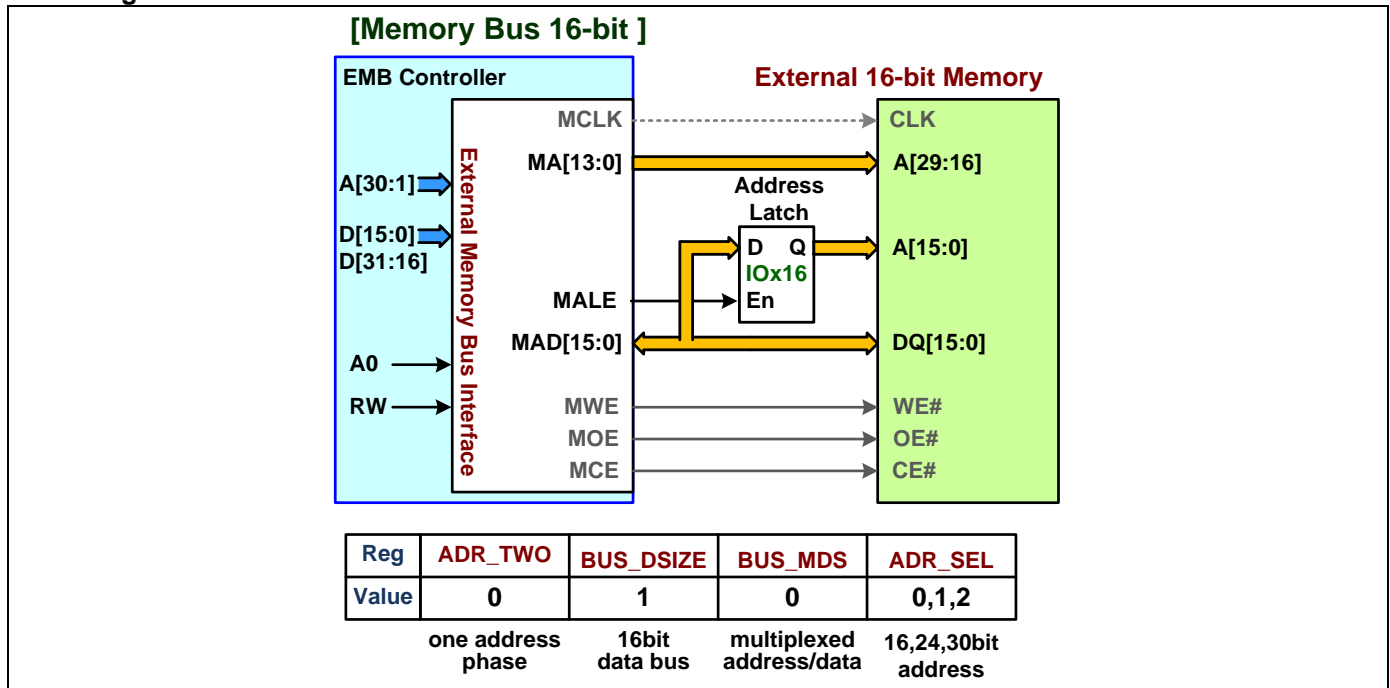
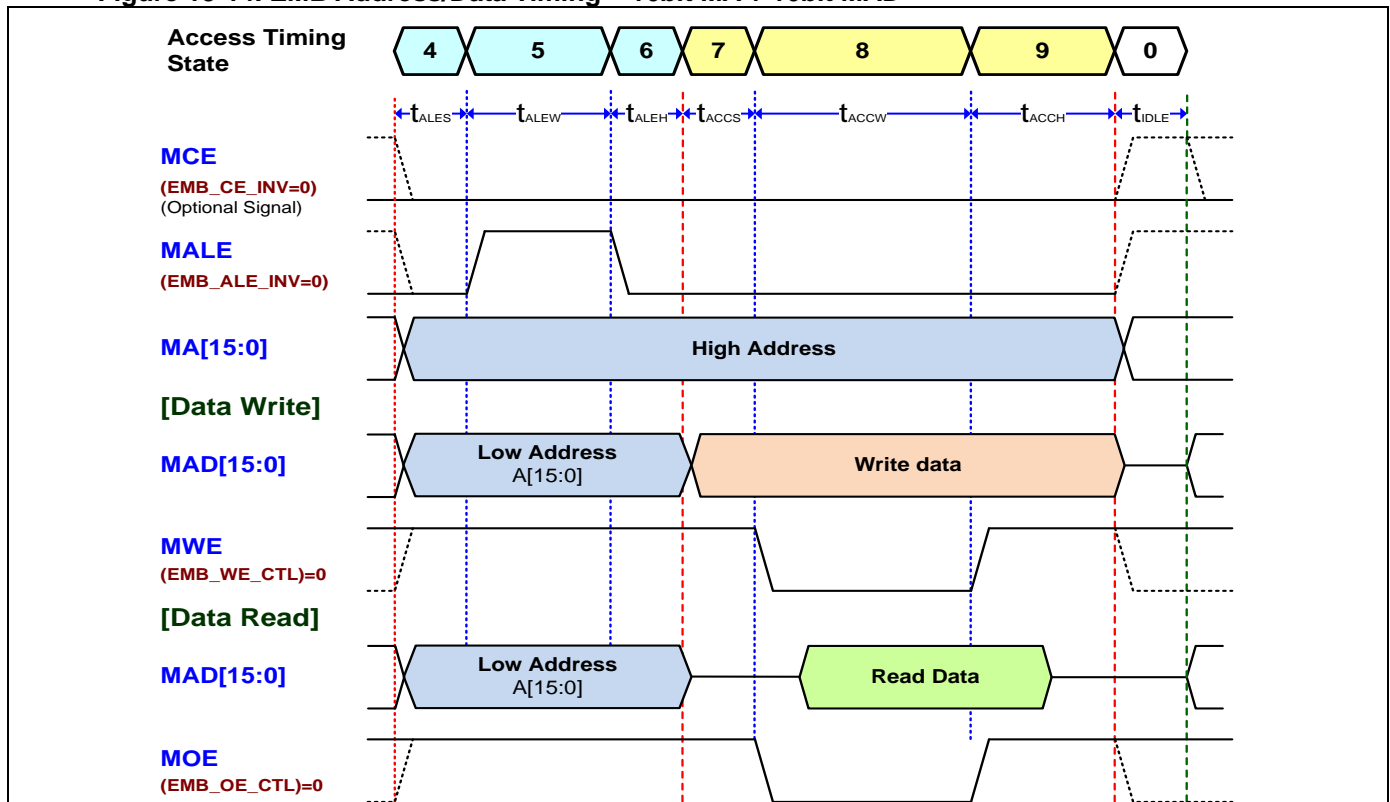


Figure 13-14. EMB Address/Data Timing – 16bit MA + 16bit MAD



13.10.3. EMB Multiplexed 16bit MAD with 2 Address Phase

This mode is configured to multiplexed 16-bit address/data with two address latch signals of **MALE** and **MALE2**. The address mode can select 16-bit, 24-bit or 30-bit address space.

Figure 13-15. EMB Address/Data Interface – 16bit MAD with 2 Address Phase

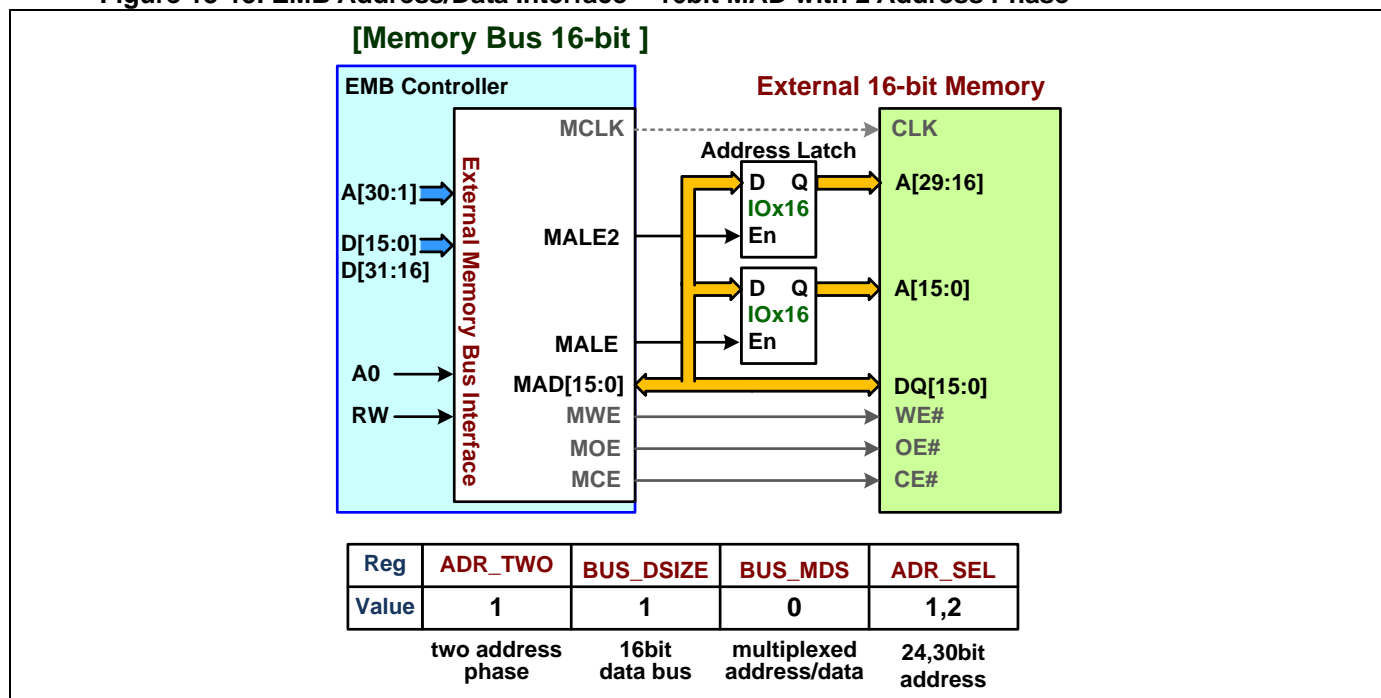
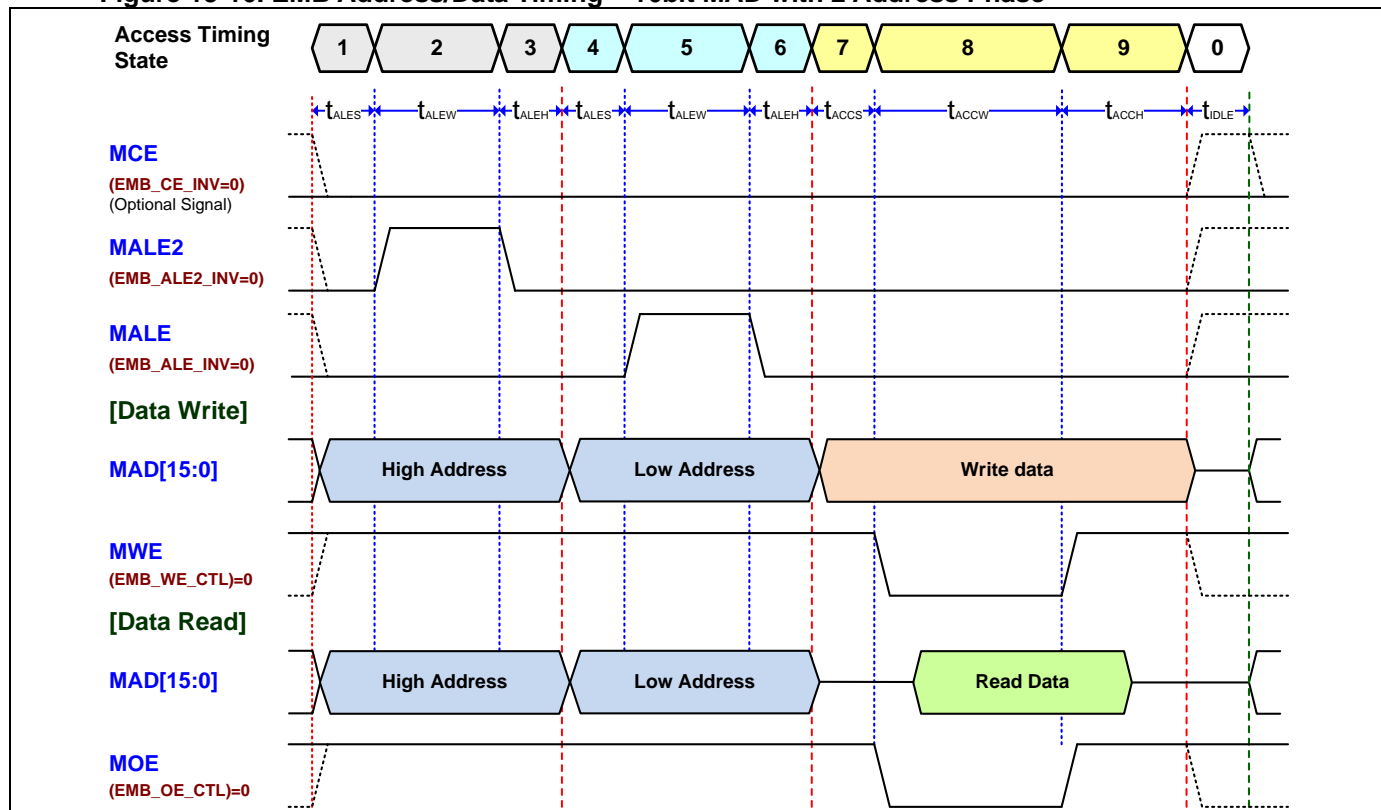


Figure 13-16. EMB Address/Data Timing – 16bit MAD with 2 Address Phase



13.10.4. EMB 16bit MA and 8bit MD

This mode is configured to separated 16-bit address and 8-bit data. The address mode is only 16-bit address space. There is no timing state-0, 1, 2, 3, 4, 5 and 6.

Figure 13-17. EMB Address/Data Interface – 16bit MA + 8bit MD

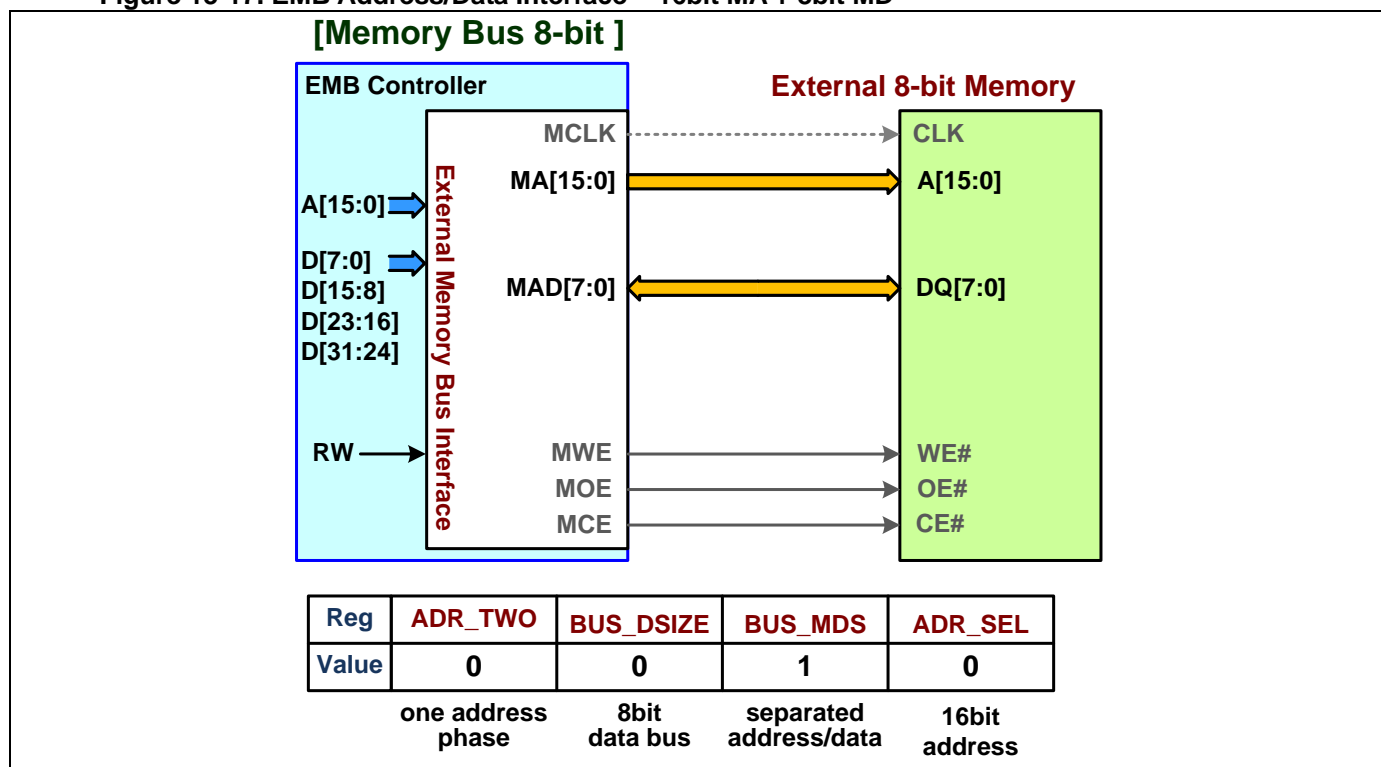
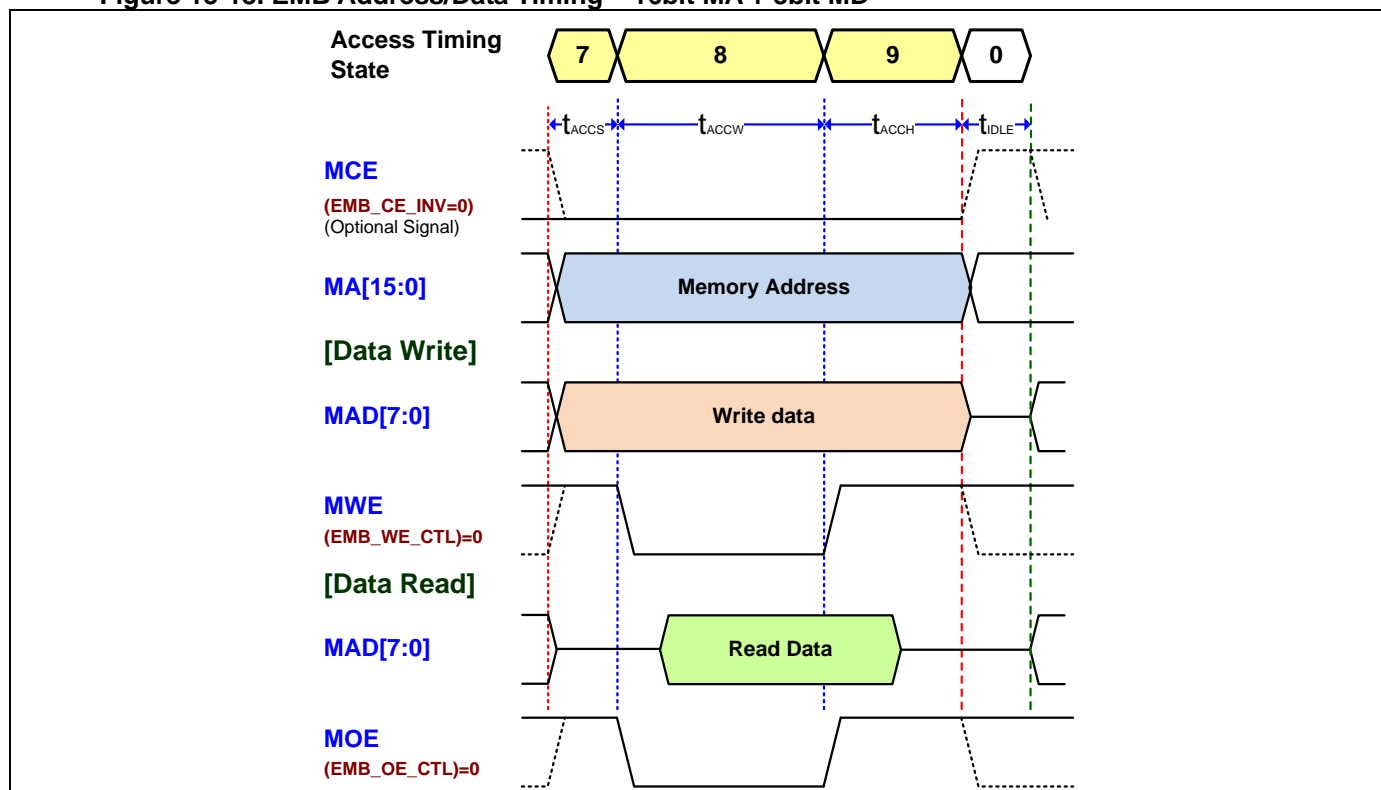


Figure 13-18. EMB Address/Data Timing – 16bit MA + 8bit MD



13.10.5. EMB 16bit MA and multiplexed 8bit MAD

This mode is configured to independent 16-bit address and multiplexed 8-bit address/data with one address latch signal of **MALE**. The address mode can select 16-bit or 24-bit address space. There is no timing state-0, 1, 2 and 3.

Figure 13-19. EMB Address/Data Interface – 16bit MA + 8bit MAD

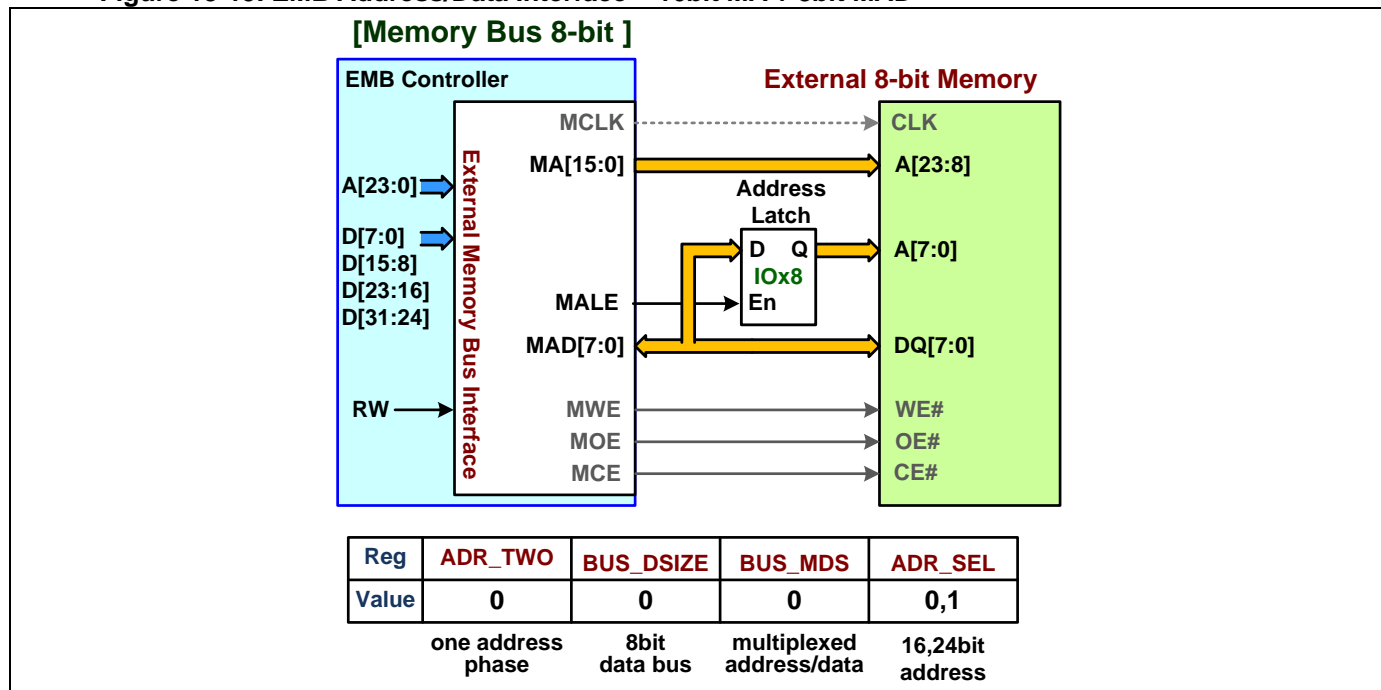
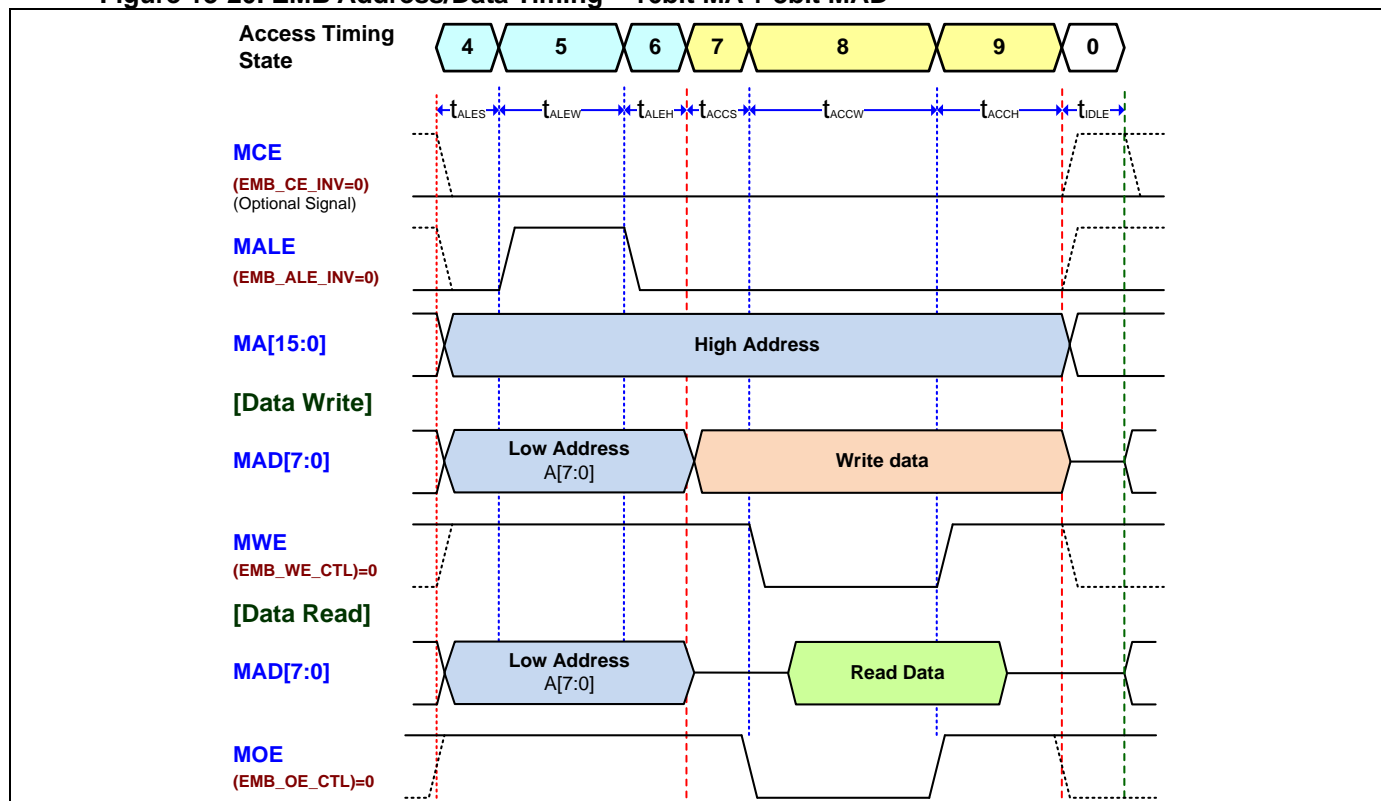


Figure 13-20. EMB Address/Data Timing – 16bit MA + 8bit MAD



13.10.6. EMB Multiplexed 8bit MAD with 2 Address Phase

This mode is configured to multiplexed 8-bit address/data with two address latch signals of **MALE** and **MALE2**. The address mode is only 16-bit address space.

Figure 13-21. EMB Address/Data Interface – 8bit MAD with 2 Address Phase

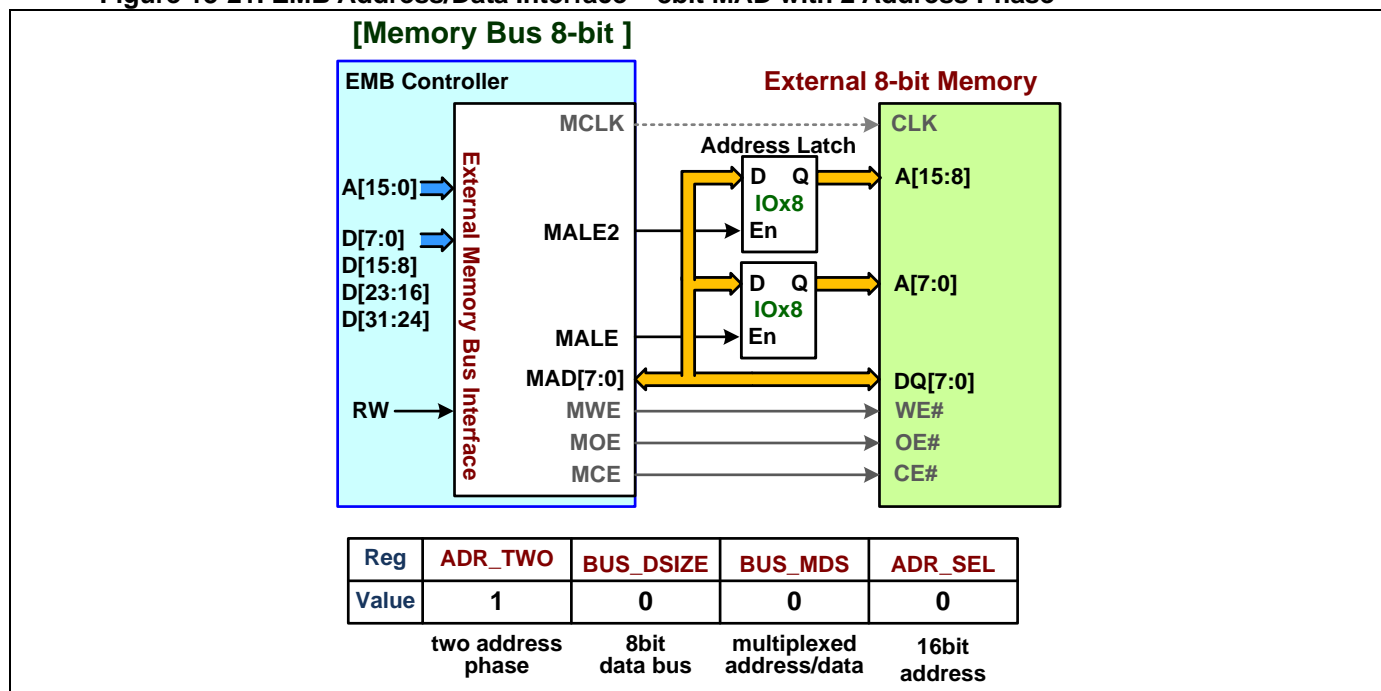
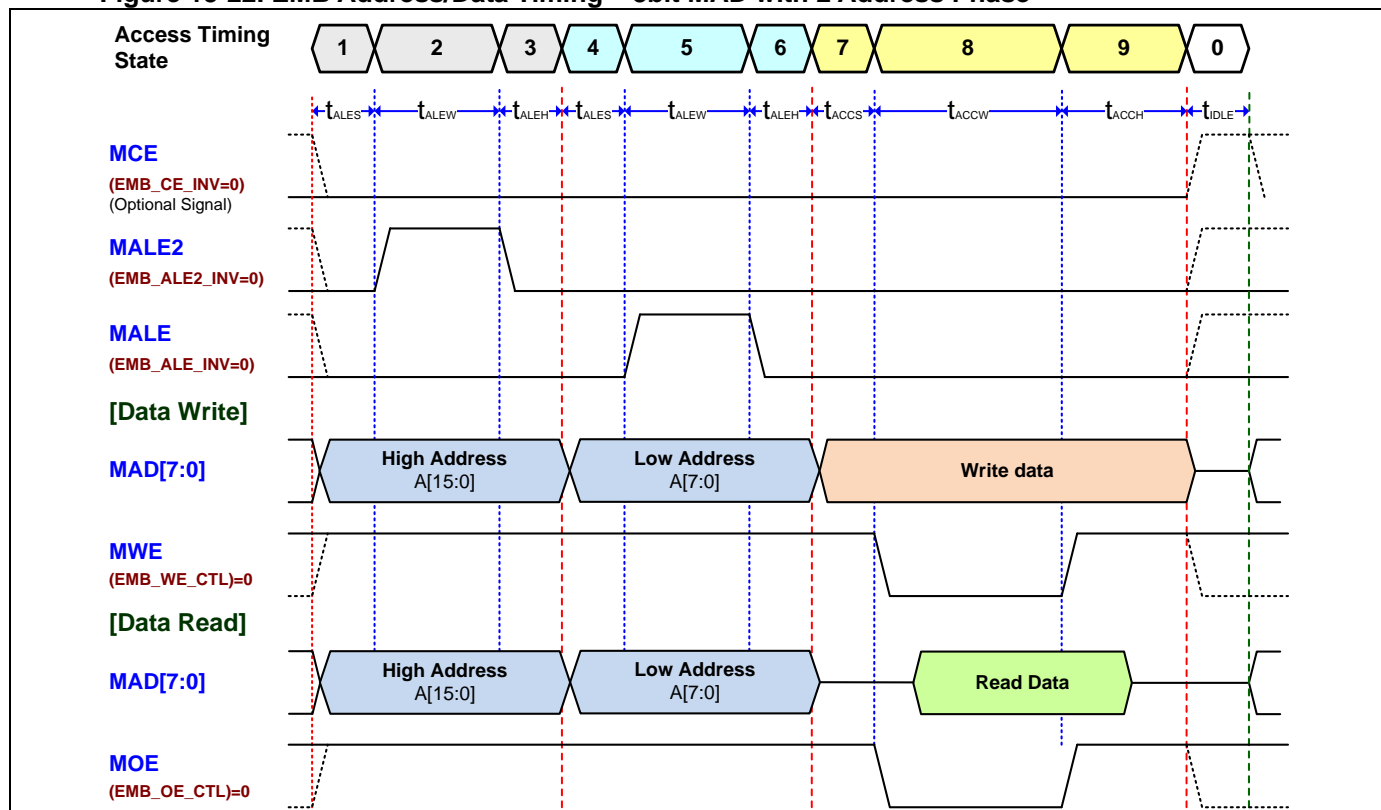


Figure 13-22. EMB Address/Data Timing – 8bit MAD with 2 Address Phase



13.11. EMB Device Interface and Timing

The EMB controller supports to access the external devices of SRAM, NOR/NAND-flash and 8080 interface LCD. User can initialize the fundamental configurations about the EMB data bus width, EMB address mode, EMB address and data multiplex mode, EMB write protection and EMB timing setting by referring the external device specification for application. Refer the sections of [“EMB Memory Control”](#) and [“EMB Address and Data Interface Mode”](#) about the fundamental configurations.

User needs to plan the EMB IO signals to map the external device IO signals by referring the pin configured specification of external device. Refer the section of [“Signal Mapping Suggestion for External Devices”](#) about the suggested EMB signals to external device pins.

Also user needs to configure the chip GPIO AFS setting for EMB control. Refer the section of [“Pin Suggestion for EMB Signal”](#) about the suggested GPIO AFS pins.

13.11.1. SRAM Interface and Access Timing

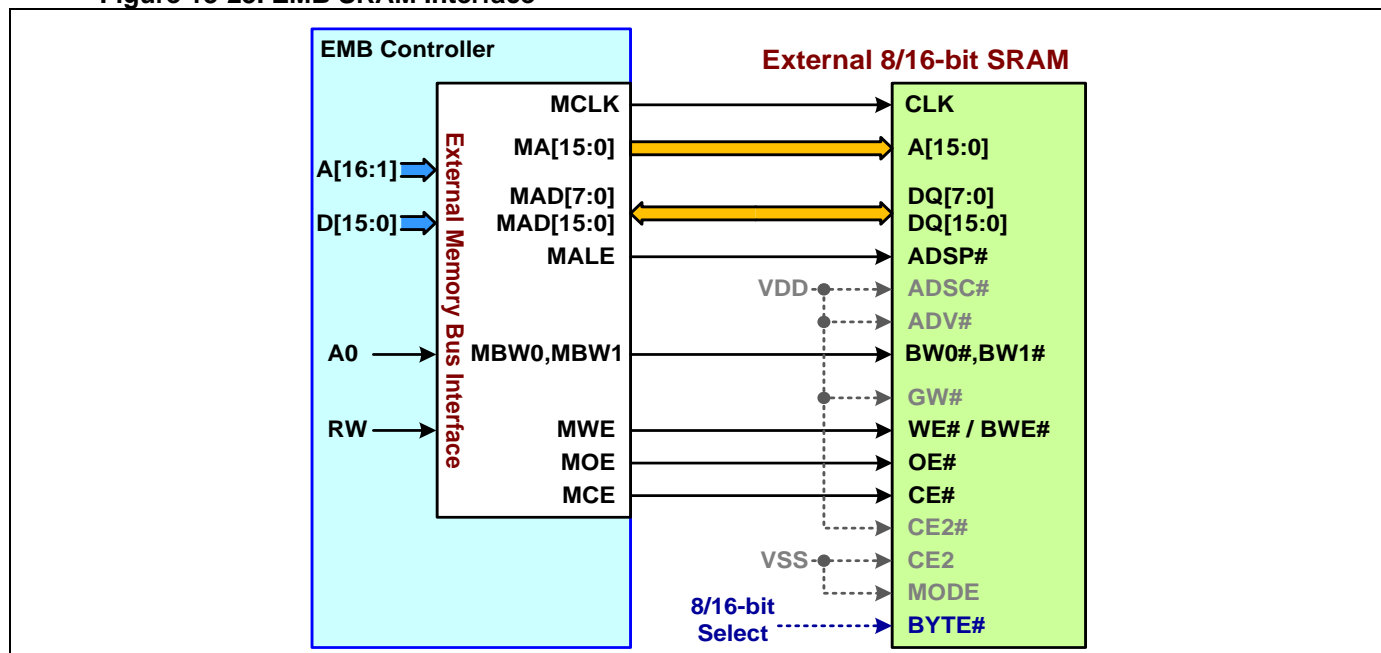
● SRAM Interface

Generally, the SRAM data bus width is 8-bit, 16-bit, 32-bit or large. The EMB can support both synchronous and asynchronous SRAM but support 8-bit and 16-bit data bus only. When connects to asynchronous SRAM, the **MCLK** is not necessary for data transactions.

Normally the SRAM device interface is including of address lines A[n:0], data lines DQ[15:0], write enable signal WE# or byte write enable signal BWE#, byte write select signals BW0# / BW1#, output enable signal OE#, chip enable signal CE# , address strobe signal ADSP# and other control signals. Additional one clock signal CLK is necessary for synchronous SRAM. For some types of SRAM, it is supported 8-bit or 16-bit data access option. There is one signal BYTE# is implemented to select 8-bit or 16-bit access bus.

The following diagram is showing the suggested connection of SRAM interface.

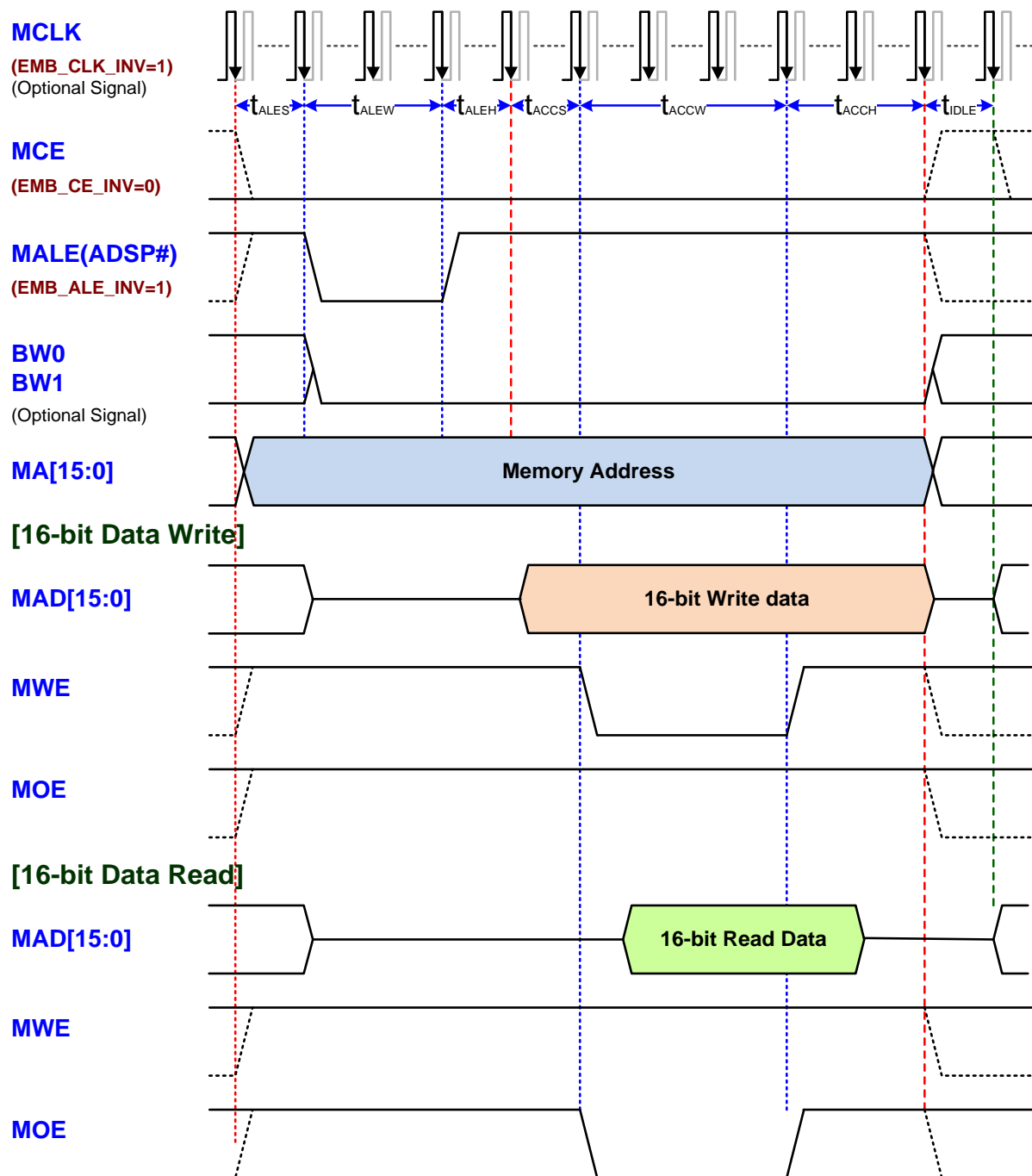
Figure 13-23. EMB SRAM Interface



- **SRAM 16-bit Data Access Timing**

User can set 16-bit EMB data width by setting **EMB_BUS_DSIZE** register for 16-bit SRAM.
The following diagram is showing the 16-bit data access timing on 16-bit SRAM interface.

Figure 13-24. EMB SRAM 16-bit Data Timing



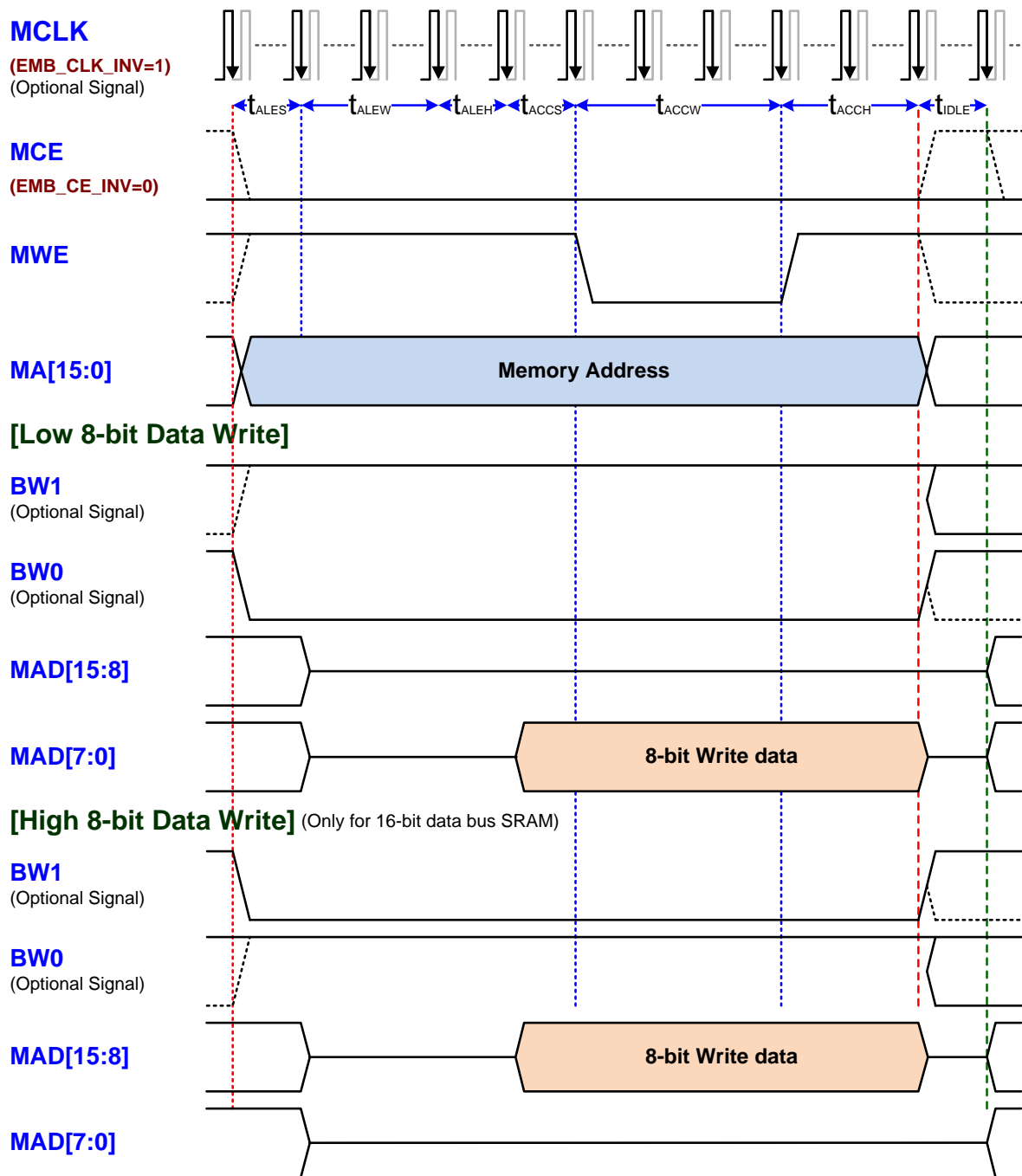
<Note-1> : MCLK is optional signal for synchronous SRAM.

<Note-2> : BW0/BW1 are optional signals for 16-bit data bus SRAM with byte write capability.

- **SRAM 8-bit Data Access Timing**

User can set 16-bit EMB data width by setting **EMB_BUS_DSIZE** register for 16-bit SRAM.
The following diagram is showing the 8-bit data access timing on 16-bit SRAM interface.

Figure 13-25. EMB SRAM 8-bit Data Write Timing



<Note-1> : BW0/BW1 are optional signals for 16-bit data bus SRAM with byte write capability.

<Note-2> : MAD[15:8] are optional signals for 16-bit data bus SRAM.

13.11.2. NOR-Flash Interface and Access Timing

● NOR-Flash Interface

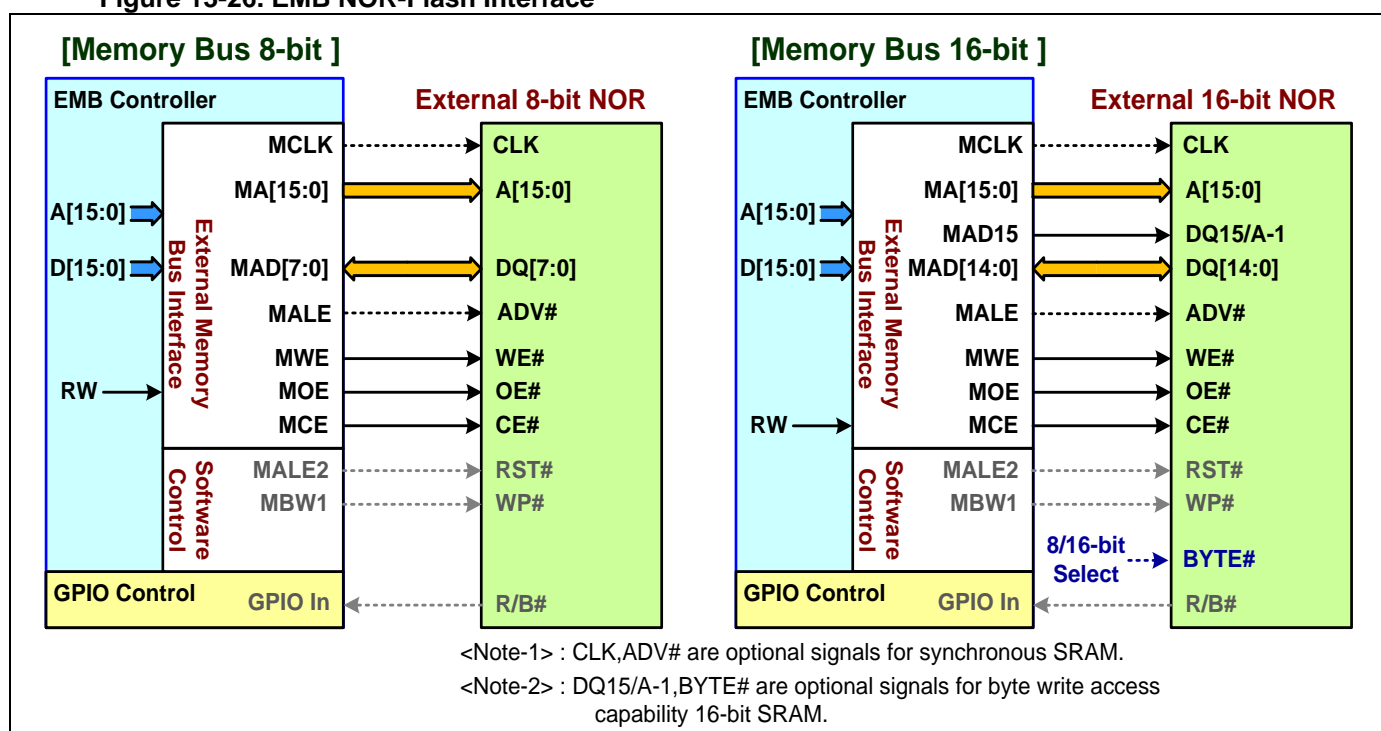
The EMB can support both synchronous and asynchronous NOR-flash. Also the EMB can support both 8-bit and 16-bit data bus for the NOR-flash. When connects to the asynchronous NOR-flash, the **MCLK** is not necessary for data transactions. When connects to the 16-bit NOR-flash with byte access, the **MBW0** signal is necessary to connect to the **BYTE#** pin of the NOR-flash.

Normally the NOR-flash interface is including of address lines A[n:0], data lines DQ[7:0] or DQ[15:0], write enable signal WE#, output enable signal OE#, chip enable signal CE#, address valid signal ADV#, hardware reset input signal RST#, hardware write protect signal WP#, and other control signals. Additional one clock signal CLK is necessary for synchronous NOR-flash. For some types of NOR-flash, it is supported 8-bit or 16-bit data access option. There is one signal **BYTE#** is implemented to select 8-bit or 16-bit access bus.

The RST# and WP# signals are able to control from **MALE2** and **MBW1** outputs in software mode control by setting **EMB_ALE2_SWEN** and **EMB_BW1_SWEN** registers if the two signals are requested in application.

The following diagram is showing the suggested connection of NOR-flash interface.

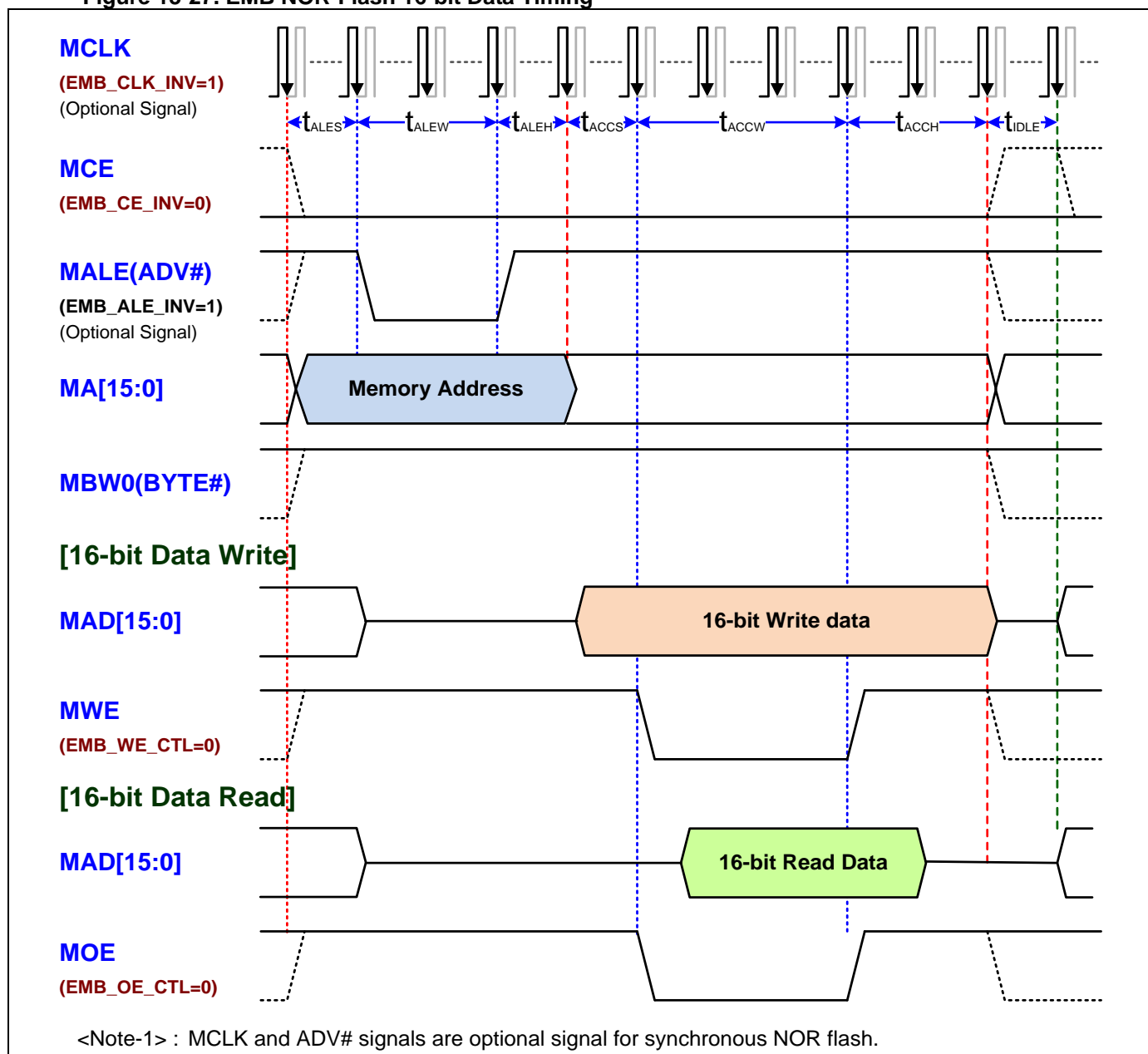
Figure 13-26. EMB NOR-Flash Interface



- **NOR-Flash 16-bit Data Access Timing**

User can set 16-bit EMB data width by setting **EMB_BUS_DSIZE** register for 16-bit NOR-flash.
The following diagram is showing the 16-bit data access timing on 16-bit NOR-flash.

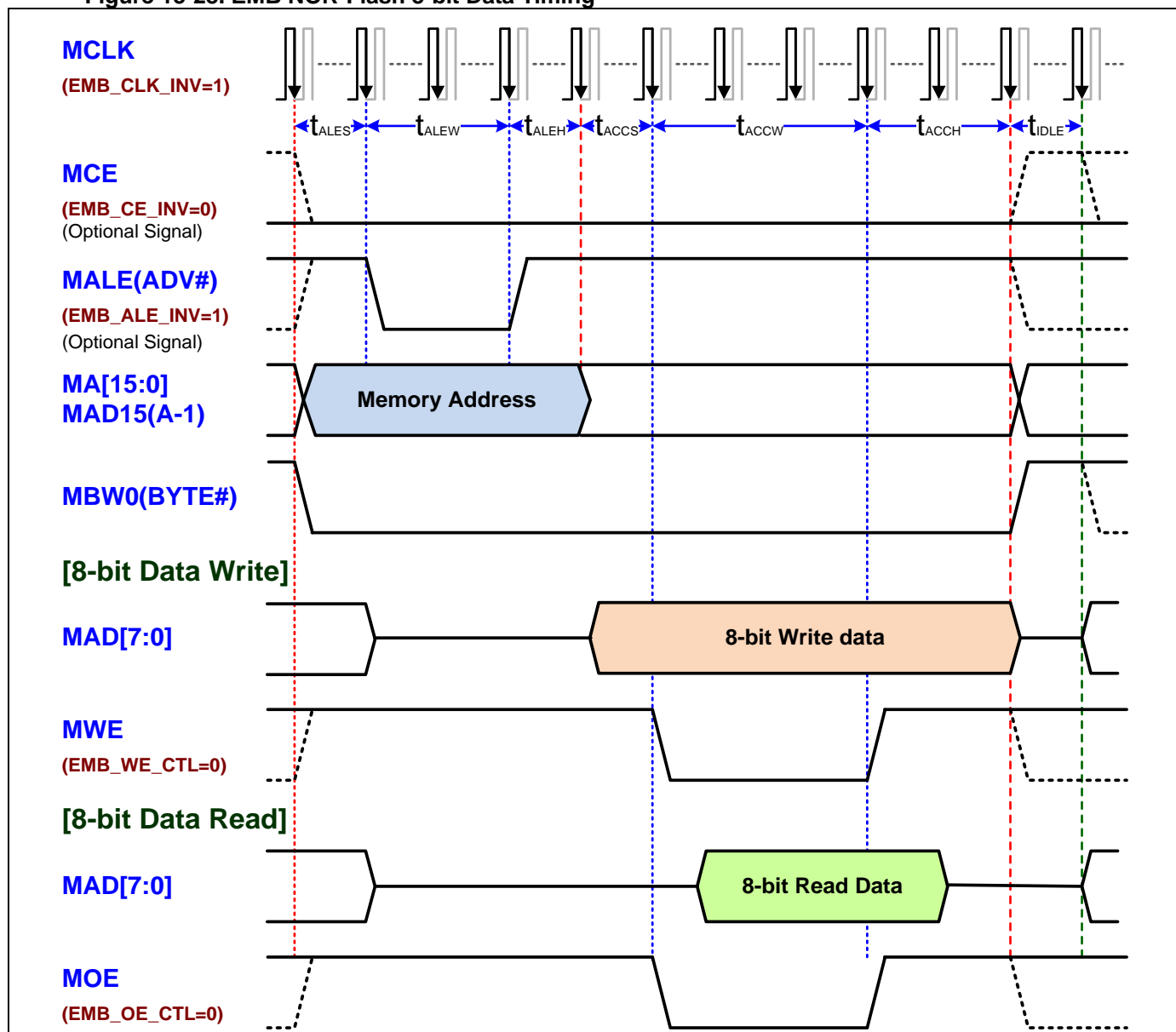
Figure 13-27. EMB NOR-Flash 16-bit Data Timing



- **NOR-Flash 8-bit Data Access Timing**

User can set 8-bit EMB data width by setting **EMB_BUS_DSIZE** register for 8-bit NOR-flash.
The following diagram is showing the 8-bit data access timing on 8-bit NOR-flash.

Figure 13-28. EMB NOR-Flash 8-bit Data Timing



13.11.3. NAND-Flash Interface and Access Timing

● NAND-Flash Interface

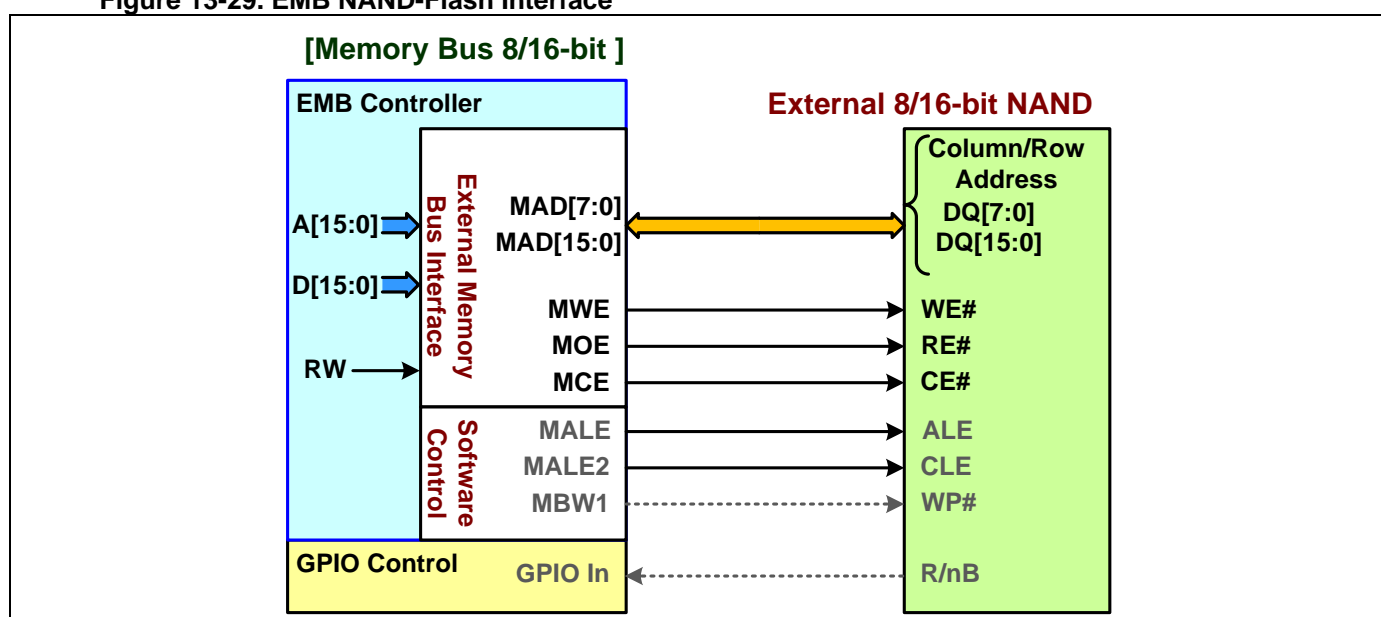
Generally, the NAND-flash is only supported asynchronous data transactions. The EMB can support both 8-bit and 16-bit data bus for the NAND-flash.

Normally the NAND-flash interface is including of multiplexed address/data lines DQ[7:0] or DQ[15:0], write enable signal WE#, read enable signal RE#, chip enable signal CE#, address latch enable signal ALE, command latch enable signal CLE, hardware write protect signal WP# and other control signals. Specially, one open drain Ready/Busy output signal R/nB is implemented to notify the state of NAND-flash. The MCU chip can get the Ready/Busy status and input the signal to any valid GPIO pin.

The ALE, CLE and WP# signals are able to control from **MALE**, **MALE2** and **MBW1** outputs in software mode control by setting **EMB_ALE_SWEN**, **EMB_ALE2_SWEN** and **EMB_BW1_SWEN** registers if these signals are requested in application.

The following diagram is showing the suggested connection of NAND-flash interface.

Figure 13-29. EMB NAND-Flash Interface

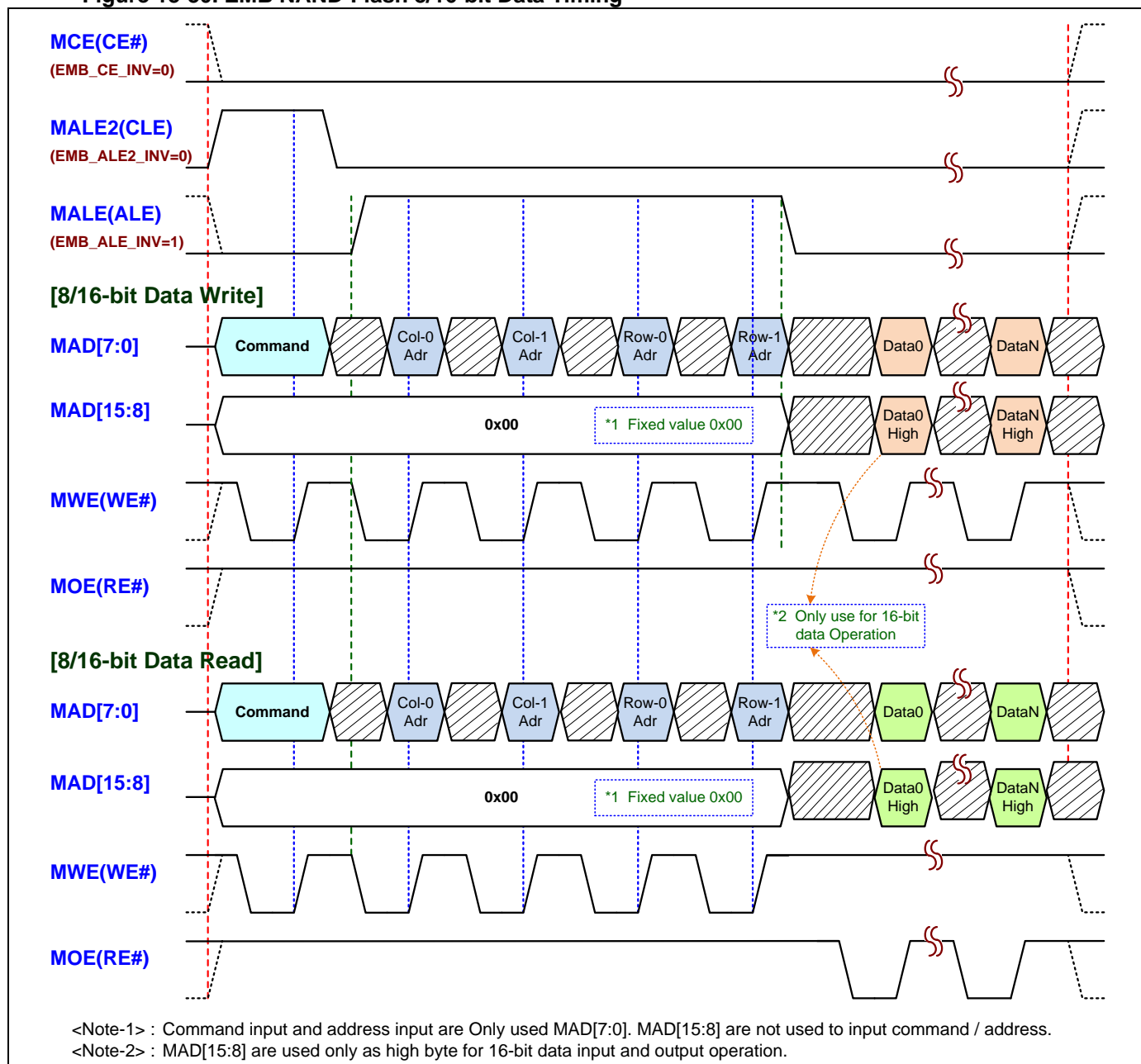


● NAND-Flash Data Access Timing

User can set 8-bit or 16-bit EMB data width by setting **EMB_BUS_DSIZE** register for 8-bit or 16-bit NAND-flash.

The following diagram is showing the data access timing of NAND-flash interface.

Figure 13-30. EMB NAND-Flash 8/16-bit Data Timing



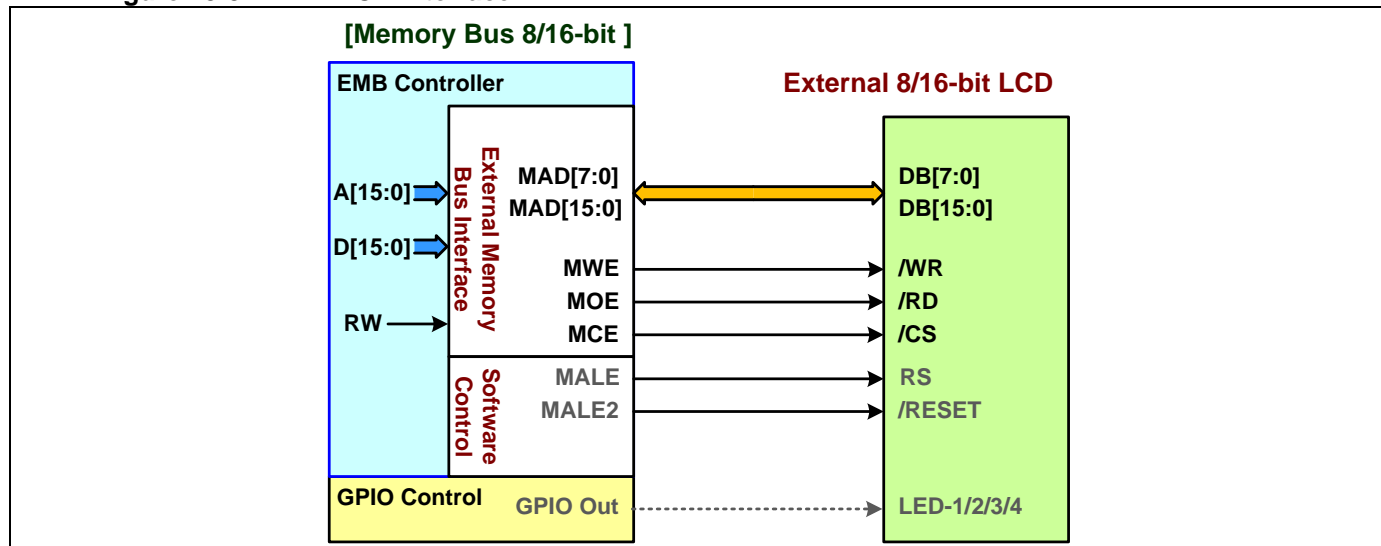
13.11.4. 8080 LCD Interface and Access Timing

- LCD Interface

The EMB can support both 8-bit and 16-bit data bus for the 8080 LCD interface. The following diagram is showing the suggested connection of LCD interface.

Normally the 8080 LCD interface is including of multiplexed address/data lines DB[7:0] or DB[15:0], write strobe signal /WR, read strobe signal /RD, chip select signal /CS, register select signal RS, hardware reset input signal /RESET and other control signals.

Figure 13-31. EMB LCD Interface

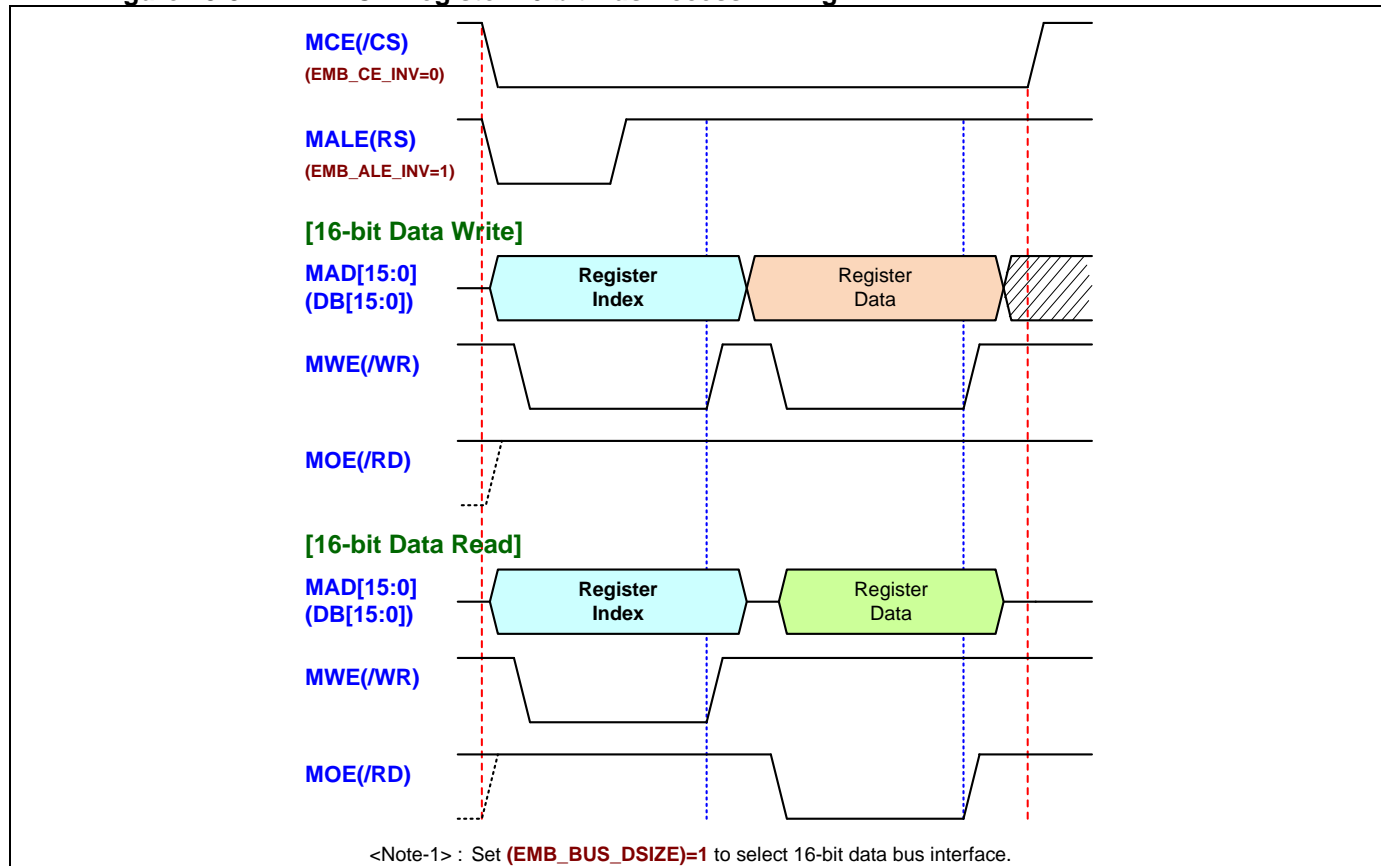


- LCD 16-bit Register Access Timing

User can set 16-bit EMB data width by setting **EMB_BUS_DSIZE** register for 16-bit LCD interface.

The following diagram is showing the 16-bit register access timing of LCD interface.

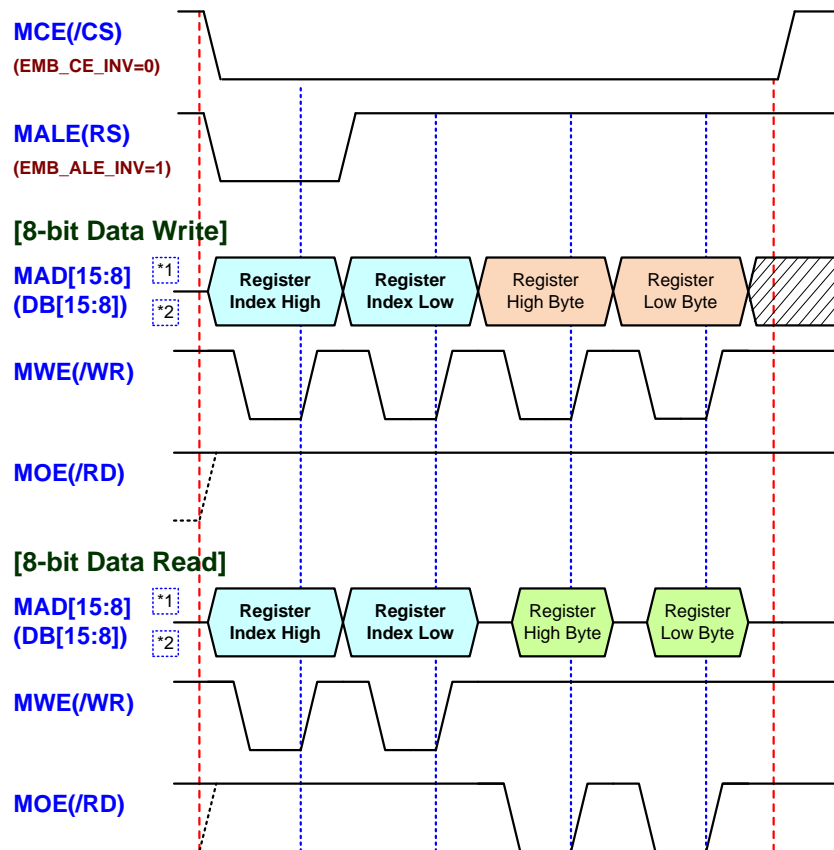
Figure 13-32. EMB LCD Register 16-bit Bus Access Timing



- **LCD 8-bit Register Access Timing**

User can set 8-bit EMB data width by setting **EMB_BUS_DSIZE** register for 8-bit LCD interface.
The following diagram is showing the 8-bit register access timing of LCD interface.

Figure 13-33. EMB LCD Register 8-bit Bus Access Timing



<Note-1> : Set (**EMB_BUS_DSIZE**)=0 to select 8-bit data bus interface.

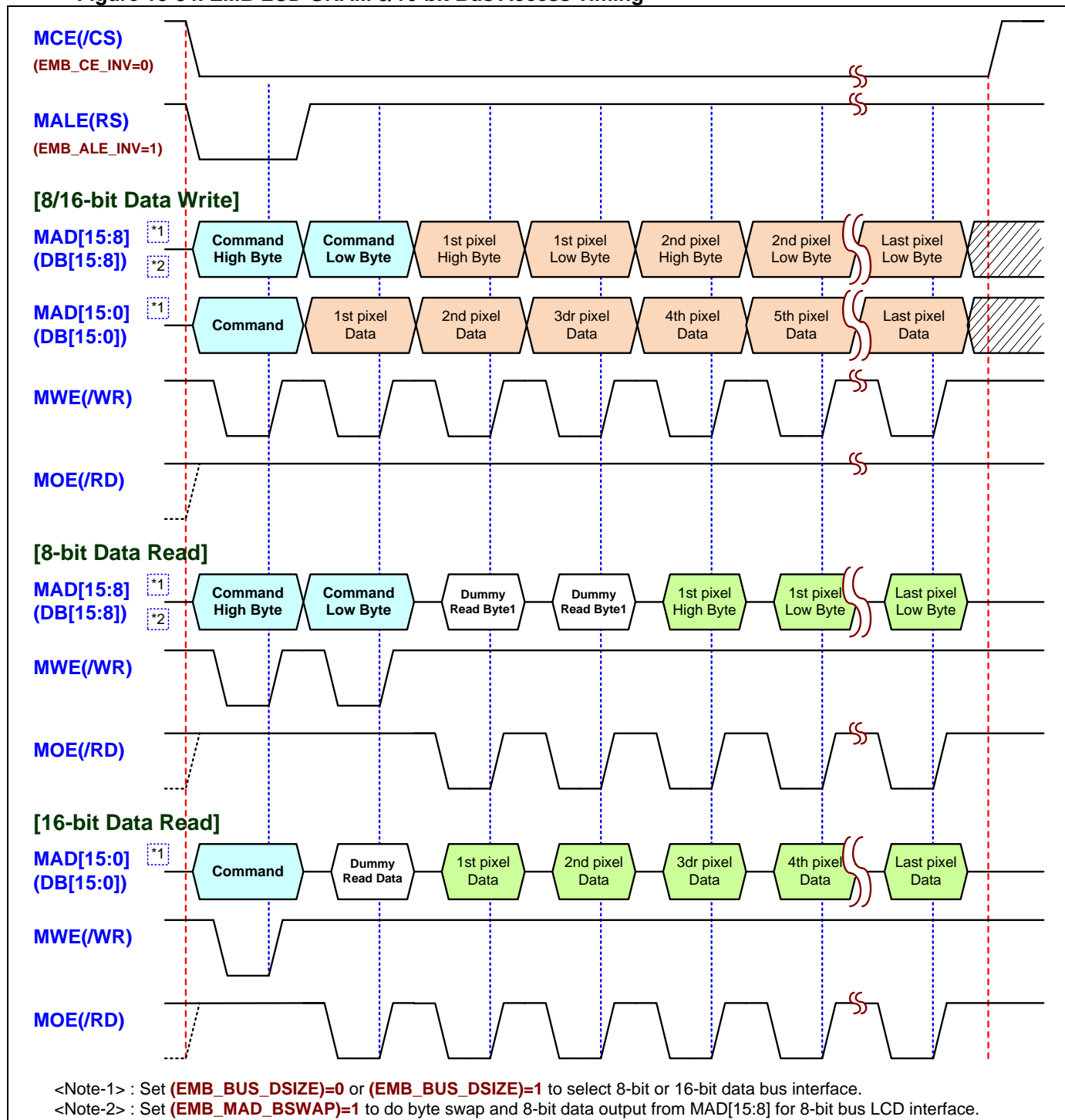
<Note-2> : Set (**EMB_MAD_BSWAP**)=1 to do byte swap and 8-bit data output from MAD[15:8] for 8-bit bus LCD interface.

● LCD GRAM Access Timing

User can set 8-bit or 16-bit EMB data width by setting **EMB_BUS_DSIZE** register for 8-bit or 16-bit LCD interface.

The following diagram is showing the GRAM access timing of LCD interface.

Figure 13-34. EMB LCD GRAM 8/16-bit Bus Access Timing



13.12. EMB DMA Operation

13.12.1. DMA Module Configure

When the chip supports a DMA (direct memory access) controller, user can configure the DMA setting of transferred source/destination devices, channel request arbitration and others in the DMA module before a DMA data transaction. The DMA source and the destination can be memory or peripheral.

Refer the DMA chapter for more detail information about the DMA module configuration.

13.12.2. EMB DMA Control

After DMA configuration is finished, user needs to set the EMB module DMA enable bit of **EMB_DMA_EN**.

Finally, the related channel request start bit of **DMA_CHn_REQ** is necessary to be set to start the DMA transaction (n = DMA channel index). Then the transferred source/destination devices will assert the RX/TX request signal to DMA controller and the DMA controller will assert the acknowledge signal to the request source/destination devices. At the time, the data transferred connection is built for DMA transaction.

13.12.3. EMB Interrupt Flag Control during DMA

During DMA operation cycle, the DMA function will control to be disabled when one of the module's WPEF/BWEF/IAEF interrupt flags has asserted. At the time, hardware will disable the **EMB_DMA_EN** bit in this condition.

Table 13-6. EMB Interrupt Flag Control for DMA Function

Action	Mask the Flag during DMA Process	DMA Disable after the Flag asserted (*1)	Normal Control
Peripheral	(Data flow flags)	(Error/Detect flags)	(Other flags)
EMB		EMB_WPEF EMB_BWEF	

<Note> *1 : When the flag is asserted, it will not force peripheral DMA disabled if the related interrupt enable bit is not enabled.

13.13. EMB Application Note

13.13.1. Pin Suggestion for EMB Signal

The following table is showing the suggested AFS pin plan to map the EMB external device signals. It is useful to plan an application circuit for easy PCB design.

The EMB external devices are including of SRAM, NOR/NAND-flashes, LCD modules and LCD RAM devices. The item of LCD RAM (SPI wire-OR) is a special application which there are this MCU chip, LCD RAM device and SPI flash(s) with data bus wire-OR together. Refer the section of "EMB and SPI Flash to LCD Transaction" for more information about the application.

The **PB[3:0]** pins are reserved for one set of I2C and one set of UART. The **PC[5:4]** pins are reserved for SWD debug signals. The **PC6** pin is reserved for MCU chip external reset. The **PC[14:13]** pins are reserved for MCU external Xtal pins. Also these pins can be to do as GPIO pins by user.

Table 13-7. EMB Interface Signal and Suggestion Pin Table

Register MCU Pin	SRAM		NOR		NAND		LCD RAM-1		LCD RAM-2	
	(EMB_MAD_SWAP) =Disable		(EMB_MAD_SWAP) =Disable		(EMB_MAD_SWAP) =Disable		(EMB_MAD_SWAP) =Enable		(EMB_MAD_SWAP) =Enable	
	(EMB_MAD_BSWAP) =Disable		(EMB_MAD_BSWAP) =Disable		(EMB_MAD_BSWAP) =Disable		(EMB_MAD_BSWAP) =Enable		(EMB_MAD_BSWAP) =Enable	
	EMB Signal	SRAM Pin	EMB Signal	NOR Pin	EMB Signal	NAND Pin	EMB Signal	LCD RAM Pin	EMB Signal	LCD RAM Pin
PA[15:0]	MA[15:0]	A[15:0]	MA[15:0]	A[15:0]						
PE0										BL_LED1 (*2)
PE1										BL_LED2 (*2)
PE2										BL_LED3 (*2)
PE3	MALE2	ADSP# (*4)	MALE2	ADV# (*4)						BL_LED4 (*2)
PB4	MALE	ADSP#	MALE	ADV#	MALE	ALE (*1)	MALE	RESET (*1)	MAD8	DB7
PB5	MOE	OE#	MOE	OE#	MOE	RE#			MAD9	DB6
PB6	MWE	WE#	MWE	WE#	MWE	WE#			MAD10	DB5

PB7	MCE	CE#	MCE	CE#	MCE	CE#	MCE	/CS (*1)	MALE2	RESET (*1)
PB8	MAD0	DQ0	MAD0	DQ0	MAD0	DQ0	MAD0	DB15	MAD0	DB15
PB9	MAD1	DQ1	MAD8	DQ8	MAD8	DQ8	MAD1	DB14	MAD1	DB14
PB10	MAD2	DQ2	MAD1	DQ1	MAD1	DQ1	MAD2	DB13	MAD2	DB13
PB11	MAD3	DQ3	MAD9	DQ9	MAD9	DQ9	MAD3	DB12	MAD3	DB12
PB12	MAD4	DQ4	MAD2	DQ2	MAD2	DQ2	MAD4	DB11	MAD4	DB11
PB13	MAD5	DQ5	MAD10	DQ10	MAD10	DQ10	MAD5	DB10	MAD5	DB10
PB14	MAD6	DQ6	MAD3	DQ3	MAD3	DQ3	MAD6	DB9	MAD6	DB9
PB15	MAD7	DQ7	MAD11	DQ11	MAD11	DQ11	MAD7	DB8	MAD7	DB8
PE8									MAD11	DB4
PE9							MOE	/RD	MOE	/RD
PC0							MWE	/WR	MWE	/WR
PC1	MAD8	DQ8	MAD4	DQ4	MAD4	DQ4	MAD8	DB7		
PC2	MAD9	DQ9	MAD12	DQ12	MAD12	DQ12	MAD9	DB6		
PC3	MAD10	DQ10	MAD5	DQ5	MAD5	DQ5	MAD10	DB5		
PC4	SWCLK									
PC5	SWDIO									
PC6	GPIO/RSTN						MBW1	RS (*1)	MALE	RS (*1)
PC7									MCE	/CS (*1)
PC8	MAD11	DQ11	MAD13	DQ13	MAD13	DQ13	MAD11	DB4		
PC9	MAD12	DQ12	MAD6	DQ6	MAD6	DQ6	MAD12	DB3	MAD12	DB3
PC10	MAD13	DQ13	MAD14	DQ14	MAD14	DQ14	MAD13	DB2	MAD13	DB2
PC11	MAD14	DQ14	MAD7	DQ7	MAD7	DQ7	MAD14	DB1	MAD14	DB1
PC12	MAD15	DQ15	MAD15	DQ15/A-1	MAD15	DQ15/A-1	MAD15	DB0	MAD15	DB0
PC13	XIN									
PC14	XOUT									
PE12	MBW0	BW0	MBW0	BYTE#						
PE13	MBW1	BW1	MBW1	WP# (*1)	MBW1	WP# (*1)				
PE14			MALE2	RST# (*1)	MALE2	CLE (*1)				
PE15				RY/BY# (*3)		RY/BY# (*3)				
PD0	MCLK	CLK	MCLK	CLK						

<Note> *1 : Output control directly by software mode register setting

*2 : Output from GPIO output pin

*3 : Input to GPIO input pin

*4 : Option Pin for ADSP# or ADV#

13.13.2. EMB and SPI Flash to LCD Transaction

This application is implemented to transfer SPI flash data to MCU chip and LCD RAM. Also the MCU chip can transfer data to LCD RAM. There are the MCU chip, 8-bit LCD RAM device and SPI flash(s) with data bus wire-OR connection. The SPI flash(s) can be one OSPI flash with 8 data lines or two QSPI flash with 4 data lines for each QSPI flash. Refer the section of “[Pin Suggestion for EMB Signal](#)” for more information about the application.

User must initialize the SPI flash(s) through SPI0 module and SPI pin configuration. Also user must initialize the LCD RAM through EMB module and EMB pin configuration. Then there are three data transferred paths.

- **SPI Flash to MCU**

MCU is inactive the LCD CS# signal and is active the NSS signal to enable the SPI flash(s). Then MCU outputs SPI clock and asserts read command to gets data from the SPI flash(s).

- **MCU to LCD**

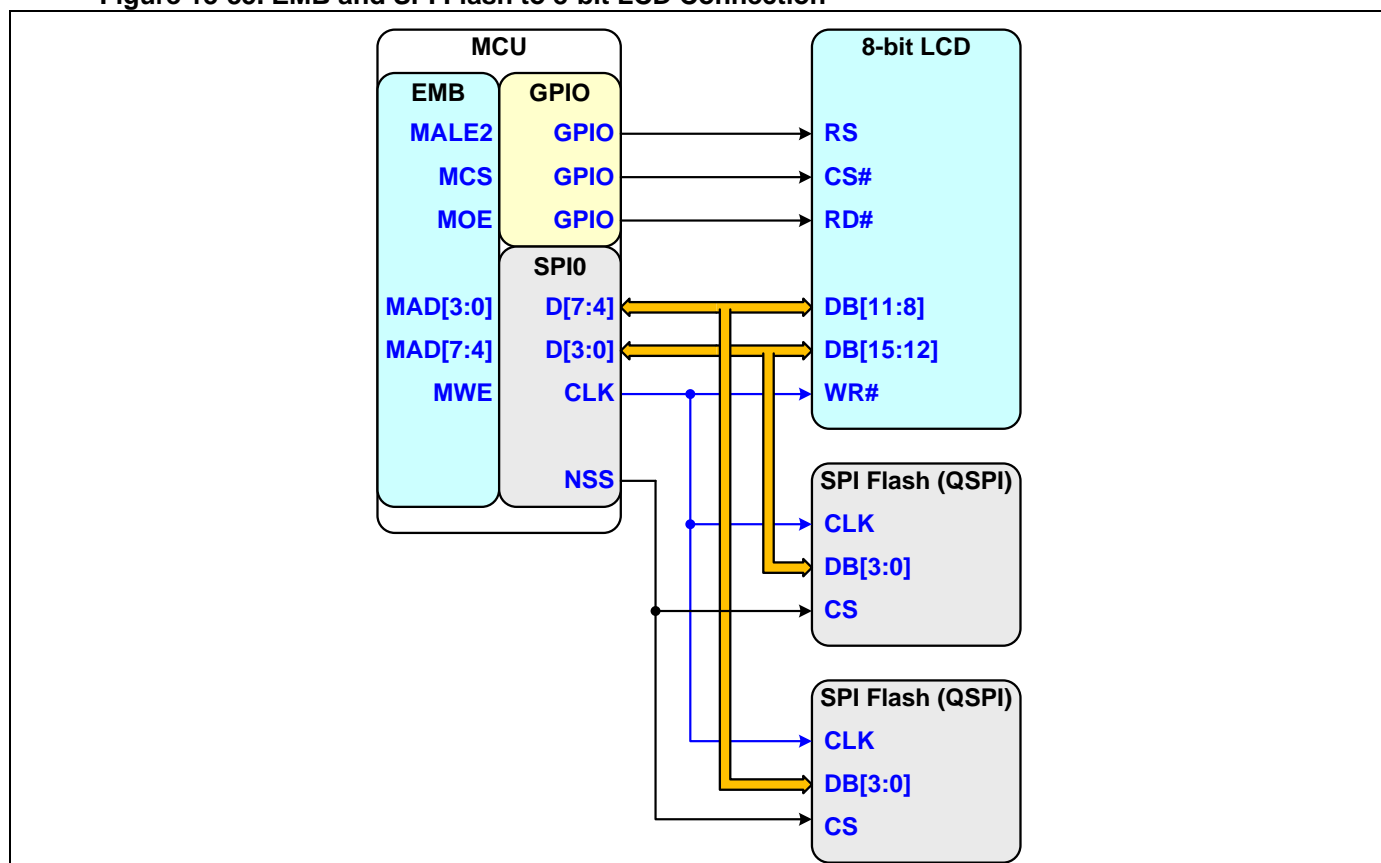
MCU is inactive the SPI flash(s) CS signal(s) and is active the LCD CS# signal to enable the LCD device. Then MCU send data to the LCD device by asserted WR# command.

- **SPI Flash to LCD**

MCU is active both the SPI flash(s) CS signal(s) and the LCD CS# signal to enable the SPI flash(s) and LCD device. And MCU must set D[7:0] or MAD[7:0] pins to input mode. Then MCU can output clock to both SPI CLK pin(s) and LCD WR# pin. Following, MCU can assert the SPI read command to trigger that SPI data directly outputs to LCD device through 8-bit LCD interface.

The following diagram is showing the system connection for “EMB and SPI Flash to 8-bit LCD”.

Figure 13-35. EMB and SPI Flash to 8-bit LCD Connection



14. APB (APB Common Control)

14.1. Introduction

ON	SLEEP	STOP
----	-------	------

The module can be running in ON and SLEEP modes only.

The chip builds in one APB (APB bus common control) module for the common control of APB devices.

Notify: The sign of (x = module index; n = OBM set index; m=OBM channel index) is using for Registers, Signals and Pins/Ports in the descriptions of this chapter.

14.2. Features

- Timer synchronous enable global control for TMx timer modules
- Timer internal trigger/clock source selection for TMx timer modules
- Infrared Remote Modulation Output
- OBM(Output Signal Break and Modulation) control
 - Support max. two sets of OBM for output signal break and modulation control
- NCO(Numerically Controlled Oscillator) output with FDC and PF modes

14.3. Implementation

14.3.1. Chip Implementation

The following table is showing the implemented APB functions of chips.

Table 14-1. APB Implementation

Chip	APB Module Sub-Functions					
	OBM0	OBM1	Timer Global Control	IR Modulation Output	NCO0	Multi-Function MUX
MG32F02A128/A064 MG32F02U128/U064 MG32F02V032	V	V	V	V	V	-
MG32F02N128/N064 MG32F02K128/K064	V	V	V	V	V	V

<Note> V: Implemented

14.4. Interrupt and Event

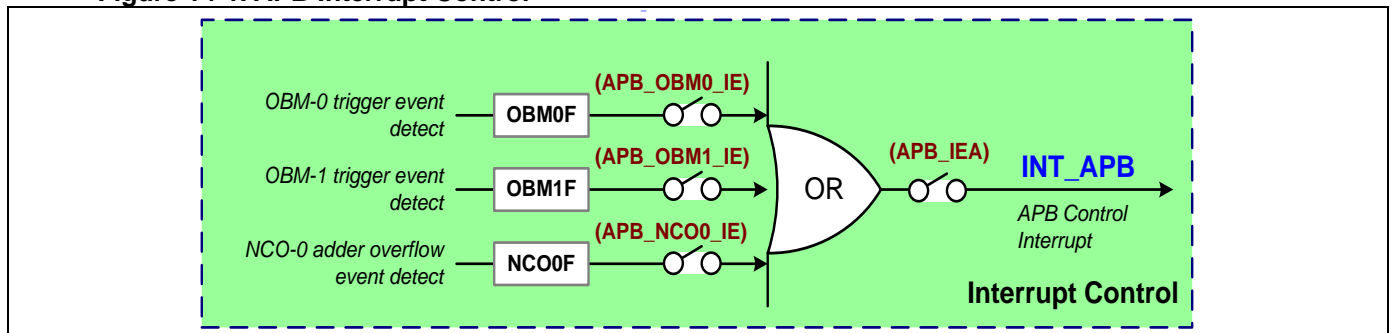
There is one signal of **INT_APB** to be generated in this APB module. **INT_APB** sends to External Interrupt Controller (EXIC) to do as an interrupt event.

14.4.1. APB Interrupt Control and Status

These interrupt flags are using for the interrupt service routine (ISR) flow control. Generally these flags are set by hardware and software clears them when the serviced job is completed in the related ISR. Each interrupt flag has one interrupt enable bit. User can enable or disable it. There is one interrupt all enable bit of **APB_IEA** to enable or disable all the interrupt sources for this module.

There are some status bits those are reading only to provide internal control status. Refer the register descriptions of the related status bits for more information.

Figure 14-1. APB Interrupt Control



14.4.2. APB Interrupt Flags

Generally these interrupt flags are set by hardware and clear by software writing 1. Refer the register descriptions for more information about the related interrupt flags and interrupt enable bit.

- **OBM0F / OBM1F**

OBM-0/1 break trigger event detect flag (**APB_OBM0F**, **APB_OBM1F**). There are the related interrupt enable register bits of **APB_OBM0_IE** and **APB_OBM1_IE**.

- **NCO0F**

NCO-0 adder overflow event detect interrupt flag (**APB_NCO0F**). There is the related interrupt enable register bit of **APB_NCO0_IE**.

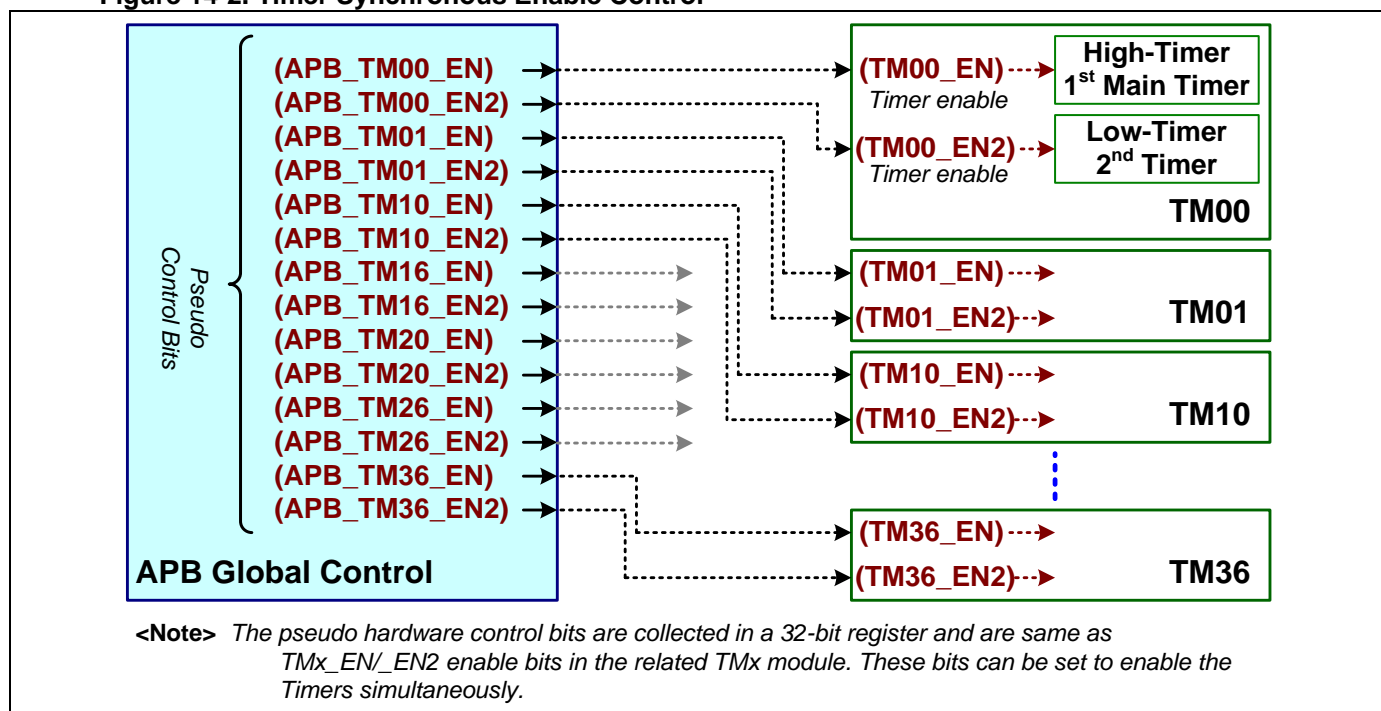
14.5. Timer Common Control

14.5.1. Timer Synchronous Enable Control

The timer builds in the TMx modules which supports three timer operation modes: (1) Cascade mode (2) Separate mode (3) Full-Counter mode. So the timer of one TMx module can be a full-counter timer or separated to two timers. There are two timer enable control bits of **TMx_EN** and **TMx_EN2** for each TMx module. The **TMx_EN** bit is used to enable the full-counter timer, High timer or Main timer for these timer operation modes. The **TMx_EN2** bit is used to enable the Low timer or 2nd timer for Cascade or Separate operation mode.

In APB module, there are the same control bits of **APB_TMx_EN** and **APB_TMx_EN2** as the timer enable bits of **TMx_EN** and **TMx_EN2**. These control register bits are built in one 32-bit register by design. However, all the full-counter timers or separated timers of all TMx modules can be enabled or disabled synchronously by setting **APB_TMx_EN** or **APB_TMx_EN2** registers for firmware easy control. (x = Timer module index)

Figure 14-2. Timer Synchronous Enable Control



[Notify]: The **APB_TM26_EN** and **APB_TM26_EN2** are not supported for MG32F02V032.

14.5.2. Timer Common Trigger/Clock Source Select

The **ITR6** and the **ITR7** are used as the common signals of trigger event or clock signal for all timer modules. User can select the trigger source signal by setting **APB_ITR6_MUX** and **APB_ITR7_MUX** registers. The **APB_ITR6** and **APB_ITR7** signals can be selected from others TMx timer, URTx, ADC0, RTC modules and EXIC global interrupt events as showing in following table.

Table 14-2. Timer Common ITR6/ITR7 Signals Table

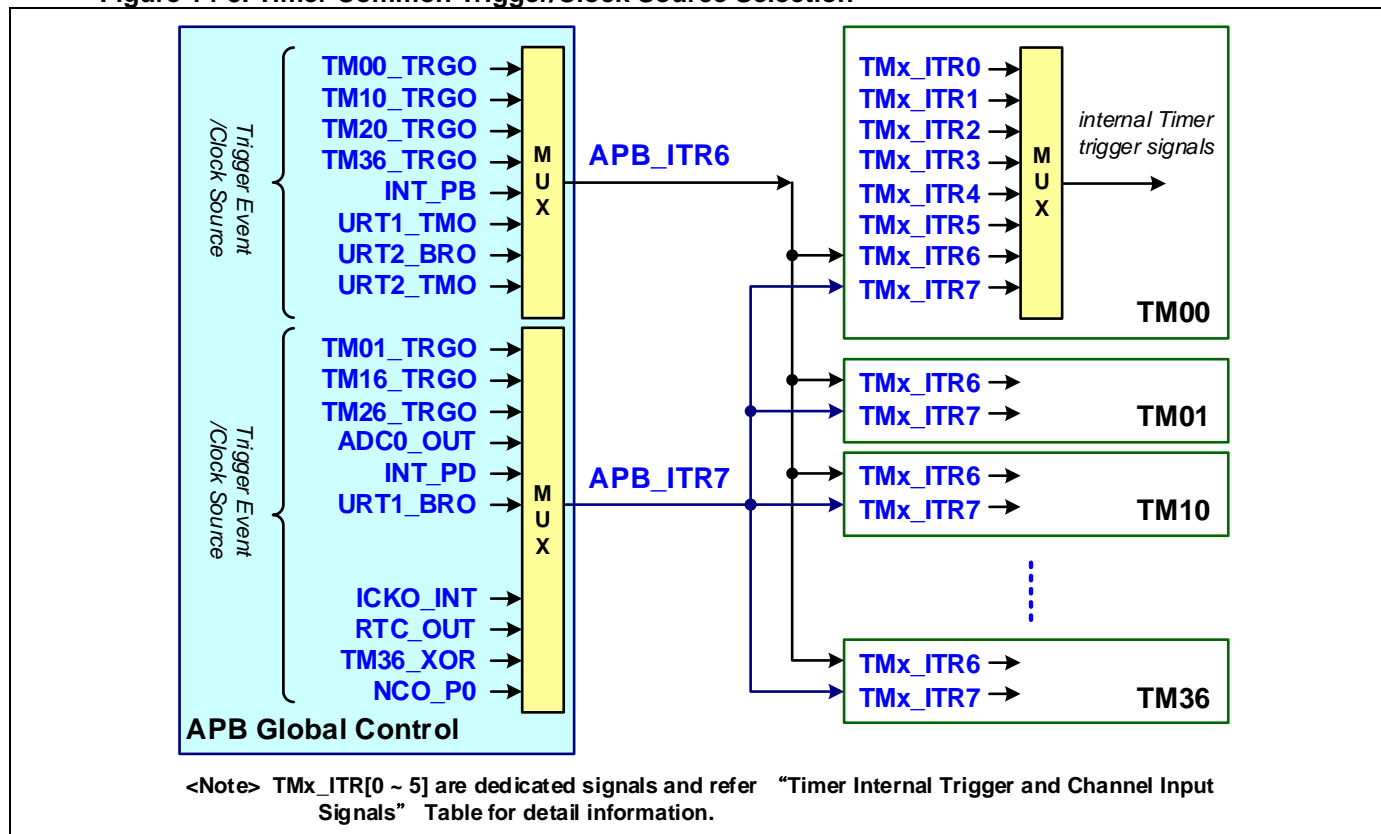
Chip	MG32F02A128/U128/A064/U064 MG32F02K128/K064/N128/N064		MG32F02V032	
ITRx	ITR6	ITR7	ITR6	ITR7
	APB Register		APB Register	
Trigger Source	ITR6_MUX	ITR7_MUX	ITR6_MUX	ITR7_MUX
TRG0 Signal	TM00_TRGO	TM01_TRGO	TM00_TRGO	TM01_TRGO
TRG1 Signal	TM10_TRGO	TM16_TRGO	TM10_TRGO	TM16_TRGO
TRG2 Signal	TM20_TRGO	TM26_TRGO	TM20_TRGO	-
TRG3 Signal	TM36_TRGO	ADC0_OUT	TM36_TRGO	ADC0_OUT
TRG4 Signal	INT_PB	INT_PD	INT_PB	INT_PD
TRG5 Signal	URT1_TMO	URT1_BRO	URT1_TMO	URT1_BRO
TRG6 Signal	URT2_BRO	-	-	-

TRG7 Signal	URT2_TMO	-	-	-
TRG8 Signal	x	ICKO_INT	x	ICKO_INT
TRG9 Signal	x	RTC_OUT	x	RTC_OUT
TRG10 Signal	x	TM36_XOR	x	TM36_XOR
TRG11 Signal	x	NCO_P0	x	NCO_P0

<Sign> "-" = reserved, "x" = non-existed bit

The following diagram is showing the timer common trigger/clock source selection. Refer the table of "Timer Common ITR6/ITR7 Signals Table" about the usable signals of ITR6 and ITR7 for used chip.

Figure 14-3. Timer Common Trigger/Clock Source Selection

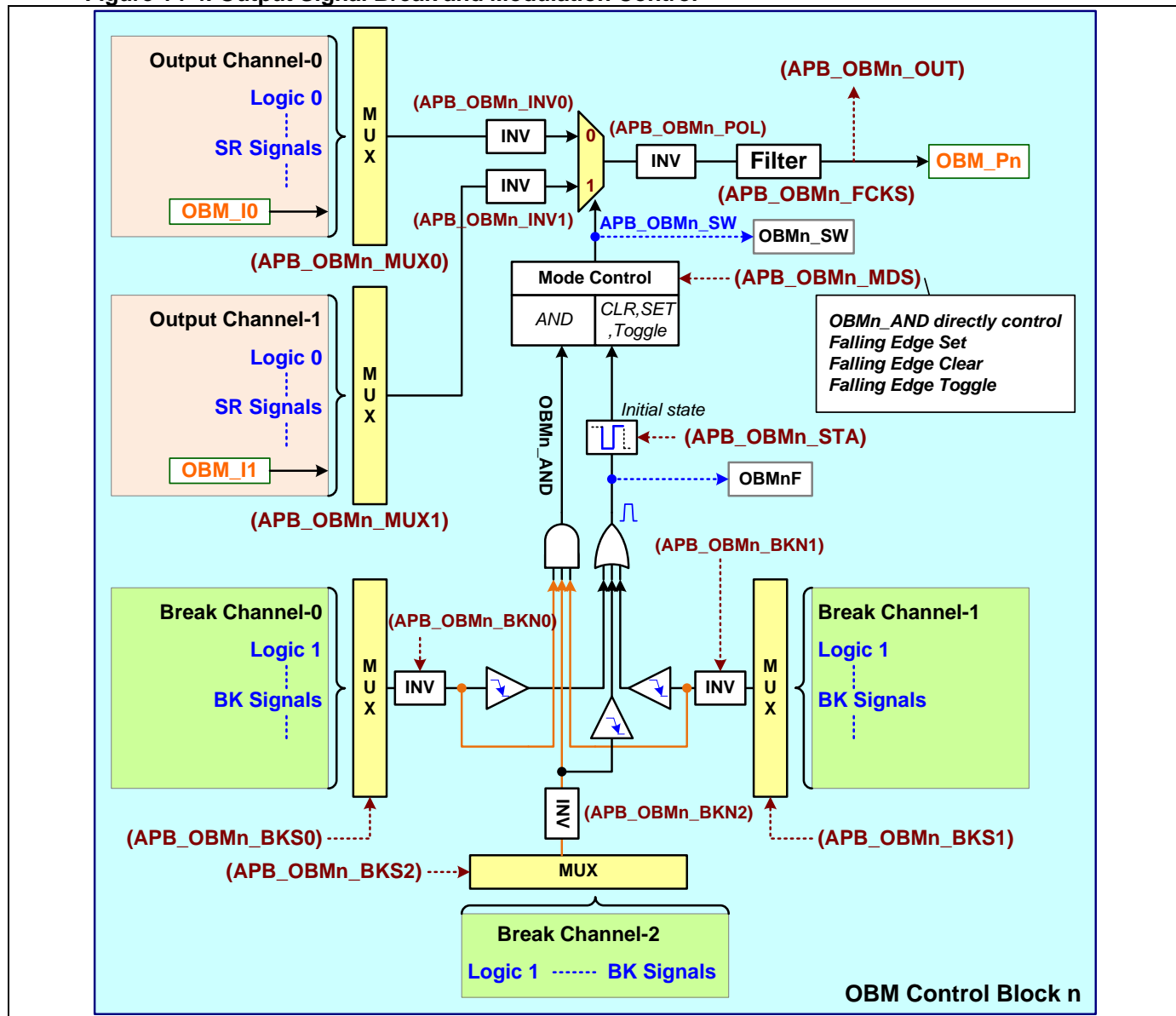


14.6. OBM Control

The APB module is including of two identical sets of output signal break and modulation (OBM) block. The OBM block is used to break one of the output signals of **TMx_CKO**, **APB_ITR6**, and **APB_ITR7** ... or do signal modulation as following block diagram.

There are two OBM output source channels, three OBM break source channels and one OBM operation mode control block for each OBM set.

Figure 14-4. Output Signal Break and Modulation Control



14.6.1. OBM Output Channel

There are two OBM output source channels which can select the source signal to be “Break” or “Modulate” in **APB_OBMn_MUXm** register. There is one signal inversion control bit of **APB_OBMn_INVm** for each output channel. (n = OBM set index; m=OBM channel index)

For the OBM output path, there is one control bit of **APB_OBMn_POL** to adjust the output signal polarity. The OBM is built in one digital deglitch filter for the output signal. User can select the filter clock source from APB clock, APB clock divided by 8 or timer TM00 trigger out (**TM00_TRGO**) by setting **APB_OBMn_FCKS** register.

There is one status bit of **APB_OBMn_OUT** to reflect the real time status of **OBMn** output signal.

The following table is showing the source signal selection by register setting for these two output channel MUX.

Table 14-3. OBM Block Output Channel Signals Table

Chip	MG32F02A128/U128/A064/U064 MG32F02K128/K064/N128/N064		MG32F02V032	
Channel	Channel-0	Channel-1	Channel-0	Channel-1
	APB Register		APB Register	
Output Source	OBMn_MUX0	OBMn_MUX1	OBMn_MUX0	OBMn_MUX1
SR0 Signal	0	0	0	0
SR1 Signal	INT_PA	INT_PB	INT_PA	INT_PB
SR2 Signal	INT_PC	INT_PD	INT_PC	INT_PD
SR3 Signal	INT_PE	-	-	-
SR4 Signal	TM00_CKO	TM01_CKO	TM00_CKO	TM01_CKO
SR5 Signal	TM10_CKO	TM16_CKO	TM10_CKO	TM16_CKO
SR6 Signal	TM20_CKO	TM26_CKO	TM20_CKO	-
SR7 Signal	TM36_CKO	-	TM36_CKO	-
SR8 Signal	TM20_OC00	TM20_OC01	TM20_OC00	TM20_OC01
SR9 Signal	TM36_OC00	TM36_OC01	TM36_OC00	TM36_OC01
SR10 Signal	TM36_OC2	TM36_OC3	TM36_OC2	TM36_OC3
SR11 Signal	-	-	-	-
SR12 Signal	OBM_I0	OBM_I1	OBM_I0	OBM_I1
SR13 Signal	ITR6	ITR7	ITR6	ITR7
SR14 Signal	-	ICKO_INT	-	ICKO_INT
SR15 Signal	-	-	-	-

<Sign> "-" = reserved, "0" = logic-0

14.6.2. OBM Break Channel

There are three OBM break source channels which can select the signals through the OBM operation mode control block to generate the breaking or modulation signal of **APB_OBMn_SW**.

User can select the break signal source in **APB_OBMn_BKS**m register for each OBM break channel. There is one signal inversion control bit of **APB_OBMn_BKN**m for each break channel. (n = OBM set index; m=OBM channel index)

There is one status bit of **APB_OBMn_SW** to reflect the real time status of OBMn break switching signal.

The following table is showing the source signal selection by register setting for these three break channel MUX.

Table 14-4. OBM Block Break Channel Signals Table

Chip	MG32F02A128/U128/A064/U064 MG32F02K128/K064/N128/N064			MG32F02V032		
Channel	Channel-0	Channel-1	Channel-2	Channel-0	Channel-1	Channel-2
	OBMn_BKSz Register			OBMn_BKSz Register		
Break Source	BKS0	BKS1	BKS2	BKS0	BKS1	BKS2
BK0 Signal	1	1	1	1	1	1
BK1 Signal	INT_PA	INT_PB	INT_PE	INT_PA	INT_PB	-
BK2 Signal	INT_PC	INT_PD	-	INT_PC	INT_PD	-
BK3 Signal	ADC0_OUT	INT_BOD1	SPI0_MOSI	ADC0_OUT	INT_BOD1	SPI0_MOSI
BK4 Signal	TM00_TRGO	TM01_TRGO	-	TM00_TRGO	TM01_TRGO	-
BK5 Signal	TM10_TRGO	TM16_TRGO	-	TM10_TRGO	TM16_TRGO	-
BK6 Signal	TM20_TRGO	TM26_TRGO	TM36_TRGO	TM20_TRGO	-	TM36_TRGO
BK7 Signal	CCL_P0	CCL_P1	-	CCL_P0	CCL_P1	-
BK8 Signal	TM20_OC00	TM20_OC10	-	TM20_OC00	TM20_OC10	-
BK9 Signal	TM36_OC2	TM36_OC3	-	TM36_OC2	TM36_OC3	-
BK10 Signal	CMP0_OUT	CMP1_OUT	-	-	-	-
BK11 Signal	-	-	-	-	-	-
BK12 Signal	URT0_TX	URT1_TX	URT2_BRO	URT0_TX	URT1_TX	-
BK13 Signal	URT2_TX	-	URT2_TMO	-	-	-
BK14 Signal	URT0_RX	URT1_RX	-	URT0_RX	URT1_RX	-
BK15 Signal	URT2_RX	-	-	-	-	-

<Note> "-" = reserved, "1" = logic-1 , SPI0_MOSI = SPI output signal

14.6.3. OBM Operation Mode

The OBM mode control block has supported AND, CLR/SET/TOGGLE operation modes. User can select the OBM operation mode in **APB_OBMn_MDS** register to control the generation of Break or Modulation signal of **APB_OBMn_SW**.

When selects AND, the **APB_OBMn_SW** signal is directly controlled by the AND signal of all break channels' output. When selects CLR/SET/TOGGLE, the **APB_OBM0_SW** signal is controlled by STA (**APB_OBMn_STA**) bit and can update by firmware. The **APB_OBMn_STA** bit is used to set the OBMn break switching signal initial state. The bit is written effectively only by written 1 to **APB_OBM0_LCK** simultaneously.

14.7. IR Control

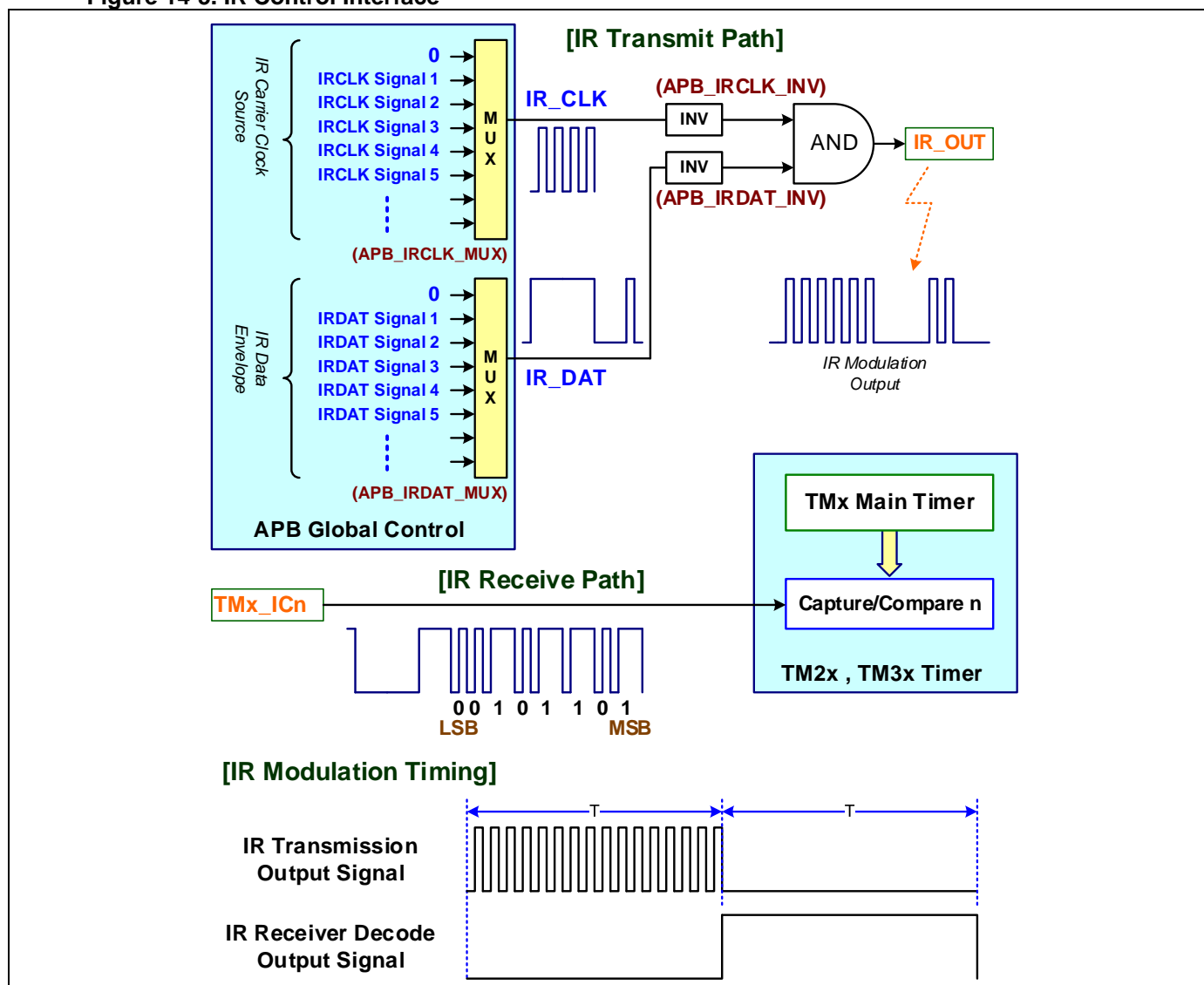
The APB module is including of one IR (Infrared Remote) modulation block. This block is used to do signal modulation for the transmission of IR control.

14.7.1. IR Control Interface

For IR transmission, the IR modulation block can output the modulated signal of IR transmission to **IR_OUT** output. For IR receiving, user can input the IR digital signal from **TMx_ICn** to timer module and use the timer capture function to record the IR information for firmware using.

The following diagram is showing the IR transmission and receiving control Interface. Refer the table of “IR Clock/Data Signals Table” in the section of “[IR Transmission Modulation](#)” about the usable input signals of IR Carrier Clock Source and IR Data Envelope for used chip.

Figure 14-5. IR Control Interface



14.7.2. IR Transmission Modulation

User can select the IR transmission modulated clock signal by setting **APB_IRCLK_MUX** register and modulated data signal by setting **APB_IRDAT_MUX** register. There are two signal inversion control bits of **APB_IRCLK_INV** and **APB_IRDAT_INV** for modulated clock signal and data signal.

The following table is showing the source signal selection by register setting for IR clock/data signals' MUX.

Table 14-5. IR Clock/Data Signals Table

Chip	MG32F02A128/A064 MG32F02U128/U064		MG32F02V032		MG32F02K128/K064 MG32F02N128/N064	
Channel	Carrier Clock	Data Envelope	Carrier Clock	Data Envelope	Carrier Clock	Data Envelope
	APB Register		APB Register		APB Register	
Clock/Data Source	IRCLK_MUX	IRDAT_MUX	IRCLK_MUX	IRDAT_MUX	IRCLK_MUX	IRDAT_MUX
Signal 0	0	0	0	0	0	0
Signal 1	TM00_CKO	TM20_TRGO	TM00_CKO	TM20_TRGO	TM00_CKO	TM20_TRGO
Signal 2	TM01_CKO	TM26_TRGO	TM01_CKO	-	TM01_CKO	TM26_TRGO
Signal 3	TM10_CKO	TM36_TRGO	TM10_CKO	TM36_TRGO	TM10_CKO	TM36_TRGO
Signal 4	TM16_TRGO	SPI0_MOSI	TM16_TRGO	SPI0_MOSI	TM16_TRGO	SPI0_MOSI
Signal 5	URT1_TMO	URT1_TX	URT1_TMO	URT1_TX	URT1_TMO	URT1_TX
Signal 6	URT2_TMO	URT2_TX	-	-	URT2_TMO	URT2_TX
Signal 7	-	-	-	-	-	-
Signal 8	x	x	x	x	NCO_P0	OBM_P0
Signal 9	x	x	x	x	SDT_P0	OBM_P1
Signal 10	x	x	x	x	-	CCL_P0
Signal 11	x	x	x	x	-	CCL_P1
Signal 12	x	x	x	x	TM16_CKO	ADC0_OUT
Signal 13	x	x	x	x	TM20_CKO	RTC_OUT
Signal 14	x	x	x	x	TM26_CKO	-
Signal 15	x	x	x	x	TM36_CKO	-

<Note> "-" = reserved, "0" = logic-0, SPI0_MOSI = SPI output signal, "x" = non-existed bit

14.8. NCO Control

The APB module is including of one Numerically Controlled Oscillator (NCO) block. The NCO block is used to generate a divided frequency with a decimal clock signal from input clock signal as following block diagram. It is useful for the accurate frequency clock requested application.

The features are including follows:

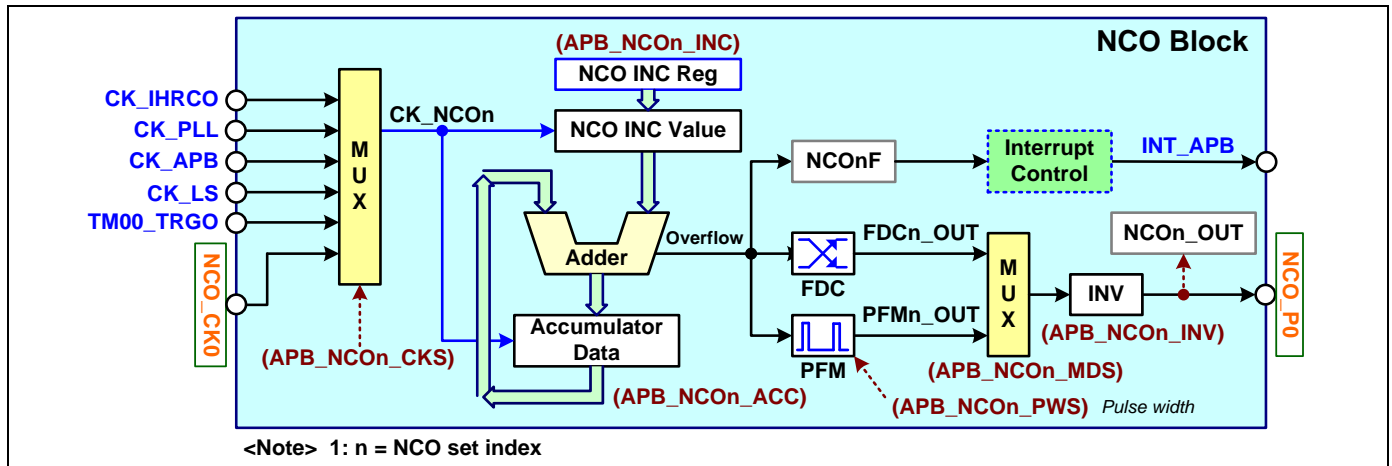
- 20-bit Increment Function
- Fixed Duty Cycle mode (FDC) mode
- Pulse Frequency (PFM) mode
- Output Pulse Width Control
- Multiple Clock Input Sources
- Output Polarity Control
- Interrupt Capability

14.8.1. NCO Block and Clock Input

The NCO block has one block enable bit in **APB_NCO_EN** register. User must enable this bit before start running the NCO block.

The following diagram is showing the NCO control block.

Figure 14-6. NCO Block



The input clock signals can come from external **NCO_CK0** pin input, internal module signal **TM00_TRGO** or internal clock source **CK_IHRCO**, **CK_PLL**, **CK_APB**, **CK_LS**. User can set **APB_NCO_n_CKS** register to select the input clock source.

A 20-bit arithmetic adder is used to accumulate a fixed increment value until the adder is overflowed. Then the adder will add the remained value of accumulator with the fixed increment value in next **CK_NCO0** clock time. The continuous same process acts like as a frequency divider and generates an overflow signal with decimal ratio frequency of **CK_NCO0** input clock.

14.8.2. NCO Clock Output

User can set the fixed increment value by setting **APB_NCO_n_INC** register and get the NCO overflow frequency as following formula.

NCO Overflow Frequency

$$\text{NCO Overflow Frequency} = \frac{\text{CK_NCO} * \text{INC_Value}}{2^{20}}$$

The NCO block supports two output modes, one is fixed duty cycle mode (FDC) and another is pulse frequency mode (PFM). User can set the **APB_NCO_n_MDS** register to choice the NCO output mode.

[Notify]: The **NCO output frequency of FDC mode is equal NCO overflow frequency/2.**

For FDC mode the NCO overflow frequency needs <= NCO Input frequency/2.

[Notify]: The **NCO output frequency** of **PFM** mode is equal **NCO overflow frequency**.

For **PFM** mode the **NCO overflow frequency** needs $< \text{NCO Input frequency}/2$.

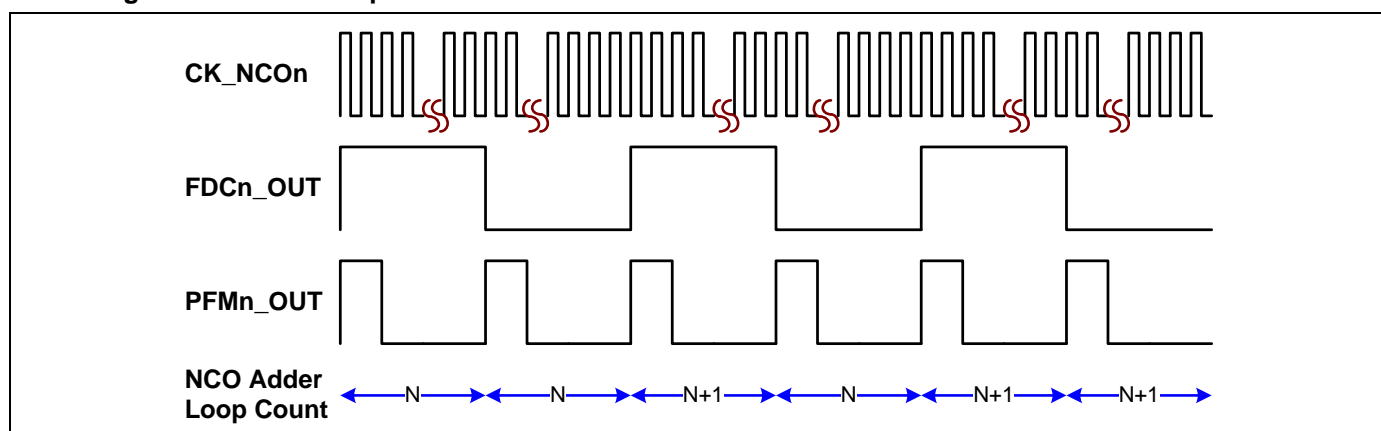
When selects the PFM mode, user can set the pulse width in $1/2/4/8 \sim 128$ clock periods of **CK_NCO0** by setting **APB_NCO_n_PWS** register.

The NCO output signal can be outputted to external pin or internal other modules. It can send to UART modules to do as clock source input. Also it can output to do as the common trigger event or clock signal **ITR7** for Timer modules. One output pin **NCO_P0** is providing to output the NCO output signal. Also this signal can be sent to URTx and TMx modules as the clock source. Others, user can invert the NCO output signal by setting **APB_NCO_n_INV** register for application request.

The NCO block is provided one NCO adder overflow interrupt flag **NCO_nF** and one interrupt enable bit **APB_NCO_n_IE** for interrupt service routine firmware using. When the NCO overflow frequency is over 4 times of APB clock frequency, the overflow interrupt flag will be responded incorrectly but the NCO frequency output is still normal. (n = NCO set index)

The following diagram is showing the NCO output waveform.

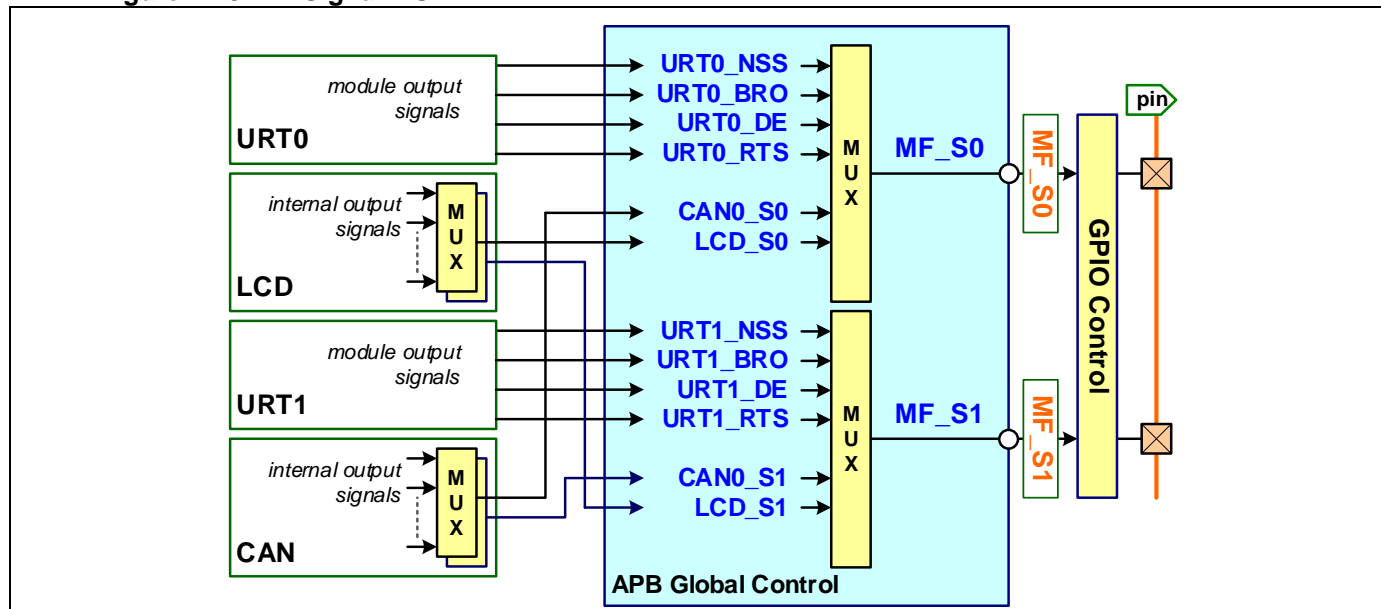
Figure 14-7. NCO Output



14.9. Multi-Function Signal

The APB module is including of two multiple signals multiplexers. These two multiplexers can use to select **MF_S0** and **MF_S1** signal from internal signals. These two selected signals can be output through AFS signals of **MF_S0** and **MF_S1** to external to pins. Refer the section of “Pin AFS Summary Table” of the related chip Data Sheet about AFS pin information.

Figure 14-8. MF Signal MUX



The following table is showing the Multi-Function source signals. For MG32F02A128/U128/A064/U064, the AFS pin setting of **USB_S0** and **USB_S1** are same as **MF_S0** and **MF_S1**.

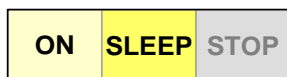
Table 14-6. Multi-Function Signals Table

Chip	MG32F02K128/K064/N128/N064		MG32F02A128/U128/A064/U064		MG32F02V032	
AFS Pin	MF_S0	MF_S1	USB_S0	USB_S1	x	x
	APB Register		x		x	
Output Signal	MF0_MUX	MF1_MUX	x	x	x	x
SIG0	URT0_NSS	URT1_NSS	USB_S0	USB_S1	x	x
SIG1	URT0_BRO	URT1_BRO				
SIG2	URT0_DE	URT1_DE				
SIG3	URT0_RTS	URT1_RTS				
SIG4	-	-				
SIG5	CAN0_S0	CAN0_S1				
SIG6	LCD_S0	LCD_S1				
SIG7	-	-				

<Sign> "-" = reserved, "x" = non-existed register or signal

15. APX (APB Extended Control)

15.1. Introduction



The module can be running in ON and SLEEP modes only.

The chip builds in one APX module for the extended function control of APB devices.

Notify: The sign of (x = module index; n = CCL set/ASB channel index; m=CCL channel index) is using for Registers, Signals and Pins/Ports in the descriptions of this chapter.

15.2. Features

- Support two sets of CCL(Configurable Custom Logic)
- ASB(ARGB Serial Bus) for addressable RGB LED display
- SDT(State Detector) for two-line sequential state detection

15.3. Implementation

15.3.1. Chip Implementation

The following table is showing the implemented APX functions of chips.

Table 15-1. APX Implementation

Chip	APX Module Sub-Functions			
	CCL0	CCL1	SDT	ASB
MG32F02A128/A064 MG32F02U128/U064	V	V	-	-
MG32F02V032	V	V	-	4-channels
MG32F02N128/N064 MG32F02K128/K064	V	V	V	4-channels

<Note> V: Implemented

15.4. Interrupt and Event

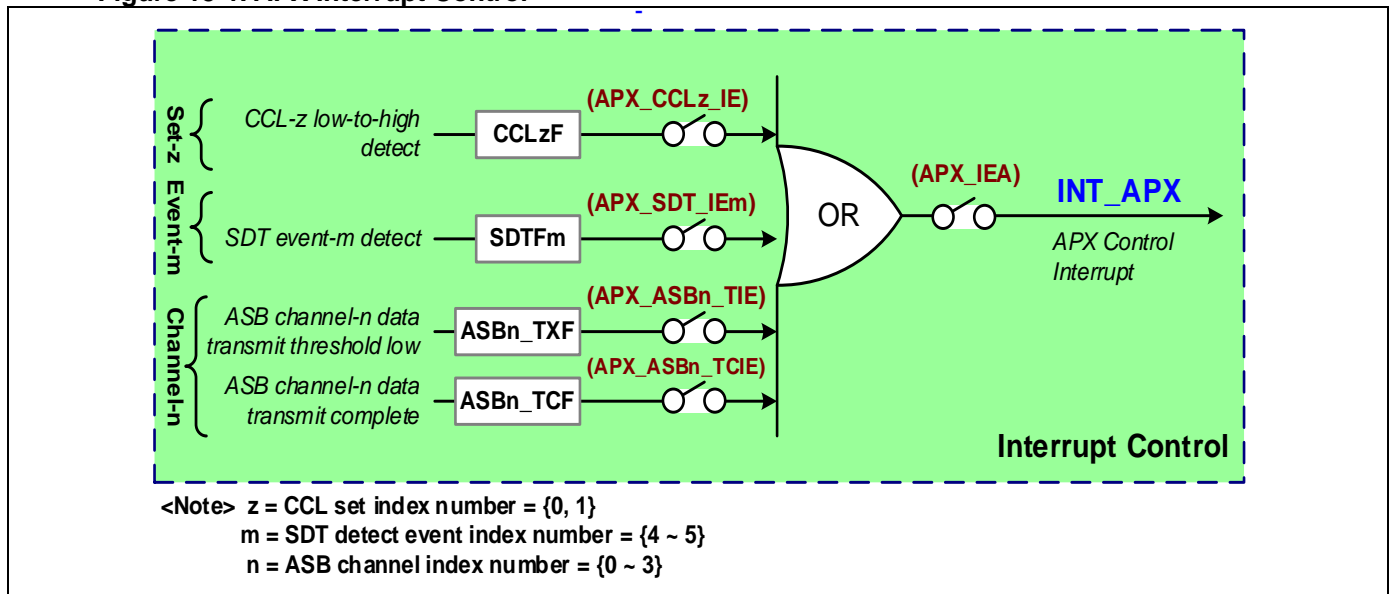
There is one signal of **INT_APX** to be generated in this APX module. **INT_APX** sends to External Interrupt Controller (EXIC) to do as an interrupt event.

15.4.1. APX Interrupt Control and Status

These interrupt flags are using for the interrupt service routine (ISR) flow control. Generally these flags are set by hardware and software clears them when the serviced job is completed in the related ISR. Each interrupt flag has one interrupt enable bit. User can enable or disable it. There is one interrupt all enable bit of **APX_IEA** to enable or disable all the interrupt sources for this module.

There are some status bits those are reading only to provide internal control status. Refer the register descriptions of the related status bits for more information.

Figure 15-1. APX Interrupt Control



15.4.2. APX Interrupt Flags

Generally these interrupt flags are set by hardware and clear by software writing 1. Refer the register descriptions for more information about the related interrupt flags and interrupt enable bit.

- **CCL0F / CCL1F**

CCL-0/ CCL-1 output low-to-high detect flag (**APX_CCL0F**, **APX_CCL1F**). There are the related interrupt enable register bits of **APX_CCL0_IE** and **APX_CCL1_IE**.

- **ASB0_TXF ~ ASB3_TXF**

ASB channel-0~3 transmission data threshold low flag (**APX_ASBn_TXF**). There are the related interrupt enable register bits of **APX_ASBn_TIE**. When transmitted FIFO is below low threshold, this flag is set. This bit is cleared when **APX_ASBn_DAT** is written or this flag set to 1 by software. (n = 0~3)

- **ASB0_TCF ~ ASB3_TCF**

ASB channel-0~3 transmission complete flag (**APX_ASBn_TCF**). There are the related interrupt enable register bits of **APX_ASBn_TCIE**. When both data FIFO and shift buffer shift out complete, then set this flag. (n = 0~3)

- **SDTF4 ~ SDTF5**

SDT state procedure-4 ~ 5 detect interrupt enable (**APX_SDTF4 ~ APX_SDTF5**). There are the related interrupt enable register bits of **APX_SDT_IE4 ~ APX_SDT_IE5**.

15.5. CCL Control

The APX module is including of two sets of Configurable Custom Logic (CCL) block. Each CCL block can be connected to the device pins or other internal peripherals' signal. The CCL can eliminate external extra logic gate devices for simple glue logic functions on user application. It supports the L-to-H signal detected event flag and the interrupt.

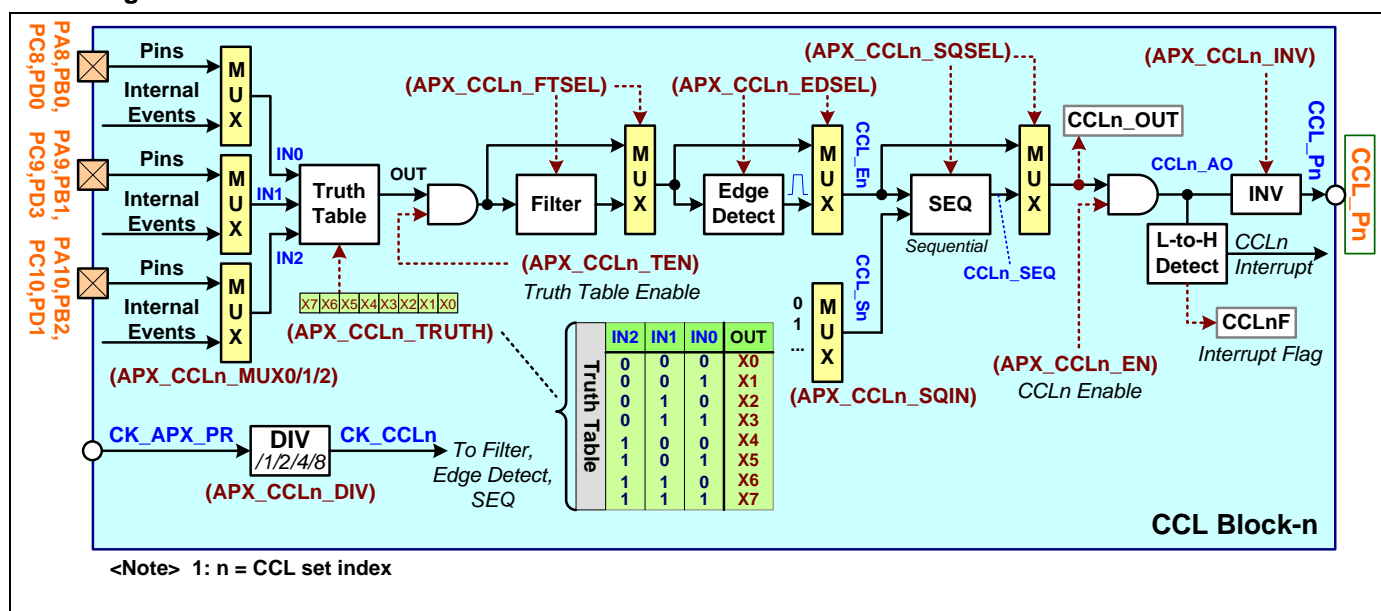
15.5.1. CCL Block

Each CCL block consists of three inputs, one truth table, one filter, one edge detector and one sequential logic block. It can generate an output as a user programmable logic expression with maximum three inputs. The **APX_CCLn_DIV** register is used to divide the CCL input clock to do as the operation clock for the filter, edge detector and sequential logic block.

Each CCL block has one block enable bit in **APX_CCLn_EN** register. User must enable this bit before start running the CCL block-n.

The following diagram is showing the CCL control block.

Figure 15-2. CCL Block



15.5.2. CCL Input and Truth Table

The truth table has three inputs with independent input Mux which can select the input source by independent setting register of **APX_CCLn_MUX0**, **APX_CCLn_MUX1** and **APX_CCLn_MUX2**. Each input can be masked independently.

The following table is showing the CCL input Mux signal source.

Table 15-2. CCL Input Mux Signal Selection

Input Mux	IN0	IN1	IN2
Register Setting	CCLn_MUX0	CCLn_MUX2	CCLn_MUX3
0	Disable	Disable	Disable
1	CCLn_SEQ	CCLn_SEQ	CCLn_SEQ
2	CCLn_AO	CCLn_AO	CCLn_AO
3	PB11	PB3	PC1
4	PA8	PA9	PA10
5	PB0	PB1	PB2
6	PB8	PB9	PB10
7	PE0(*1)	PE1(*1)	PE2(*1)
8	CMP0_OUT(*1)	CMP1_OUT(*1)	ADC0_OUT
9	URT0_TX	URT1_TX	URT4_TX(*1)
10	SPI0_MOSI	SPI0_MISO	SPI0_CLK
11	TM36_OC00	TM36_OC10	TM36_OC2
12	TM26_OC00(*1)	TM26_OC10(*1)	TM20_OC00
13	SDT_I0	SDT_I1	SDT_P0
14	OBM_P0(*1)	OBM_I0(*1)	OBM_I1(*1)
15	URT0_BRO(*1)	URT1_BRO(*1)	URT4_BRO(*1)

<Note> *1: These signals are not supported for MG32F02V032.

User can input an 8-bit value to the **APX_CCLn_TRUTH** register to define the truth table output. That makes the truth table logic to do like as any combinational AND, NAND, OR, NOR, XOR, XNOR, INV gate or other function which is defined by the **APX_CCLn_TRUTH** register.

The following table is showing the truth table setting examples for combinational AND, NAND, OR, NOR, XOR, XNOR, INV gate.

Table 15-3. CCL Truth Table Setting Examples for Three Inputs

Truth Table Logic Output Setting for Three Inputs						
IN2, IN1, IN0	AND	NAND	OR	NOR	XOR	XNOR
Register	0x80	0x7F	0xFE	0x01	0x96	0x69
0, 0, 0	0	1	0	1	0	1
0, 0, 1	0	1	1	0	1	0
0, 1, 0	0	1	1	0	1	0
0, 1, 1	0	1	1	0	0	1
1, 0, 0	0	1	1	0	1	0
1, 0, 1	0	1	1	0	0	1
1, 1, 0	0	1	1	0	0	1
1, 1, 1	1	0	1	0	1	0
Truth Table Logic Output Setting for Two Inputs (IN2 is masked)						
IN2, IN1, IN0	AND	NAND	OR	NOR	XOR	XNOR
Register	0x08	0x07	0x0E	0x01	0x06	0x09
x, 0, 0	0	0	0	1	0	1
x, 0, 1	0	1	1	0	1	0
x, 1, 0	0	1	1	0	1	0
x, 1, 1	1	1	1	0	0	1
Truth Table Logic Output Setting for One Input (IN1/IN2 is masked)						
IN2, IN1, IN0	INV					
Register	0x01					
x, x, 0	1					
x, x, 1	0					

<Note> "x" is input masked, Register : Truth table register **APX_CCLn_TRUTH**

15.5.3. CCL Filter and Edge Detect

The truth table output signal can be clock synchronized or filtered by through the optional filter to remove spikes. User can set the **APX_CCLn_FTSEL** register to select the filter modes.

One optional edge detector can detect the input rising, falling or dual edge by setting **APX_CCLn_EDSEL** register and generate an output pulse.

15.5.4. CCL Sequential Logic and Output

The optional sequential logic can be used to do as D flip flop, JK flip flop, D latch and RS latch by setting **APX_CCLn_SQSEL** register. It can generate the complex waveform from two inputs, one is from the **CCL_En** signal and one is from adjacent **CCL_En+1** signal or others by setting **APX_CCLn_SQIN** register.

The CCL output signal can be outputted to external pin **CCL_Pn**. Others, user can invert the CCL output signal by setting **APX_CCLn_INV** register for application request.

The CCL block is provided one CCL output status bit **CCLn_OUT**, one output low-to-high edge detect interrupt flag **CCLnF** and one interrupt enable bit **APX_CCLn_IE** for interrupt service routine firmware using.

The following table is showing the sequential IO table setting for D flip flop, JK flip flop, D latch and RS latch.

Table 15-4. CCL Sequential I/O Table

Mode	APX Reg	Input		Output
	CCL0_SQSEL	CCL_En	CCL_Sn	CCLn_SEQ
D flip flop	Signal	D	Gate	Qn+1
	1	0	0	Qn
		0	1	0
		1	0	Qn
		1	1	1
JK flip flop	Signal	J	K	Qn+1
	2	0	0	Qn
		0	1	0
		1	0	1
		1	1	/Qn
D latch	Signal	D	Gate	Qn+1
	3	0	0	Qn
		0	1	0
		1	0	Qn
		1	1	0
RS latch	Signal	S	R	Qn+1
	4	0	0	Qn
		0	1	0
		1	0	1
		1	1	unknown

<Note> "Qn" : current state, "Qn+1" : next state

15.6. ASB Control

The APX module is including of one ARGB Serial Bus (ASB) block for ARGB (addressable RGB) LED display application. The ASB block can support to connect and control maximum four channels of ARGB LED chain. It also supports the related data transmission control flags and the interrupts for independent channel control.

The ASB block is implemented both asynchronous ARGB mode and synchronous Shift mode with DMA capability for RGB LED application. It is designed the programmable ARGB data code-0/1 high time and **RESET code** time control for flexible ARGB timing. It also can set the code level of ARGB **RESET code**, **IDLE code** and **SYNC code** by firmware register control.

Specially, the ASB block is supported the hardware auto **RESET code** generation and channel synchronization function to improve LED display performance and firmware easy control in application.

15.6.1. ASB Block

The ASB block is implemented four independent channels of ARGB LED display control bus. There is one independent output pin **ASB_Pn** to connect external ARGB devices for each channel. (n= {0, 1, 2, 3})

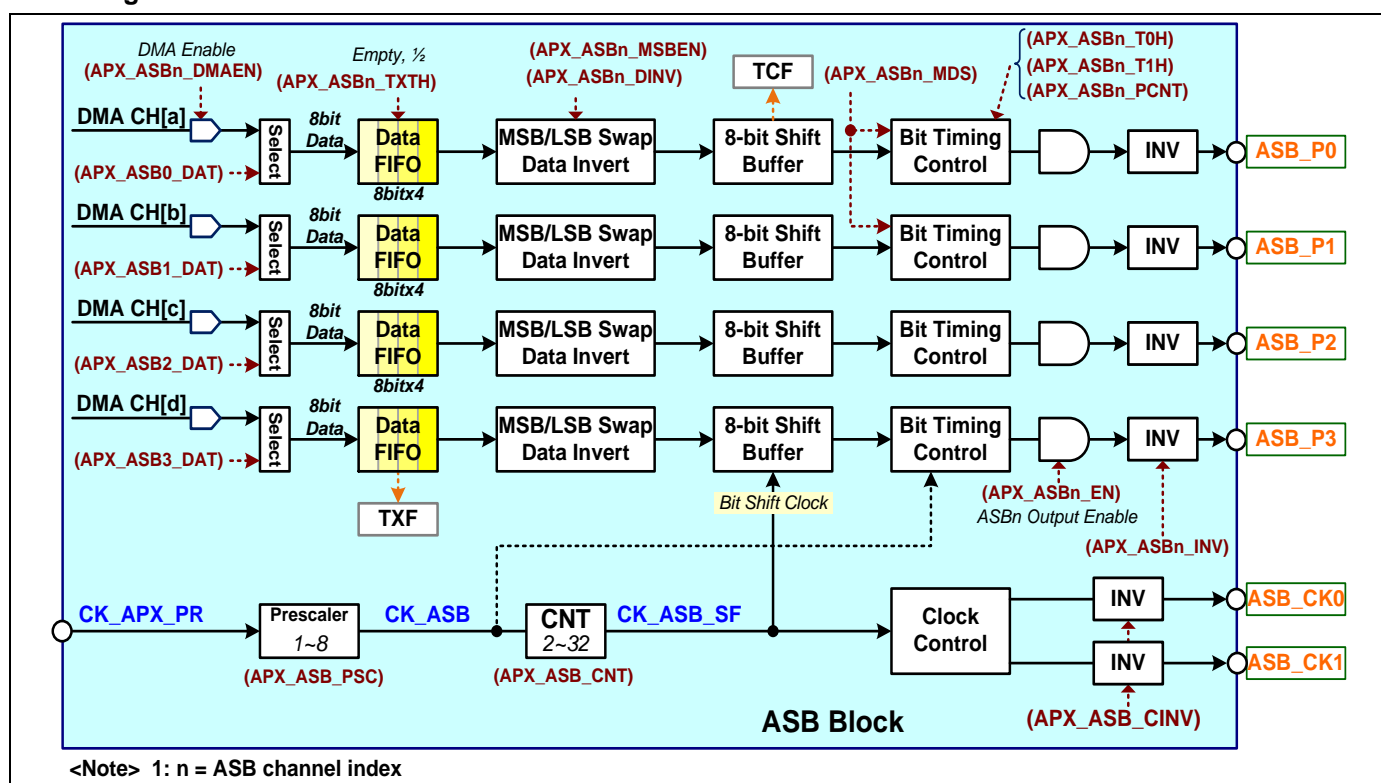
Each channel is including of 4-stages 8-bit FIFO with threshold control, MSB/LSB swap and data inversion logic, 8-bit shift buffer, bit timing control logic and output signal inversion logic. Also each channel can connect to one independent DMA channel for transferring ARGB LED data from internal SRAM or internal Flash. Please refer the section of “[APX DMA Operation](#)” for more information about DMA operation.

The ASB block is implemented the ASB process clock control logic to control the ASB data transfer speed. The **APX_ASB_PSC** register is used to divide the ASB input clock **CK_APX_PR** to do as the ASB block operation clock **CK_ASB** and ASB bit timing control clock for ARGB mode. The **APX_ASB_CNT** register is used to divide the **CK_ASB** clock to do as the ASB shift buffer operation clock **CK_ASB_SF** and bit shift out clock for Shift mode.

For synchronous Shift mode, there are two output pins **ASB_CK0** and **ASB_CK1** with output inversion control to connect external synchronous devices. The **ASB_CK0** pin is only used for ASB channel-0 and **ASB_CK1** pin is only used for ASB channel-1. The **APX_ASB_CINV** register is used to invert the bit shift-out clock for both **ASB_CK0** and **ASB_CK1** pins.

The following diagram is showing the ASB control block.

Figure 15-3. ASB Block



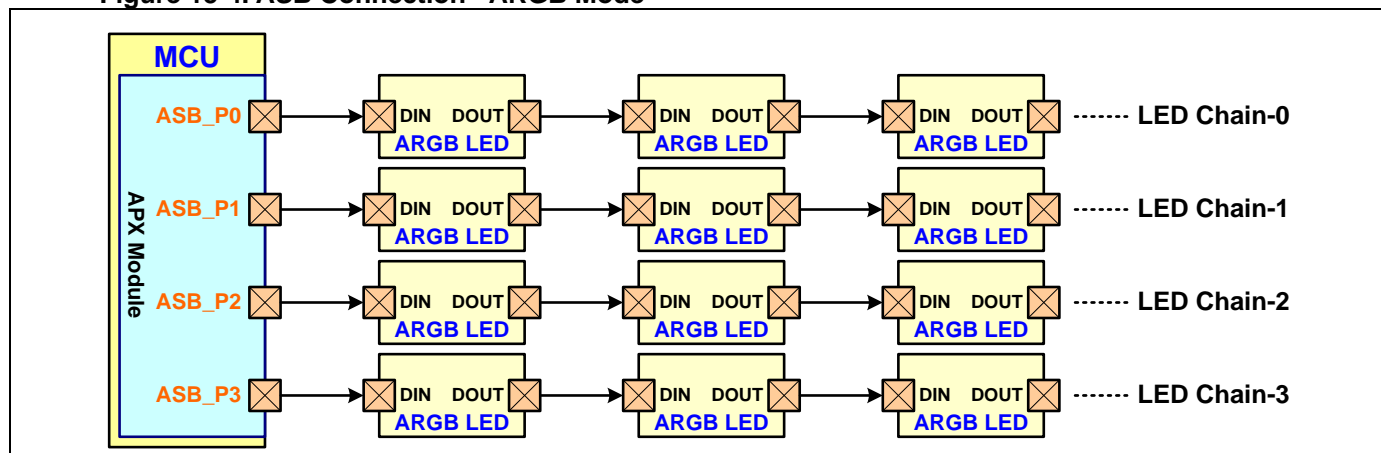
15.6.2. ASB Connection for Application

The ASB block supports two output modes, one is asynchronous ARGB mode and another is synchronous Shift mode. Refer the section of “[ASB Channel Control](#)” for more detail information about ASB mode control.

The following diagram is showing the ASB application connection for asynchronous ARGB mode. The independent **ASB_Pn** pin is always as data output signal to connect to external ARGB devices for each channel. The external ARGB devices are cascaded connection through the input DIN pin to the output DOUT pin for each cascaded ARGB device. (n= {0, 1, 2, 3})

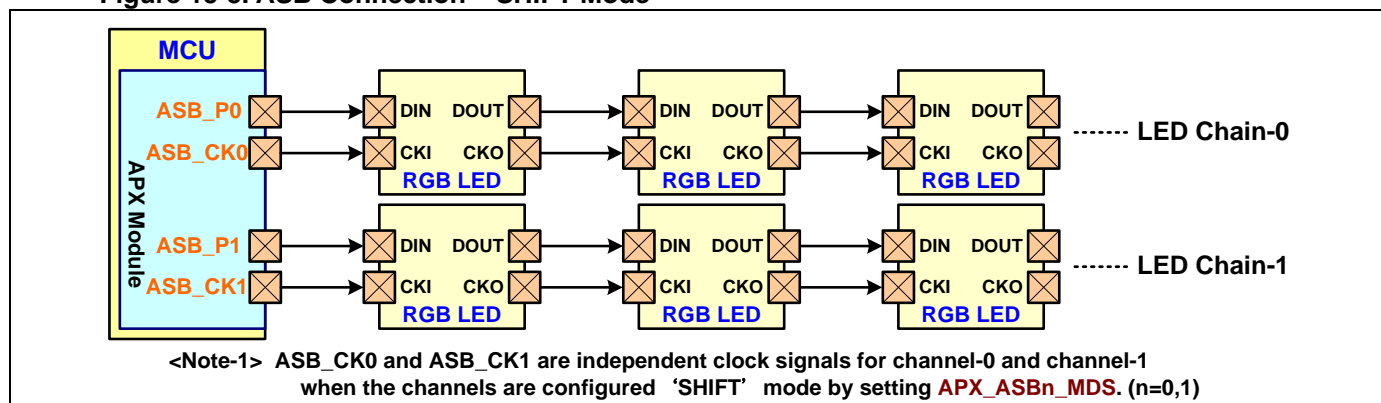
The MCU sends the first LED pixel data to the first ARGB device and the first ARGB device gets the first LED pixel data by internal data latch. In subsequent, The MCU sends the second LED pixel data to the first ARGB device and is passed to the second ARGB device. Then the second ARGB device gets the second LED pixel data. In following, the MCU sends the last LED pixel data and is passing to the last ARGB device. Finally, MCU transmits all the LED pixel data for one transferred frame. Usually the number of cascade is no transmission limit of the ARGB devices but limited signal transmission speed and device power supply voltage issue on PCB design.

Figure 15-4. ASB Connection - ARGB Mode



The following diagram is showing the ASB application connection for synchronous Shift mode. The independent **ASB_Pn** pin is as data output signal and extra **ASB_CKn** pin is as shift clock output signal to connect to external synchronous devices for channel-0 and channel-1. The ASB channel-2 and channel-3 are not supported the Shift mode. It is same as ARGB mode, the MCU send the RGB LED pixel data through first RGB device to last RGB device for the cascaded RGB devices.

Figure 15-5. ASB Connection – SHIFT Mode



15.6.3. ASB Channel Control

The ASB block is implemented four independent channels of ARGB LED display control bus. Each ASB channel has one channel enable bit in **APX_ASBn_EN** register. User must enable this bit for used ASB channel before start running the ASB block. There is a pseudo control bit **APX_ASBn_ENX** for each channel and it is identical to **APX_ASBn_EN** register bit. All the pseudo control bits of all channels are implemented in a same register of **APX_CR0** for firmware easy using. ($n = \{0, 1, 2, 3\}$)

The ASB block supports two output modes, one is asynchronous ARGB mode and another is synchronous Shift mode. User can set the **APX_ASBn_MDS** register to choice the ASB output mode for channel-0 and channel-1 independently. For ASB channel-2 and channel-3, these two channels are only support asynchronous ASB mode.

For ASB ARGB mode, the data transfer is using unipolar NRZ communication protocol. Each ASB code bit is composited a starting high level and an ending low level. The ASB code bit 0 or 1 is with the same time period. The ASB code bit is 0 or 1 that is distinguished by the duty cycle of high level and low level.

In application, the different LED devices have the different specifications about the duty cycle of code-0/1 and the minimum time of **RESET code**. The ASB block is designed the programmable ARGB data code-0/1 high time to change the duty cycle and the programmable **RESET code** time for flexible ARGB LED timing control. User can program the ASB code-0 and ASB code-1 high time by setting **APX_ASB_T0H** and **APX_ASB_T1H** registers.

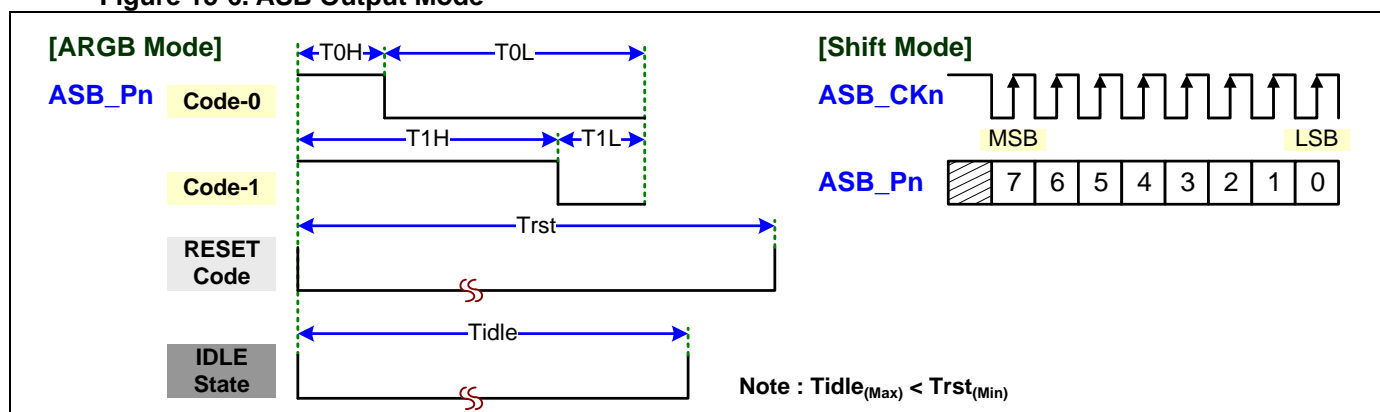
Usually the ASB bus is reset after the last LED pixel data transmission completed at the DIN pin of external first ARGB device. Then the MCU must send the **RESET code** to external ARGB devices. Also user can program the ASB **RESET code** time by setting **APX_ASB_TRST** register. After the **RESET code** cycle, the ASB bus is entering idle state if the ASB data transmission is stopped and no next pixel data transmission at the time.

The ASB block also can set the code level of ARGB **RESET code** and **IDLE code** whether is low level or high level by setting **APX_ASB_RST** and **APX_ASB_IDLE** registers. Usually, both **RESET code** and **IDLE code** are active low level.

For ASB Shift mode, the ASB data transfer is synchronous by the shift clock **ASB_CKn**. The ASB data signal **ASB_Pn** is update after the shift clock changing from high-to-low falling edge by MCU output. The external RGB device can sample and latch the ASB pixel data at the shift clock changing from low-to-high rising edge.

The following diagram is showing the ASB output ARGB mode and Shift mode timing diagram. User can set the time of **T0H** and **T1H** to easy change duty cycle of code-0 and code-1 by register control.

Figure 15-6. ASB Output Mode



Refer the ASB block, each channel is implemented the MSB/LSB swap and data inversion logic. User can set **APX_ASBn_MSBEN** register to configure the output data order by Lsb first or Msb first and enable to inverse the transmitted data bits from "0" to "1" or "1" to "0" by setting **APX_ASBn_DINV** register. Also each channel is implemented the output signal inversion logic and user can set the **APX_ASBn_INV** register to inverse the output signal through the related output pin **ASB_Pn** to connect external ARGB devices.

15.6.4. ASB Data Transmit

The ASB block is implemented four independent channels of ARGB serial bus. It builds 4-stages 8-bit FIFO with threshold control and one independent **ASB_Pn** pin to connect external cascaded devices for each channel. The chip can send the successive LED pixel data to the external cascaded LED devices by through the ASB block for each ASB channel independent. ($n = \{0, 1, 2, 3\}$)

For each ASB channel, there is one ASB 8-bit data port which is used to fill the ASB byte data for DMA hardware transmission or MCU firmware writing through **APX_ASBn_DAT** register. There is a pseudo ASB data register **APX_ASBn_DATX** for each channel and it is identical to **APX_ASBn_DAT** register. All the pseudo ASB data register of all channels are implemented in a same register of **APX_ASBDAT** for firmware easy using.

When the ASB data is filled into the ASB 8-bit data port, the data will be sent to the 4-stages 8-bit FIFO for each ASB channel. The **APX_ASBn_TXTH** register is used to set the data low threshold of 8-bit data FIFO. When ASB data transmission is processing and the ASB channel is not operating in DMA mode, hardware will assert the **APX_ASBn_TXF** flag if the data byte number of data FIFO is equal to or lower than the threshold. Also it will induce the related MCU interrupt if the **APX_ASBn_TIE** bit was enabled. Use can fill the next ASB successive data to the ASB data register in the related interrupt service routine when detect the **APX_ASBn_TXF** flag. When the ASB channel is operating in DMA mode, the **APX_ASBn_TXF** flag is masked during the DMA transfer process and the ASB data is filled to the ASB data port automatically. Please refer the section of “[APX DMA Operation](#)” for more information about DMA operation.

When the ASB data port does not be filled any more data to stop ASB data transmission and shift buffer data has shifted out clearly, the TCF flag (**APX_ASBn_TCF**) can denote that the previous ASB data transmission is complete.

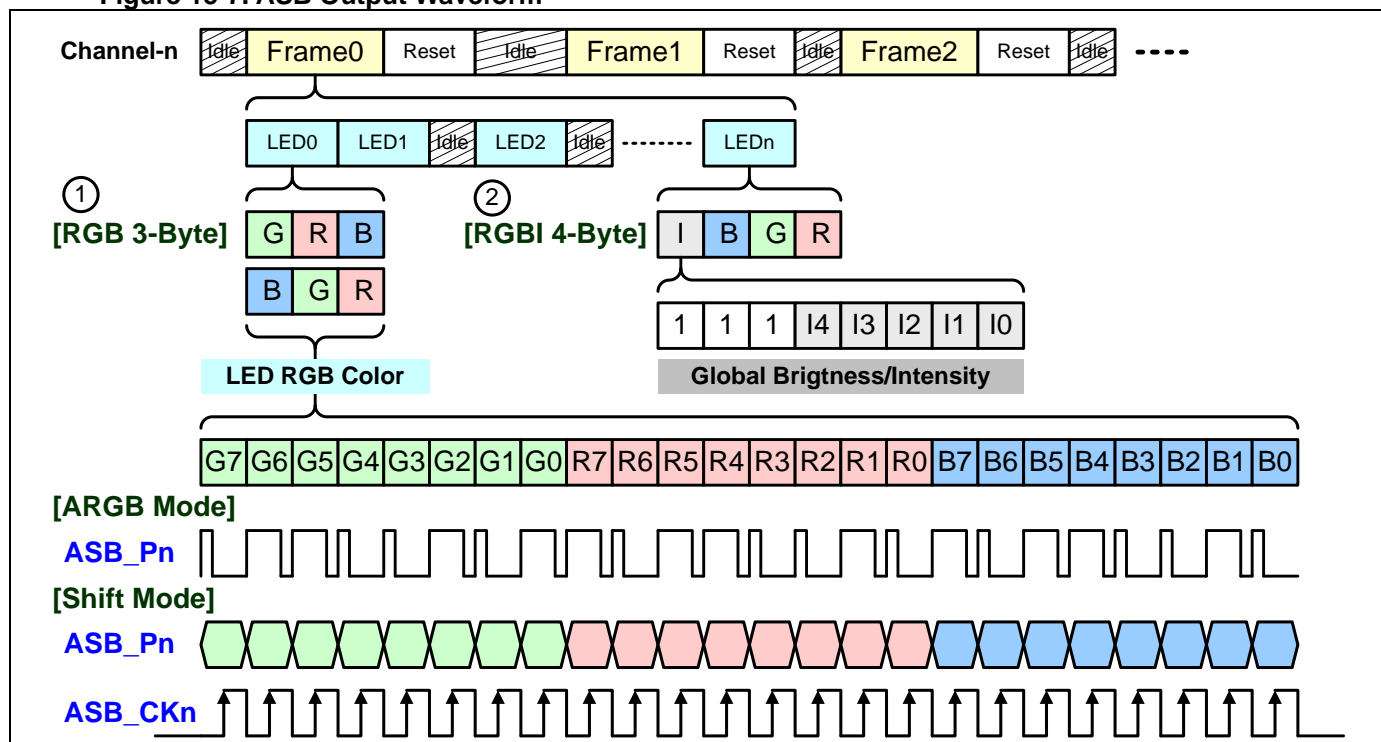
For MCU firmware data control, user can force to clear the transmitted data FIFO by setting the independent **APX_ASBn_FCLR** register bit for each ASB channel. This register bit is set by software and clear by hardware. When use enables the register bit of **APX_ASBn_FCLR**, the transmitted data FIFO and shift buffer will be flushed and **APX_ASBn_TXF** flag is set.

The following diagram is showing the ASB output waveforms. The pixel data of one LED device can be RGB 3-byte data format or RGBI 4-byte data format. Each ASB data transferred frame is composited successive N-pixel data and one **RESET code** for N cascaded LED devices. When ASB transmission data is filled slower than data shifts out, the ASB data transferred frame will be to have to insert the idle state (**IDLE code**). Between two transferred frames, the ASB bus is also in idle state usually.

*[Notify]: The time of idle state must be short than the time of **RESET code**, otherwise the idle state (**IDLE code**) will be identified to a **RESET code** by external LED devices and induce the bus and LED devices reset.*

Please refer the section of “[ASB Channel Control](#)” for more information about **RESET code** and **IDLE code** setting and refer the section of “[ASB Bus Reset](#)” for detail information about **RESET code** generation.

Figure 15-7. ASB Output Waveform



15.6.5. ASB Bus Reset

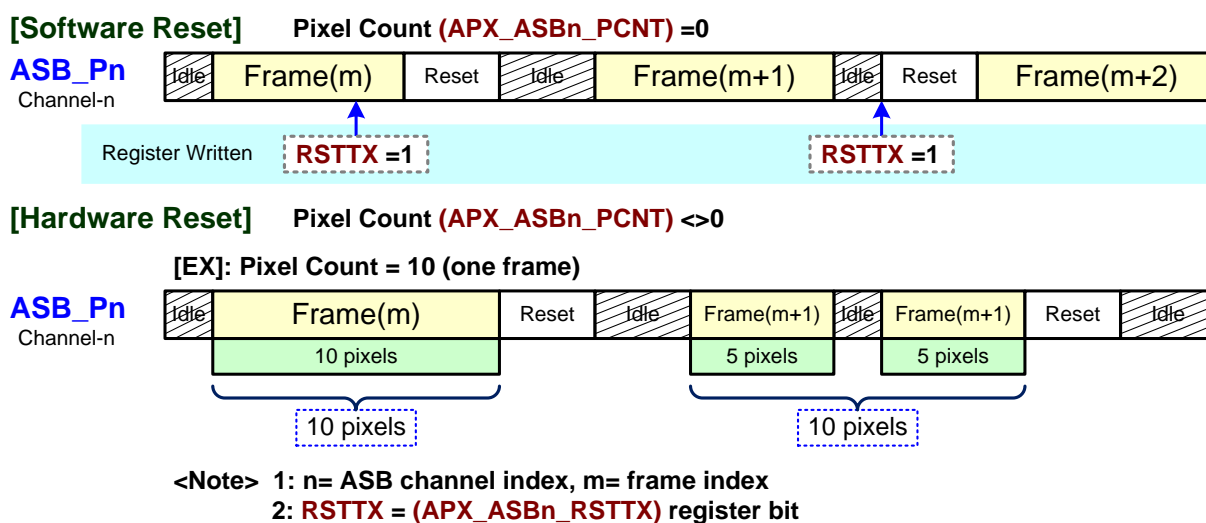
When the ASB data transmission is completed for one transferred frame, the MCU must send the **RESET code** to external ARGB devices to reset the ASB bus. The ASB block provides two methods to generate the **RESET code** for each ASB channel. One is software reset that user can set the register bit of **APX_ASBn_RSTTX** to directly trigger ASB block to generate the **RESET code** by MCU firmware control. Please refer the section of “[ASB Channel Control](#)” for more information about **RESET code** setting.

Another is hardware reset that the ASB block automatically generates the **RESET code** by chip hardware control. The ASB block is implemented a pixel counter and uses to supports the **RESET code** generation of one transferred frame. When the transmitted data number is reached the pixel count, the chip will automatically insert a **RESET code** after the last pixel data. User can set the pixel number of one transferred frame by setting **APX_ASBn_PCNT** register. Set the register value 0 that is disabled to insert a **RESET code** automatically by hardware and is only able to software trigger reset by MCU firmware control. Others, this pixel count register can be also used to control the ASB channel synchronization. Please refer the section of “[ASB Channel Synchronous Mode](#)” for detail information about ASB channel synchronization control.

For the pixel counter of each ASB channel, user can independently define the byte length of a pixel which is 3-byte or 4-byte by setting **APX_ASBn_PLEN** register for the pixel data format of using LED devices in application.

The following diagram is showing the ASB **RESET code** generation timing by software reset or hardware reset. For the software reset, user can trigger the register bit of **APX_ASBn_RSTTX** to force **RESET code** generation at any time during the ASB data transmission period. The chip will generate the **RESET code** directly in idle state or after shift out over the integrated 8-bit data for current transmission ASB data byte. For the hardware reset, the chip will automatically insert a **RESET code** after the transmitted data number is reached the pixel count setting value 10 in the example.

Figure 15-8. ASB RESET code Generation Timing



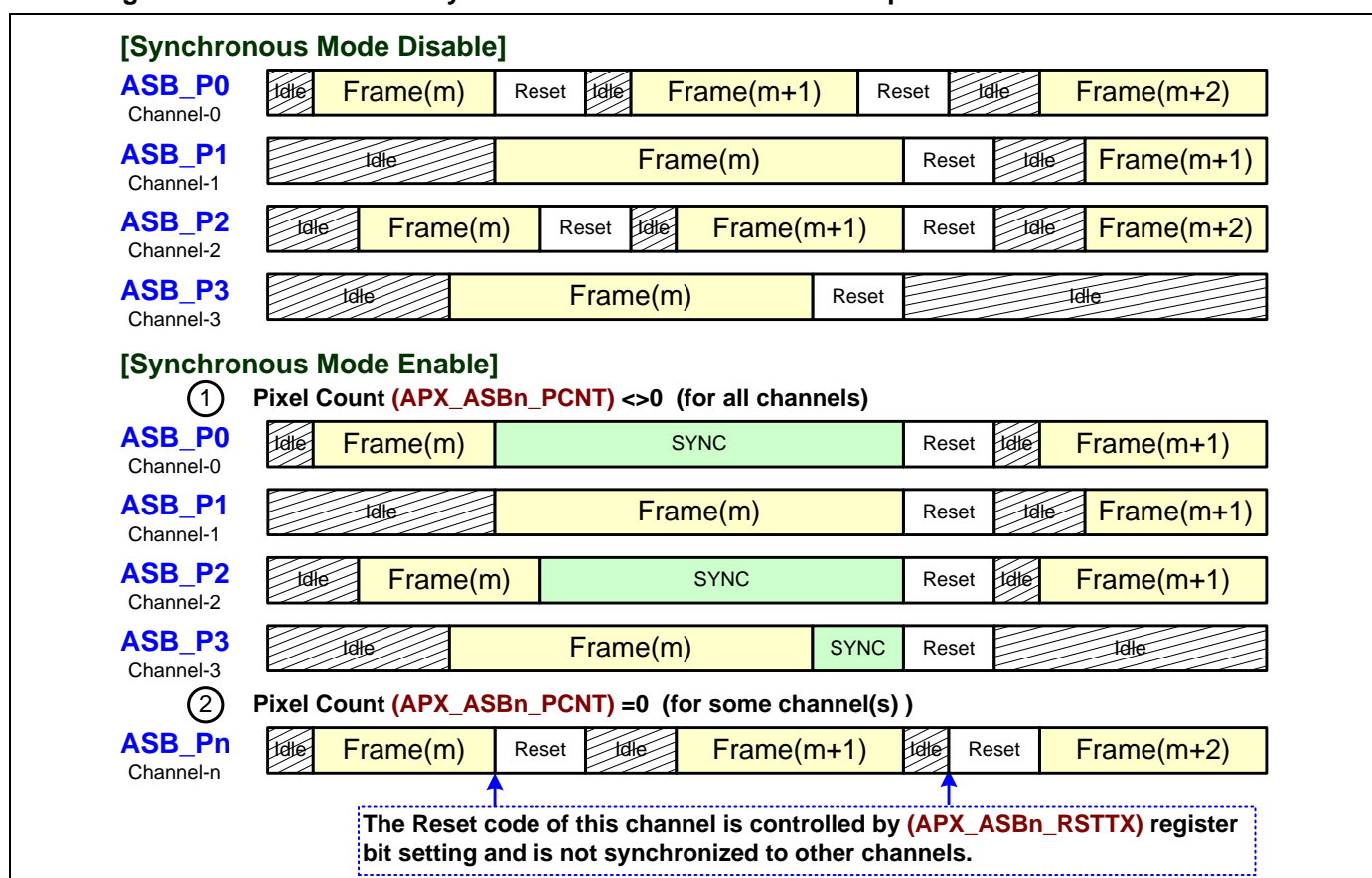
15.6.6. ASB Channel Synchronous Mode

The ASB block is supported channel synchronous mode to synchronize the transmission ending for synchronization enabled ASB channels. When user wants to run the channel synchronous mode, use needs to enable the function by setting **APX_ASB_SYEN** register at first. Then, any ASB channel is going to join in the channel synchronization which must be set non-zero value for the reset pixel count in **APX_ASBn_PCNT** register. When set the reset pixel count register value to 0, it will disable to insert a **RESET code** automatically and does not join in the channel synchronization.

The following diagram is showing an example of the ASB channel asynchronous and synchronous mode waveform. When channel synchronous mode is disabled, each ASB channel can independently start the ASB data transmission at any time and independently control the time of **RESET code** sending to end the transmission. When channel synchronous mode is enabled, each ASB channel can also independently start the ASB data transmission at any time. The chip can detect the transmission complete time for all synchronization enabled channels and know the last transmission complete time. Then at the last transmission complete time, the chip will insert the **RESET code** synchronously for all synchronization enabled channels. Any synchronization enabled channel is transmission complete before the last transmission complete time, it will be inserted the **SYNC code** after the ending of data transmission until the last transmission complete time.

For example in the following diagram for synchronous mode enable, the channel-1 is the last channel of transmission complete. The other channels are inserted the **SYNC code** after the ending of data transmission until the ending of data transmission of channel-1. The ASB block can set the code logic of **SYNC code** whether is code-0 or code-1 by setting **APX_ASB_SYNC** register.

Figure 15-9. ASB Channel Synchronous Mode Waveform Example



15.7. SDT Control

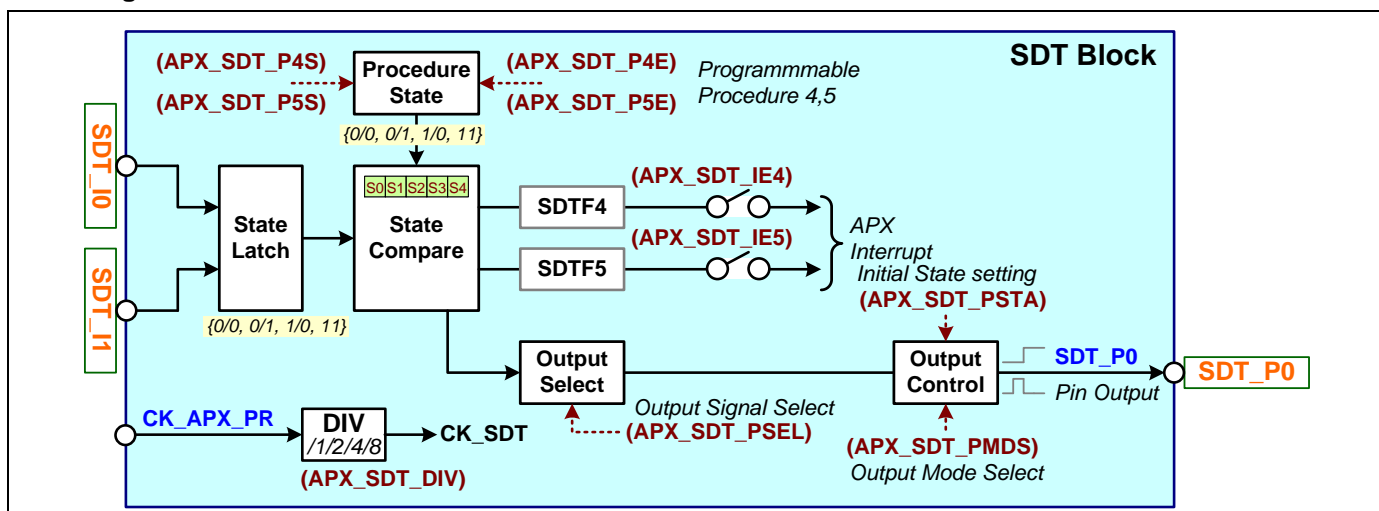
The APX module is including of one Sequential State Detector (SDT) block for two-line sequential state detection. The SDT can be connected to the device pins and detect the sequential stats for special procedures. It supports the related detected event flag and the interrupt for different procedure.

15.7.1. SDT Block

The SDT block is including of six state procedure control logic, state compare logic, output event control logic and one IO pin output mode control logic. The **APX_SDT_DIV** register is used to divide the SDT input clock to do as the SDT block operation clock.

Each SDT block has one block enable bit in **APX_SDT_EN** register. User must enable this bit before start running the SDT block. The following diagram is showing the SDT control block.

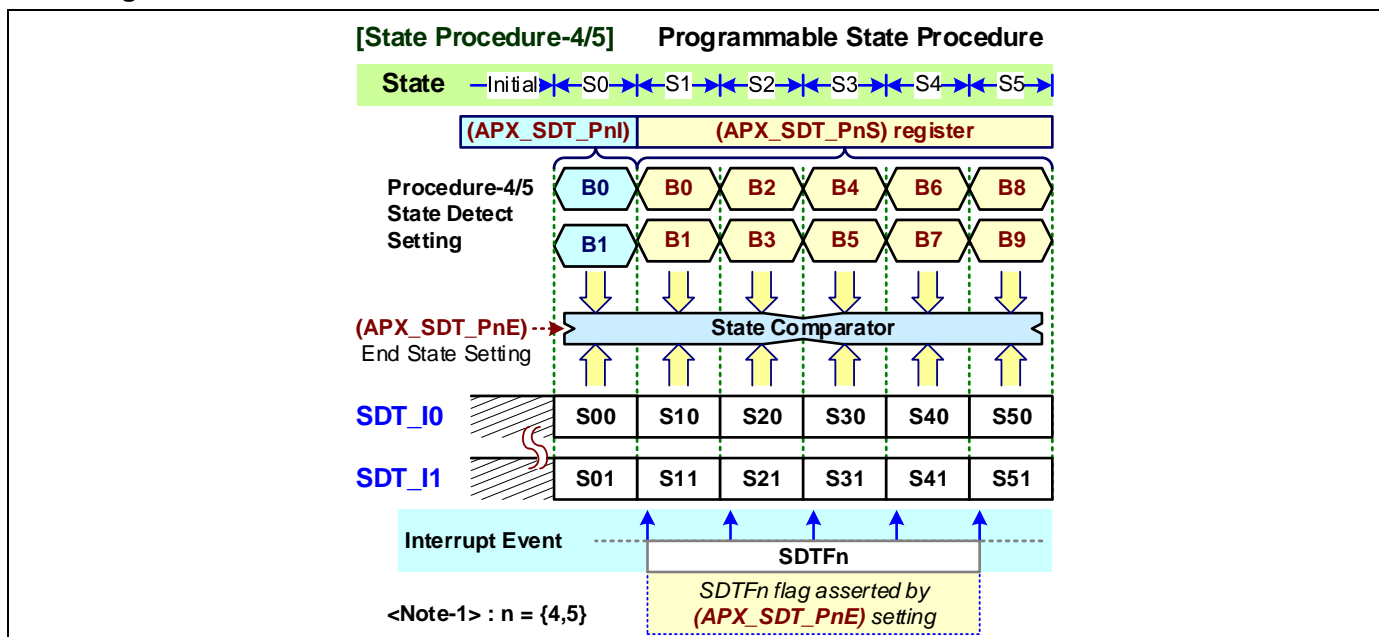
Figure 15-10. SDT Block



15.7.2. SDT Input and State

There two input signals are from **SDT_I0** and **SDT_I1** pins and use to detect the sequential state by state procedure control logics and the state compare logic. There are maximum six sets of state procedure to be provided to detect the input signals' state. Also the SDT block is provided six state procedures detect interrupt **SDTnF** flags and six interrupt enable bits **APX_SDT_IEn** for interrupt service routine firmware using.

Figure 15-11. SDT Block



The state procedure-4, 5 are two independent programmable procedures those can be defined the detection state flow and the last state index by configurable registers. User needs to set the wished initial states value of 0/0, 0/1, 1/0, 11 into initial state registers of **APX_SDT_P4I** or/and **APX_SDT_P5I**. Following, user can set the wished sequential states value of 0/0, 0/1, 1/0, 11 into sequential registers of **APX_SDT_P4S** or/and **APX_SDT_P5S** for **SDT_I0** / **SDTI1** input detection.

Also user needs set the detected end state index number 1~5 in **APX_SDT_P4E** or/and **APX_SDT_P5E** registers. The chip will assert the **SDT4F** or/and **SDT5F** flags if chip has compared **SDT_I0** / **SDTI1** input signal and matched to each register setting state until setting last state.

15.7.3. SDT Output

These procedures' detection events can be chose to output to IO pin **SDT_P0** by through **SDT_P0** signal.

User can select one of the output source events from state procedure-0~5 detect event by setting **APX_SDT_PSEL** register. The **SDT_P0** signal can be set the initial state in **APX_SDT_PSTA** register. There are two output modes can be set in **APX_SDT_PMDS** register.

15.8. APX DMA Operation

15.8.1. DMA Module Configure

When the chip supports a DMA (direct memory access) controller, user can configure the DMA setting of transferred source/destination devices, channel request arbitration and others in the DMA module before a DMA data transaction. The DMA source and the destination can be memory or peripheral.

Refer the DMA chapter for more detail information about the DMA module configuration.

15.8.2. APX DMA Control

After DMA configuration is finished, user needs to set the ASB function of APX module DMA enable bit of **APX_ASBN_DMAEN** for each ASB channel. Each channel can connect to one independent DMA channel for transferring ARGB LED data from internal SRAM or internal Flash. (n= {0, 1, 2, 3})

Finally, the related channel request start bit of **DMA_CHn_REQ** is necessary to be set to start the DMA transaction (n = DMA channel index). Then the transferred source/destination devices will assert the RX/TX request signal to DMA controller and the DMA controller will assert the acknowledge signal to the request source/destination devices. At the time, the data transferred connection is built for DMA transaction.

During the DMA transaction cycle, the transmitted data flag of **APX_ASBN_TXF** is masked by hardware. In general, the transmitted complete flag of **APX_ASBN_TCF** will be asserted after the finished of DMA transaction.

15.8.3. APX Interrupt Flag Control during DMA

During DMA operation cycle, the module's interrupt flags will control and act two types as following table. One is masked during the DMA process. Another is normally as same as the action of not processing in DMA operation.

Table 15-5. APX Interrupt Flag Control for DMA Function

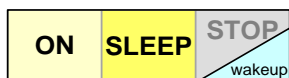
Action	Mask the Flag during DMA Process	DMA Disable after the Flag asserted (*1)	Normal Control	
Peripheral	(Data flow flags)	(Error/Detect flags)	(Other flags)	
APX	APX_ASBN_TCF (*2) APX_ASBN_TXF (*2) (n=0,1,2,3)		APX_CCLnF (n=0,1)	APX_SDTFn (n=4,5)

<Note> *1: When the flag is asserted, it will not force peripheral DMA disabled if the related interrupt enable bit is not enabled.

*2: This flag will be asserted after DMA TX complete.

16. I2C (Inter Integrated Circuit)

16.1. Introduction



The module can be running in ON and SLEEP modes but can wakeup from STOP mode.

The I2C interface is a two-wire, bi-directional serial bus. It is ideally suited for typical microcontroller applications. The I2C protocol allows the systems designer to interconnect up to 128 different devices using only two bi-directional bus lines, one for clock (SCL) and one for data (SDA). The I2C bus provides control of SDA, SCL generation and synchronization, arbitration logic, and START/STOP control and generation. The only external hardware needed to implement this bus is a single pull-up resistor for each of the I2C bus lines. All devices connected to the bus have individual addresses, and mechanisms for resolving bus contention are inherent in the I2C protocol. Please refer the "I2C-bus specification and user manual" for more information about the I2C protocol.

The I2C module builds in the shadow buffer and data register to improve transmit and receive communication performance.

The module can be running in ON and SLEEP mode only but can be wakeup MCU from STOP mode by slave address hardware detection.

Notify: The sign of (x = module index) is using for Registers, Signals and Pins/Ports in the descriptions of this chapter. [EX]: **I2Cx_EN** ~ x indicates module index number 0, 1.

16.2. Features

- Provide max. two I2C modules : I2C0 , I2C1
- I2C module common functions
 - Support master and slave mode
 - Support programmable clock rate control
 - Support programmable high/low period control for master mode
 - Support clock stretching for slave mode
 - Built-in high pre-drive control circuit
 - Generation and detection of 7-bit addressing
 - Support general call function
 - Support multi-master processing capability
 - Support multiple slave address decoding with two sets of address
 - Support bus error , invalid No-ACK , data overrun and multi-master arbitration error detection
 - Support both Byte mode and Buffer mode flow control
 - Support Byte mode bus event code for simplex firmware control
 - Support Buffer mode 4-byte data buffer and 32-bit data register for high speed communication
 - Received and transmitted data are buffered with DMA capability
 - Support SMBus timeout detection
 - Support slave address hardware detection wakeup from STOP mode

16.3. Implementation

16.3.1. Chip Implementation

The following table is showing the implemented I2C modules of chips.

Table 16-1. I2C Implementation

Chip	I2C Module		I2C Functions	I2C Clock
	I2C0	I2C1	STOP Wakeup	Max. Rate
MG32F02A128/A064 MG32F02U128/U064 MG32F02V032 MG32F02N128/N064 MG32F02K128/K064	V	V	V	1MHz

<Note> V: Implemented; Measure maximum clock rate at VDD>=3.3V.

16.3.2. Modules' Functions

The following table is showing the implemented functions of I2C modules.

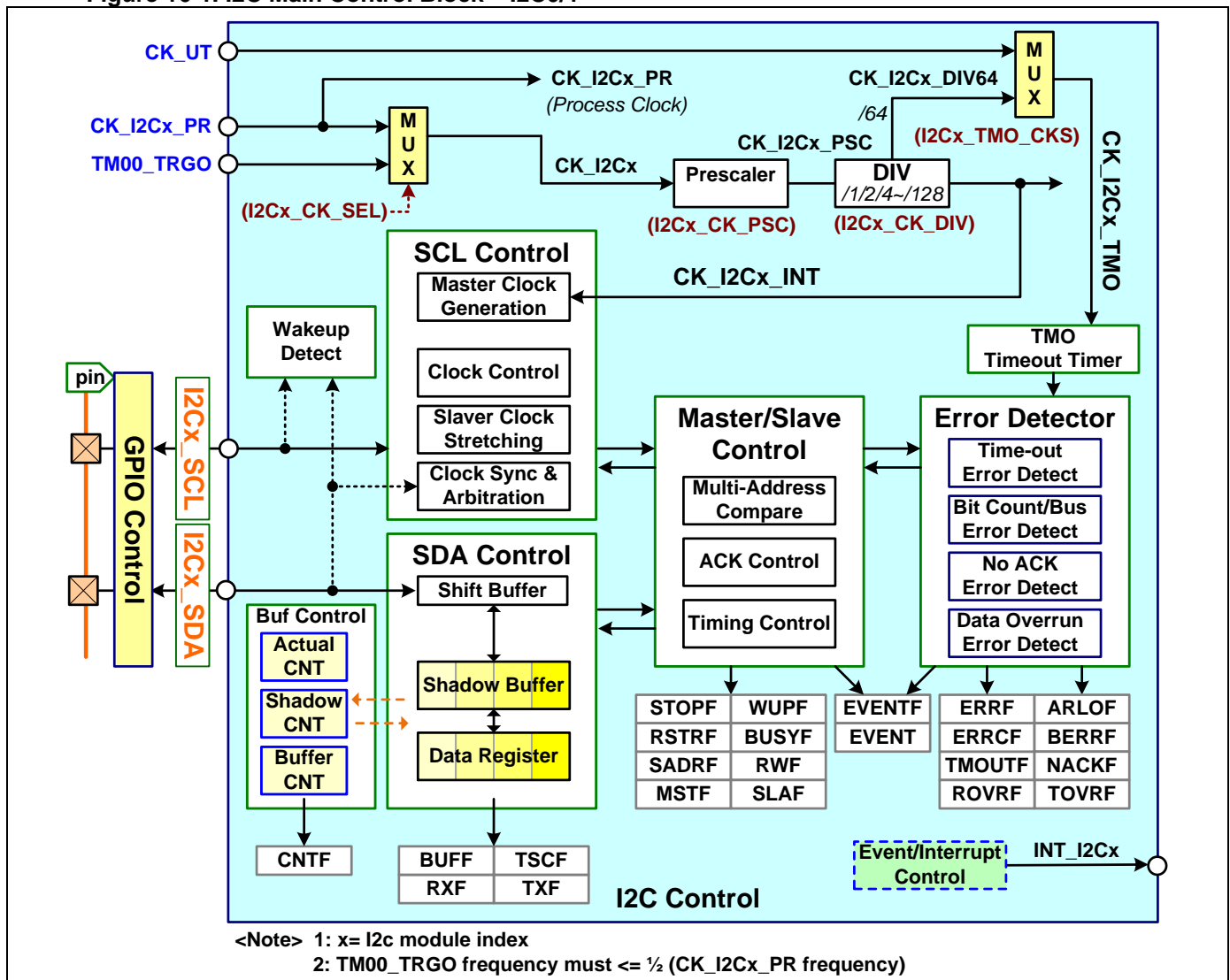
Table 16-2. I2C Modules' Functions

Module	I2C0/1	Comment
Chip	All	
Module Functions		
Master mode	yes	
Slave mode	yes	
Multi-Master	yes	for Byte mode only
General call	yes	command support - Reset , Write programmable slave address , Master address decode
Multi-Slave Address	2 sets	
Data Byte mode	yes	8bit shift buffer + 8bit data register; Software flow control mode
Data Buffer mode	yes	8bit shift buffer + 32bit shadow buffer + 32bit data register
Shadow Buffer	4-byte	internal data control buffer
Standard/Fast mode	yes	
Fast mode plus (1M/s)	yes	support for Buffer and DMA mode
Built-in pre-drive high	yes	pre-drive both SCL and SDA by hardware
SCL stretching	optional	ACK cycle SCL stretching; hardware high level checking
Address detect wake up	yes	slave address detect and wake up on STOP mode
TX NACK ignore	yes	master TX ignore receiving NACK for Buffer mode
Slave address mask	yes	support slave address mask register
Programmable SCL High/Low time	yes	
SCL/SDA input filter	by IO Control	
SCL/SDA input Schmitt trigger	yes	
Time-out detect	yes	SMBus timeout, Detect SCL low or SCL/SDA both high timeout
Arbitration lost detect	yes	for Multi-Master mode
Bus error detect	yes	Bit-count mismatch error before valid 'Start' or 'Stop'; Data change between SCL high
Invalid NACK detect	yes	
Data overrun detect	yes	for SCL clock stretching disabled
DMA request capability	yes	

16.4. Control Block

The following diagram is showing the I2C Control block.

Figure 16-1. I2C Main Control Block – I2C0/1



16.5. IO Lines

16.5.1. IO Signals

- **I2Cx_SCL**

It is the I2C clock SCL signal and uses as output for I2C master mode or as input for I2C slave mode.

- **I2Cx_SDA**

It is the I2C data SDA signal and uses as bidirectional IO for both I2C master and slave modes.

16.5.2. IO Configure

User must configure the related IO pins in order to use the IO lines of this module. The IO operation modes, output high speed option, pull-high option, output drive strength, IO deglitch filter and input inverse selection are programmable by pin independent. Please refer the section of “[IO Mode](#)” in GPIO chapter for more detail descriptions of IO mode configuration. For the signal noise of I2C SCL and SDA lines, user can set the input digital deglitch register **Px_FCKs** to filter the noise. Please refer the “[Input Filter Control](#)” in the section of “[IO Configuration](#)” for more information.

[Notify]: For Fast-mode and Fast-mode Plus, input filters on the SDA and SCL inputs suppress noise spikes of less than 50 ns by the reference of “I2C-bus specification and user manual”.

Each IO signal may be mapped and selected on several IO pins by configuring the IO AFS matrix. Please

refer the section of “[Alternate Function Select](#)” in GPIO chapter for more detail descriptions of IO AFS configuration. About the actual IO pin AFS information, please refer the section of “[Pin Alternate Functions Selected Table](#)” in Pin Description chapter of the chip Data Sheet.

16.6. Enabling and Clock

16.6.1. I2C Global Enable

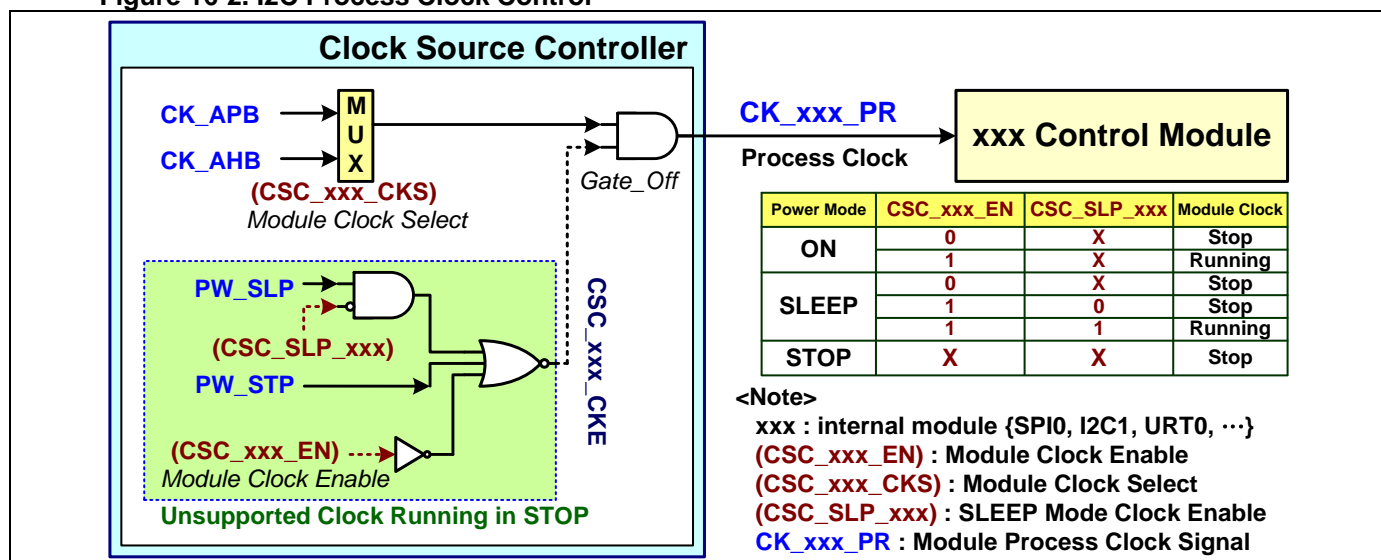
There is a global enable bit of **I2Cx_EN** for all functions of this module. When this bit is disabled, all the I2C functions are not working.

16.6.2. I2C Clock Control

- **Module Process Clock**

The module process clock of **CK_I2Cx_PR** is using for the interface control logic between APB bus and the module. It is coming from CSC (Clock Source Controller) module. It can be enabled in **CSC_I2Cx_EN** register and select the clock source from APB clock or AHB clock in **CSC_I2Cx_CKS** register. User can plan the module clock is running or not beforehand for chip entering **SLEEP** mode by setting **CSC_SLP_I2Cx** register. Refer the System Clock chapter for more information.

Figure 16-2. I2C Process Clock Control



- **Module Internal Clock**

The I2C module is able to output the internal clock of **CK_I2Cx_INT** as the sampling clock for received data signal and the clock source for transmitted data signal. User can select the clock source from the module process clock of **CK_I2Cx_PR** or the timer trigger output signal of **TM00_TRGO** by setting **I2Cx_CK_SEL** register. Also the module provides one clock prescaler and one clock divider to generate the internal clock of **CK_I2Cx_INT**. The clock prescaler can divide the input clock in **I2Cx_CK_PSC** register and the register value must be set > 0. The clock divider can divide the input clock by 1/2/4/8/ ~128 in **I2Cx_CK_DIV** register.

[Notify]: Usually the internal clock frequency needs slow down at least 1/2 than module process clock.

The TMO timer clock source can be select from the clock of **CK_UT** or **CK_I2Cx_DIV64** by setting **I2Cx_TMO_CKS** register. The clock of **CK_I2Cx_DIV64** is fixed from the frequency divided by 64 of **CK_I2Cx_PSC** clock signal.

16.7. Interrupt and Event

There are two types' signals of **INT_I2Cx**, **WUP_I2Cx** to be generated in this I2C control module. **INT_I2Cx** sends to EXIC External Interrupt Controller to do as an interrupt event. **WUP_I2Cx** sends to External Interrupt Controller (EXIC) to do as the system wakeup events.

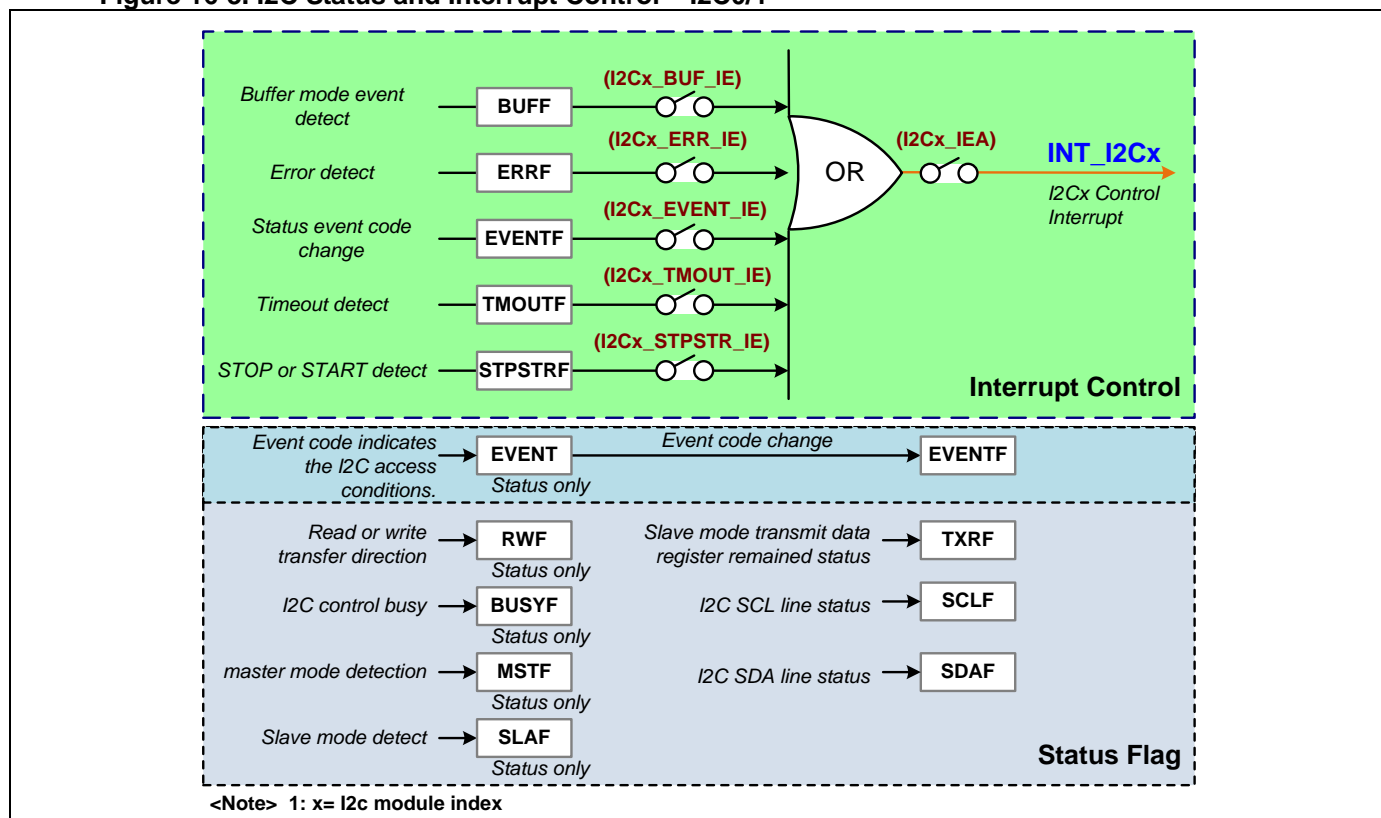
16.7.1. I2C Interrupt Control and Status

These interrupt flags are using for the interrupt service routine (ISR) flow control. Generally these flags are set by hardware and software clears them when the serviced job is completed in the related ISR. Each interrupt flag has one interrupt enable bit. User can enable or disable it. There is one interrupt all enable bit of **I2Cx_IEA** to enable or disable all the interrupt sources for this module.

There are some status bits those are reading only to provide internal control status. One busy flag of **I2Cx_BUSYF** is used to indicate the data transfer busy status. Refer the register descriptions of the related status bits for more information.

During STOP mode, the I2C control module can detect the I2C salve address which is set in **I2Cx_SADR** or **I2Cx_SADR2** registers. When the owned I2C salve address is detected, the chip will be waked up if the **I2Cx_WUP_IE** and **I2Cx_IEA** registers are enabled.

Figure 16-3. I2C Status and Interrupt Control – I2C0/1



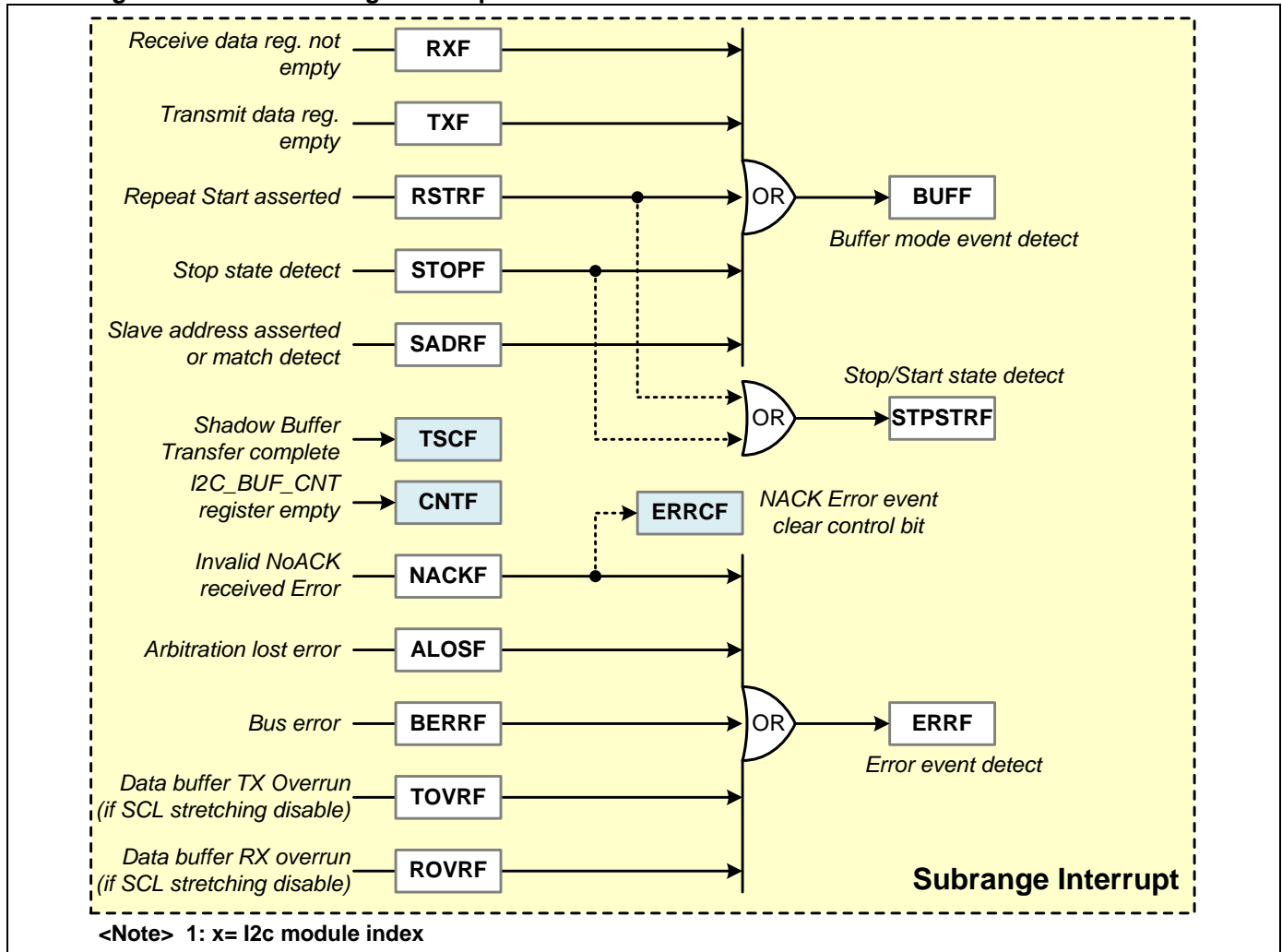
16.7.2. I2C Subrange Interrupt

There are some subrange interrupt flags for the interrupt flags of **I2Cx_BUFF**, **I2Cx_ERRF** and **I2Cx_STPSTRF**. These subrange interrupt flags can be asserted by hardware and induce to assert the upper level flag of **I2Cx_BUFF**, **I2Cx_ERRF** and **I2Cx_STPSTRF** for interrupt generation. User can use these upper level flags for interrupt flow control and no any hardware event control.

The flags of **I2Cx_TSCF**, **I2Cx_CNTF** and **I2Cx_ERRCF** are set by chip for firmware control only. They do not do as interrupt events.

The following diagram is showing the I2C subrange interrupt control block.

Figure 16-4. I2C Subrange Interrupt Control – I2C0/1



16.7.3. I2C Interrupt Flags

Generally these interrupt flags are set by hardware and clear by software writing 1. Refer the register descriptions for more information about the related interrupt flags and interrupt enable bit.

- **EVENTF**

I2C status event interrupt Flag (**I2Cx_EVENTF**). There is a related interrupt enable register bit of **I2Cx_EVENT_IE**. This flag is using for I2C byte mode interrupt control. It is asserted each time the event code register of **I2Cx_EVENT** is updated by hardware. There is an **I2Cx_EVENTF2** flag which is identical to **I2Cx_EVENTF** flag and is designed in the same 32-bit register with I2C_EVENT (event code value) for firmware easy using.

- **BUFF**

I2C buffer mode event flag (**I2Cx_BUFF**). There is a related interrupt enable register bit of **I2Cx_BUFF_IE**. This flag is using for I2C buffer mode interrupt control. When any the flag of RXF, TXF, RSTRF, STOPF or SADRF is asserted, this flag is induced to assert and generates the interrupt event if the **I2Cx_BUFF_IE** bit is enabled.

- **RXF**

I2C receive data register not empty flag (**I2Cx_RXF**). This is a subrange interrupt flag for BUFF interrupt flag. When the shadow buffer receives data, hardware will copy the shadow buffer data to data register **I2Cx_DAT** and this flag is set. This bit is cleared when **I2Cx_DAT** is read or this flag set to 1 by software.

Notify: The RXF flag is not cleared when I2Cx_DAT is read by SWD debugging.

- **TXF**

I2C transmit data register empty flag (**I2Cx_TXF**). This is a subrange interrupt flag for BUFF interrupt flag. When the shadow buffer is empty for transmission, the data register **I2Cx_DAT** will copy to the shadow buffer and this flag is set. This bit is cleared when **I2Cx_DAT** is written or this flag set to 1 by software.

— SADRf

I2C slave address matched flag (**I2Cx_SADRf**). This is a subrange interrupt flag for BUFF interrupt flag. This flag is also asserted for master mode if transmit mode slave address unmatched or received mode slave address asserted. When wakeup from **STOP** mode by detection matched slave address, user needs to clear this bit to disable the clock stretching and releases clock signal for external master.

— RSTRf

I2C repeat Start asserted flag (**I2Cx_RSTRf**). This is a subrange interrupt flag for BUFF interrupt flag. When hardware detect the I2C repeat Start condition, this flag is set.

— STOPF

I2C Stop detection (**I2Cx_STOPF**). This is a subrange interrupt flag for BUFF interrupt flag. When hardware detect the I2C Stop condition, this flag is set.

● ERRF

I2C error interrupt flag (**I2Cx_ERRF**). There is a related interrupt enable register bit of **I2Cx_ERR_IE**. It indicates any of invalid no ack, bus arbitration lost bus error or data overrun error. When any the flag of RXOVRF, TXOVRF, BERRF, ALOSF or NACKF is asserted, this flag is induced to assert and generates the interrupt event if the **I2Cx_ERR_IE** bit is enabled.

— ROVRF

I2C receive overrun flag (**I2Cx_ROVRF**). This is a subrange interrupt flag for ERRF interrupt flag. When receives overrun, hardware will stop to receive next data into data shadow buffer until this flag is cleared.

— TOVRF

I2C transmit underrun flag (**I2Cx_TXOVRF**). This is a subrange interrupt flag for ERRF interrupt flag. When transmits underrun and I2C is slave mode, hardware will send 0xFF value for next data until the data buffer is not empty.

— NACKF

I2C “Not Acknowledge” received error flag (**I2Cx_NACKF**). This is a subrange interrupt flag for ERRF interrupt flag.

— ALOSF

I2C bus arbitration lost error flag (**I2Cx_ALOSF**). This is a subrange interrupt flag for ERRF interrupt flag.

— BERRF

I2C bus error flag for invalid Stop/Start state (**I2Cx_BERRF**). This is a subrange interrupt flag for ERRF interrupt flag.

● TMOUTF

I2C bus timeout detect flag (**I2Cx_TMOUTF**). There is a related interrupt enable register bit of **I2Cx_TMOUT_IE**.

● WUPF

STOP mode wakeup by I2C event detect flag (**I2Cx_WUPF**). There is a related interrupt enable register bit of **I2Cx_WUP_IE**. When hardware detect that the slave address is matched to the setting of **I2Cx_SADR** (**I2Cx_SADR_EN**=1) or **I2Cx_SADR2** (**I2Cx_SADR2_EN**=1) during **STOP** mode, this flag is asserted.

● STPSTRF

I2C Stop or Start detection flag (**I2Cx_STPSTRF**). There is a related interrupt enable register bit of **I2Cx_STPSTR_IE**. When any the flag of RSTRF or STOPF is asserted, this flag is induced to assert and generates the interrupt event if the **I2Cx_STPSTR_IE** bit is enabled.

16.7.4. I2C Control Flags

The flags of **I2Cx_TSCF**, **I2Cx_CNTF** and **I2Cx_ERRCF** are set by chip for firmware control only. They do not do as interrupt events.

● TSCF

I2C shadow buffer transfer complete flag (**I2Cx_TSCF**). This flag is set by hardware and clear by hardware or software writing 1. This flag is using for internal hardware control.

● CNTF

I2C buffer count **I2Cx_BUF_CNT** register empty status (**I2Cx_CNTF**). This flag is set by hardware and clear by software writing 1.

● ERRCF

I2C master mode NACK error flag and state control bit (**I2Cx_ERRCF**). This flag is set by hardware and clear by software writing 1 or hardware auto clear during START/STOP state. This bit is asserted if occurs NACK during slave-address cycle or data cycle of receive access.

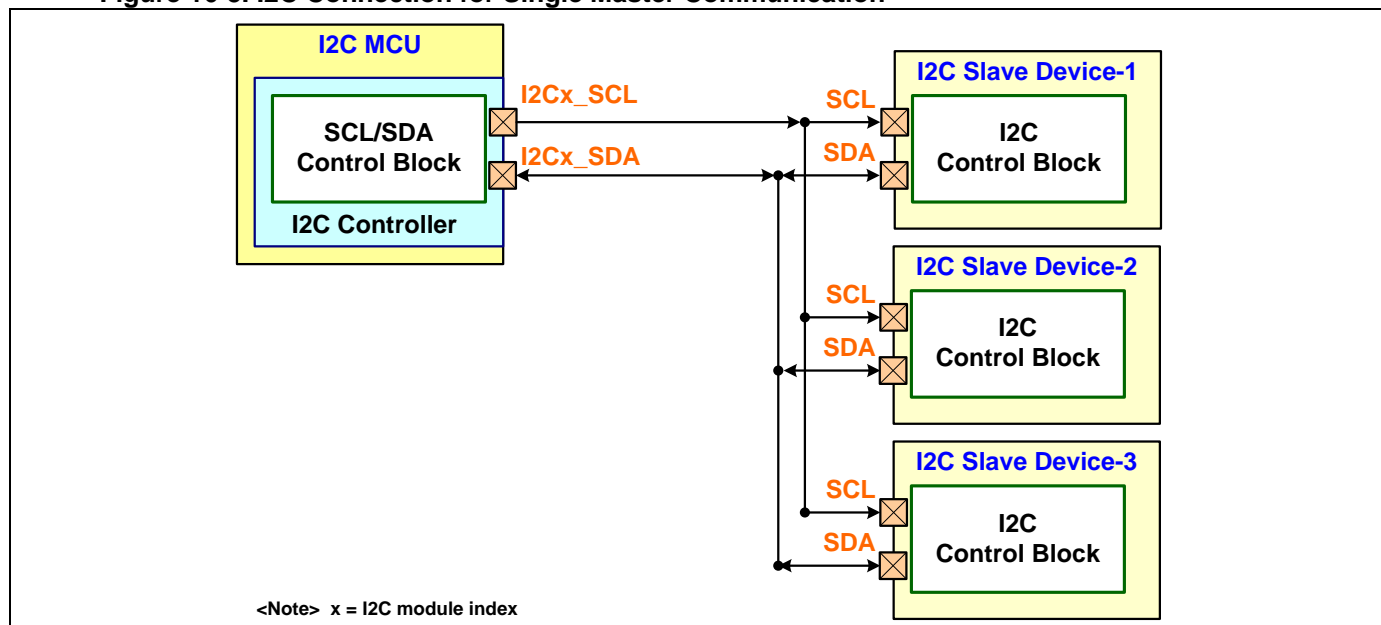
16.8. I2C Connection for Application

Usually user need set the IO mode of the using IO pins to open-drain for I2C SCL and SDA signals.

The following diagrams are showing the I2C application connections of Single Master, Multi-Master and Slave Mode communication. There are two read only status bits to directly reflect the I2C SCL and SDA line status in the I2C module. User can read these two status bits in **I2Cx_SCLF** and **I2Cx_SDAF** register bits for firmware using.

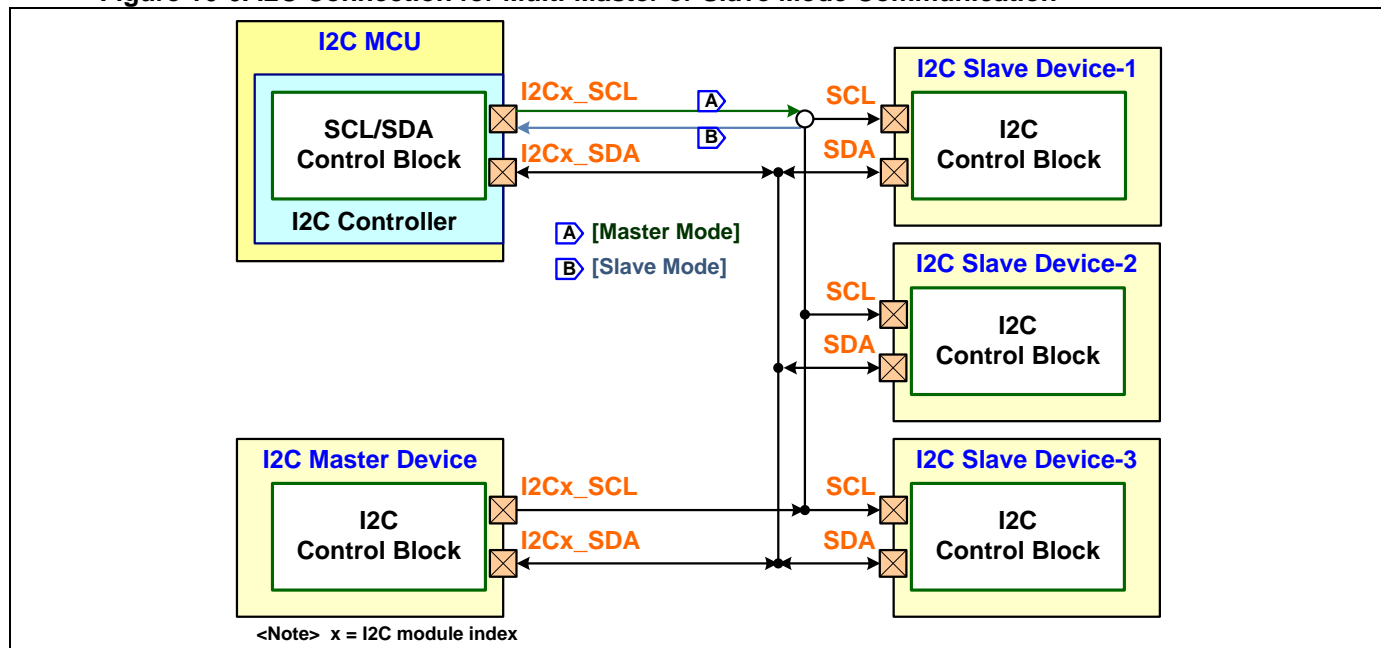
- I2C Connection for Single Master Communication

Figure 16-5. I2C Connection for Single Master Communication



- I2C Connection for Multi-Master and Slave Mode Communication

Figure 16-6. I2C Connection for Multi-Master or Slave Mode Communication



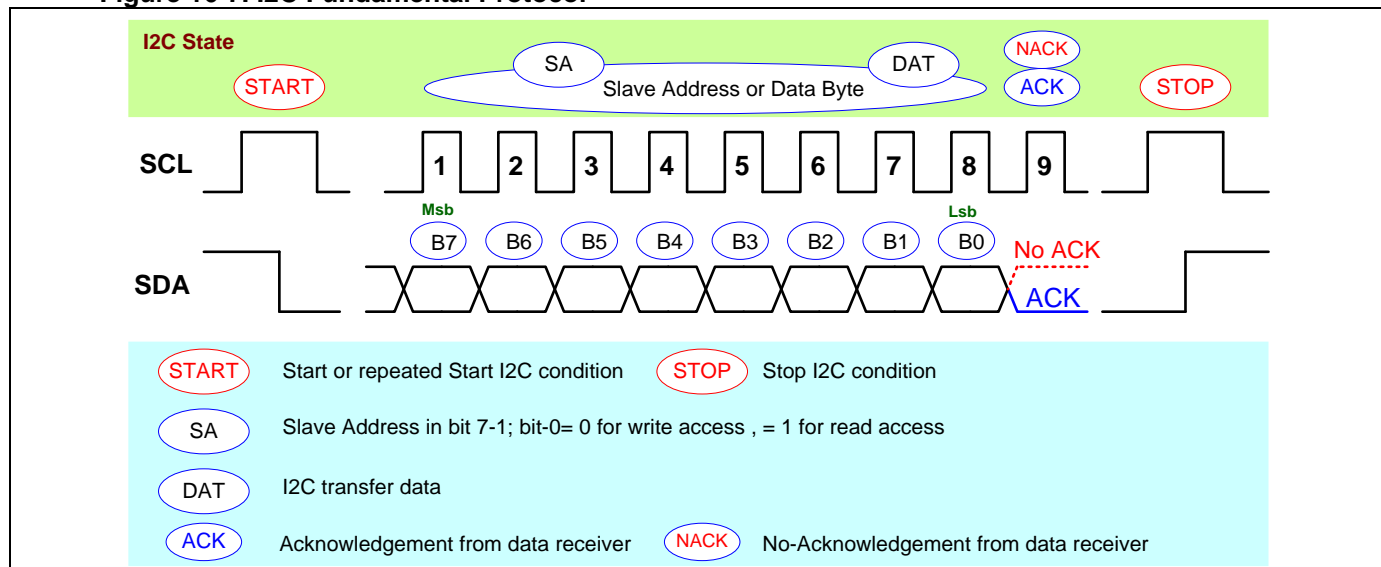
16.9. I2C Fundamental Control

16.9.1. I2C Fundamental Protocol

The fundamental function of the I2C controller is designed by the standard I2C communication protocol. Refer the “I2C-bus specification and user manual” for more information about the I2C protocol.

The following diagram is showing the I2C fundamental protocol.

Figure 16-7. I2C Fundamental Protocol



16.9.2. I2C Control Mode

The I2C supports the operation modes of single master/multi-master/slave mode by setting **I2Cx_MDS** register. The multi-master mode is only supported for Byte mode only.

The I2C supports the optional data control mode of Byte mode and Buffer mode by setting **I2Cx_BUF_EN** register. When selects Byte mode, firmware can easy use the event flag of EVENTF (**I2Cx_EVENTF**) and event code register of **I2Cx_EVENT** for data transfer flow control. When Buffer mode is enabled, a shadow buffer is using to speed up for data flow control. The RXF and TXF flags will use to indicate the data register receiving not-empty and transmission empty.

16.9.3. I2C Slave Address

The I2C controller supports two sets of slave address for slave mode address detection. User can set these two slave addresses in **I2Cx_SADR** and **I2Cx_SADR2** registers and enable independently by setting **I2Cx_SADR_EN** and **I2Cx_SADR2_EN** registers. For the I2C slave address in **I2Cx_SADR** register, the mask register of **I2Cx_SA_MSK** is provided to configure and mask address bits of the receiving slave address for I2C slave mode. The register zero bit makes the result which the related address bit is considered as 'don't care' for comparison. The mask register is no effect on **I2Cx_SADR2** register setting. Also the I2C controller supports to detect the I2C general call address **0x00** and is able to enable independently in **I2Cx_GC_EN** register.

User can get the detected slave address code in **I2Cx_SA_CODE** register for I2C slave mode. When operates in slave mode, the I2C controller will grab the slave address code into this register always.

16.9.4. I2C Data Register and Shift Buffer

One data registers of **I2Cx_DAT** is designed for 8/16/32-bit data access. When buffer mode is enabled, reads the data register will clear the RXF flag and writes the data register will clear TXF flag.

The I2C controller implements one 8-bit shift buffer for data receiving and transmission. There is one register of **I2Cx_SBUF** which can be read to get the data bits of received and transmitted shift buffer in real time.

16.9.5. I2C Access Command

There are three command register bits of STA/STO/AA to control the I2C Start, Stop and ACK states for the I2C data receiving and transmission. These bits are implemented in **I2Cx_STA**, **I2Cx_STO** and **I2Cx_AA** registers. Specially, the **I2Cx_STA** register bit can be written only when the protected bit of **I2Cx_STA_LCK** is written "1" simultaneously. Also the **I2Cx_STO** and **I2Cx_AA** register bits have the same protected function with independent register bits of **I2Cx_STO_LCK**, **I2Cx_AA_LCK**.

- **STA - I2C START enable bit**

When the STA bit is set to enter a master mode, the I2C hardware checks the status of the serial bus and generates a START condition if the bus is free. If the bus is not free, then I2C waits for a STOP condition and generates a START condition after a delay. If STA is set while I2C is already in a master mode and one or more bytes are transmitted or received, I2C transmits a repeated START condition. STA may be set at any time. STA may also be set when I2C is an addressed slave. When the STA bit is reset, no START condition or repeated START condition will be generated.

- **STO - I2C STOP enable bit**

When the STO bit is set while I2C is in a master mode, a STOP condition is transmitted to the serial bus. When the STOP condition is detected on the bus, the I2C hardware clears the STO flag. In a slave mode, the STO flag may be set to recover from a bus error condition. In this case, no STOP condition is transmitted to the bus. However, the I2C hardware behaves as if a STOP condition has been received and switches to the defined not addressed slave receiver mode. The STO flag is automatically cleared by hardware. If the STA and STO bits are both set, then a STOP condition is transmitted to the bus if I2C is in a master mode (in a slave mode, I2C generates an internal STOP condition which is not transmitted), and then transmits a START condition.

- **AA - I2C assert Acknowledge enable bit**

If the AA bit is set to '1', an ACK will be returned during the ACK clock pulse on the SCL line when:

- (1) The own slave address has been received.
- (2) A data byte has been received while I2C is in the master/receiver mode.
- (3) A data byte has been received while I2C is in the addressed slave/receiver mode.

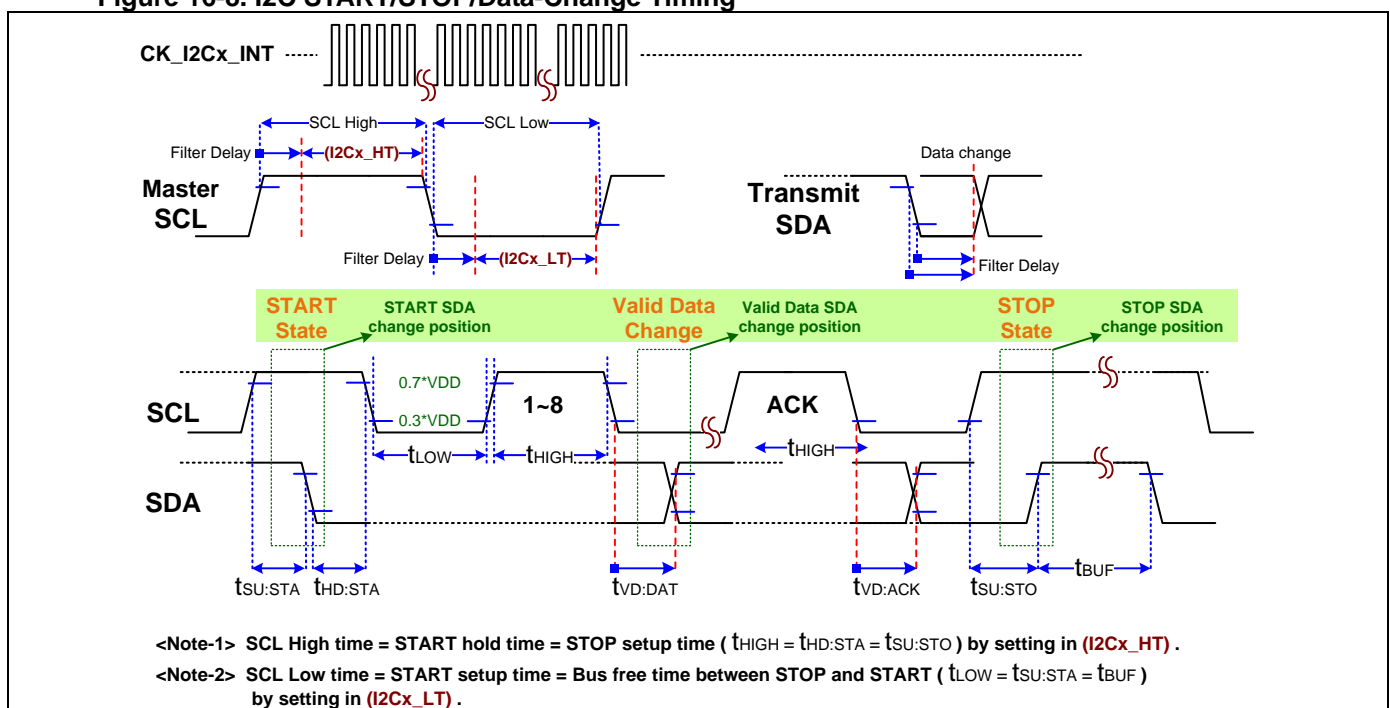
If the AA bit is reset to '0', a NACK will be returned during the ACK clock pulse on SCL when:

- (1) A data has been received while I2C is in the master/receiver mode.
- (2) A data byte has been received while I2C is in the addressed slave/receiver mode.

16.9.6. I2C START/STOP/Data-Change

Two registers of **I2Cx_HT** and **I2Cx_LT** are simply used to configure the I2C master mode timing. User can set the SCL high time by setting **I2Cx_HT** register and set the SCL low time by setting **I2Cx_LT** register.

Figure 16-8. I2C START/STOP/Data-Change Timing



16.10. I2C Byte Mode Control

User can easy use the event flag of EVENTF (**I2Cx_EVENTF**) and event code register of **I2Cx_EVENT** for data transfer flow control.

16.10.1. I2C Byte Mode Transmit and Receive

The EVENTF flag is asserted each time the hardware detects the new I2C condition or error event. At the time, also the event code register of **I2Cx_EVENT** is updated and the I2C controller will generate an interrupt event if the interrupt enable bit of **I2Cx_EVENT_IE** is enabled. User can read the **I2Cx_EVENT** register to get an event code which indicates the detected I2C event.

User can implement the I2C communication firmware by interrupt event trigger and read the I2C event code to get the I2C current state. User can set I2C access commands of STA/STO/AA step by step in **I2Cx_STA**, **I2Cx_STO** and **I2Cx_AA** registers for I2C communication control.

Refer the section of "[I2C Access Command](#)" for more information about the I2C access command.

16.10.2. I2C Event Code

The following table is showing the I2C Event Code of Summary mode.

● I2C Event Code Summary

Table 16-3. I2C Event Code Table

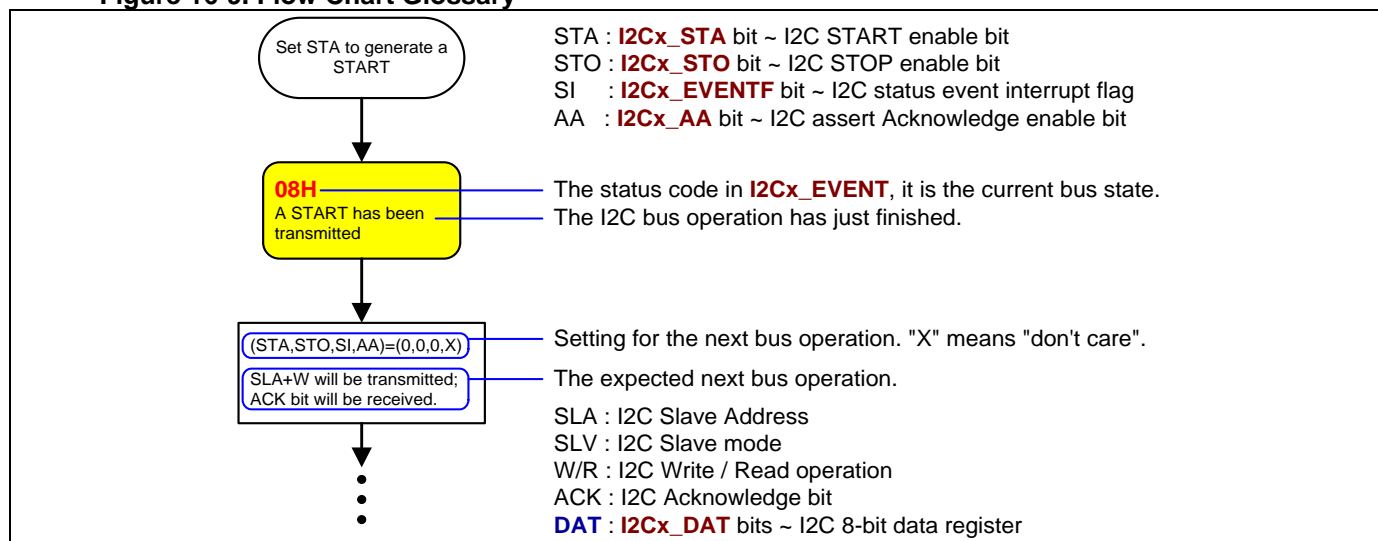
Event Code	Hardware and Bus Status	Master/Slave Mode			
		MT	MR	SR	ST
0x00	Bus error (due to an illegal START or STOP condition)	V	V	V	V
0x08	START transmitted	V	V		
0x10	Repeated START transmitted	V	V		
0x18	SLA+W transmitted and ACK received	V			
0x20	SLA+W transmitted and NoACK received	V			
0x28	DAT transmitted and ACK received	V			
0x30	DAT transmitted and NoACK received	V			
0x38	Arbitration lost in SLA+W or DAT	V			
	Arbitration lost in SLA+R or DAT		V		
	Arbitration lost in NACK bit		V		
0x40	SLA+R transmitted and ACK received		V		
0x48	SLA+R transmitted and NoACK received		V		
0x50	DAT received and ACK received		V		
0x58	DAT received and NoACK received		V		
0x60	Own SLA+W received and ACK returned			V	
0x68	Own SLA+W received, Arbitration lost and ACK returned			V	
0x70	General Call address received and ACK returned			V	
0x78	General Call address received and Arbitration lost			V	
0x80	Previously addressed with own SLA, DAT received and ACK returned			V	
0x88	Previously addressed with own SLA, DAT received and NoACK returned			V	
0x90	DAT received and ACK returned (General Call)			V	
0x98	DAT received and NoACK returned (General Call)			V	
0xA0	STOP or Repeated START received			V	
0xA8	Own SLA+R received and ACK returned				V
0xB0	Own SLA+R received and Arbitration lost				V
0xB8	DAT transmitted and ACK received				V
0xC0	DAT transmitted and NoACK received				V
0xC8	Last DAT transmitted and ACK received				V
0xF8	STOP or bus is released ; No relevant state information available (EVENTF = 0 and no interrupt asserted)	V	V	V	V

<Note> DAT = I2Cx_DAT data register (x: module index)
SLA+W/R = slave address with write/read command bit

16.10.3. I2C Byte Mode Control Flow

The following figure is showing the I2C Flow Chart Glossary for following I2C flow charts.

Figure 16-9. Flow Chart Glossary



● I2C Master Transmitter Mode

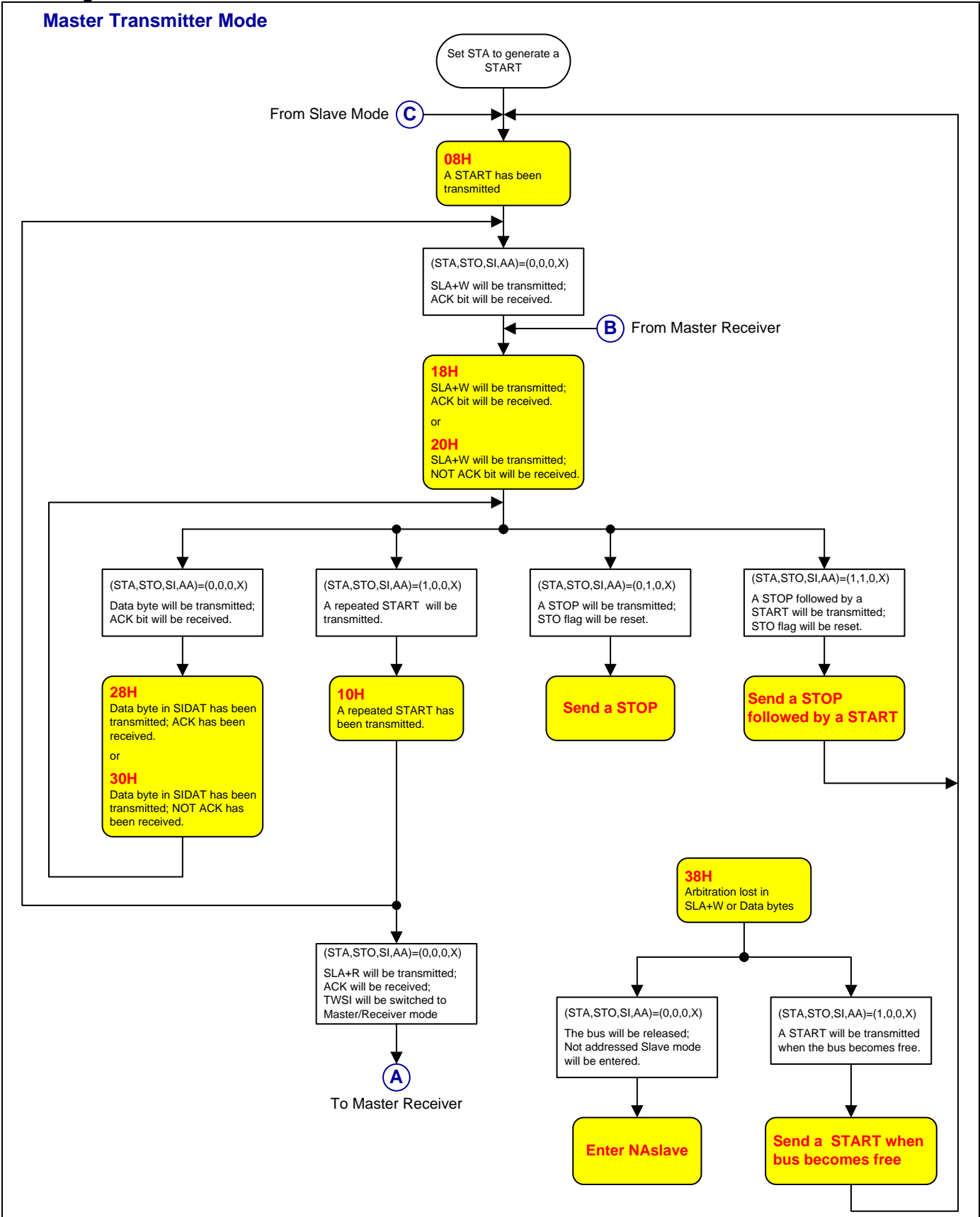
The following table is showing the I2C Event Code and Command of master transmitter mode.

Table 16-4. I2C Master Transmitter Mode Event Table

Event Code	Hardware and Bus Status	To/From I2Cx_DAT	STA	STO	AA	Next Action Taken by Hardware
0x08	A START condition has been transmitted	Load SLA+W	X	0	X	SLA+W will be transmitted; ACK bit will be received
0x10	A repeated START condition has been transmitted.	Load SLA+W	X	0	X	SLA+W will be transmitted; ACK bit will be received
		Load SLA+R; Clear STA	X	0	X	SLA+W will be transmitted; the I2C block will be switched to MASTER receiver mode.
0x18	SLA+W has been transmitted; ACK has been received.	Load data byte	0	0	X	Data byte will be transmitted; ACK bit will be received.
		No DAT action	1	0	X	Repeated START will be transmitted.
		No DAT action	0	1	X	STOP condition will be transmitted; STO flag will be reset.
		No DAT action	1	1	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset.
0x20	SLA+W has been transmitted; NOT ACK has been received.	Load data byte	0	0	X	Data byte will be transmitted; ACK bit will be received.
		No DAT action	1	0	X	Repeated START will be transmitted.
		No DAT action	0	1	X	STOP condition will be transmitted; STO flag will be reset.
		No DAT action	1	1	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset.
0x28	Data byte in DAT has been transmitted; ACK has been received.	Load data byte	0	0	X	Data byte will be transmitted; ACK bit will be received.
		No DAT action	1	0	X	Repeated START will be transmitted.
		No DAT action	0	1	X	STOP condition will be transmitted; STO flag will be reset.
		No DAT action	1	1	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset.
0x30	Data byte in DAT has been transmitted; NOT ACK has been received.	Load data byte	0	0	X	Data byte will be transmitted; ACK bit will be received.
		No DAT action	1	0	X	Repeated START will be transmitted.
		No DAT action	0	1	X	STOP condition will be transmitted; STO flag will be reset.
		No DAT action	1	1	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset.
0x38	Arbitration lost in SLA+R/W or Data bytes.	No DAT action	0	0	X	I2C-bus will be released; not addressed slave will be entered.
		No DAT action	1	0	X	A START condition will be transmitted when the bus becomes free.

<Note> DAT = I2Cx_DAT register (x: module index)
SLA+W/R = slave address with write/read command bit

Figure 16-10. I2C Master Transmitter Mode Flow Chart



- I2C Master Receiver Mode

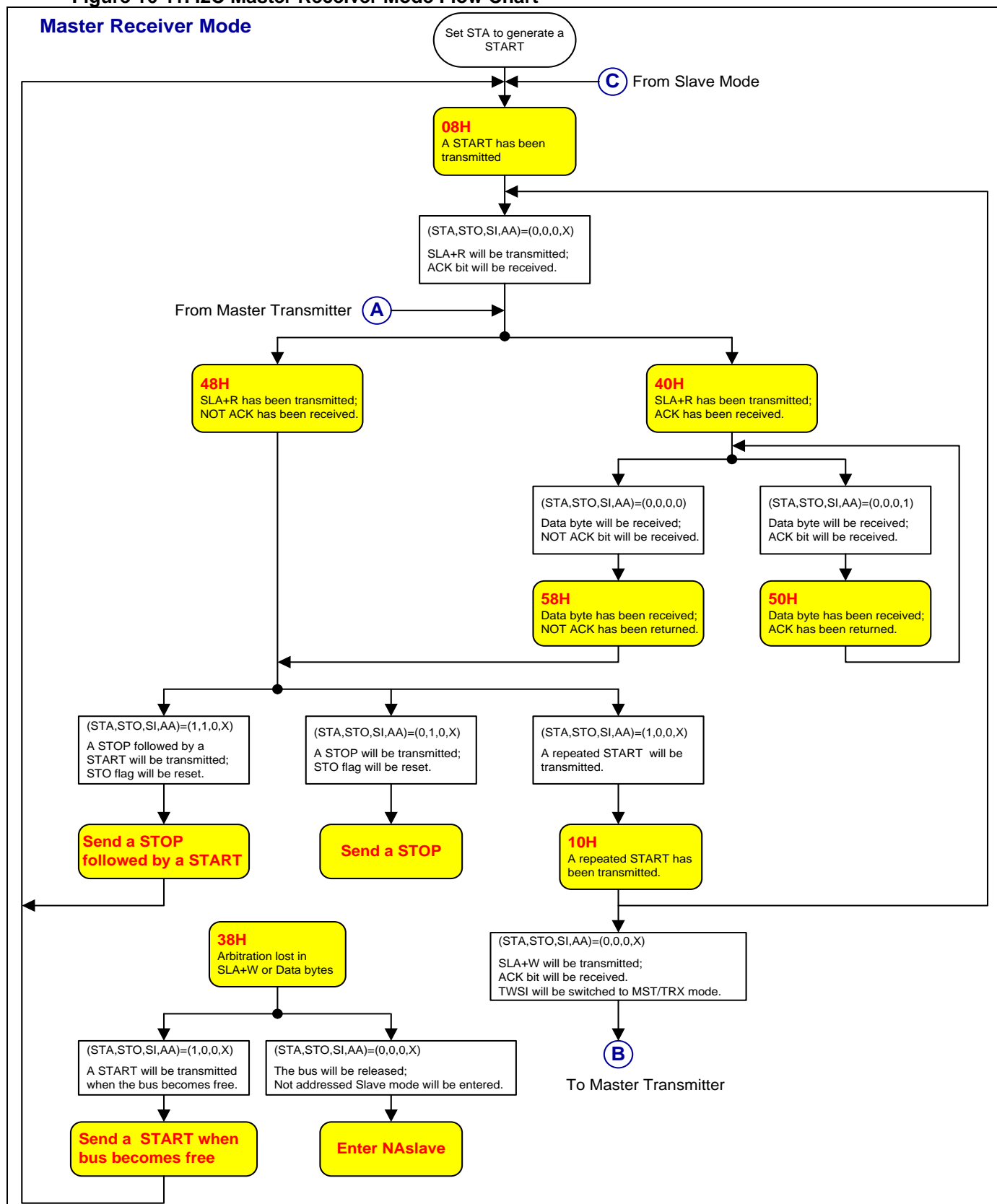
The following table is showing the I2C Event Code and Command of master receiver mode.

Table 16-5. I2C Master Receiver Mode Event Table

Event Code	Hardware and Bus Status	To/From I2Cx_DAT	STA	STO	AA	Next Action Taken by Hardware
0x08	A START condition has been transmitted.	Load SLA+R	X	0	X	SLA+R will be transmitted; ACK bit will be received.
0x10	A repeated START condition has been transmitted.	Load SLA+R	X	0	X	As above.
		Load SLA+W	X	0	X	SLA+W will be transmitted; the I2C block will be switched to MASTER/TRX mode.
0x38	Arbitration lost in NOT ACK bit.	No DAT action	0	0	X	I2C-bus will be released; the I2C block will enter a slave mode.
		No DAT action	1	0	X	A START condition will be transmitted when the bus becomes free.
0x40	SLA+R has been transmitted; ACK has been received.	No DAT action	0	0	0	Data byte will be received; NOT ACK bit will be returned.
		No DAT action	0	0	1	Data byte will be received; ACK bit will be returned.
0x48	SLA+R has been transmitted; NOT ACK has been received.	No DAT action	1	0	X	Repeated START condition will be transmitted.
		No DAT action	0	1	X	STOP condition will be transmitted; STO flag will be reset.
		No DAT action	1	1	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset.
0x50	Data byte has been received; ACK has been returned.	Read data byte	0	0	0	Data byte will be received; NOT ACK bit will be returned.
		Read data byte	0	0	1	Data byte will be received; ACK bit will be returned.
0x58	Data byte has been received; NOT ACK has been returned.	Read data byte	1	0	X	Repeated START condition will be transmitted.
		Read data byte	0	1	X	STOP condition will be transmitted; STO flag will be reset.
		Read data byte	1	1	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset.

The following figure is showing the suggested flow chart of I2C master receiver mode.

Figure 16-11. I2C Master Receiver Mode Flow Chart



- **I2C Slave Transmitter Mode**

The following table is showing the I2C Event Code and Command of slave transmitter mode.

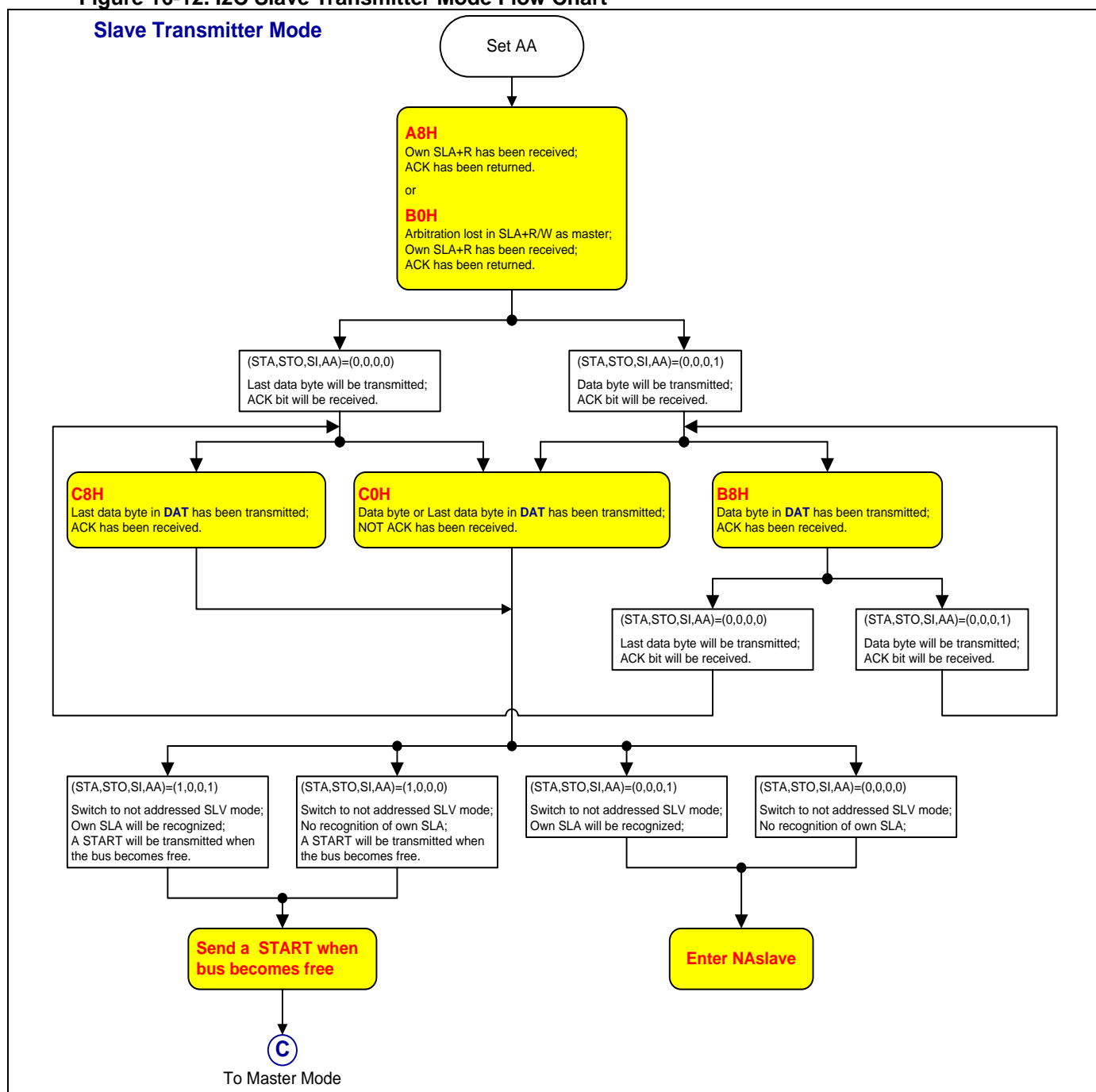
Table 16-6. I2C Slave Transmitter Mode Event Table

Event Code	Hardware and Bus Status	To/From I2Cx_DAT	STA	STO	AA	Next Action Taken by Hardware
0xA8	Own SLA+R has been received; ACK has been returned.	Load data byte	X	0	0	Last data byte will be transmitted and ACK bit will be received.
		Load data byte	X	0	1	Data byte will be transmitted; ACK will be received.
0xB0	Arbitration lost in SLA+R/W as master; Own SLA+R has been received, ACK has been returned	Load data byte	X	0	0	Last data byte will be transmitted and ACK bit will be received.
		Load data byte	X	0	1	Data byte will be transmitted; ACK will be received.
0xB8	Data byte in DAT has been transmitted; ACK has been received.	Load data byte	X	0	0	Last data byte will be transmitted and ACK bit will be received.
		Load data byte	X	0	1	Data byte will be transmitted; ACK will be received.
0xC0	Data byte in DAT has been transmitted; NOT ACK has been received.	No DAT action	0	0	01	Switched to not addressed SLAVE mode; no recognition of own SLA or General Call address.
		No DAT action	0	0	1	Switched to not addressed SLAVE mode; Own SLA will be recognized; General Call address will be recognized if ADR[0] = logic 1.
		No DAT action	1	0	0	Switched to not addressed SLAVE mode; no recognition of own SLA or General Call address. A START condition will be transmitted when the bus becomes free.
		No DAT action	1	0	1	Switched to not addressed SLAVE mode; Own SLA will be recognized; General Call address will be recognized if ADR[0] = logic 1. A START condition will be transmitted when the bus becomes free.
0xC8	Last data byte in DAT has been transmitted (AA = 0); ACK has been received.	No DAT action	0	0	0	Switched to not addressed SLAVE mode; no recognition of own SLA or General Call address.
		No DAT action	0	0	1	Switched to not addressed SLAVE mode; Own SLA will be recognized; General Call address will be recognized if ADR[0] = logic 1.
		No DAT action	1	0	0	Switched to not addressed SLAVE mode; no recognition of own SLA or General Call address. A START condition will be transmitted when the bus becomes free.
		No DAT action	1	0	1	Switched to not addressed SLAVE mode; Own SLA will be recognized; General Call address will be recognized if ADR[0] = logic 1. A START condition will be transmitted when the bus becomes free.

The following figure is showing the suggested flow chart of I2C slave transmitter mode.

Figure 16-12. I2C Slave Transmitter Mode Flow Chart

Slave Transmitter Mode



- I2C Slave Receiver Mode

The following table is showing the I2C Event Code and Command of slave receiver mode.

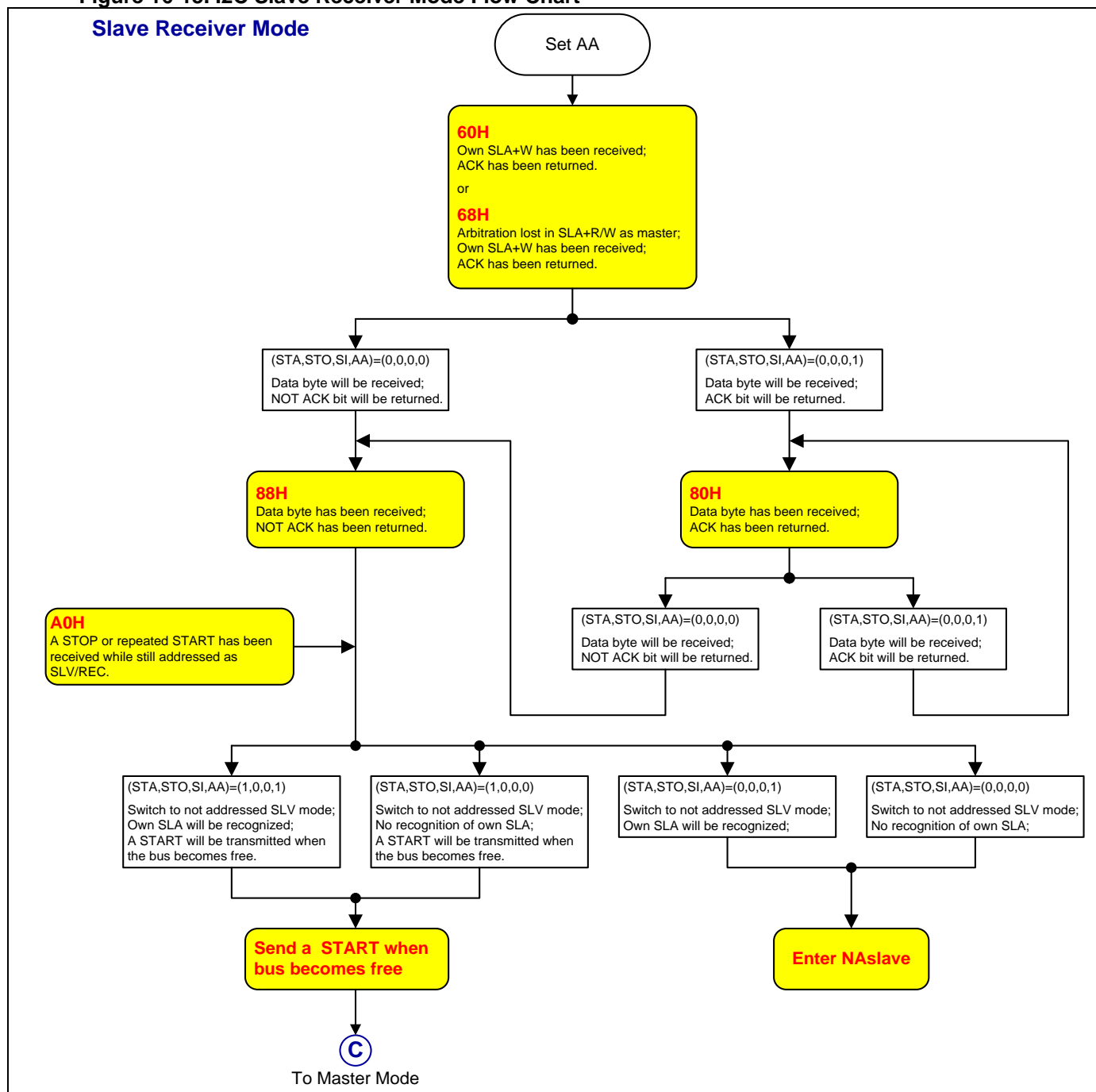
Table 16-7. I2C Slave Receiver Mode Event Table

Event Code	Hardware and Bus Status	To/From I2Cx_DAT	STA	STO	AA	Next Action Taken by Hardware
0x60	Own SLA+W has been received; ACK has been returned.	No DAT action	X	0	0	Data byte will be received and NOT ACK will be returned.
		No DAT action	X	0	1	Data byte will be received and ACK will be returned.
0x68	Arbitration lost in SLA+R/W as master; Own SLA+W has been received, ACK returned.	No DAT action	X	0	0	Data byte will be received and NOT ACK will be returned.
		No DAT action	X	0	1	Data byte will be received and ACK will be returned.
0x70	General Call address (0x00) has been received; ACK has been returned.	No DAT action	X	0	0	Data byte will be received and NOT ACK will be returned.
		No DAT action	X	0	1	Data byte will be received and ACK will be returned.
0x78	Arbitration lost in SLA+R/W as master; General Call address has been received, ACK has been returned.	No DAT action	X	0	0	Data byte will be received and NOT ACK will be returned.
		No DAT action	X	0	1	Data byte will be received and ACK will be returned.
0x80	Previously addressed with own SLA address; DATA has been received; ACK has been returned.	Read data byte	X	0	0	Data byte will be received and NOT ACK will be returned.
		Read data byte	X	0	1	Data byte will be received and ACK will be returned.
0x88	Previously addressed with own SLA; DATA byte has been received; NOT ACK has been returned.	Read data byte	0	0	0	Switched to not addressed SLAVE mode; no recognition of own SLA or General Call address.
		Read data byte	0	0	1	Switched to not addressed SLAVE mode; Own SLA will be recognized; General Call address will be recognized if ADR[0] = logic 1.
		Read data byte	1	0	0	Switched to not addressed SLAVE mode; no recognition of own SLA or General Call address. A START condition will be transmitted when the bus becomes free.
		Read data byte	1	0	1	Switched to not addressed SLAVE mode; Own SLA will be recognized; General Call address will be recognized if ADR[0] = logic 1. A START condition will be transmitted when the bus becomes free.
0x90	Previously addressed with General Call; DATA byte has been received; ACK has been returned.	Read data byte	X	0	0	Data byte will be received and NOT ACK will be returned.
		Read data byte	X	0	1	Data byte will be received and ACK will be returned.
0x98	Previously addressed with General Call; DATA byte has been received; NOT ACK has been returned.	Read data byte	0	0	0	Switched to not addressed SLAVE mode; no recognition of own SLA or General Call address.
		Read data byte	0	0	1	Switched to not addressed SLAVE mode; Own SLA will be recognized; General Call address will be recognized if ADR[0] = logic 1.
		Read data byte	1	0	0	Switched to not addressed SLAVE mode; no recognition of own SLA or General Call address. A START condition will be transmitted when the bus becomes free.
		Read data byte	1	0	1	Switched to not addressed SLAVE mode; Own SLA will be recognized; General Call address will be recognized if ADR[0] = logic 1. A START condition will be transmitted when the bus becomes free.
0xA0	A STOP condition or repeated START condition has been received while still addressed as Slave Receiver or Slave Transmitter.	No DAT action	0	0	0	Switched to not addressed SLAVE mode; no recognition of own SLA or General Call address.
		No DAT action	0	0	1	Switched to not addressed SLAVE mode; Own SLA will be recognized; General Call address will be recognized if ADR[0] = logic 1.
		No DAT action	1	0	0	Switched to not addressed SLAVE mode; no recognition of own SLA or General Call address. A START condition will be transmitted when the bus becomes free.
		No DAT action	1	0	1	Switched to not addressed SLAVE mode; Own SLA will be recognized; General Call address will be recognized if ADR[0] = logic 1. A START condition will be transmitted when the bus becomes

The following figure is showing the suggested flow chart of I2C slave receiver mode.

Figure 16-13. I2C Slave Receiver Mode Flow Chart

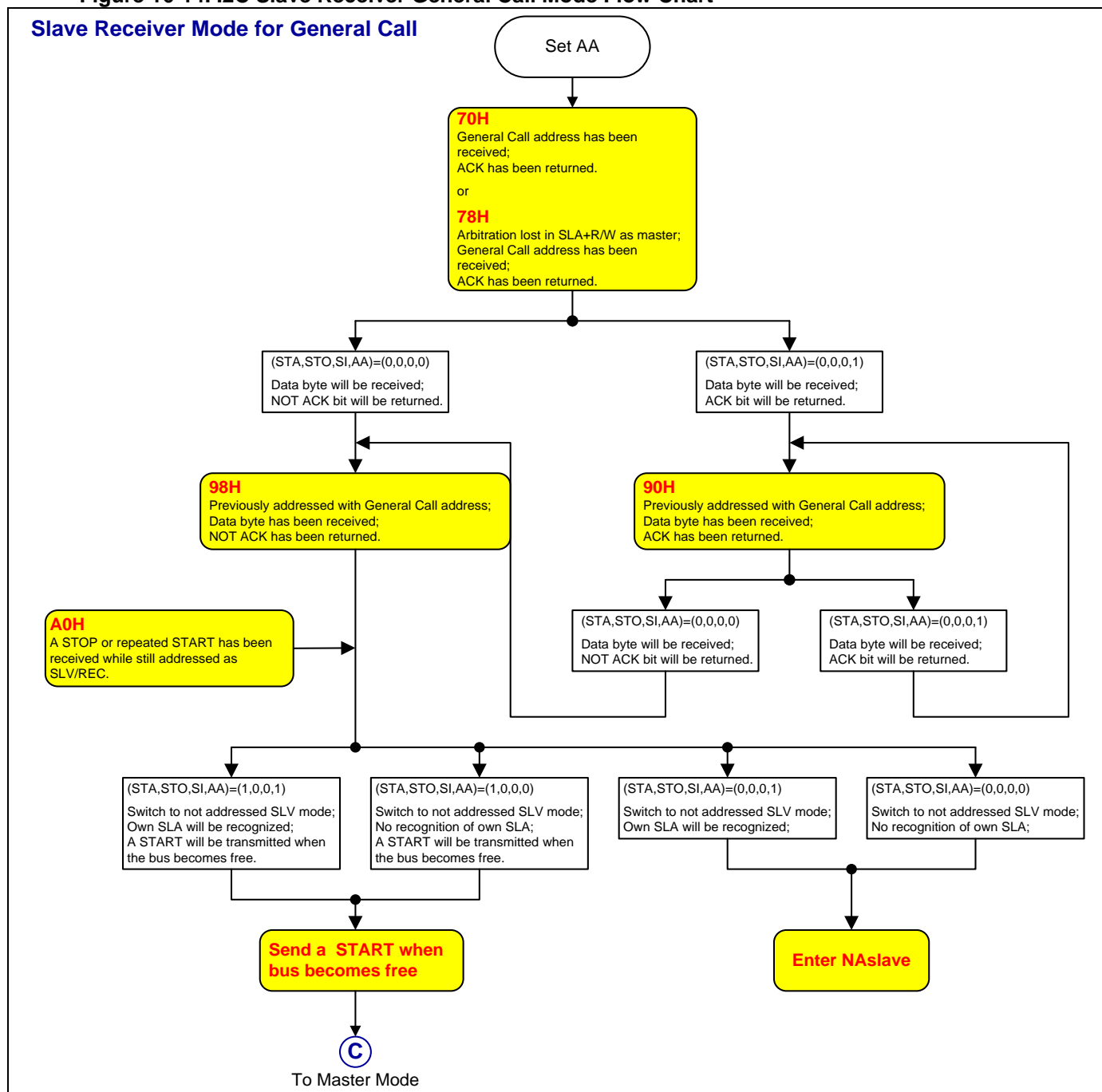
Slave Receiver Mode



● I2C Slave General Call

The following figure is showing the suggested flow chart of I2C slave General Call mode.

Figure 16-14. I2C Slave Receiver General Call Mode Flow Chart



● I2C Miscellaneous Event Table

Table 16-8. I2C Miscellaneous Event Table

Event Code	Hardware and Bus Status	To/From I2Cx_DAT	STA	STO	AA	Next Action Taken by Hardware
0xF8	No relevant state information available; Bus is released; EVENTF = 0 and no interrupt asserted.	No DAT action				Wait or proceed current transfer.
0x00	Bus error during MASTER or selected slave modes, due to an illegal START or STOP condition. State 0x00 can also occur when interference causes the I2C block to enter an undefined state.	No DAT action	0	1	x	Only the internal hardware is affected in the MASTER or addressed SLAVE modes. In all cases, the bus is released and the I2C block is switched to the not addressed SLAVE mode. STO is reset.

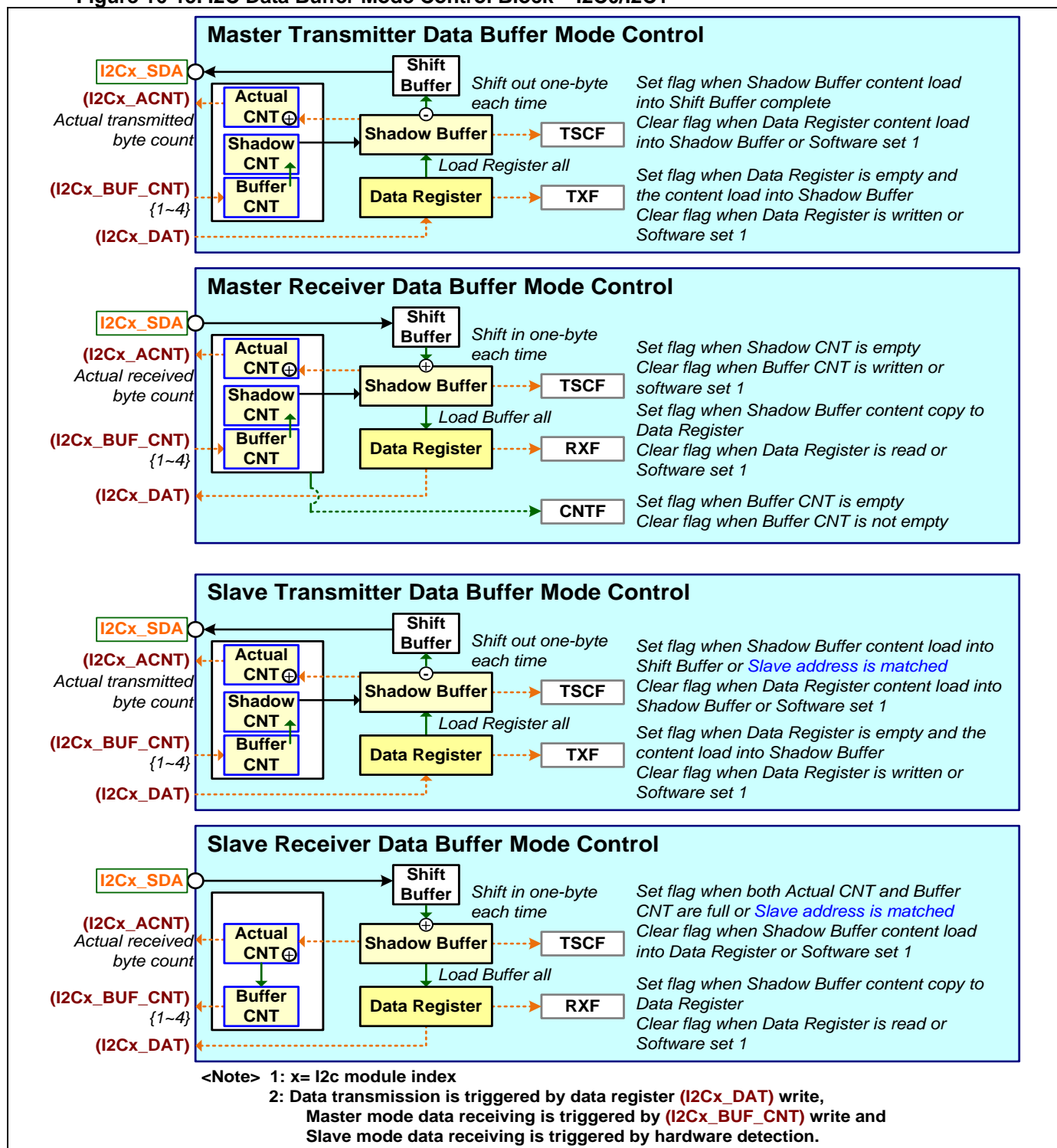
16.11. I2C Buffer Mode Control

User can easy use the event flags of TXF (**I2Cx_TXF**) and RXF (**I2Cx_RXF**) for data transfer flow control.

16.11.1. I2C Buffer Mode Data Buffer

The module implements an 8-bit shift buffer, a 32-bit shadow buffer and a 32-bit data register for data flow control of Data Buffer mode. The following diagram is showing the I2C Data Buffer mode control block.

Figure 16-15. I2C Data Buffer Mode Control Block – I2C0/I2C1



16.11.2. I2C Data Buffer Control

The RXF flag (**I2Cx_RXF**) will be activated at every time that the Shadow Buffer content is copied to data register (**I2Cx_DAT**). Also the related interrupt will be asserted if the **I2Cx_BUFF_IE** register is enabled.

The TXF flag (**I2Cx_TXF**) will be activated at every time that the data register content is copied to Shadow Buffer. Also the related interrupt will be asserted if the **I2Cx_BUFF_IE** register is enabled. The TSCF flag (**I2Cx_TSCF**) will be activated at every time that the Shadow Buffer content is copied to Shift Buffer.

There is one I2C shadow buffer transfer complete TSCF flag (**I2Cx_TSCF**). Usually it is using for internal hardware control only.

For master transmitter mode, this flag is set when shadow buffer content loads into shift buffer completely and clear when data register content load into shadow buffer or software set 1. For master receiver mode, this flag is set when Buffer CNT (shadow buffer count register of **I2Cx_BUF_CNT**) is empty and clear when Buffer CNT is written or software sets 1.

For slave transmitter mode, this flag is set when shadow buffer content load into shift buffer completely or slave address matched. Clear flag when data register content load into shadow buffer or software setting 1. For slave receiver mode, this flag is set when the ACNT (actual count register of **I2Cx_ACNT**) and Buffer CNT are full or slave address is matched. Clear flag when shadow buffer content load into data register Software set 1.

The **I2Cx_BUF_CNT** register is used to do as the transmitted or received data byte count threshold. When transmitted or received data arrives at the threshold and the interrupt enable bit of **I2Cx_BUFF_IE** is enabled, the interrupt is generated. When writes this register, hardware will auto clear the **I2Cx_CNTRF**.

The **I2Cx_ACNT** register can be read to get the transmitted or received data actual byte count value. When transmitted or received data complete by last data transfer or error conditions, the actual transmitted or received data byte number is recorded in this register. The count value is not calculated and included the NACK error byte. For other conditions, this register value is no meaning.

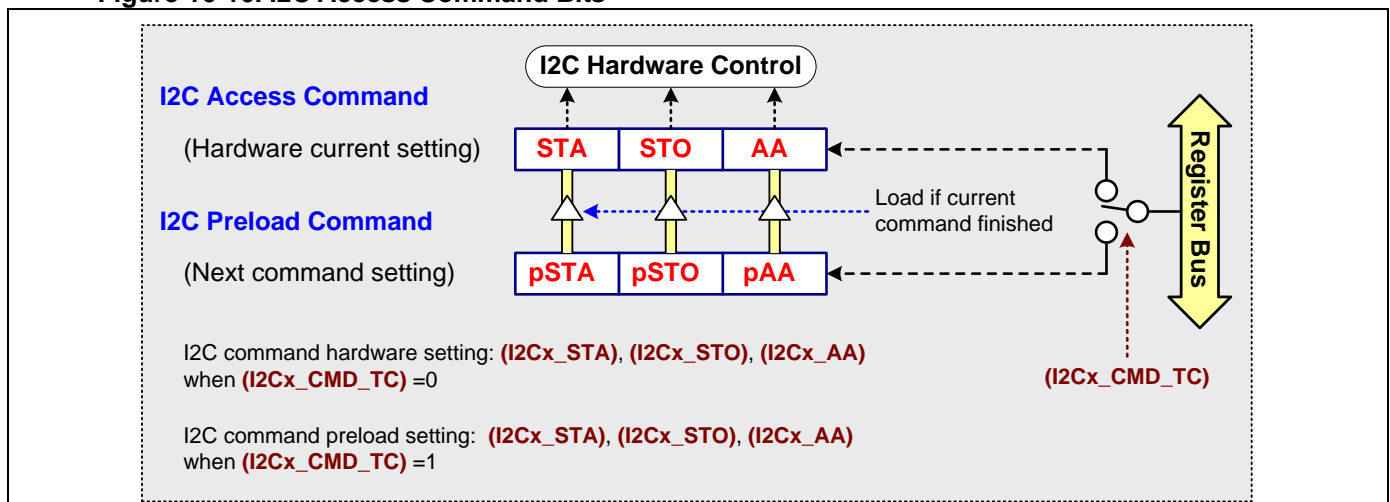
16.11.3. I2C Buffer Mode Access Command

As same as Byte mode, the same three command register bits of STA/STO/AA are used to control the I2C Start, Stop and ACK states for the I2C data receiving and transmission. These bits are implemented in **I2Cx_STA**, **I2Cx_STO** and **I2Cx_AA** registers. Refer the section of "[I2C Access Command](#)" for more information about the I2C access command.

There are extra three command register bits of pSTA/pSTO/pAA are using as the preload command to set for next I2C command. These bits can directly be initialized in the registers of **I2Cx_PSTA**, **I2Cx_PSTO** and **I2Cx_PAA**. Also these bits can be initialized in the same registers of **I2Cx_STA**, **I2Cx_STO** and **I2Cx_AA** by the control of **I2Cx_CMD_TC** register. When the **I2Cx_CMD_TC** register is set 0, writes to the three register bits which will initial the I2C access command for hardware current setting. When the **I2Cx_CMD_TC** register is set 1, writes to the three register bits which will initial the I2C preload command for next command setting.

Though the command bits of STA/STO/AA and pSTA/pSTO/pAA can be initialized in the same register bits of **I2Cx_STA**, **I2Cx_STO** and **I2Cx_AA** by register write access, reads the three register bits that is only able to get the command bits of STA/STO/AA.

Figure 16-16. I2C Access Command Bits



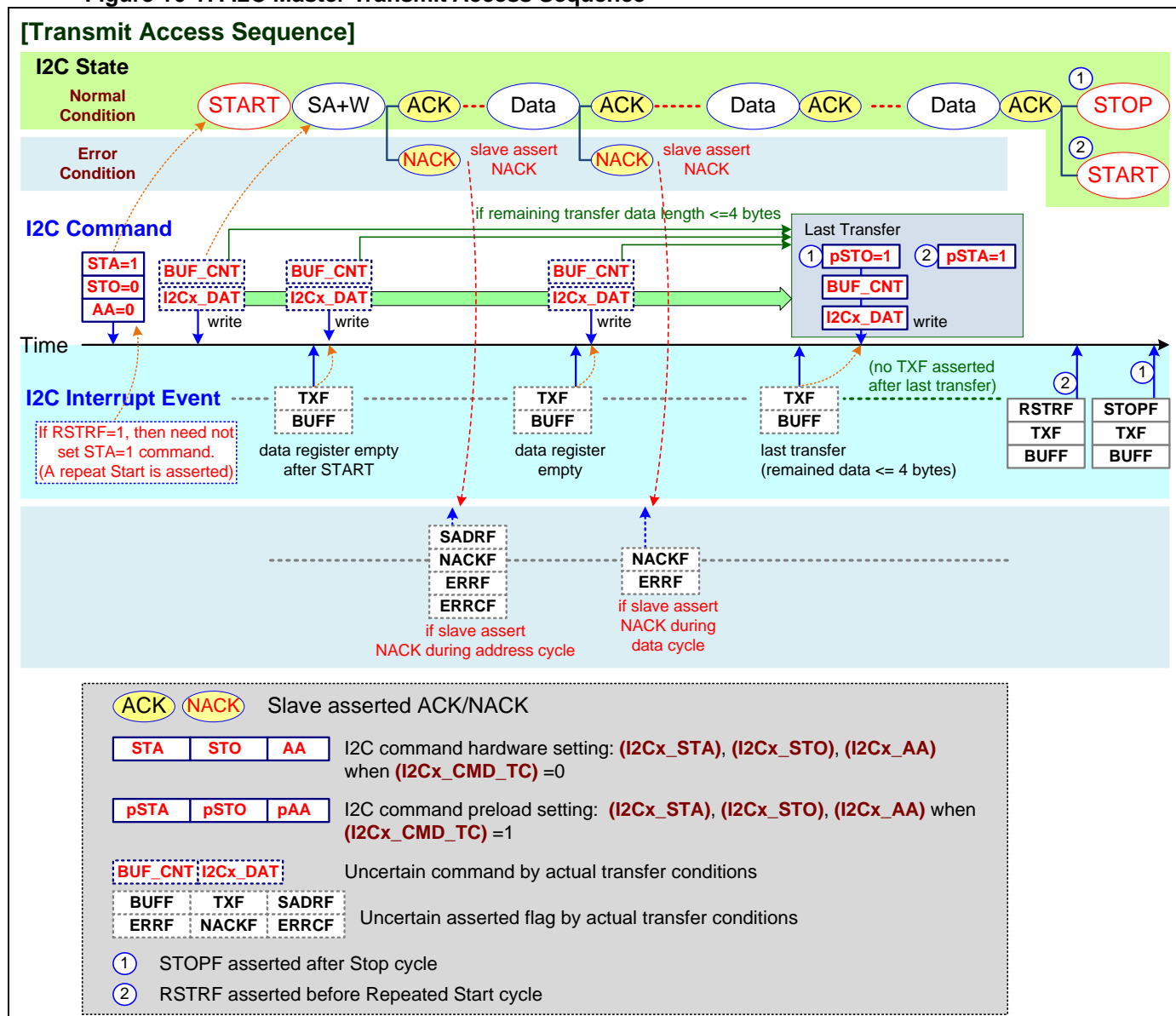
16.11.4. I2C Master Transmit and Receive

● I2C Master Transmit Access Sequence

User can set the STA/STO/AA command to start the I2C master transmission access. According to the interrupt of TXF/BUFF, user fills the transmitted data to I2C data register until last data transmission. User can initialize the preload command to stop I2C master transmission at the last I2C data register written for I2C data transmission.

The following diagram is showing the master mode transmit access sequence.

Figure 16-17. I2C Master Transmit Access Sequence



User can set the STA/STO/AA command to start the I2C master transmission access. According to the interrupt of RXF/BUFF, user gets the received data from I2C data register and returns ACK by setting AA command until last data receiving. User can initialize the preload command to stop I2C master receiving at the last I2C data register read for I2C data receiving.

Figure 16-18. I2C Master Receive Access Sequence



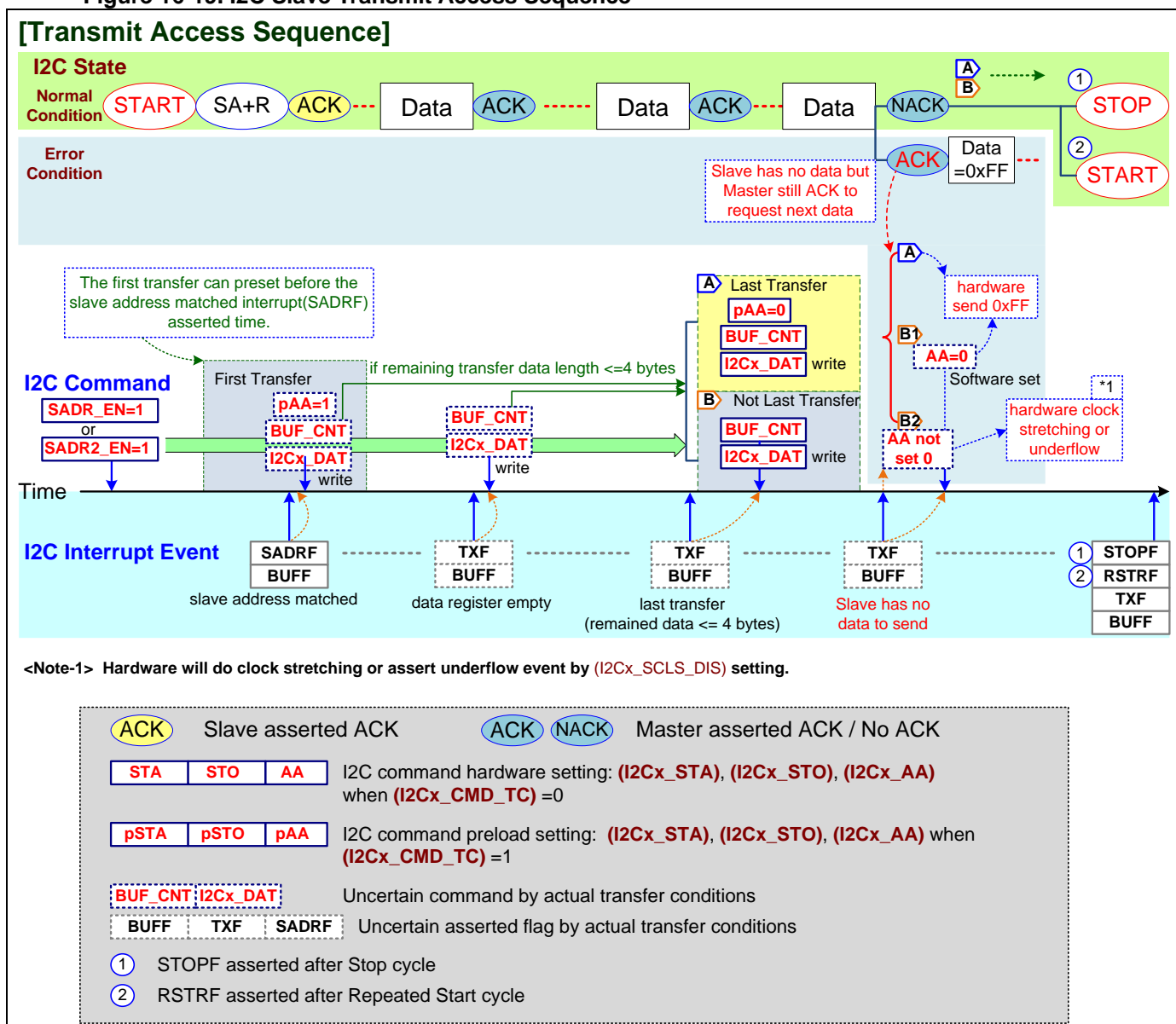
16.11.5. I2C Slave Transmit and Receive

● I2C Slave Transmit Access Sequence

When the I2C controller has detected the matched slave address and hardware asserts the SADRf flag, User can initialize the firmware setting and write the transmitted I2C data to I2C data register beforehand for I2C slave transmit access. According to the interrupt of TXF/BUFF, user fills the transmitted data to I2C data register until last data transmission. If the firmware is able to recognize the last I2C data transmission by the pre-defined data flow protocol, user can initialize the preload pAA command at the last I2C data register written for I2C data transmission.

The following diagram is showing the slave mode transmit access sequence.

Figure 16-19. I2C Slave Transmit Access Sequence



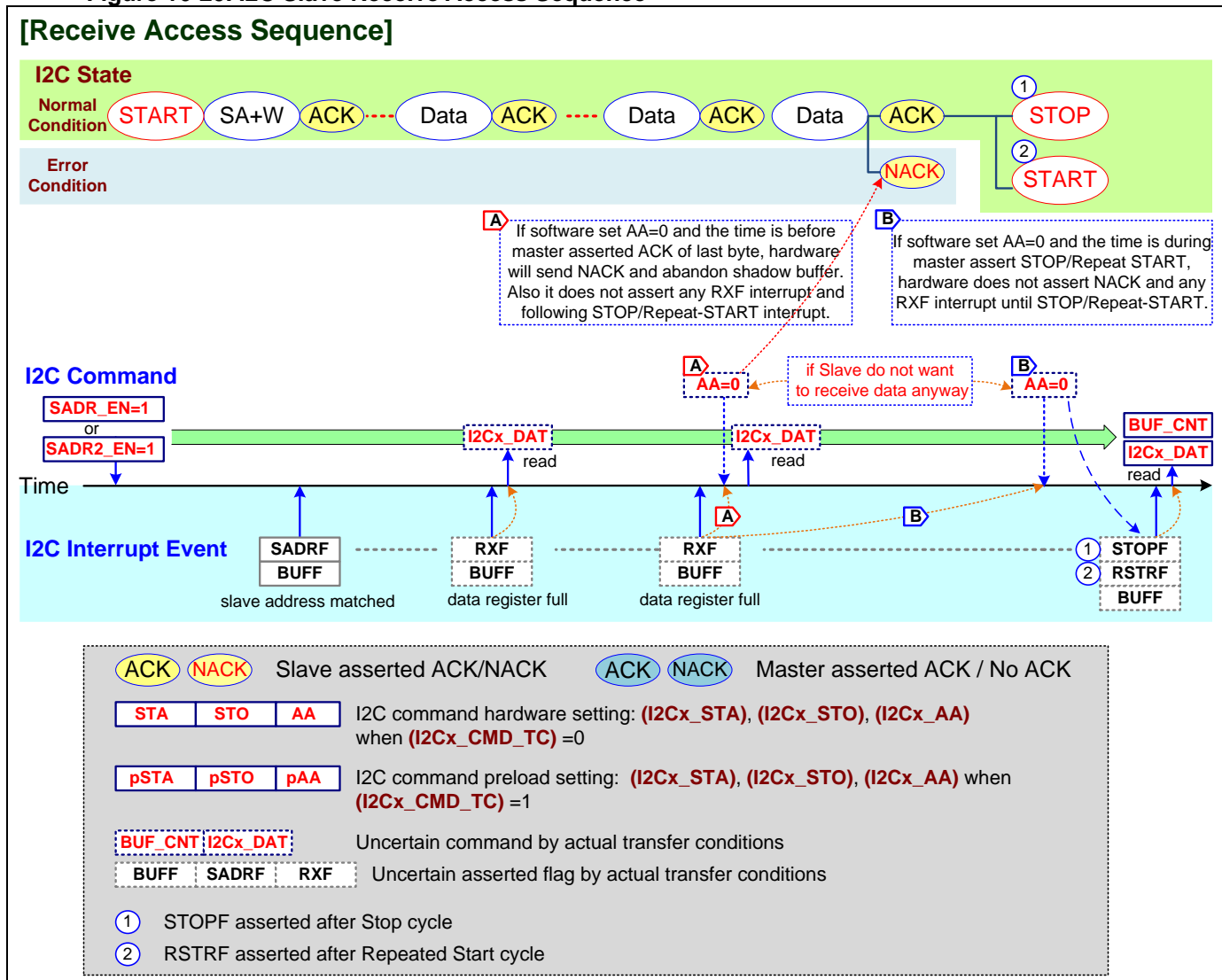
<Note-1> Hardware will do clock stretching or assert underflow event by (I2Cx_SCLSDIS) setting.

● I2C Slave Receive Access Sequence

When the I2C controller has detected the matched slave address and hardware asserts the SADRf flag, User can initialize the firmware setting for I2C slave receive access. According to the interrupt of RXF/BUFF, user gets the received data from I2C data register and returns ACK by setting AA command until last data receiving.

The following diagram is showing the slave mode receive access sequence.

Figure 16-20. I2C Slave Receive Access Sequence



16.12. I2C Function Control

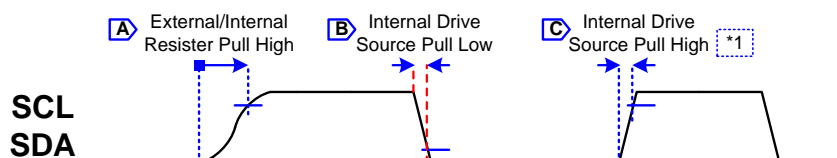
16.12.1. I2C Signal Drive Control Timing

The I2C controller supports to drive SCL and SDA signals for high level. User can select pre-drive time of 0T (disable pre-drive), 1T, 2T, and 3T in **CK_I2Cx** clock time by setting **I2Cx_PDRV_SEL** register.

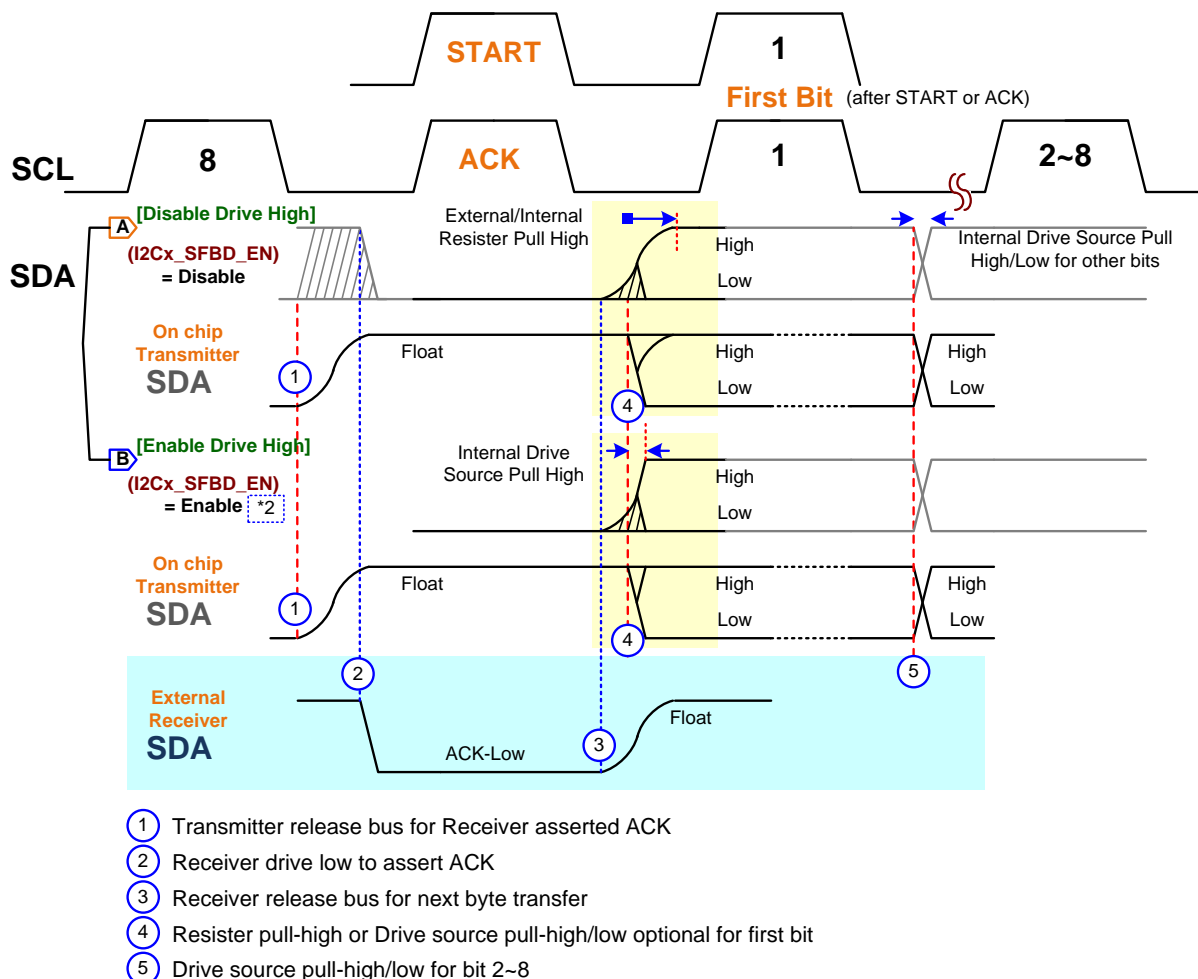
There is an **I2Cx_SFBD_EN** register to enable drive-high for SDA first bit when data is transmitted. This bit is no effect and disabled when **I2Cx_PDRV_SEL=0**.

Figure 16-21. I2C Signal Drive Control Timing

[Signal Pull Types]



[SDA First Bit Drive High Option for transmitter]



<Note-1> When (I2Cx_PDRV_SEL)=0T, chip will pull high by internal resistor for both SCL and SDA signals.

When (I2Cx_PDRV_SEL)≠0T, chip will pre-drive high for both SCL and SDA signals.

<Note-2> To set (I2Cx_SFBD_EN)=Enable is no effect if (I2Cx_PDRV_SEL)=0T and force to disable this function.

The following table is showing the output driving setting for I2C SCL and SDA signals.

Table 16-9. I2C Signal Output Driving Setting

I2C Signal	I2Cx Register		Address/Data Bits	
	PDRV_SEL	SFBD_EN	Master mode	Slave mode
SCL	0	x	ODO	
	1,2,3	x	PDRV	ODO
I2C Signal	I2Cx Register		Data 1st Bit	Address Bits Data th2~8 Bits
	PDRV_SEL	SFBD_EN	Master/Slave	Master/Slave
SDA	0	x	ODO	
	1,2,3	0	ODO	PDRV
		1	PDRV	

<Note> I2C IO pins set open drain mode ; x: Don't care

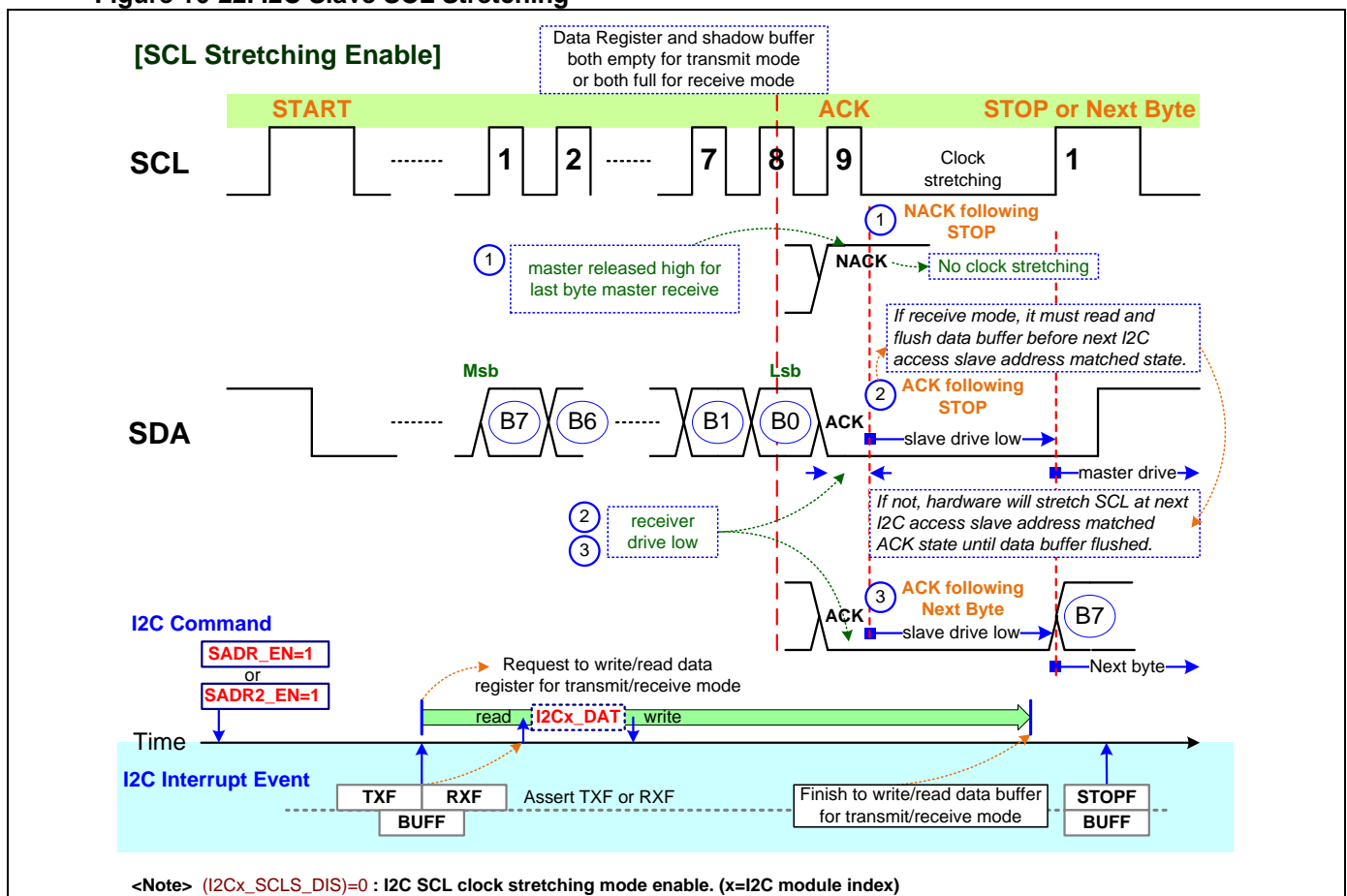
ODO: open drain output

PDRV: pre-driving 1~3 clock for high output

16.12.2. I2C Slave SCL Stretching

The I2C supports the SCL clock signal stretching low function for slave mode. This function is able to enable or disable by setting **I2Cx_SCLS_DIS** register and default is enabled. When enables, the chip will stretch SCL signal to low level if the I2C data buffer is full for receiving or empty for transmission.

Figure 16-22. I2C Slave SCL Stretching



16.12.3. Wakeup from STOP

The I2C supports to wakeup chip in **STOP** mode by the events of timer overflow, timer input periodic clock and alarm compare output. There are independent wakeup enable bits of **I2C_TF_WPEN**, **I2C_PC_WPEN** and **I2C_ALM_WPEN** for these three wakeup events.

When the chip is entering **STOP** mode and any of these I2C wakeup events is happened, the I2C will send the wakeup event to Power Controller (PW) to do as the system wakeup events. If the related control registers are enabled, the wakeup event will make to wakeup the chip.

During STOP mode, the I2C control module can detect the I2C slave address which is set in **I2Cx_SADR** or **I2Cx_SADR2** registers. When the owned I2C slave address is detected, the chip will be waked up if the **I2Cx_WUP_IE** and **I2Cx_IEA** registers are enabled.

Refer the descriptions of System Power and Interrupt chapters for more information about the wakeup events and control.

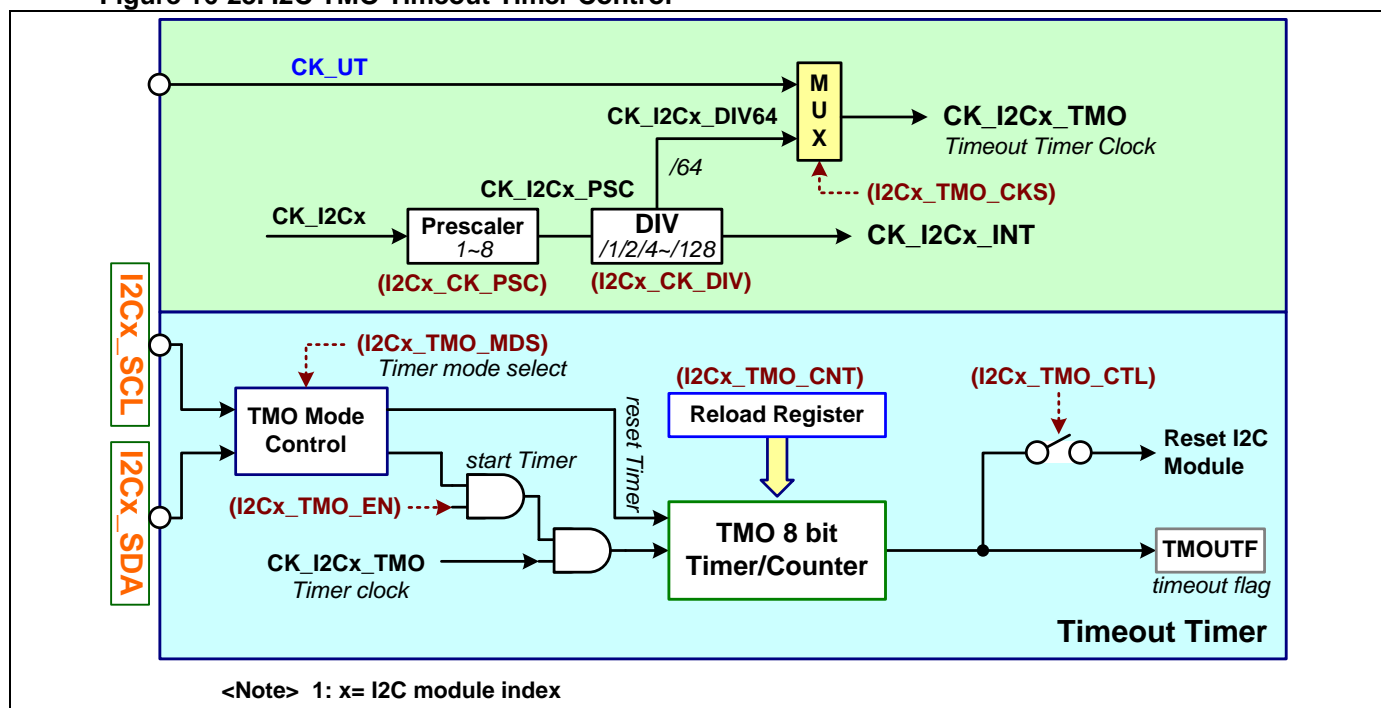
16.12.4. I2C Timeout Timer Control

The module provides one 8-bit timeout timer (TMO) for I2C access time-out control. It can configure as an I2C timeout timer or a general using timer by setting **I2Cx_TMO_MDS** register and enable by setting **I2Cx_TMO_EN** register. When the TMO uses as an I2C timeout timer, user can select to detect and count the time of SCL low state or bus idle state (both SCL and SDA high) by setting **I2Cx_TMO_MDS** register.

The TMO input timer clock source can be select from the clock of **CK_UT** or **CK_I2Cx_DIV64** by setting **I2Cx_TMO_CKS** register. The clock of **CK_I2Cx_DIV64** is fixed from the frequency divided by 64 of **CK_I2Cx_PSC** clock signal.

The TMO provides a register of **I2Cx_TMO_CNT** register and use to set the TMO timeout time. When the TMO timeout is happened, user can enable to reset the I2C controller by setting **I2Cx_TMO_CTL** register.

Figure 16-23. I2C TMO Timeout Timer Control



When the TMO timer is configured as a general using timer, the timer enabled function is no effect by **I2Cx_EN** register.

The following table is showing the I2C TMO Timeout Timer On/Off Control.

Table 16-10. I2C TMO Timeout Timer On/Off Control

TMO Timer Function	I2Cx Register			Timer On/Off
	TMO_MDS	EN	TMO_EN	
I2C Timeout Timer	0x	0	x	Timer Off
		1	0	Timer Off
			1	Timer On
General Using Timer	10	x	0	Timer Off
			1	Timer On

<Sign> x: Don't care

16.12.5. I2C NACK Control

Usually the I2C device must assert the ACK bit after receives 8-bit data or slave address except the last byte for master transmission mode. When detect the abnormal NACK, the chip will make the NACKF flag active and assert the related interrupt if the interrupt enable bit (**I2Cx_NACKF**) is enabled. Please refer the sections of "[I2C Interrupt Flags](#)" and "[I2C Master Transmission NACK Detect](#)" for more information about NACK condition.

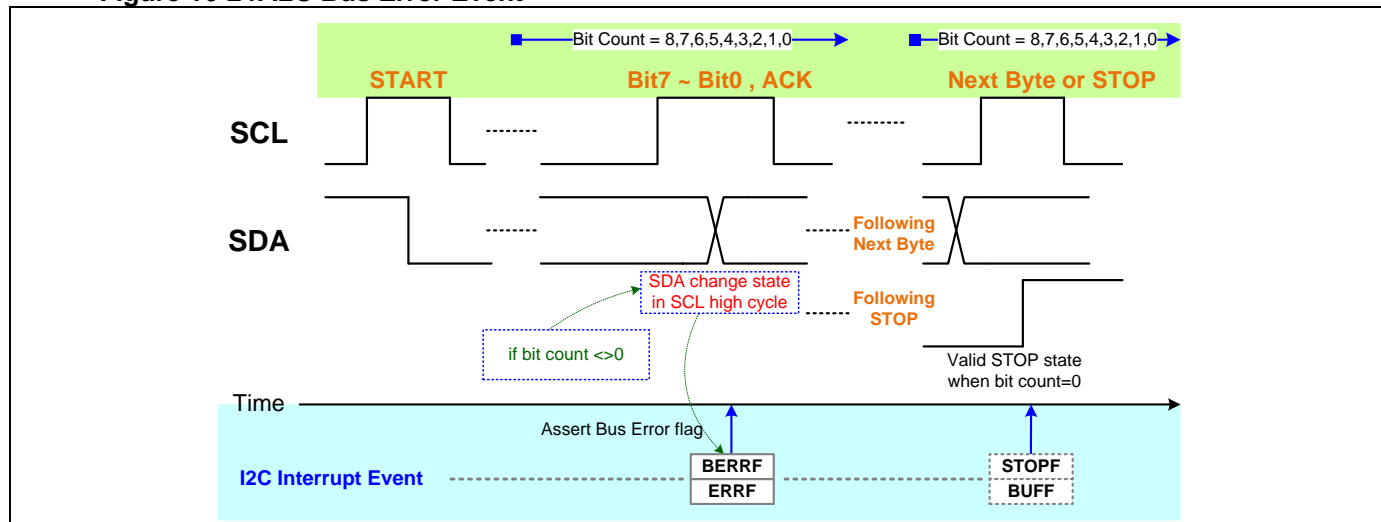
By application for master transmission mode, user can ignore receiving NACK and enable the I2C to transmit next data continuously when receive a NACK bit for Buffer mode by setting **I2Cx_NACK_EN** register.

16.13. I2C Error Management

16.13.1. I2C Bus Error Event

When the I2C controller detects a START or STOP state during the data bits transferred cycle but the data bits of one byte are not yet transferred end (bit count $\neq 0$), the BERRF flag (**I2Cx_BERRF**) will be activated.

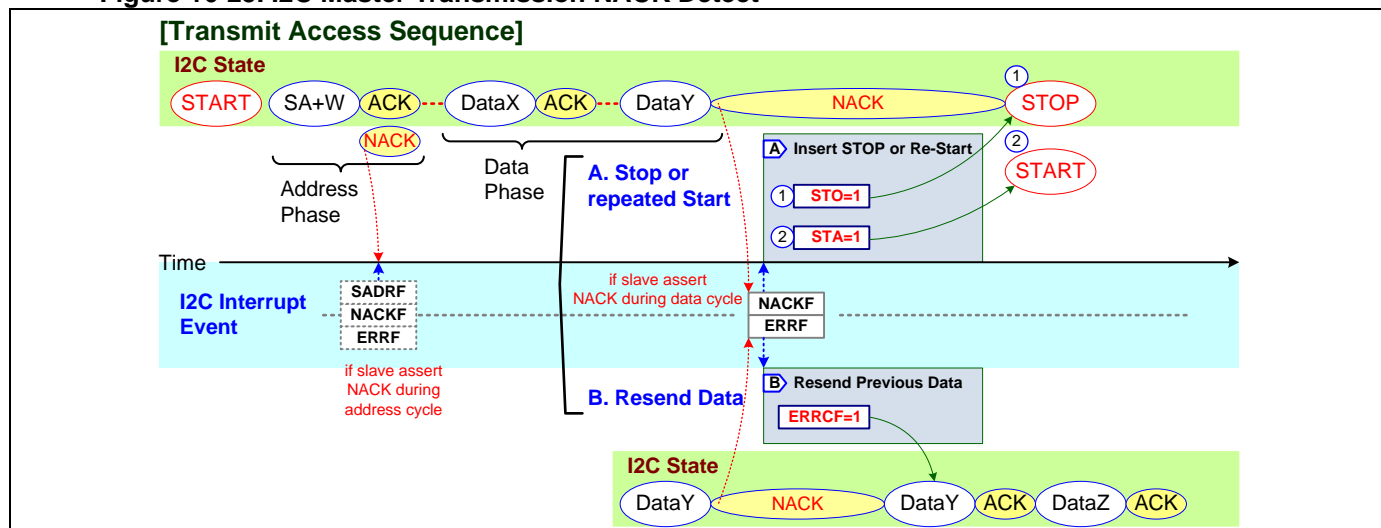
Figure 16-24. I2C Bus Error Event



16.13.2. I2C Master Transmission NACK Detect

For I2C master transmission mode, the I2C controller receives a NACK state and the NACK flag (**I2Cx_NACKF**) will be activated. When the NACK is detected in slave address cycle, the SADRf flag (**I2Cx_SADRf**) will be activated simultaneously. When the NACK is detected in data byte cycle, user can directly set the **I2Cx_STA** or **I2Cx_STO** register bits to assert Repeat-START or STOP state. Or user can set **I2Cx_ERRCF** register to "1" to resend previous data byte.

Figure 16-25. I2C Master Transmission NACK Detect

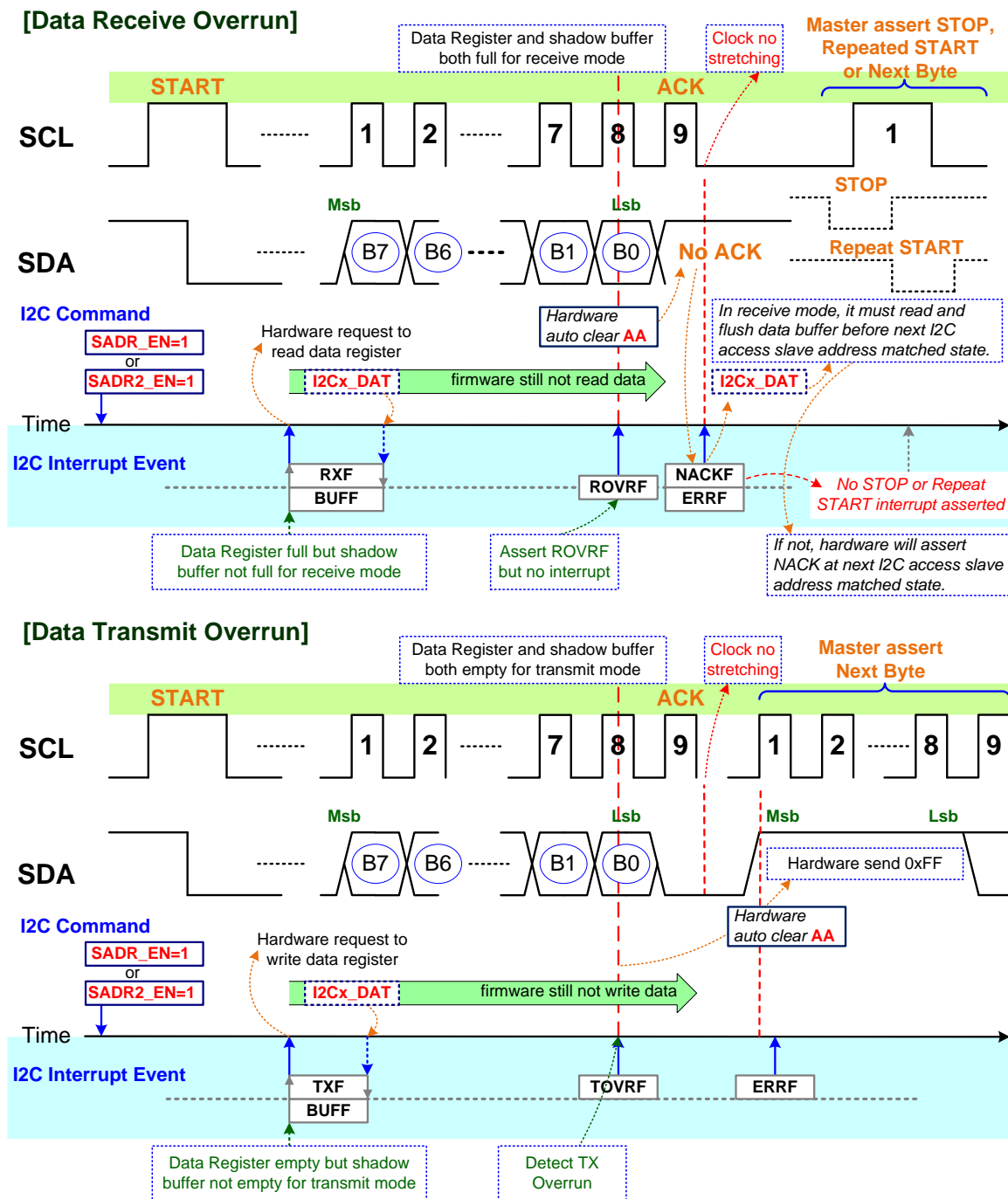


There is one read only status bit of **I2Cx_TXRF** which indicates the I2C transmission data register remained status. When the I2C communication is occurred the bus NACK error and **I2Cx_NACKF** is active, this bit is used to indicate the data register content whether has remain data and not yet transmitted. In the condition, the I2C master will assert STOP usually. User can calculate the corrected total transfer data count by reading **I2Cx_ACNT** register. The **I2Cx_TXRF** bit is cleared in slave address matched state and updated after last byte NACK state.

16.13.3. I2C Slave Data Overrun under SCL Stretching Disable

When Shadow Buffer is full and RX data register (**I2Cx_DAT**) is not empty for slave mode, the ROVRF flag (**I2Cx_ROVRF**) will be activated if the Shift Buffer receives full. When both Shadow Buffer and TX data register (**I2Cx_DAT**) are empty for slave mode, the TOVRF flag (**I2Cx_TOVRF**) will be activated if the Shift Buffer shifts out completely and external I2C master still requests data.

Figure 16-26. I2C Slave Data Overrun under SCL Stretching Disable



<Note> (I2Cx_SCLS_DIS)=1 : I2C SCL clock stretching mode disable. (x=I2C module index)

16.14. I2C DMA Operation

The I2C support the capability of received and transmitted data are buffered with DMA in Byte mode.

16.14.1. DMA Module Configure

When the chip supports a DMA (direct memory access) controller, user can configure the DMA setting of transferred source/destination devices, channel request arbitration and others in the DMA module before a DMA data transaction. The DMA source and the destination can be memory or peripheral.

Refer the DMA chapter for more detail information about the DMA module configuration.

16.14.2. I2C DMA Control

After DMA configuration is finished, user needs to set the I2C module DMA enable bits of **I2Cx_DMA_RXEN** and **I2Cx_DMA_TXEN**.

Finally, the related channel request start bit of **DMA_CHn_REQ** is necessary to be set to start the DMA transaction (n = DMA channel index). Then the transferred source/destination devices will assert the RX/TX request signal to DMA controller and the DMA controller will assert the acknowledge signal to the request source/destination devices. At the time, the data transferred connection is built for DMA transaction.

- **I2C RX to DMA**

The **I2Cx_DMA_RXEN** register bit is used to enable DMA data transfer from I2C receiving input to DMA destination.

During the DMA transaction cycle, the received data flag of **I2Cx_RXF** is masked by hardware.

- **I2C TX from DMA**

The **I2Cx_DMA_TXEN** register bit is used to enable DMA data transfer from DMA source to I2C transmitted output.

During the DMA transaction cycle, the transmitted data flag of **I2Cx_TXF** is masked by hardware. In general, the transmitted complete flag of **I2Cx_TSCF** will be asserted after the finished of DMA transaction.

When hardware detects one of errors or special conditions, hardware will clear both the **I2Cx_DMA_TXEN** bit and the **I2Cx_DMA_RXEN** bit.

16.14.3. I2C Interrupt Flag Control during DMA

During DMA operation cycle, the module's interrupt flags will control and act three types as following table. One is masked during the DMA process. Another is to disable the DMA function after the flag has asserted. At the time, hardware will disable both the **I2Cx_DMA_TXEN** bit and the **I2Cx_DMA_RXEN** bit in this condition. Others are normally as same as the action of not processing in DMA operation.

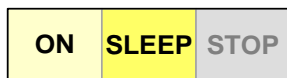
Table 16-11. I2C Interrupt Flag Control for DMA Function

Action	Mask the Flag during DMA Process	DMA Disable after the Flag asserted (*1)	Normal Control	
Peripheral	(Data flow flags)	(Error/Detect flags)	(Other flags)	
I2Cx	I2Cx_EVENTF I2Cx_BUFF	I2Cx_ERRF(*1) I2Cx_STPSTRF(*1)	I2Cx_TMOUTF I2Cx_WUPF	I2Cx_RXF I2Cx_TXF I2Cx_RSTRF I2Cx_STOPF I2Cx_CNTF I2Cx_ERRCF I2Cx_SADRF I2Cx_TSCF I2Cx_ROVRF I2Cx_TOVRF I2Cx_NACKF I2Cx_ALOSF I2Cx_BERRF

When the flag is asserted, it will not force peripheral DMA disabled if the related interrupt enable bit is **Note-1** : not enabled. Specially, the flag asserted that will force peripheral DMA disabled for I2Cx_ERRF or I2Cx_STPSTRF however the related interrupt is enabled or not.

17. UART (Universal Asynchronous Receiver Transmitter)

17.1. Introduction



The module can be running in ON and SLEEP modes only.

The UART module support full-duplex transmission, meaning it can transmit and receive simultaneously. The module builds in the shadow buffer and data register by transmit and receive independently to improve transmit and receive communication performance. It can commence reception of a second byte before a previously received byte has been read from the register. However, if the first byte still hasn't been read by the time reception of the second byte is complete, one of the bytes will be lost.

There are two types of UART module, one is advanced UART modules and another is basic UART modules. The advanced UART modules are including of URT0 ~ URT2. The basic UART modules are including of URT4 ~ URT7 those are only support standard asynchronous communication mode through RX and TX two pins.

The advanced UART module can operate in multiple modes with asynchronous *and* synchronous communication. The asynchronous communication supports UART, IrDA, *LIN*, *SmartCard* and multi-processor modes. The synchronous communication supports one line SYNC and two lines SPI master/slave modes. The UART operates as a full-duplex Universal Asynchronous Receiver and Transmitter, which can transmit and receive simultaneously and at different baud rates.

Notify: The sign of (x = module index) is using for Registers, Signals and Pins/Ports in the descriptions of this chapter. [EX]: **URTx_EN** ~ x indicates module index number 0, 1, 2, 4, 5, 6, 7.

17.2. Features

- Provide UART modules : URT0~2, URT4~7
- UART module common functions
 - Provide precise UART baud-rate control by programmable oversampling rate
 - Support baud rate up to 6 Mbit/s
 - Programmable data word length - 7 or 8 bits
 - Programmable 4~32 oversampling rate
 - Hardware parity checking and parity generation
 - Separate enable bits for transmitter and receiver
 - Swappable TX/RX pin configuration
 - Separate signal polarity control for transmission and reception
- URT0/1/2 advanced UART modules
 - Support UART, Synchronous, SPI master/slave, SmartCard, LIN, Multi-processor modes
 - Selectable MSB or LSB first data order
 - Configurable stop bits - 0.5, 1, 1.5 or 2 stop bits
 - Selectable one or three point for data majority vote
 - Support a timeout timer for Idle/RX/Break/Calibration timeout detection
 - Support UART mute mode which enter or exit by multiple events hardware auto detection
 - Support 4-byte data buffer and 32-bit data register for high speed communication
 - Support auto baud-rate detection and calibration
 - Support multiprocessor communication for master and slave mode - Idle-Line , Address-Bit
 - Support low speed UART-like frame format IrDA
 - Support transceiver hardware flow control by CTS/RTS signals only
 - Provide driver enable signal to activate the transmission for bidirectional communication
 - Support RX timeout detection for aging character detection
 - Support transmission-error hardware detection and auto resent control for Smart-card application
 - Support receiving parity error hardware detection and auto retry control for Smart-card application
 - Received and transmitted data are buffered with DMA capability
- URT4/5/6/7 basic UART modules
 - Support fundamental UART mode
 - Support TX/RX independent 8-bit data register for simplex firmware control

— Configurable stop bits - 1 or 2 stop bits

17.3. Implementation

17.3.1. Chip Implementation

The following table is showing the implemented UART modules of chips.

Table 17-1. UART Implementation

Chip	UART Module 0~3						UART Module 4~7		
	UART Module			UART Max. Baud Rate	SPI Max. Clock Rate		UART Module		UART Max. Baud Rate
	URT0/1	URT2	URT3	UART	SPI Master	SPI Slave	URT4	URT5/6/7	UART
MG32F02A128/A064	V	V	-	6Mbps	18MHz	12MHz	V	V	6Mbps
MG32F02U128/U064	V	-	-	6Mbps	24MHz	16MHz	V	-	6Mbps
MG32F02V032	V	-	-	6Mbps	24MHz	16MHz	V	-	6Mbps
MG32F02N128/N064	V	V	-	6Mbps	22MHz	16MHz	V	V	6Mbps
MG32F02K128/K064	V	V	-	6Mbps	22MHz	16MHz	V	V	6Mbps

<Note> V: Implemented; Measure maximum clock rate at VDD>=3.3V.

17.3.2. Modules' Functions

The following table is showing the implemented functions of UART modules.

Table 17-2. UART Modules' Functions

Module	URT0/1/2	URT4/5/6/7	Comment
Chip	All chips with implemented URT0/1/2	All chips with implemented URT4/5/6/7	
Module Functions			
UART - asynchronous	yes	yes	
Prescaler/Baud-Rate counter	6/8-bit	6/8-bit	Prescaler counter and Baud-Rate counter bits
Synchronous - SPI mode	Master/Slave		Synchronous - Master/Slave 2 data lines
SmartCard - ISO7816-3	yes		
LIN	yes		
Multiprocessor - Address Bit	yes		
Multiprocessor - Idle Line	yes		
IrDA - UART Like	yes		low speed UART-like frame format IrDA (SIR Normal Mode)
Hardware flow control	yes		only support CTS/RTS
External Clock pin	1		
Timer BRO,TMO pins	2		
Swappable TX/RX	yes	yes	
NCO output as clock input	yes	yes	NCO_Pn input as UART clock input source
Shadow Buffer	4-byte		
Data 7-bit option	yes	yes	
TX parity bit generation	yes	yes	hardware auto generate the parity bit from the data byte
Msb/Lsb transfer option	yes		
Configurable stop bits	0.5, 1, 1.5, 2	1, 2	programmable Stop bit length
Auto Baud-Rate calibration	yes		auto Baud-Rate detection and calibration
Mute mode auto enter/exit	yes		enter mute mode if address match does not occur
Break condition detect	yes		
Idle line detect	yes		
Programmable over sampling number	4~32	4~32	
Programmable clock phase/polarity	yes		for synchronous mode
General timer control	yes	yes	baud-rate timer and timeout timer as general timer
Drive enable	yes		Drive enable signal of the transmission mode for the external transceiver.
RX parity error detect	yes	yes	checks parity of received data byte
Frame error detect	yes	yes	
Data overrun detect	yes	yes	receive buffer over threshold level; transmit buffer empty
TX error detect	yes		SmartCard/LIN
Noise character detect	yes		skip or not for noise character
Idle timeout detect	yes		for SmartCard application
RX timeout detect	yes		aging character detection
Break timeout detect	yes		for LIN application
Calibration timeout detect	yes		for LIN application
DMA request capability	yes		

17.4. Control Block

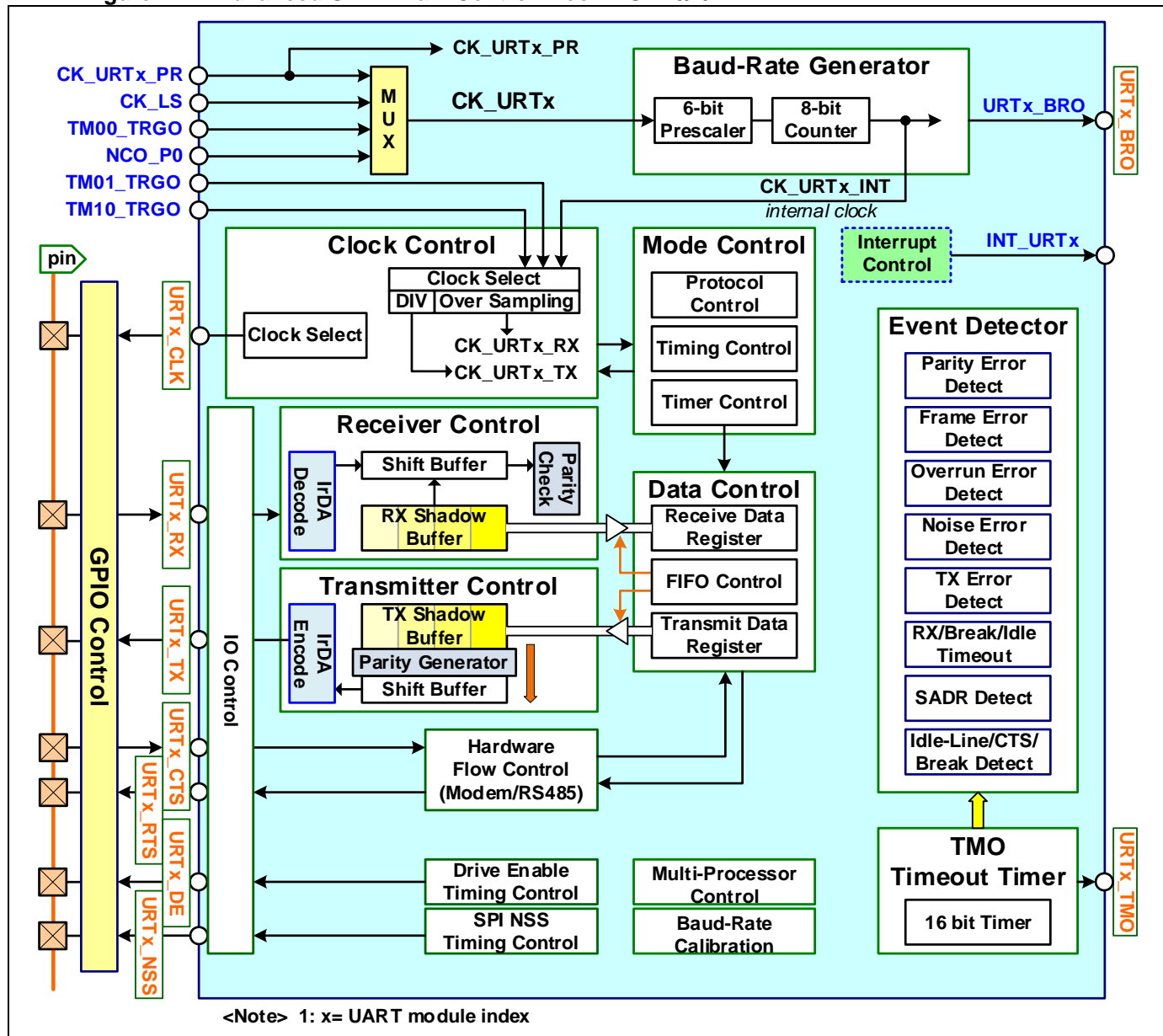
There are two types of URTx module, one is basic UART module and another is advanced UART module.

17.4.1. Advanced UART Control Block

The advanced UART module can operate in multiple modes: asynchronous communication, synchronous communication, SPI master, **SmartCard**, **LIN**, multi-processor mode. The asynchronous communication operates as a full-duplex Universal Asynchronous Receiver and Transmitter (UART), which can transmit and receive simultaneously and at different baud rates.

The following diagram is showing the advanced UART Control block.

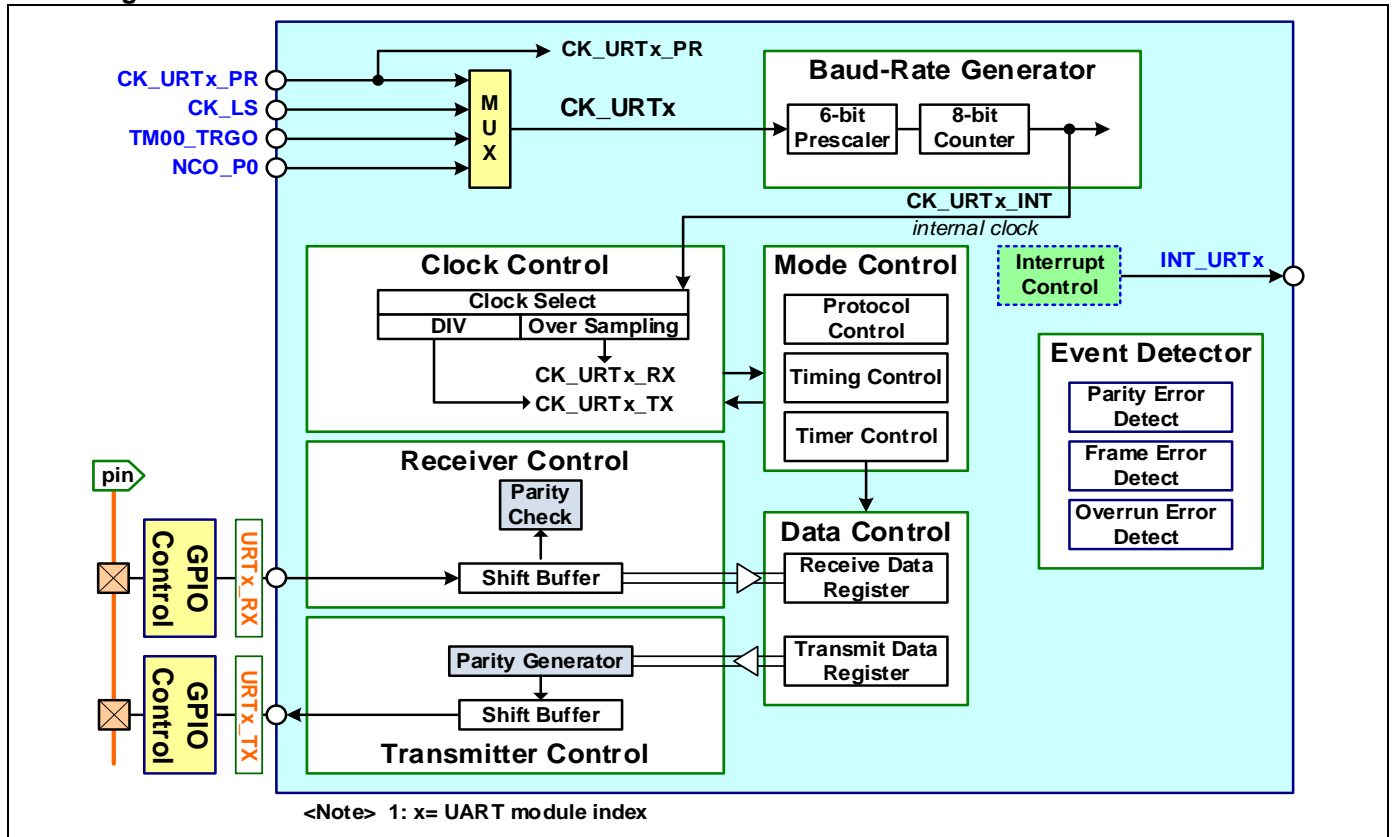
Figure 17-1. Advanced UART Main Control Block – URT0/1/2



17.4.2. Basic UART Control Block

The basic UART module can only operate in standard asynchronous communication.
The following diagram is showing the basic UART Control block.

Figure 17-2. Basic UART Main Control Block – URT4/5/6/7



17.5. IO Lines

17.5.1. IO Signals

- **URT_x_CLK**
It is the UART clock signal and uses as output for UART synchronous mode or SPI master mode.
- **URT_x_RX**
It is the UART receive RX data signal or SPI MISO input signal for SPI master mode.
- **URT_x_TX**
It is the UART transmit TX data signal or SPI MOSI input signal for SPI master mode.
- **URT_x_NSS**
It is the SPI slave select or chip select output signal for SPI master mode.
- **URT_x_CTS**
It is the UART clear to send incoming flow control input signal.
- **URT_x_RTS**
It is the UART request to send outgoing flow control input signal.
- **URT_x_DE**
It is the UART data enable output signal for external drive device.
- **URT_x_BRO**
It is the UART output signal from UART baud-rate timer overflow signal.
- **URT_x_TMO**
It is the UART output signal from UART timeout timer overflow signal.

17.5.2. IO Configure

User must configure the related IO pins in order to use the IO lines of this module. The IO operation modes, output high speed option, pull-high option, output drive strength, IO deglitch filter and input inverse selection are programmable by pin independent. Please refer the section of “[IO Mode](#)” in GPIO chapter for more detail descriptions of IO mode configuration.

Each IO signal may be mapped and selected on several IO pins by configuring the IO AFS matrix. Please refer the section of “[Alternate Function Select](#)” in GPIO chapter for more detail descriptions of IO AFS configuration. About the actual IO pin AFS information, please refer the section of “[Pin Alternate Functions Selected Table](#)” in Pin Description chapter of the chip Data Sheet.

17.6. Enabling and Clock

17.6.1. UART Global Enable

There is a global enable bit of **URTx_EN** for all functions of this module. When this bit is disabled, all the UART functions are not working.

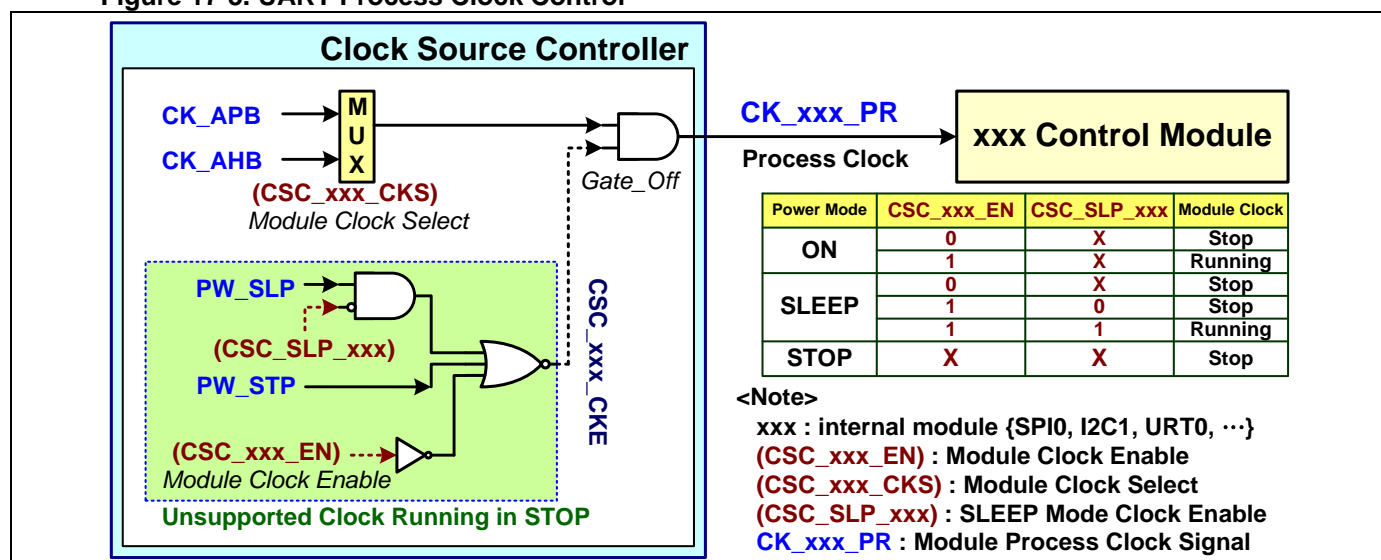
17.6.2. UART Clock Control

● Module Process Clock

The module process clock of **CK_URTx_PR** is using for the interface control logic between APB bus and the module. It is coming from CSC (Clock Source Controller) module. It can be enabled in **CSC_URTx_EN** register and select the clock source from APB clock or AHB clock in **CSC_URTx_CKS** register.

In **ON** mode, the process clock is running only if **CSC_URTx_EN** register is enabled. User can plan the module clock is running or not beforehand for chip entering **SLEEP** mode by setting **CSC_SLP_URTx** register. In **STOP** mode, the process clock is always stopping. Refer the System Clock chapter for more information.

Figure 17-3. UART Process Clock Control



● Module Internal Clock

User can select the clock source from the module process clock of **CK_URTx_PR**, **NCO_P0**, internal ILRCO clock of **CK_ILRCO** or the timer trigger output signal of **TM00_TRGO** by setting **URTx_CK_SEL** register.

The UART module is built in a Baud-Rate generator to output the internal clock **CK_URTx_INT** as the sampling clock for received data signal and the clock source for transmitted data signal. Also it is able to output the clock **CK_URTx_SC** for **SmartCard** application. User can configure the Baud-Rate generator by setting **URTx_PSR** and **URTx_RLR** registers. The Baud-Rate counter is implemented 8-bit counter.

[Notify]: Usually the internal clock frequency needs slow down at least 1/2 than module process clock.

For asynchronous communication, the data transmission and receiving are able to use different Baud-Rate for the signals of **URTx_RX** and **URTx_TX**. The registers of **URTx_TX_CKS** and **URTx_RX_CKS** are used independently to select the clock source for input signal of **URTx_RX** sampling and output signal of **URTx_TX** generation.

For synchronous communication, the **URT_x_CLK** can configure as clock output and enable by setting **URT_x_CLK_EN** register. User can set **URT_x_CLK_CKS** register to select output **CK_URT_x_SC** for **SmartCard** application or **CK_URT_x_OUT** for others synchronous communication application.

An output signal of **URT_x_BRO** can come from the timer overflow signal of Baud-Rate generator and use as a clock signal output.

The following diagrams are showing the UART clock control block for different chips.

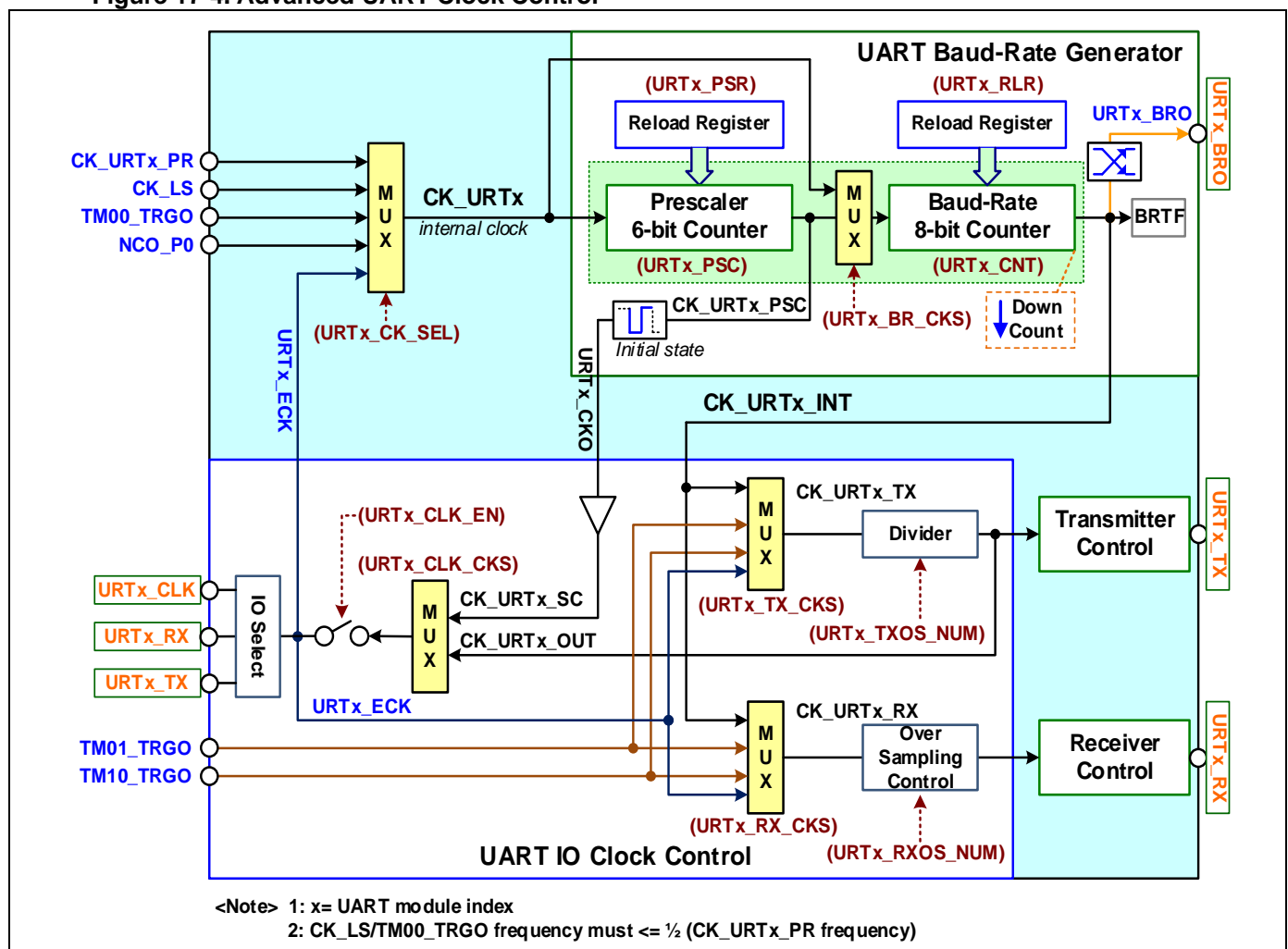
❖ Advanced UART Clock Control

The external clock input can come from **URT_x_ECK** signal to do as the Baud-Rate generator clock input by setting **URT_x_CK_SEL** register when uses the Baud-Rate generator as a general using timer. User can select the **URT_x_ECK** signal from external signal of **URT_x_CLK**, **URT_x_RX** or **URT_x_TX** by setting **URT_x_ECK_CKS** and **URT_x_IO_SWP** register. Refer the section of “**UART IO Control**” for more information.

For some application, the input clock **URT_x_ECK** can also input as the TX divider or RX over sampling control input clock by setting **URT_x_TX_CKS** or **URT_x_RX_CKS** registers. It sends to the MUX of **URT_x_TX_CKS** and **URT_x_RX_CKS** as one selected input clock.

There is one MUX to select the 8-bit Baud-Rate counter input clock coming from **CK_URT_x** signal or the Prescaler clock output **CK_URT_x_PSC** by setting **URT_x_BR_CKS** register. For SmartCard application, user can select the 8-bit Baud-Rate counter input clock coming directly from **CK_URT_x** signal. Others, user can select the Prescaler clock output **CK_URT_x_PSC**. Refer the section of “**UART SmartCard Clock Output**” for more information about SmartCard clock output.

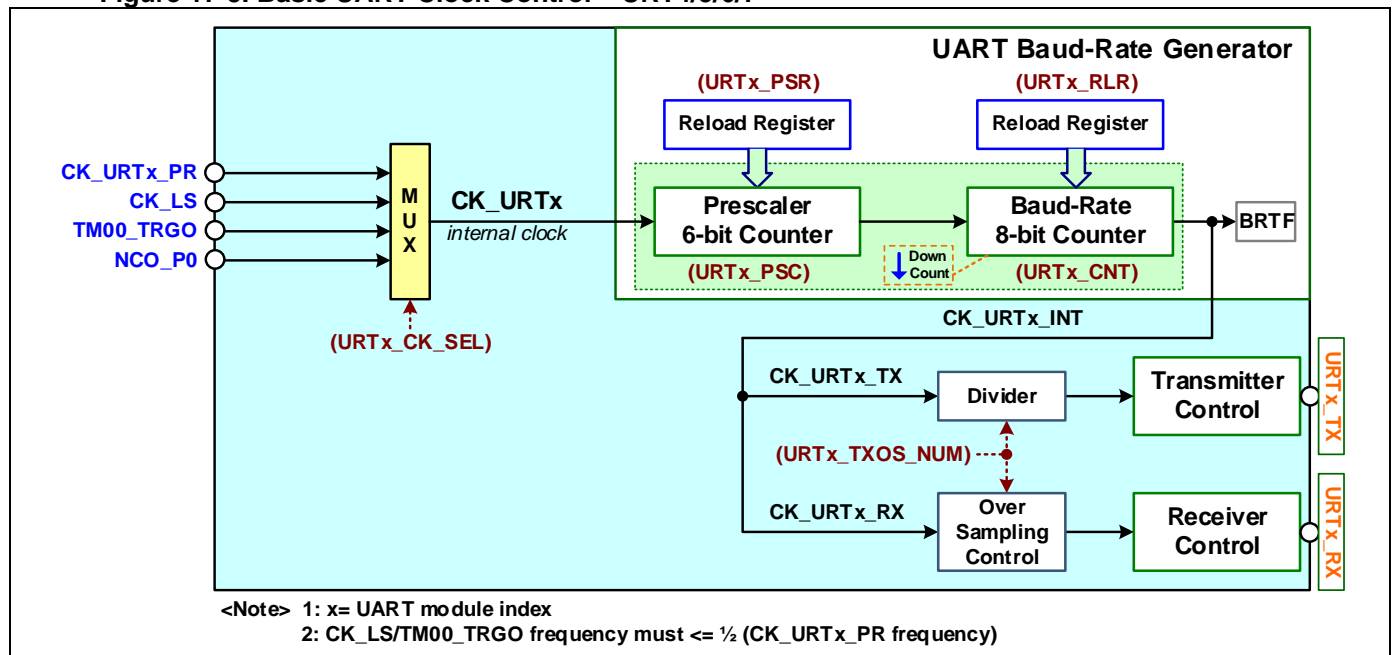
Figure 17-4. Advanced UART Clock Control



❖ Basic UART Clock Control

The basic UART modules are only support standard asynchronous communication mode through RX and TX two pins.

Figure 17-5. Basic UART Clock Control – URT4/5/6/7



17.6.3. UART PSC Timeout Signal Output

The PSC prescaler timer can output the timeout signal to do as a clock signal of **CK_URT_x_PSC** to external port of **URT_x_CLK**. User can set the initial state of timeout signal by setting **URT_x_CKO_STA** register. Specially, the initial state register can be written only when the register of **URT_x_CKO_LCK** is written "1" simultaneously.

17.7. Interrupt and Event

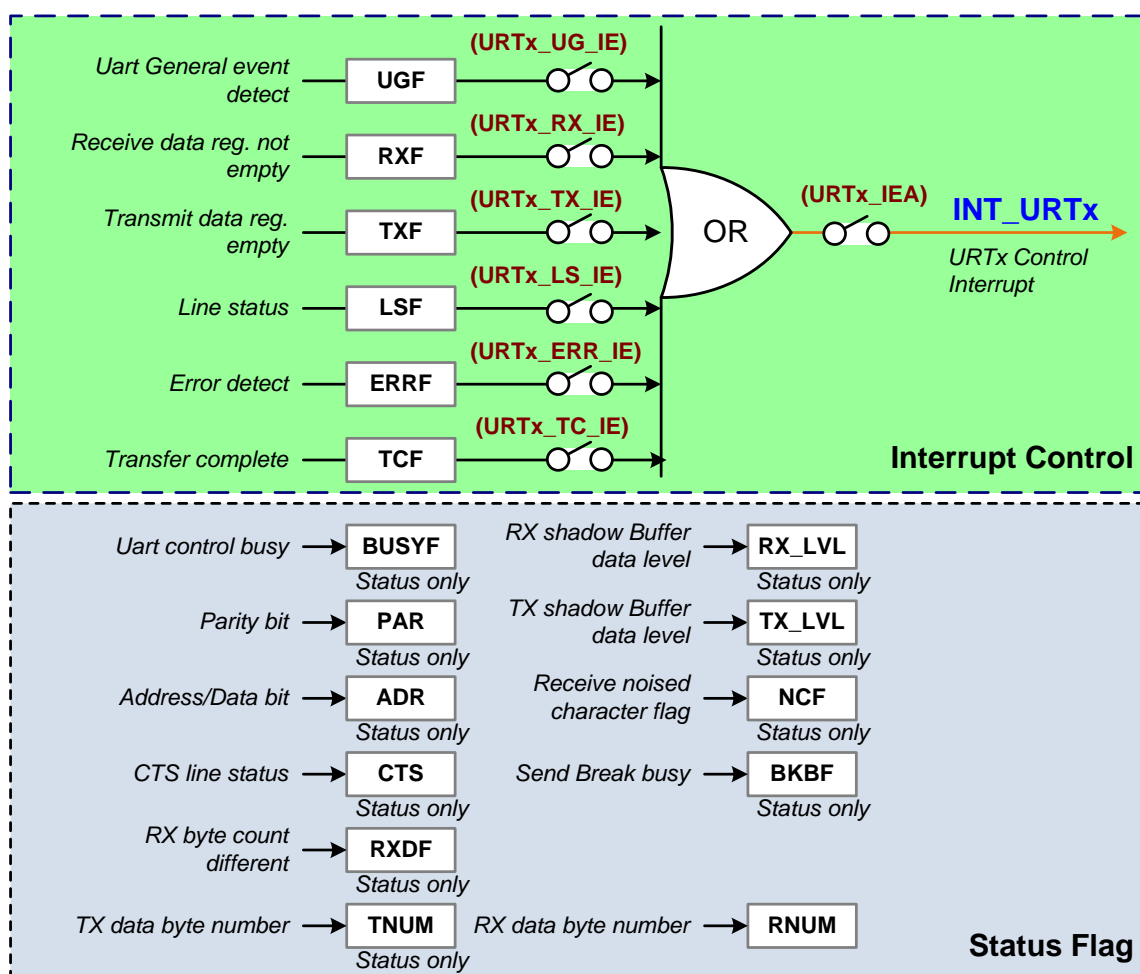
There is one signal of **INT_URT_x** to be generated in this UART control module. **INT_URT_x** sends to EXIC External Interrupt Controller to do as an interrupt event.

17.7.1. UART Interrupt Control and Status

These interrupt flags are using for the interrupt service routine (ISR) flow control. Generally these flags are set by hardware and software clears them when the serviced job is completed in the related ISR. Each interrupt flag has one interrupt enable bit. User can enable or disable it. There is one interrupt all enable bit of **URT_x_IEA** to enable or disable all the interrupt sources for this module.

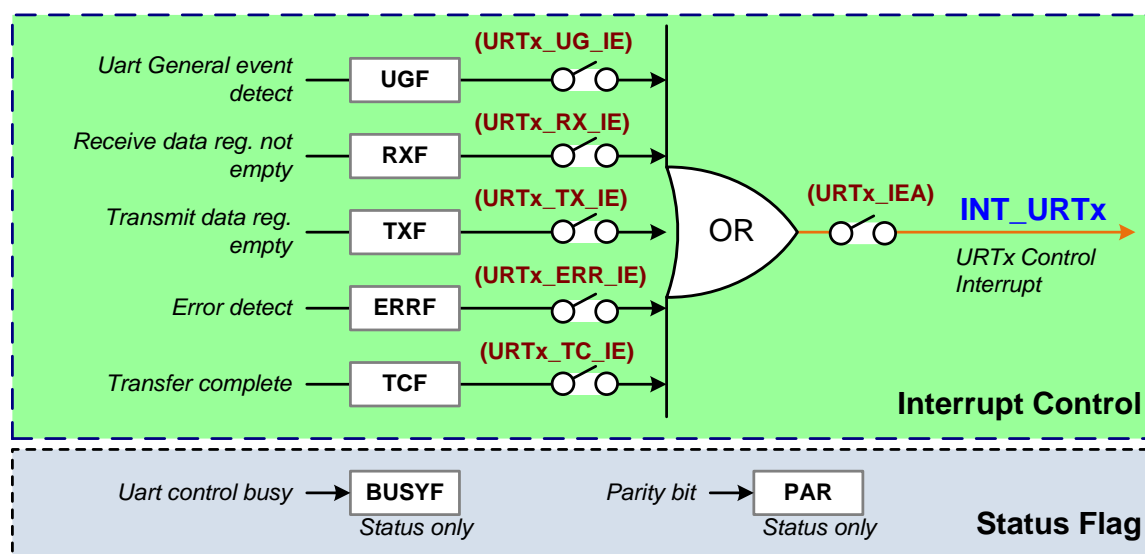
There are some status bits those are reading only to provide internal control status. One busy flag of **URT_x_BUSYF** is used to indicate the data transfer busy status. When detect valid Start bit, this bit is set and clear after Stop bit. Refer the register descriptions of the related status bits for more information.

Figure 17-6. UART Status and Interrupt Control – URT0/1/2



<Note> 1: x= UART module index

Figure 17-7. UART Status and Interrupt Control – URT4/5/6/7

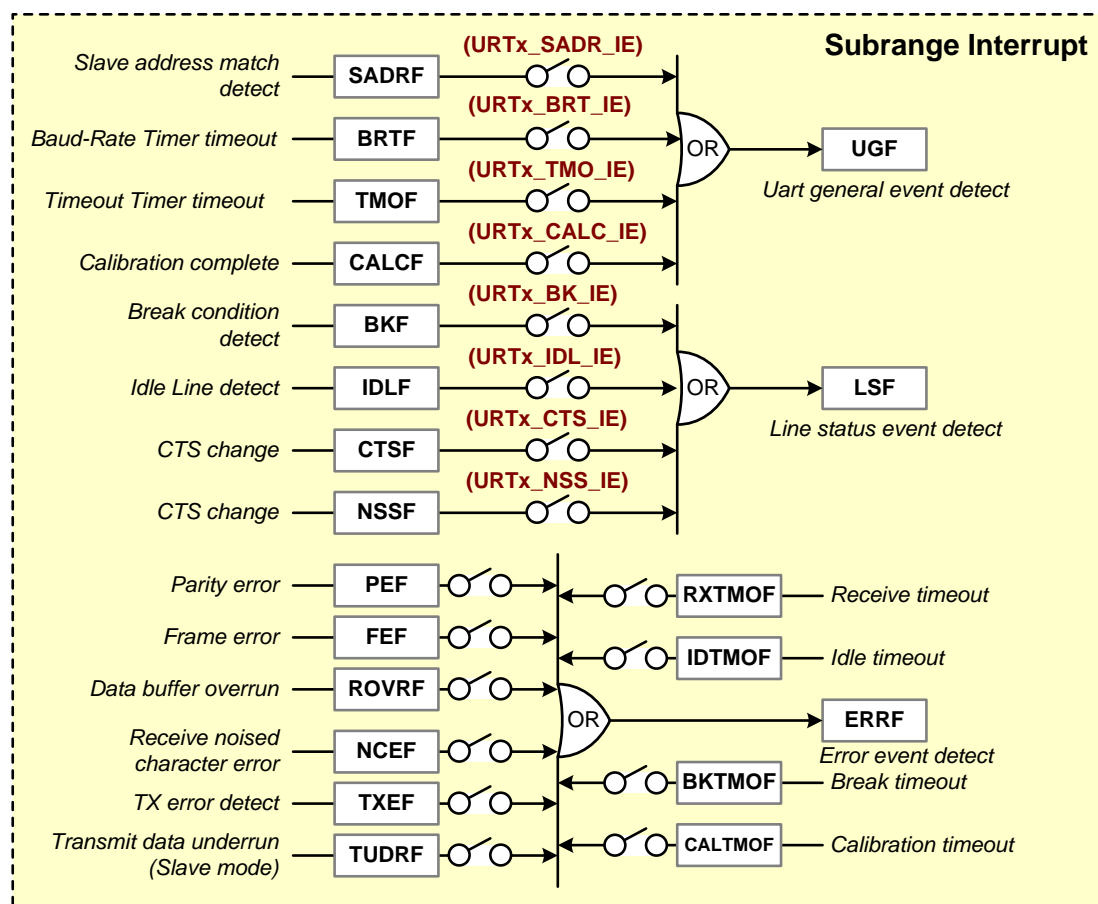


<Note> 1: x= UART module index

17.7.2. UART Subrange Interrupt

There are some subrange interrupt flags for the interrupt flags of **URTx_UGF**, **URTx_LSF** and **URTx_ERRF**. The following diagram is showing the UART subrange interrupt control block of URT0/1/2.

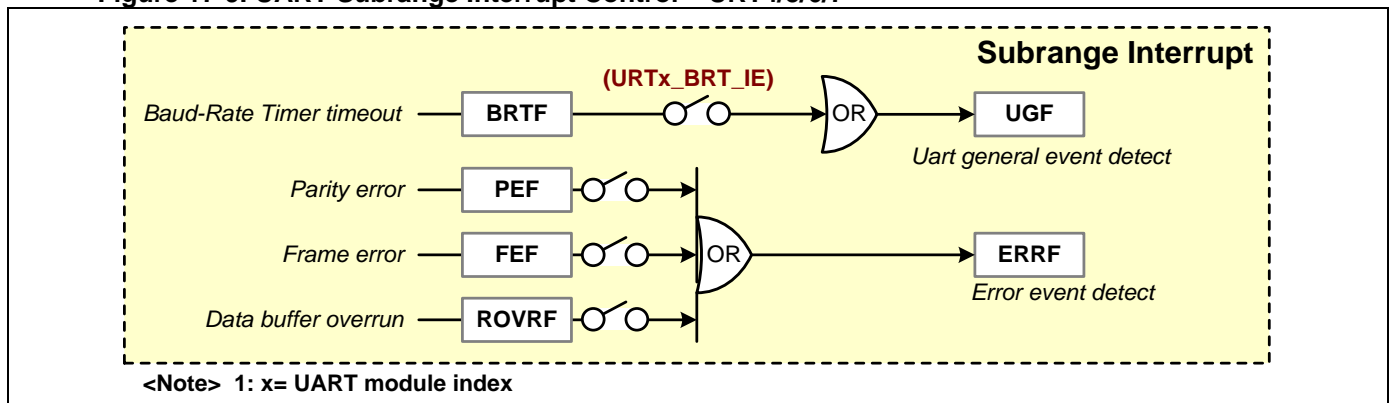
Figure 17-8. UART Subrange Interrupt Control – URT0/1/2



<Note> 1: x= UART module index

The following diagram is showing the UART subrange interrupt control block of URT4/5/6/7.

Figure 17-9. UART Subrange Interrupt Control – URT4/5/6/7



17.7.3. UART Interrupt Flags

Generally these interrupt flags are set by hardware and clear by software writing 1. Refer the register descriptions for more information about the related interrupt flags and interrupt enable bit.

- **UGF**

UART general event flag (**URT_x_UGF**). There is a related interrupt enable register bit of **URT_x_UG_IE**. It indicates any of **URT_x_SADRF**, **URT_x_BRTF**, **URT_x_TMOF** or **URT_x_CALCF** flag is asserted when this flag is set.

- **TCF**

UART transmission complete flag (**URT_x_TCF**). When both shadow buffer and data register are empty and shift buffer shift out complete, then set this flag. There is a related interrupt enable register bit of **URT_x_TC_IE**.

- **ERRF**

UART error interrupt flag (**URT_x_ERRF**). There is a related interrupt enable register bit of **URT_x_ERR_IE**. It indicates any of parity error, frame error, overrun error, received time out and noise error.

- **LSF**

UART general event flag (**URT_x_LSF**). There is a related interrupt enable register bit of **URT_x_LS_IE**. It indicates any of the line statue flag of break condition, idle line and CTS detection is asserted.

- **RXF**

UART receive data register not empty flag (**URT_x_RXF**). There is a related interrupt enable register bit of **URT_x_RX_IE**. When received data buffer level **URT_x_RX_LVL** is greater than or equal to the shadow buffer threshold **URT_x_RX_TH** setting, this flag is set and the data buffer content copy to data register **URT_x_RDAT**. This bit is cleared when **URT_x_RDAT** is read or this flag set to 1 by software.

Notify: The RXF flag is not cleared when URT_x_RDAT is read by SWD debugging.

- **TXF**

UART transmit data register empty flag (**URT_x_TXF**). There is a related interrupt enable register bit of **URT_x_TX_IE**. When transmitted shadow buffer is empty and the data register **URT_x_TDAT** will copy to the shadow buffer, this flag is set. This bit is cleared when **URT_x_TDAT** is written or this flag set to 1 by software.

- **SADRF**

UART slave address matched flag (**URT_x_SADRF**). This flag is using for multi-processor mode. There is a related interrupt enable register bit of **URT_x_SADR_IE**.

- **BRTF**

UART baud-rate generator timer timeout flag (**URT_x_BRTF**). There is a related interrupt enable register bit of **URT_x_BRT_IE**.

- **TMOF**

UART timeout timer timeout flag (**URT_x_TMOF**). There is a related interrupt enable register bit of **URT_x_TMO_IE**.

- **CALCF**

UART auto baud-rate calibration complete flag (**URT_x_CALCF**). There is a related interrupt enable register bit of **URT_x_CALC_IE**.

- **BKF**

UART break condition detect flag (**URTx_BKF**). There is a related interrupt enable register bit of **URTx_BK_IE**. Refer the section of "[UART Break and Parity/Frame Error Detection](#)" for more information.

- **IDLF**

UART idle line detect flag (**URTx_IDLF**). There is a related interrupt enable register bit of **URTx_IDL_IE**.

- **CTSF**

UART CTS change detect interrupt flag (**URTx_CTSF**). There is a related interrupt enable register bit of **URTx_CTS_IE**.

- **PEF**

UART parity error flag (**URTx_PEF**). There is a related interrupt enable register bit of **URTx_PE_IE**. When is running multi-processor mode, the parity value is including of address bit. Refer the section of "[UART Break and Parity/Frame Error Detection](#)" for more information.

- **FEF**

UART frame error flag (**URTx_FEF**). There is a related interrupt enable register bit of **URTx_FE_IE**. Refer the section of "[UART Break and Parity/Frame Error Detection](#)" for more information.

- **NCEF**

UART receive noised character error flag (**URTx_NCEF**). There is a related interrupt enable register bit of **URTx_NCE_IE**.

- **ROVRF**

UART receive overrun flag (**URTx_ROVRF**). There is a related interrupt enable register bit of **URTx_ROVR_IE**. When receive overrun, hardware will stop to receive next data into data shadow buffer until this flag is cleared. This flag is indicated for following two conditions. (1) When RX shadow buffer is arrived over the RX threshold and the data register has not read out. If shift buffer is filled of next data, this flag is asserted. (2) When Parity error, Frame error, Break detect or Slave-Address detect, has happened and caused RX shadow buffer input holding. If shift buffer is filled of next data, this flag is asserted.

- **TXEF**

UART TX error detect flag (**URTx_TXEF**). There is a related interrupt enable register bit of **URTx_TXE_IE**. Refer the section of "[UART TX Error Detect and Resend Control](#)" for more information.

- **RXTMOF**

UART receive time out flag (**URTx_RXTMOF**). There is a related interrupt enable register bit of **URTx_RXTMO_IE**. Refer the section of "[UART TMO Timeout Control](#)" for more information.

- **IDTMOF**

UART idle state time out flag (**URTx_IDTMOF**). There is a related interrupt enable register bit of **URTx_IDTMO_IE**. Refer the section of "[UART TMO Timeout Control](#)" for more information.

- **BKTMOF**

UART break receive time out flag (**URTx_BKTMOF**). There is a related interrupt enable register bit of **URTx_BKTMO_IE**. Refer the section of "[UART TMO Timeout Control](#)" for more information.

- **CALTMOF**

UART auto baud-rate calibration sync field receive time-out time out flag (**URTx_CALTMOF**). There is a related interrupt enable register bit of **URTx_CALTMO_IE**. Refer the section of "[UART TMO Timeout Control](#)" for more information.

- **TUDRF**

SPI slave mode transmit underrun flag (**URTx_TUDRF**). There is a related interrupt enable register bit of **URTx_TUDR_IE**.

- **NSSF**

SPI slave mode NSS signal inactive detect interrupt flag (**URTx_NSSF**). There is a related interrupt enable register bit of **URTx_NSS_IE**. When the module is configured to SPI slave mode, this flag is asserted if the input NSS signal has changed from active to inactive.

*[Notify]: The **NSSF** flag is not supported for MG32F02A128/U128/A064/U064.*

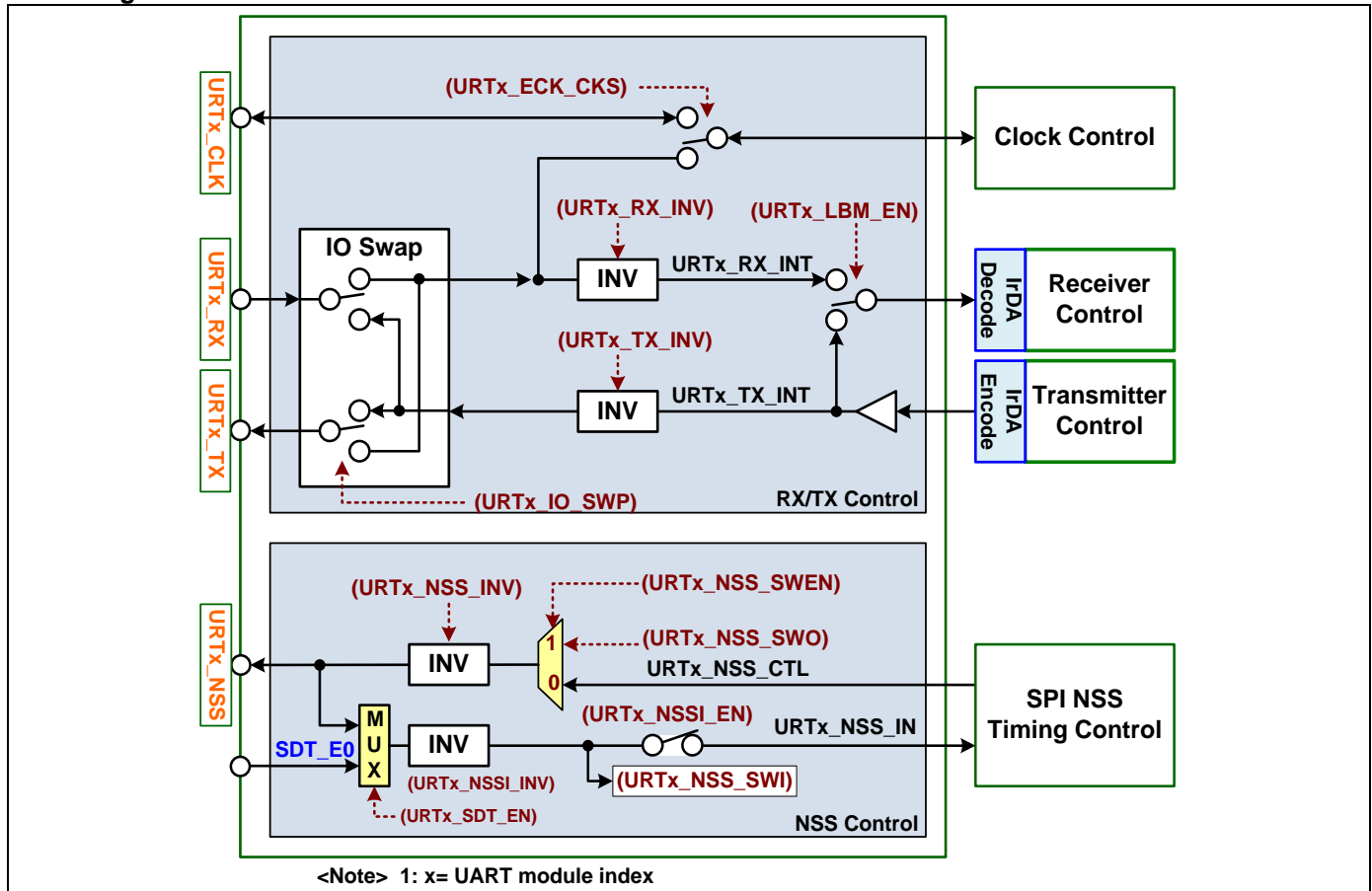
17.8. UART Module IO Control

This module provides two data signals – **URT_x_RX** and **URT_x_TX**, one clock signal – **URT_x_CLK**, two hardware flow control signals – **URT_x_CTS** and **URT_x_RTS**, one data enable signal – **URT_x_DE** and one slave select output signal – **URT_x_NSS**.

17.8.1. UART IO Control

The following diagrams are showing the UART RX/TX/NSS IO control block for different chips.

Figure 17-10. UART RX/TX IO Control



All the **URT_x_ZZZ_INV** (ZZZ: register string) registers are used to inverse the related signal. The **URT_x_IO_SWP** register is used to enable to swap the signals of **URT_x_RX** and **URT_x_TX**.

Table 17-3. UART Pin Swap Function Mapping

	URTx Register		UART Function
Chip	All Chips		
Pin	IO_SWAP	ECK_CKS	
URTx_TX	0	x	UART TX
	1	0	UART RX
	1	1	UART Clock
URTx_RX	0	0	UART RX
	0	1	UART Clock
	1	x	UART TX
URTx_CLK	x	0	UART Clock
	x	1	no using

<Sign> x : Don't care

For synchronous SPI master mode, the **URT_x_NSS_SWEN** register is used to enable **URT_x_NSS** signal by software control. When it enables, user can directly control the output by setting **URT_x_NSS_SWO** register.

The **URT_x_ECK_CKS** register is used to select the clock IO port for UART synchronous mode. When the

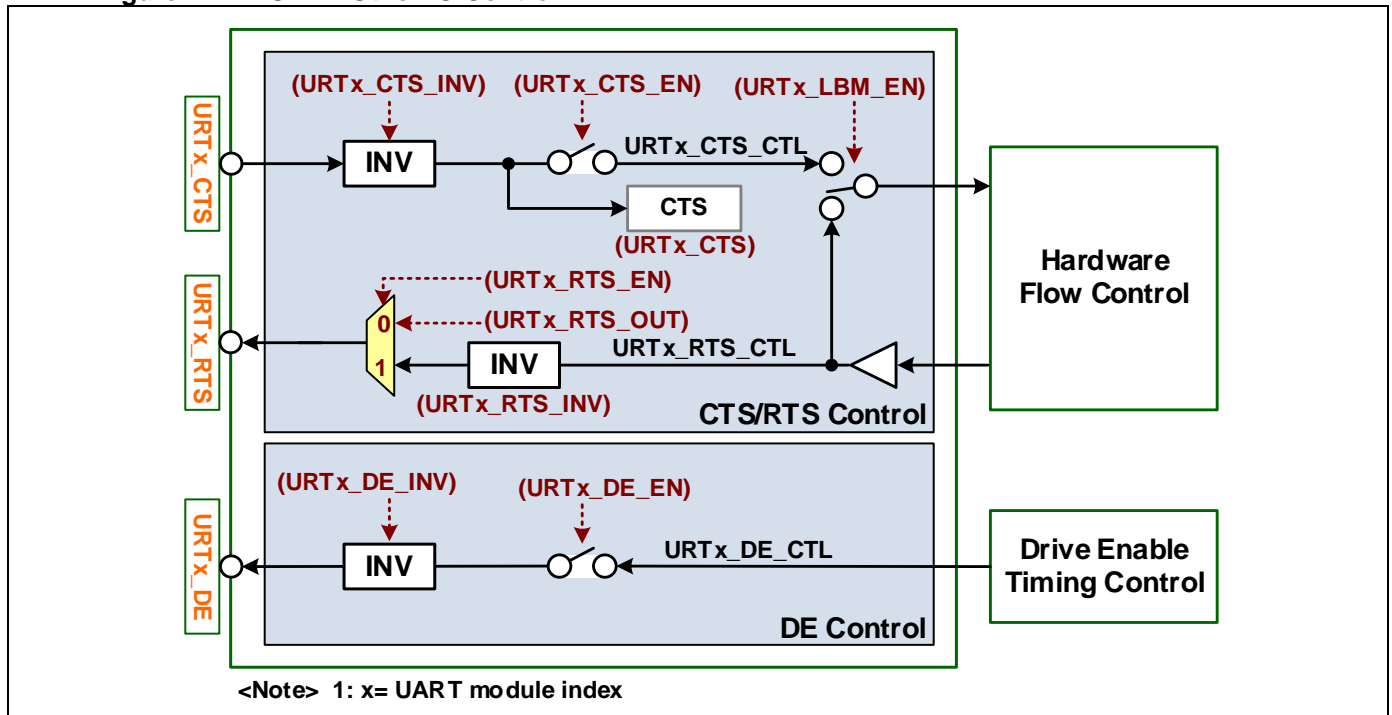
clock is input mode, the clock can input from the IO function of **URT_x_CLK** or the selected IO function of **URT_x_RX** or **URT_x_TX** by setting **URT_x_IO_SWP** register. When the clock is output mode, the output clock also can set **URT_x_ECK_CKS** register to select the output pin which can be the IO function of **URT_x_CLK** or the selected IO function of **URT_x_RX** or **URT_x_TX** by setting **URT_x_IO_SWP** register.

For synchronous SPI slave mode, user can enable the NSS input from **URT_x_NSS** input by setting **URT_x_NSSI_EN**. The **URT_x_NSSI_INV** register is used to inverse the **URT_x_NSS** input signal. Also user can directly get the input of **URT_x_NSS** state by reading **URT_x_NSS_SWI** register.

The registers of **URT_x_CTS_EN**, **URT_x_RTS_EN** and **URT_x_DE_EN** are used to enable the IO function of **URT_x_CTS**, **URT_x_RTS** and **URT_x_DE** signals. See “UART Hardware Flow Control” section for more information about CTS/RTS control.

The following diagram is showing the UART CTS/RTS/DE IO control block.

Figure 17-11. UART Other IO Control



17.8.2. UART IO Mode

Usually user can set the IO mode of the using IO pins to push-pull or open-drain for UART output signals. Also user can set the IO mode of the using IO pins to digital input or open-drain for UART input signals. For UART mode bidirectional signal, the IO mode of the using IO pins need set to open-drain.

For SPI mode bidirectional signal, the IO mode of the using IO pins need set to push-pull or open-drain. When selects push-pull mode, the SPI data signal will output by push-pull mode and be open-drain for idle or input state. When selects open-drain mode, the SPI data signal will always be open-drain for idle, input or output state.

17.8.3. UART Loop Back Mode

The UART module is built-in a loop back mode for internal self-testing by setting **URT_x_LBM_EN** register. When enables, the received input is taken from transmitted output to replace from input pin for RX and CTS input.

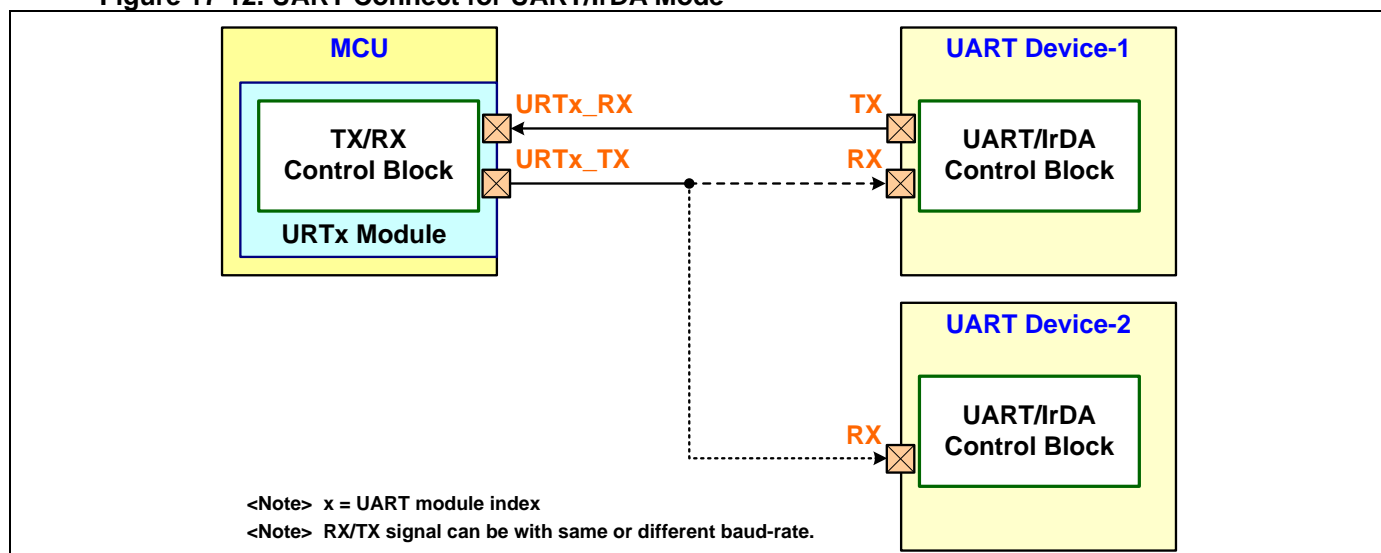
17.9. UART Connection for Application

17.9.1. UART Connect for UART/IrDA Mode

The following diagram is showing the UART application connection of UART/IrDA communication. The **URT_x_TX** signal is always as data output signal and the **URT_x_RX** signal is always as data input signal.

There is a UART HFC mode to do UART communication with two more signals of **URT_x_CTS** ("Clear to Send") and **URT_x_RTS** ("Request to Send"). Please refer the section of "UART Connection for Hardware Flow Control" for more detail information about UART HFC mode control.

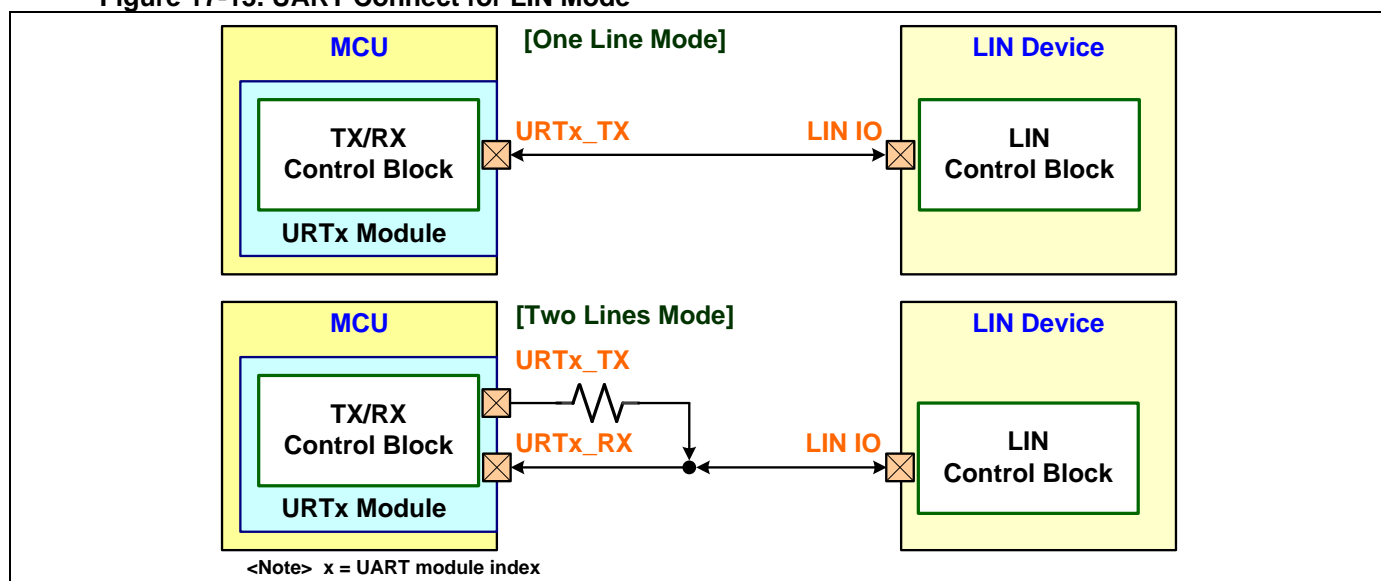
Figure 17-12. UART Connect for UART/IrDA Mode



17.9.2. UART Connect for LIN Mode

The following diagram is showing the UART application connection of LIN communication. The **URT_x_TX** signal is bi-directional as data input or output signal for one line mode. The **URT_x_TX** signal is always as data output signal and the **URT_x_RX** signal is always as data input signal for two lines mode.

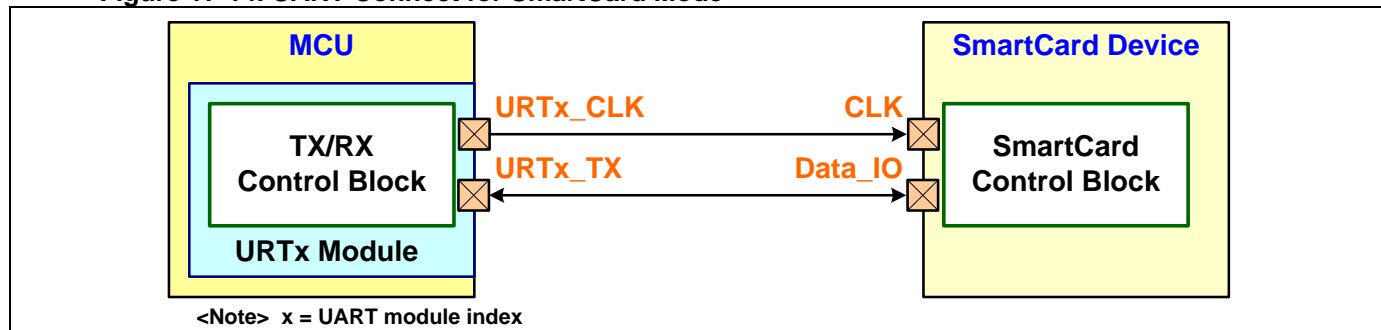
Figure 17-13. UART Connect for LIN Mode



17.9.3. UART Connect for SmartCard Mode

The following diagram is showing the UART application connection of SmartCard communication. The **URT_x_CLK** signal is outputted as clock signal. The **URT_x_TX** signal is bi-directional as data input or output signal.

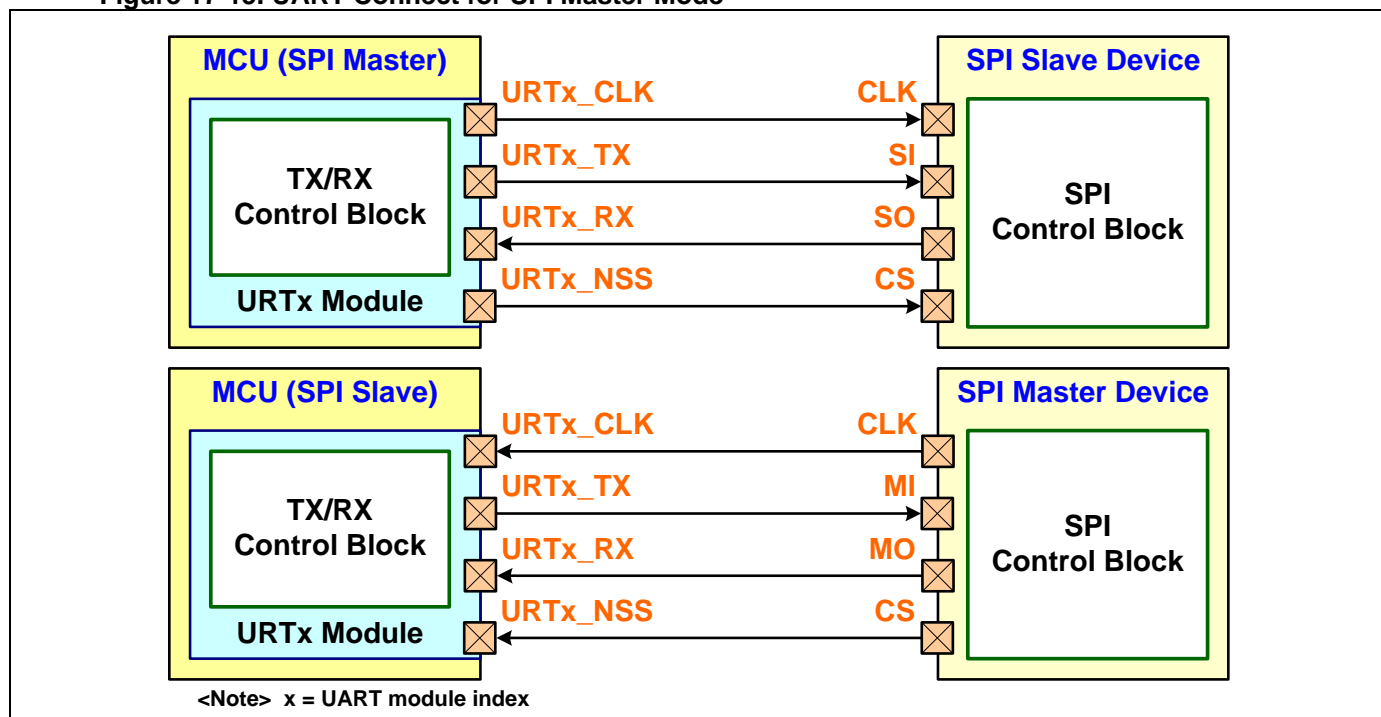
Figure 17-14. UART Connect for SmartCard Mode



17.9.4. UART Connect for SPI Mode

The following diagram is showing the UART application connection of SPI Master/Slave communication. The **URT_x_CLK** signal is as clock signal outputted for master mode and inputted for slave mode. The **URT_x_TX** signal is always as data output signal and the **URT_x_RX** signal is always as data input signal.

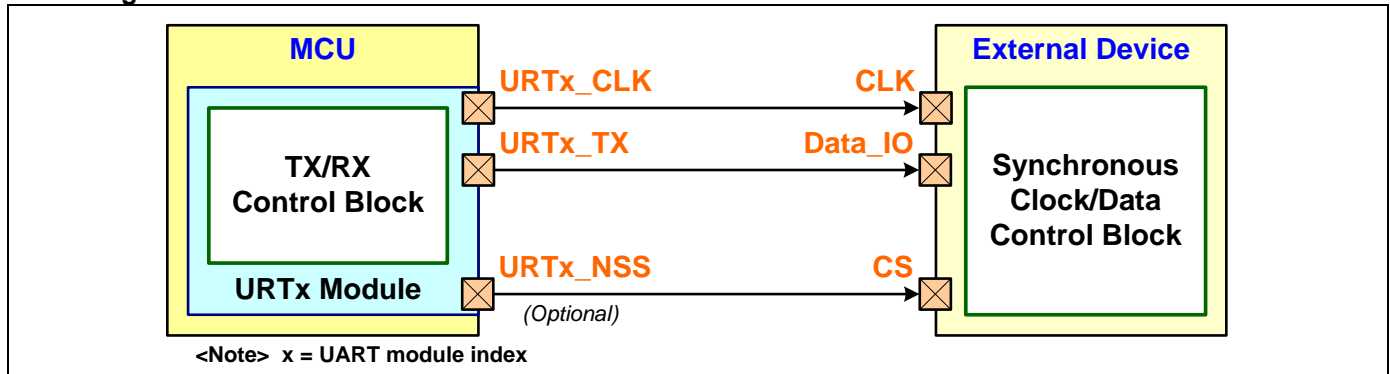
Figure 17-15. UART Connect for SPI Master Mode



17.9.5. UART Connect for SYNC Mode

The following diagram is showing the UART application connection of SYNC communication. The **URTx_CLK** signal is as clock signal outputted to external device inputted. The **URTx_TX** signal is bi-directional as data input or output signal for one line mode.

Figure 17-16. UART Connect for SYNC Mode



17.10. UART Character Format

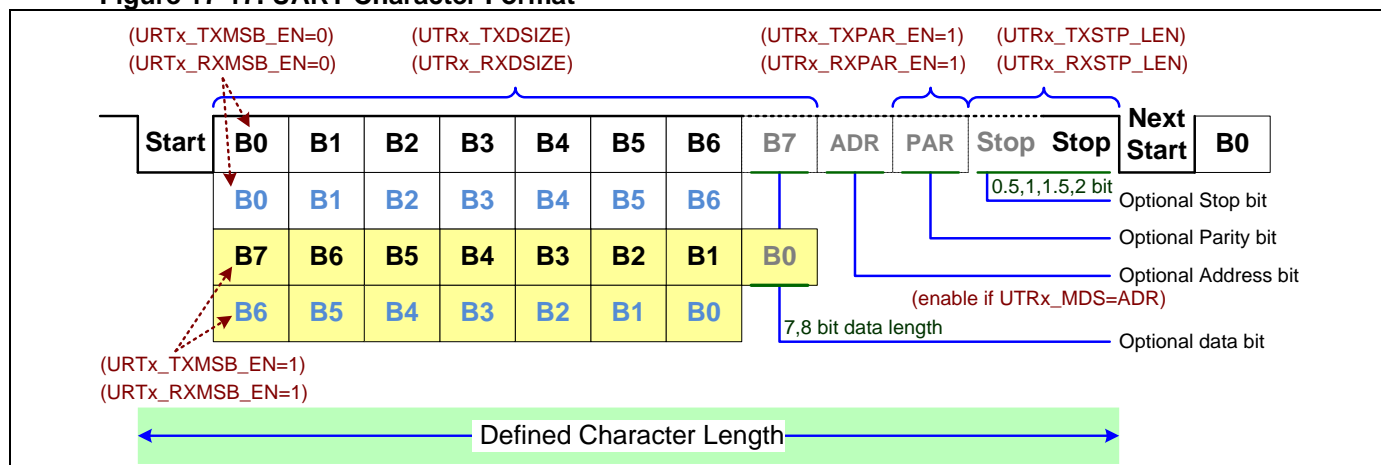
17.10.1. UART Data Character Format Setting

The UART character is defined as the data unit for UART transaction. Generally, the character is including of one Start bit, 8-bit or 7-bit data bits and one Stop bit. Others, it also can insert one parity bit (PAR) and one address bit (ADR) for multi-processor mode.

The related control registers of character setting are independent for transmission and receiving. The data bits can be configure as 8-bit or 7-bit data bits by setting **URT_x_TXDSIZE** and **URT_x_RXDSIZE** registers. The Stop bit can select the bit length by setting **URT_x_TXSTP_LEN** and **URT_x_RXSTP_LEN** registers. The parity bit can be enabled by setting **URT_x_TXPAR_EN** and **URT_x_RXPAR_EN** registers. These two registers are not able to be set for SYNC mode. The address bit is automatically inserted by hardware when the UART mode is set to "ADR" (Address-bit mode for multi-processor) in **URT_x_MDS** register. For URT4/5/6/7 modules, the registers of **URT_x_RXSTP_LEN**, **URT_x_RXPAR_EN** and **URT_x_RXDSIZE** are replaced by using TX/RX combined common registers of **URT_x_TXSTP_LEN**, **URT_x_TXPAR_EN** and **URT_x_TXDSIZE**.

User can select to the first transferred bit from Lsb or Msb by setting **URT_x_TXMSB_EN** and **URT_x_RXMSB_EN** registers. For URT4/5/6/7 modules, these modules are not support the **URT_x_TXMSB_EN** and **URT_x_RXMSB_EN** registers and the first transferred bit is fixed from Lsb.

Figure 17-17. UART Character Format



The following table is showing the register setting of character for different character format.

Table 17-4. UART Data Character Format Setting

URT _x Register					UART Data Character												
TXDSIZE RXDSIZE	TXMSB_EN RXMSB_EN	MDS	TXPAR_EN RXPAR_EN	TXSTP_LEN RXSTP_LEN													
0 (8-bit)	0	0~2	0	1 (1-bit)	S	B0	B1	B2	B3	B4	B5	B6	B7	P			
0 (8-bit)	0	0~2	0	3 (2-bit)	S	B0	B1	B2	B3	B4	B5	B6	B7	P	P		
0 (8-bit)	0	0~2	1	1 (1-bit)	S	B0	B1	B2	B3	B4	B5	B6	B7	PAR	P		
0 (8-bit)	0	0~2	1	3 (2-bit)	S	B0	B1	B2	B3	B4	B5	B6	B7	PAR	P	P	
0 (8-bit)	0	3	0	1 (1-bit)	S	B0	B1	B2	B3	B4	B5	B6	B7	ADR	P		
0 (8-bit)	0	3	0	3 (2-bit)	S	B0	B1	B2	B3	B4	B5	B6	B7	ADR	P	P	
0 (8-bit)	0	3	1	1 (1-bit)	S	B0	B1	B2	B3	B4	B5	B6	B7	ADR	PAR	P	
0 (8-bit)	0	3	1	3 (2-bit)	S	B0	B1	B2	B3	B4	B5	B6	B7	ADR	PAR	P	P
0 (8-bit)	1	0~2	0	1 (1-bit)	S	B7	B6	B5	B4	B3	B2	B1	B0	P			
0 (8-bit)	1	0~2	0	3 (2-bit)	S	B7	B6	B5	B4	B3	B2	B1	B0	P	P		
0 (8-bit)	1	0~2	1	1 (1-bit)	S	B7	B6	B5	B4	B3	B2	B1	B0	PAR	P		
0 (8-bit)	1	0~2	1	3 (2-bit)	S	B7	B6	B5	B4	B3	B2	B1	B0	PAR	P	P	
0 (8-bit)	1	3	0	1 (1-bit)	S	B7	B6	B5	B4	B3	B2	B1	B0	ADR	P		
0 (8-bit)	1	3	0	3 (2-bit)	S	B7	B6	B5	B4	B3	B2	B1	B0	ADR	P	P	
0 (8-bit)	1	3	1	1 (1-bit)	S	B7	B6	B5	B4	B3	B2	B1	B0	ADR	PAR	P	
0 (8-bit)	1	3	1	3 (2-bit)	S	B7	B6	B5	B4	B3	B2	B1	B0	ADR	PAR	P	P
1 (7-bit)	0	0~2	0	1 (1-bit)	S	B0	B1	B2	B3	B4	B5	B6		P			
1 (7-bit)	0	0~2	0	3 (2-bit)	S	B0	B1	B2	B3	B4	B5	B6		P	P		
1 (7-bit)	0	0~2	1	1 (1-bit)	S	B0	B1	B2	B3	B4	B5	B6		PAR	P		

1 (7-bit)	0	0~2	1	3 (2-bit)	S	B0	B1	B2	B3	B4	B5	B6	PAR	P	P
1 (7-bit)	0	3	0	1 (1-bit)	S	B0	B1	B2	B3	B4	B5	B6	ADR	P	
1 (7-bit)	0	3	0	3 (2-bit)	S	B0	B1	B2	B3	B4	B5	B6	ADR	P	P
1 (7-bit)	0	3	1	1 (1-bit)	S	B0	B1	B2	B3	B4	B5	B6	ADR	PAR	P
1 (7-bit)	0	3	1	3 (2-bit)	S	B0	B1	B2	B3	B4	B5	B6	ADR	PAR	P
1 (7-bit)	1	0~2	0	1 (1-bit)	S	B6	B5	B4	B3	B2	B1	B0	P		
1 (7-bit)	1	0~2	0	3 (2-bit)	S	B6	B5	B4	B3	B2	B1	B0	P	P	
1 (7-bit)	1	0~2	1	1 (1-bit)	S	B6	B5	B4	B3	B2	B1	B0	PAR	P	
1 (7-bit)	1	0~2	1	3 (2-bit)	S	B6	B5	B4	B3	B2	B1	B0	PAR	P	P
1 (7-bit)	1	3	0	1 (1-bit)	S	B6	B5	B4	B3	B2	B1	B0	ADR	P	
1 (7-bit)	1	3	0	3 (2-bit)	S	B6	B5	B4	B3	B2	B1	B0	ADR	P	P
1 (7-bit)	1	3	1	1 (1-bit)	S	B6	B5	B4	B3	B2	B1	B0	ADR	PAR	P
1 (7-bit)	1	3	1	3 (2-bit)	S	B6	B5	B4	B3	B2	B1	B0	ADR	PAR	P

B0~B7 = Data bit 0~7, S = Start bit, P = Stop bit

ADR : Address bit (1=slave address, 0=data), PAR : Parity bit (including of B0~B7 and ADR)

17.10.2. UART Parity Bit

User can set the function of parity bit in **URT_xTXPAR_POL**, **URT_xRXPOL**, **URT_xTXPAR_STK** and **URT_xRXPOL_STK** registers. The following table is showing the register setting of parity bit.

Table 17-5. UART Parity Bit Definitions

Parity Function	URT _x Register		Parity Bit Value
	TXPAR_POL RXPOL	TXPAR_STK RXPOL_STK	
Even	0	0	$PAR = B0 \oplus B1 \oplus B2 \oplus B3 \oplus B4 \oplus B5 \oplus B6 \oplus B7 \oplus ADR$
Odd	1	0	$PAR = \text{Not} (B0 \oplus B1 \oplus B2 \oplus B3 \oplus B4 \oplus B5 \oplus B6 \oplus B7 \oplus ADR)$
Fixed 0	0	1	$PAR = 0$
Fixed 1	1	1	$PAR = 1$

PAR : Parity bit (including of B0~B7 and ADR)

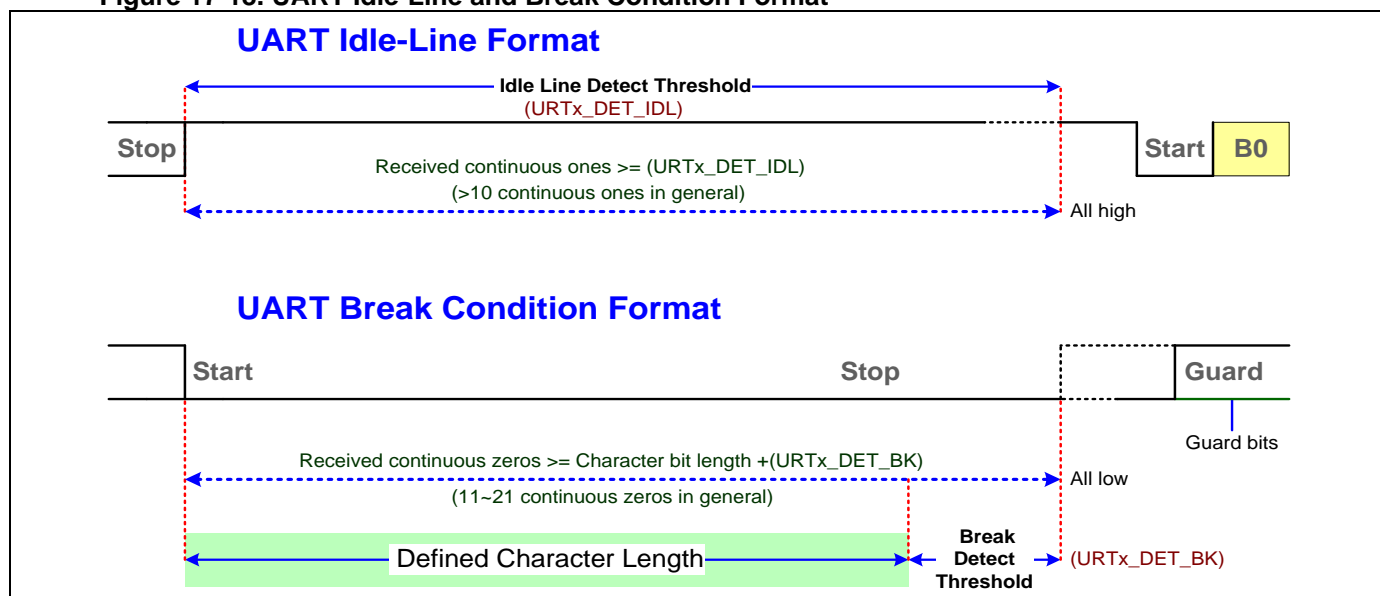
17.10.3. UART Idle-Line Format Setting

The UART module can detect an Idle-Line and user can define the bit time of Idle-Line from last Stop bit by setting **URT_xDET_IDL** register.

17.10.4. UART Break Condition Format Setting

The UART module can detect a Break condition and user can define the bit time of Break condition as one character length plus the extra bit time by setting **URT_xDET_BK** register.

Figure 17-18. UART Idle-Line and Break Condition Format



17.11. UART Fundamental Control

17.11.1. UART Control Mode Setting

The UART module is able to configure the control mode from one of **UART** (asynchronous mode), **SYNC** (synchronous mode), **IDLE** (multi-processor idle mode) and **ADR** (multi-processor address-bit mode). User can configure the control mode by setting **URT_x_MDS** register.

The following table is showing the register control availability of character data control register for different control modes.

Table 17-6. UART Control Mode vs. Character Data Control Register

UART Control Mode	URT _x Character Data Control Register					
	TXMSB_EN RXMSB_EN	TXDSIZE TXSTP_LEN	RXDSIZE RXSTP_LEN	TXPAR_EN TXPAR_POL TXPAR_STK	RXPAR_EN RXPAR_POL RXPAR_STK	TXOS_NUM RXOS_NUM
UART (Asynchronous 1/2-line)	v	v	v	v	v	3~31 (*1)
SYNC (Synchronous 1/2-line)	v	B				3~31 (*1)
IDLE (Multi-Processor Idle Line)	v	v	v	v	v	3~31 (*1)
ADR (Multi-Processor Address-bit)	v	v	v	v	v	3~31 (*1)

<Sign> v: Using available for independent TX or RX control , B: Using for both TX/RX control , Blank: Control no effect

<Note> *1: Value 3~31 indicates actual over-sampling number value 4~32.

For advanced UART modules of URT0/1/2, these control registers of previous table list are almost separated and independent control for UART data transmission and receiving. For basic UART modules of URT4/5/6/7, UART data transmission and receiving are almost simple control by combined common registers.

The following table is showing the register control availability for advanced UART modules and basic UART modules.

Table 17-7. UART Module vs. Character Data Control Register

UART Modules	URT _x Character Data Control Register						
	TXMSB_EN RXMSB_EN	TXOS_NUM TXDSIZE TXSTP_LEN	RXOS_NUM RXDSIZE RXSTP_LEN	TXPAR_EN	RXPAR_EN	TXPAR_POL TXPAR_STK	RXPAR_POL RXPAR_STK
URT0/1/2	v	v	v	v	v	v	v
URT4/5/6/7	-	B	-	B	-	v	v

<Sign> v: Using available for independent TX or RX control , B: Using for both TX/RX control , - : Not supported

17.11.2. UART Operation Mode Setting

The following table is showing the register setting of UART operation mode configuration for modules of URT0/1/2. The modules of URT4/5/6/7 are only support UART mode. User can set these registers to configure an operation mode for application product. Refer the related register description and the section of "[UART Connection for Application](#)" for more information.

The modes of UART, IrDA, HFC and SPI are used two data lines to do full duplex communication. The modes of LIN and SC (SmartCard) are used one data line or two data lines to do half duplex communication. The SYNC mode is like as SPI one line master mode that is specially used the **URT_x_RX_EN** register to control bidirectional data transfer for half duplex communication. The modes of SC, SYNC and SPI are using with one clock line to do synchronous communication.

Table 17-8. UART Operation Mode Setting

Operation Mode	URT _x Setting Register										
	MDS	DAT_LINE	HDX_EN	CLK_EN	CLK_CKS	TXE_MDS	RXE_MDS	TXPAR_EN RXPAR_EN	IR_EN	RTS_EN CTS_EN	SYNC_MDS
UART (asynchronous mode)	0	2	0	0	x	0	0	0/1	0	0	0
IrDA (Infrared Data Association)	0	2	0	0	x	0	0	0/1	1	0	0
HFC (hardware flow control)	0	2	0	0	x	0	0	0/1	0	1	0
LIN - one line	0	1	1	0	x	2	0	0	0	0	0

(Local Interconnect Network)											
LIN - two line (Local Interconnect Network)	0	2	1	0	x	2	0	0	0	0	0
SC (SmartCard)	0	1	1	1	1	1	1	1	0	0	0
SYNC (Synchronous 1-line)	1	1	0	1	0	0	0	0	0	0	0
SPI Master (Synchronous 2-line)	1	2	0	1	0	0	0	0	0	0	0
SPI Slave (Synchronous 2-line)	1	2	0	1	0	0	0	0	0	0	1
IDLE (Multi-Processor Idle Line)	2	2	0	0	x	0	0	0/1	0	0	0
ADR (Multi-Processor Address-bit)	3	2	0	0	x	0	0	0/1	0	0	0

<Sign> x: don't care ; 0/1: can be 0 or 1

17.11.3. UART Transmit

By UART data transaction, user needs to set the **URTx_TX_EN** register to enable transmission block functions and configure the operation mode for the data transmission. Refer the section of "[UART Operation Mode Setting](#)" for the operation mode settings.

When detect the TXF (**URTx_TXF**) flag, user can write the transmission data to the TX data register (**URTx_TDAT**) and the chip will automatically clear the TXF flag. After the time, User can write the next transmission data to the same TX data register when detects the TXF flag again. Repeat the sequence until the data transmission complete. For URT0/1/2/3 modules, the TX data register (**URTx_TDAT**) is implemented 32-bit which can support 8/16/32-bit write operation. For URT4/5/6/7 modules, the TX data register is only implemented 8-bit.

When the UART is running in one line bi-direction mode, the TCF flag (**URTx_TCF**) can denote that the previous data transmission is complete.

User can read **URTx_TNUM** register to get the remained data byte number of data register in real time. It usually is used to calculate the last transmission data length when communication error is happened.

User can configure the Stop bit length for UART data transmission by setting the **URTx_TXSTP_LEN** register. The actual Stop bit length is affected by the transmission oversampling setting number in **URTx_TXOS_NUM** register. For URT4/5/6/7 modules, the Stop bit length is only supported 1-bit and 2-bit in the register of **URTx_TXSTP_LEN**.

The following table is showing the UART TX Stop bit Length setting table.

Table 17-9. UART TX Stop Bit Length Setting

Stop Bit Request	URTx Register		Actual STOP Bit Length
	TXSTP_LEN	TXOS_NUM	
0.5-Bit	0	3,5,7, ~ , 31	0.5 bit
		4,6,8, ~ , 30	0.5 bit + one CK_URTx_TX clock time
1-Bit	1	3 ~ 31	1 bit
1.5-Bit	2	3,5,7, ~ , 31	1.5 bit
		4,6,8, ~ , 30	1.5 bit + one CK_URTx_TX clock time
2-Bit	3	3 ~ 31	2 bit

<Note> TX Oversampling number = TXOS_NUM+1

17.11.4. UART Receive

By UART data transaction, user needs to set the **URTx_RX_EN** register to enable receiving block functions and configure the operation mode for the data receiving. Refer the section of “[UART Operation Mode Setting](#)” for the operation mode settings.

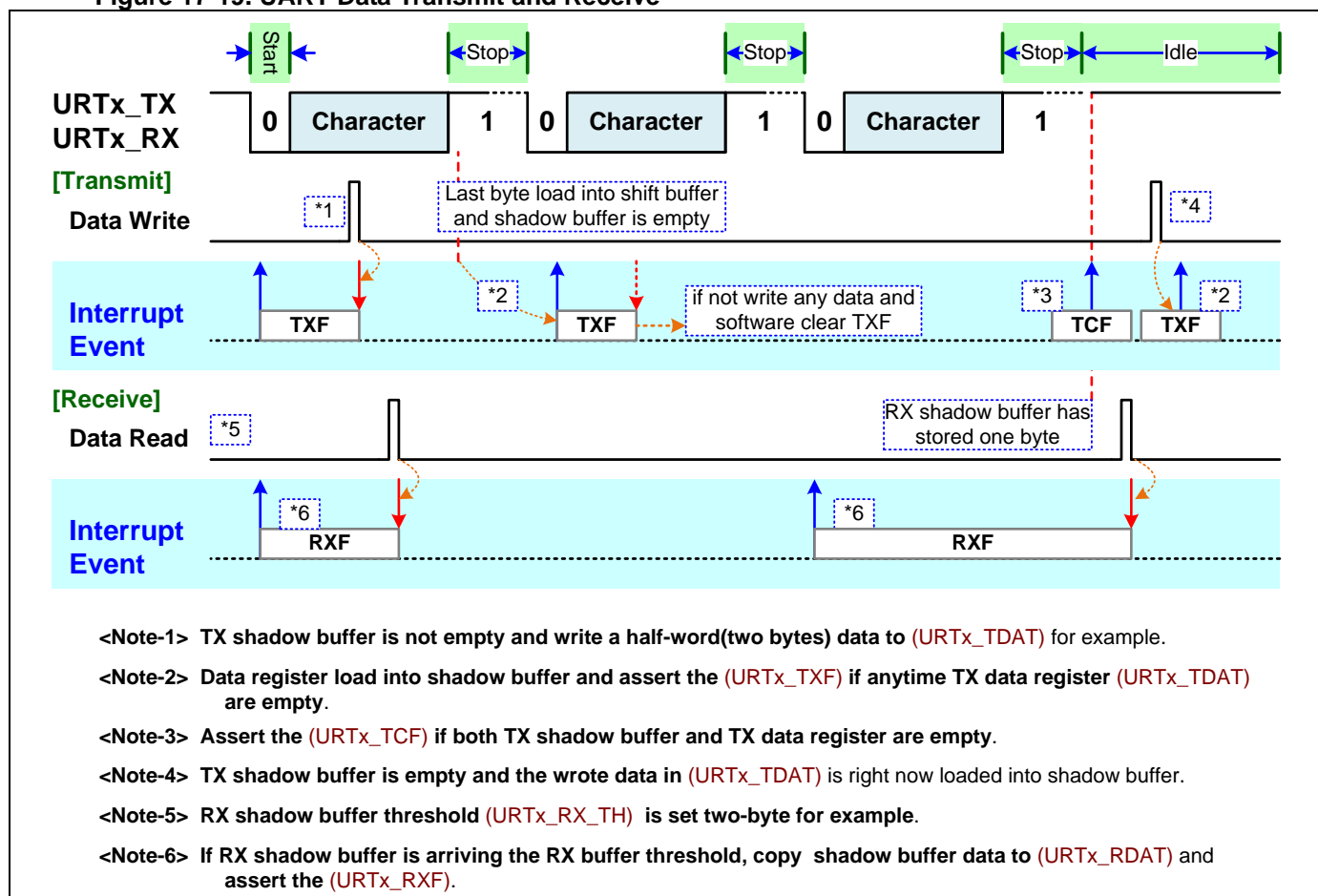
As same as data transmission, user can read the received data from the RX data register (**URTx_RDAT**) when detect the RXF flag (**URTx_RXF**) and the chip will automatically clear the RXF flag. By same way, User can read the received data when every time detects the RXF flag. Repeat the sequence until the data receiving complete. For URT0/1/2/3 modules, the RX data register (**URTx_RDAT**) is implemented 32-bit which can support 8/16/32-bit read operation. For URT4/5/6/7 modules, the RX data register is only implemented 8-bit.

The **URTx_RX_TH** register is used to set the data byte threshold of shadow buffer. When the data byte number of shadow buffer is equal to the threshold, hardware will copy the shadow buffer content to data register.

For last tail data control, user can check the RXDF flag (**URTx_RXDF**) and **URTx_RNUM** register. RXDF flag indicates received data byte number is different from previous received data byte number. **URTx_RNUM** register indicates received data byte number when data shadow buffer last transfer to **URTx_RDAT** register. Firmware can write an initial value for received byte number comparison.

User can configure the Stop bit length for UART data receiving by setting the **URTx_RXSTP_LEN** register. For URT4/5/6/7 modules, the **URTx_RXSTP_LEN** register is replaced by using TX/RX combined common register of **URTx_TXSTP_LEN**.

Figure 17-19. UART Data Transmit and Receive



17.12. UART Data Buffer

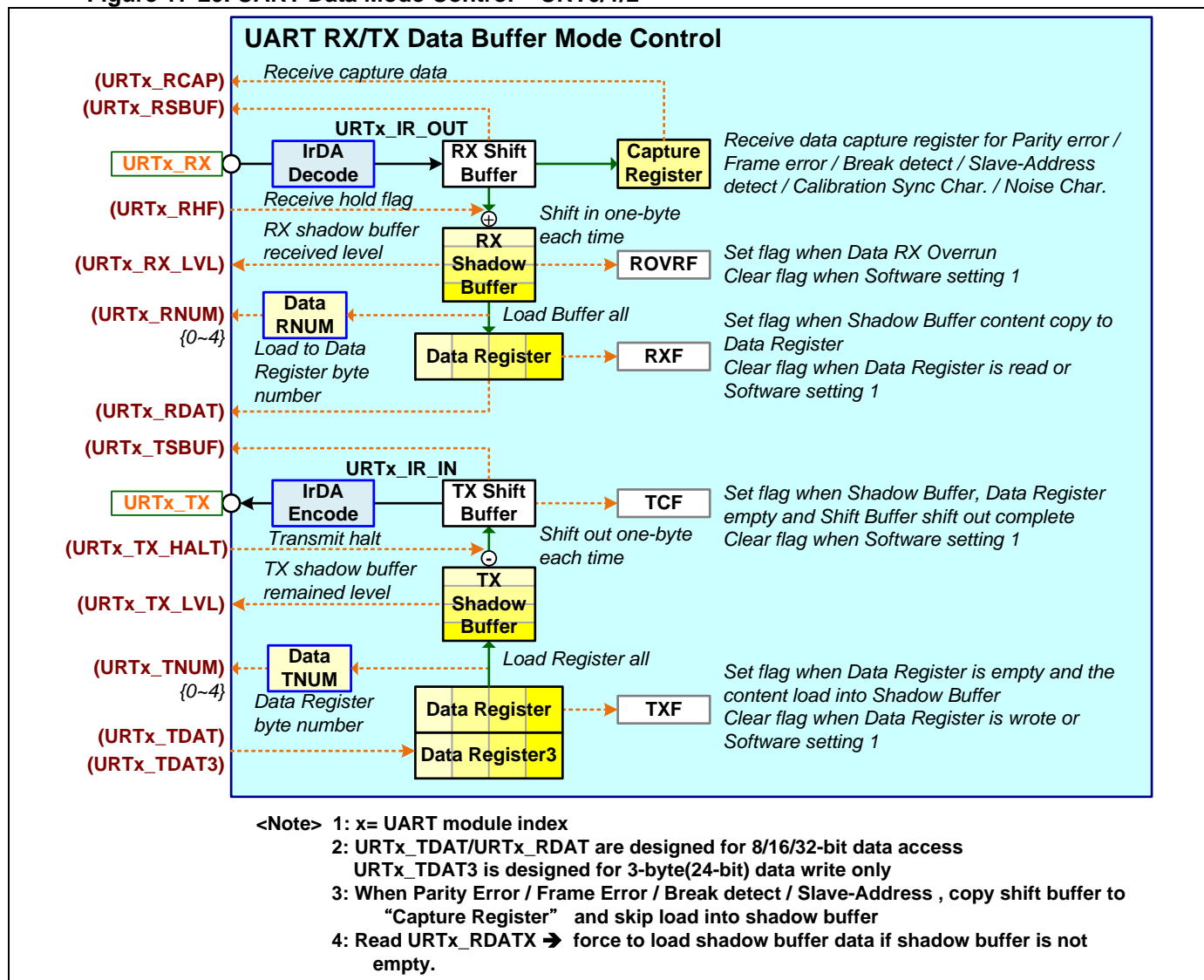
The UART module implements two 8-bit shift buffers, two 32-bit shadow buffer and two 32-bit data register for data flow control and reduce the CPU overhead. User can easy use the event flags of TXF (**URT_x_TXF**) and RXF (**URT_x_RXF**) for data transfer flow control.

❖ Advanced UART

The data path of advanced UART is including of IrDA Decode/Encode, RX/TX shift buffer, RX/TX shadow buffer and RX/TX 32-bit data registers.

The following diagram is showing the UART Data Buffer mode control block for URT0/1/2.

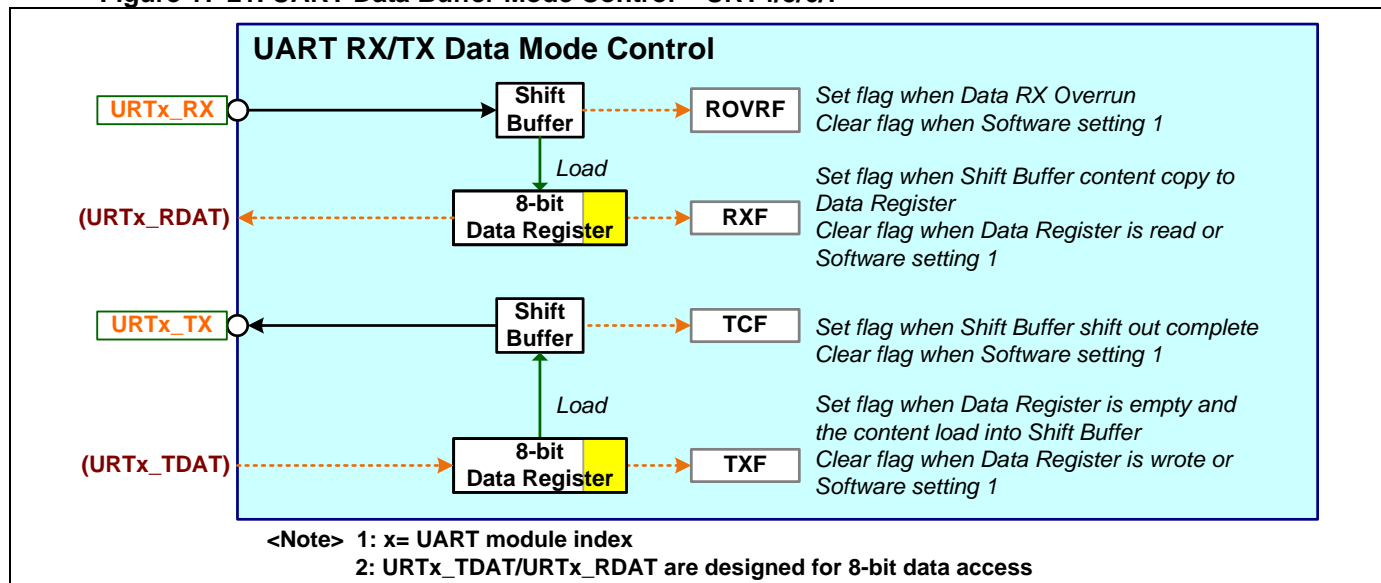
Figure 17-20. UART Data Mode Control – URT0/1/2



❖ Basic UART

The data path of basic UART is including of RX/TX shift buffer and RX/TX 8-bit data registers. The following diagram is showing the UART Data Buffer mode control block for URT4/5/6/7.

Figure 17-21. UART Data Buffer Mode Control – URT4/5/6/7



17.12.1. UART Data Buffer Control

The RXF flag (**URT_x_RXF**) will be activated at every time that the RX Shadow Buffer content is copied to RX data register. Also the related interrupt will be asserted if the **URT_x_RX_IE** register is enabled. The RHF flag (**URT_x_RHF**) is generate by some data hold trigger events and use to hold the received data to load into shadow buffer. See “UART Receive Hardware Hold and Capture Control” section for more information.

The TXF flag (**URT_x_TXF**) will be activated at every time that the TX data register content is copied to TX Shadow Buffer. Also the related interrupt will be asserted if the **URT_x_TX_IE** register is enabled. When both Shadow Buffer and TX data register are empty, the TCF flag (**URT_x_TCF**) will be activated if the Shift Buffer shifts out completely. User can set the **URT_x_TX_HALT** register to halt the data transmission for software data flow control.

The data registers of **URT_x_TDAT** and **URT_x_RDAT** are designed for 8/16/32-bit data access and the register of **URT_x_TDAT3** is designed for 3-byte (24-bit) data write only. When user write one any of 8/16/32-bit data to the **URT_x_TDAT3** register, the chip will do as a 3-byte (24-bit) data to be written into.

17.12.2. UART Data Inverse

User can enable to inverse the received or transmitted data bits from “0” to “1” or “1” to “0” by setting **URT_x_RDAT_INV** or **URT_x_TDAT_INV** registers. When enables the data inversion function, the received or transmitted data bits are inverted but Start, Stop, Address and Parity bits are not inverted.

17.12.3. UART Shift Buffer

The UART module implements two independent 8-bit shift buffers for data receiving and transmission. There are two registers of **URT_x_RSBUF** and **URT_x_TSBUF** those can be read to get the data bits of received and transmitted shift buffer in real time. Also user can read **URT_x_ADR** and **URT_x_PAR** register bits to get the Address and Parity bits from received shift buffers in real time.

17.12.4. UART Data Buffer Clear

For firmware data control, user can force to clear the received or transmitted data buffer by setting **URT_x_RDAT_CLR** or **URT_x_TDAT_CLR** registers. These two register bit are set by software and clear by hardware.

When enables the register bit of **URT_x_RDAT_CLR**, the received data register and shadow buffer will be flushed. Also **URT_x_RXF** flag and **URT_x_RX_LVL** is cleared. When enables the register bit of **URT_x_TDAT_CLR**, the transmitted data register and shadow buffer will be flushed. Also **URT_x_TXF** flag is set and **URT_x_TX_LVL** is cleared.

17.13. UART Multi-Processor

17.13.1. UART Multi-Processor Operation Mode

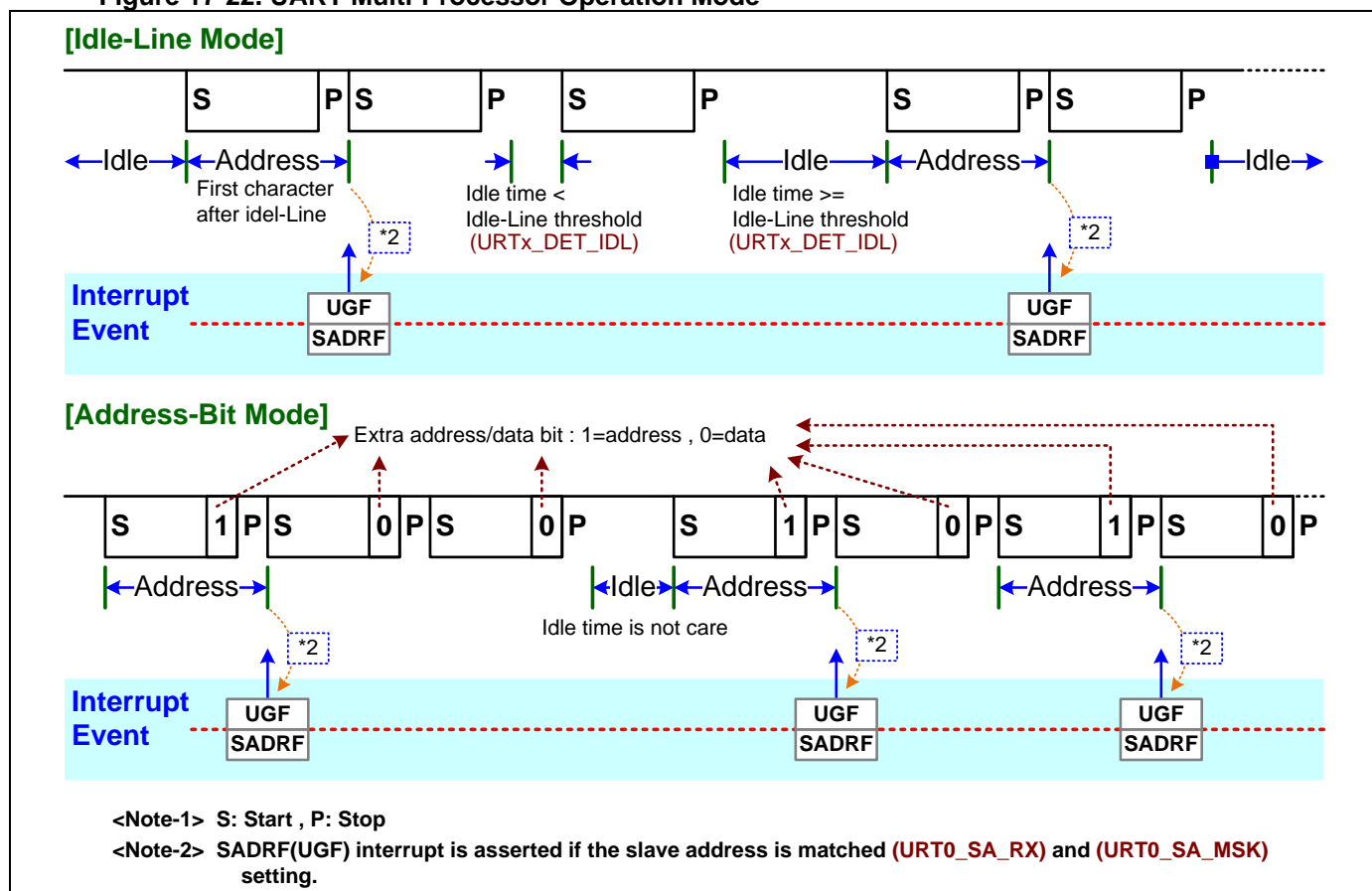
The UART module implements two operation modes of Idle-Line mode or Address-Bit mode for multi-processor communication by setting the **URTx_MDS** register.

When receives the address character for Idle-Line mode or the address-bit for Address-Bit mode, the SADRf flag (**URTx_SADRf**) will be activated and asserts the interrupt if the related interrupt enable register of **URTx_SADR_IE** is enabled.

The slave address is defined in **URTx_SA_RX** and **URTx_SA_MSK** registers.

The following diagram is showing the time sequence of UART Multi-Processor operation mode.

Figure 17-22. UART Multi-Processor Operation Mode



17.13.2. UART Multi-Processor and Mute mode

When the UART is running in multi-processor mode, user can configure by registers to enter the mute mode when receives the salve address is unmatched and exit the mute mode when receives the salve address is matched. See the description of "UART Mute Mode Control" for more information about mute mode.

17.13.3. UART Multi-Processor Address Setting

User can set the received address in **URTx_SA_RX** register and set the address mask by setting **URTx_SA_MSK** register.

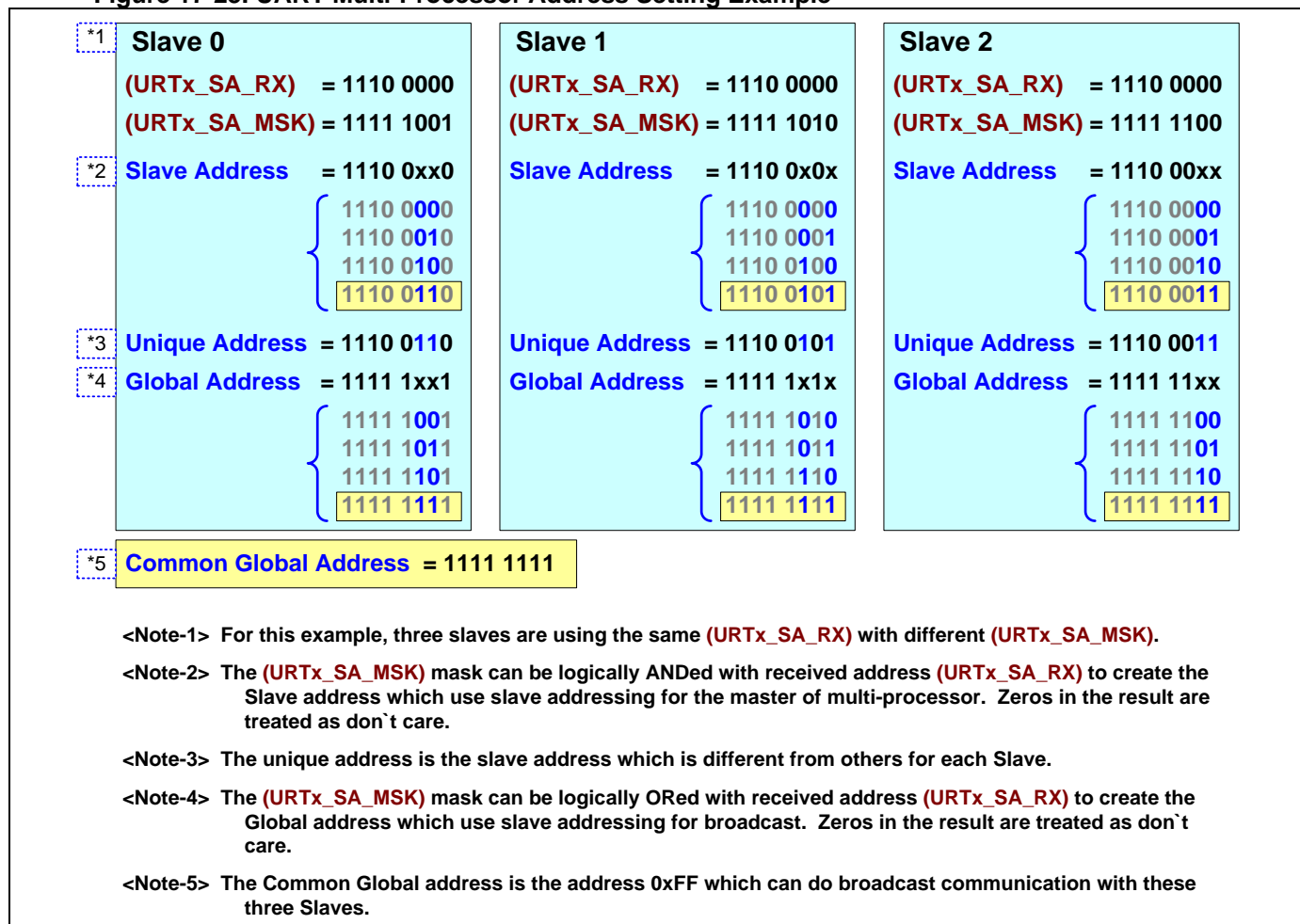
The following diagram is showing the example of UART Multi-Processor slave address setting.

For this example, three slaves are using the same slave address (**URTx_SA_RX**) with different address mask (**URTx_SA_MSK**). The address mask register of **URTx_SA_MSK** can be logically ANDed with received address register of **URTx_SA_RX** to create the final **slave address** which use slave addressing for the master of multi-processor. Zeros in the result are treated as don't care.

The unique address is the slave address which is different from others for each Slave. The address mask register of **URTx_SA_MSK** can be logically ORed with received address register of **URTx_SA_RX** to create the **global address** which use slave addressing for broadcast. The Common Global address is the address **0xFF**

which can do broadcast communication with these three Slaves. The multi-processor global slave address can be enabled in the register of **URT_x_GSA_EN**.

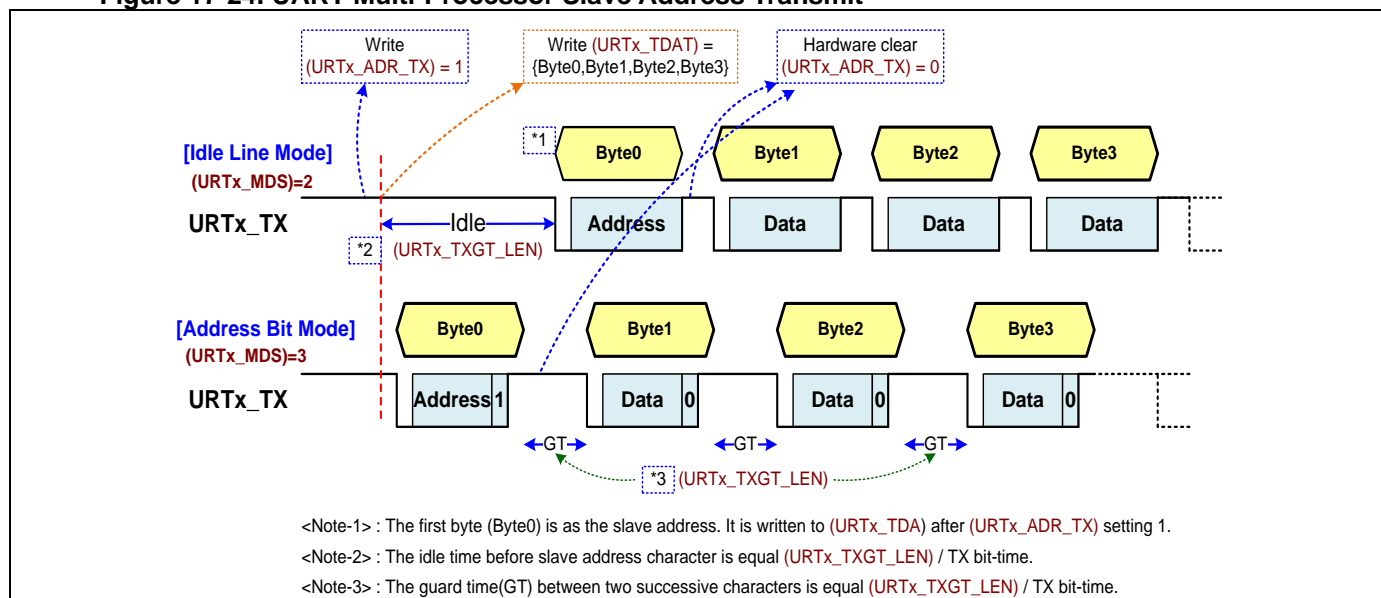
Figure 17-23. UART Multi-Processor Address Setting Example



17.13.4. UART Multi-Processor Slave Address Transmit

User can send an address character for Idle-Line mode or an address-bit for Address-Bit mode by setting **URT_x_ADR_TX** register to 1. For Idle-Line mode, user can set the idle line time by setting **URT_x_TXGT_LEN** register.

Figure 17-24. UART Multi-Processor Slave Address Transmit

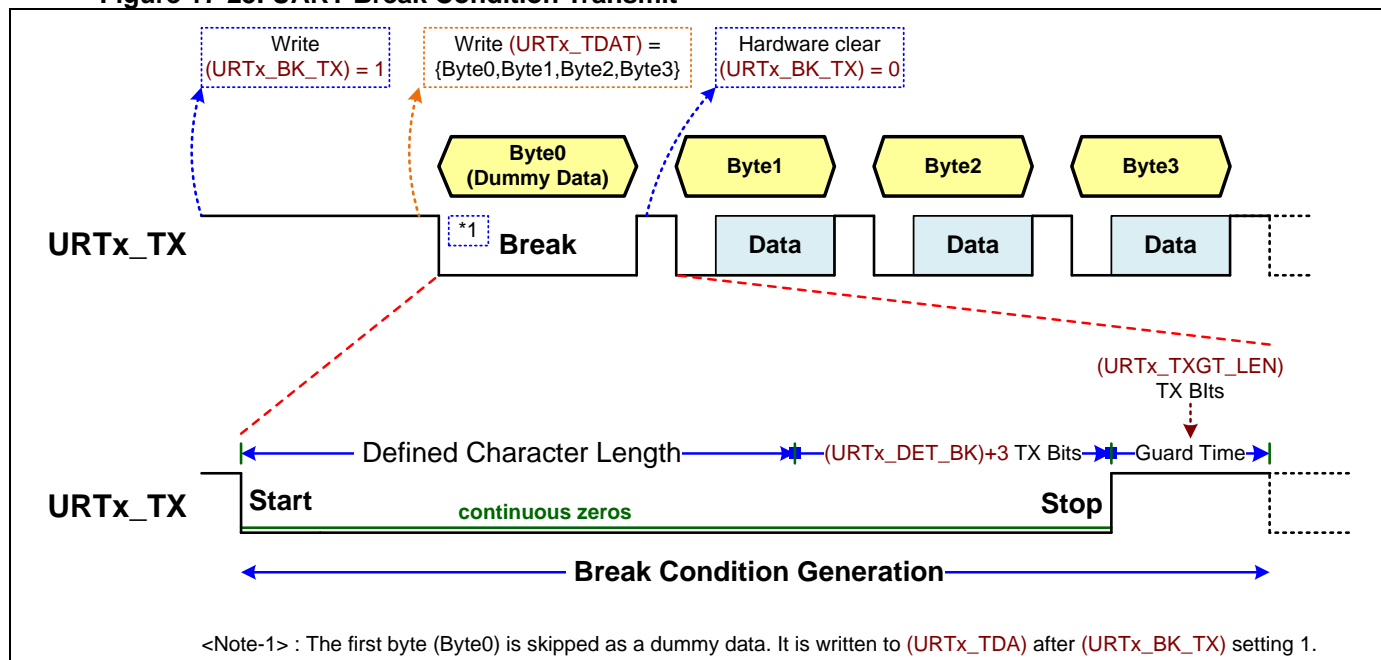


17.14. UART Break Condition Transmit

User can send a Break condition character by setting **URT_x_BK_TX** register to 1. A flag of **URT_x_BKBF** can be read to indicate the busy status of Break sending process. When this bit is "1", it is meaning the Break sending process is not yet finished. When this bit is "0", it is meaning that is finished.

The Break condition character time is equal a character bit time plus (**URT_x_DET_BK** register setting value +3). The **URT_x_TXGT_LEN** can use to set the guard time between two transmitted characters.

Figure 17-25. UART Break Condition Transmit



The following table is showing the UART break condition send and detect control setting table.

Table 17-10. UART Break Condition Send and Detect Control

Control Mode	URT _x Register			Break Sending Length / BKF flag asserted Position
	DET_BK	TXSTP_LEN	RXSTP_LEN	
Send Break Condition (*1)	0	0,1,2,3	-	Break Length = 14 bits
	1	0,1,2,3	-	Break Length = 16 bits
Detect Break Condition (*2)	0	-	0,1,2	BKF asserted at 10.5th bit
			3	BKF asserted at 11.5th bit
	1	-	0,1,2	BKF asserted at 12.5th bit
			3	BKF asserted at 13.5th bit

*1 : Set TXDSIZE=8bit, TXPAR_EN=Disable, MDS= {0~2}

*2 : Set RXDSIZE=8bit, RXPAR_EN=Disable, MDS= {0~2}

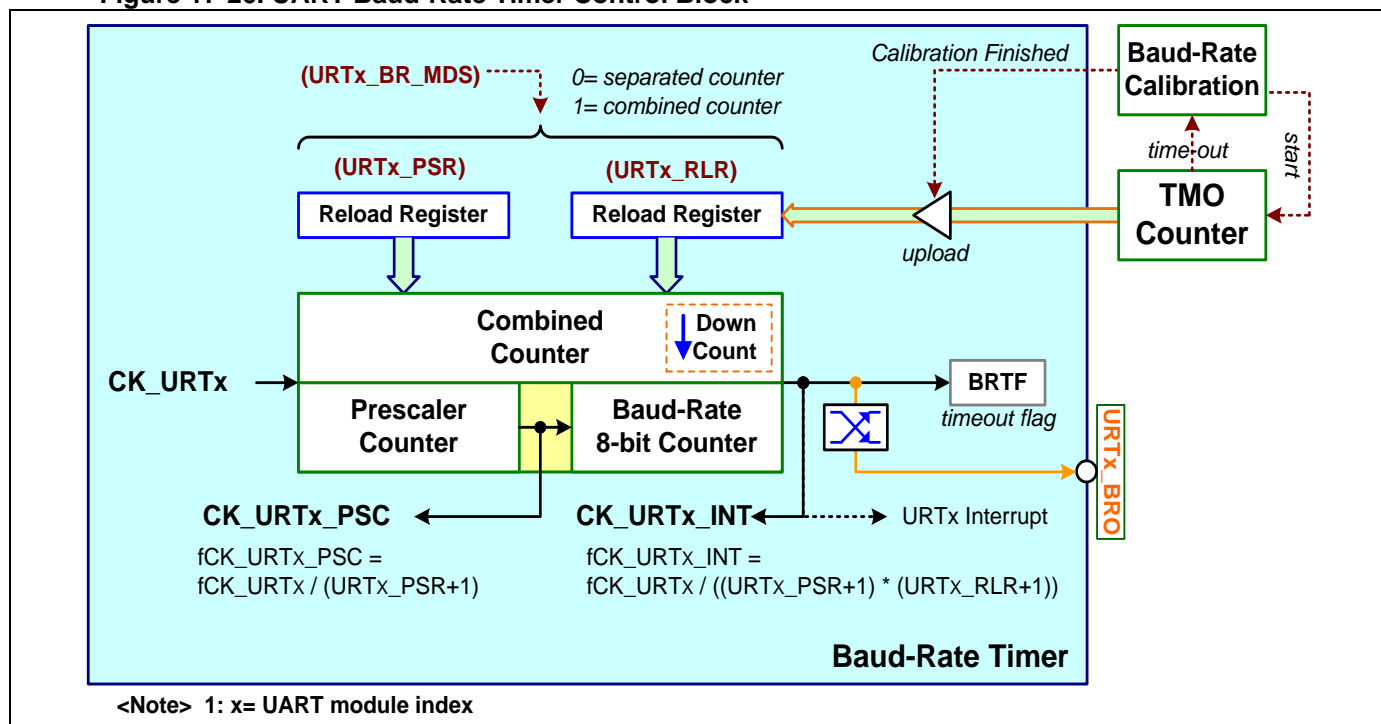
17.15. UART Baud-Rate Control

The Baud-Rate timer (BR) can be configured to separated counter mode or combined counter mode to do as an UART Baud-Rate generator or a general using timer by setting **URT_x_BR_MDS** register and enable by setting **URT_x_BR_EN** register.

The Baud-Rate timer generator is able to output the internal clock for UART communication Baud-Rate control. User can configure the Baud-Rate generator by setting **URT_x PSR** and **URT_x RLR** registers.

Also user can get the real time counter value of BR timer by reading **URT_x_BR_PSC** and **URT_x_BR_CNT** registers. The following diagram is showing the UART Baud-Rate Timer Control block.

Figure 17-26. UART Baud-Rate Timer Control Block



17.15.1. UART Baud-Rate Timer On/Off Control

When the BR timer is configured as a general using timer, the timer enabled function is no effect by **URTx EN** register. The following table is showing the UART Baud-Rate Timer On/Off Control.

Table 17-11. UART BR Baud-Rate Timer On/Off Control

BR Timer Function	Counter Mode	URTx Register			Timer On/Off
		BR_MDS	EN	BR_EN	
URTx Baud-Rate Generator	Separated	0	0	x	Timer Off
			1	0	Timer Off
				1	Timer On
	Combined	1	0	x	Timer Off (*1)
			1	0	Timer Off
				1	Timer On
General Using Timer	Combined	1	x	0	Timer Off
				1	Timer On

<Sign> x : Don't care; *1 : BR EN=1, BR Timer On but BR generator cannot work for UART

17.15.2. UART Baud-Rate Calibration

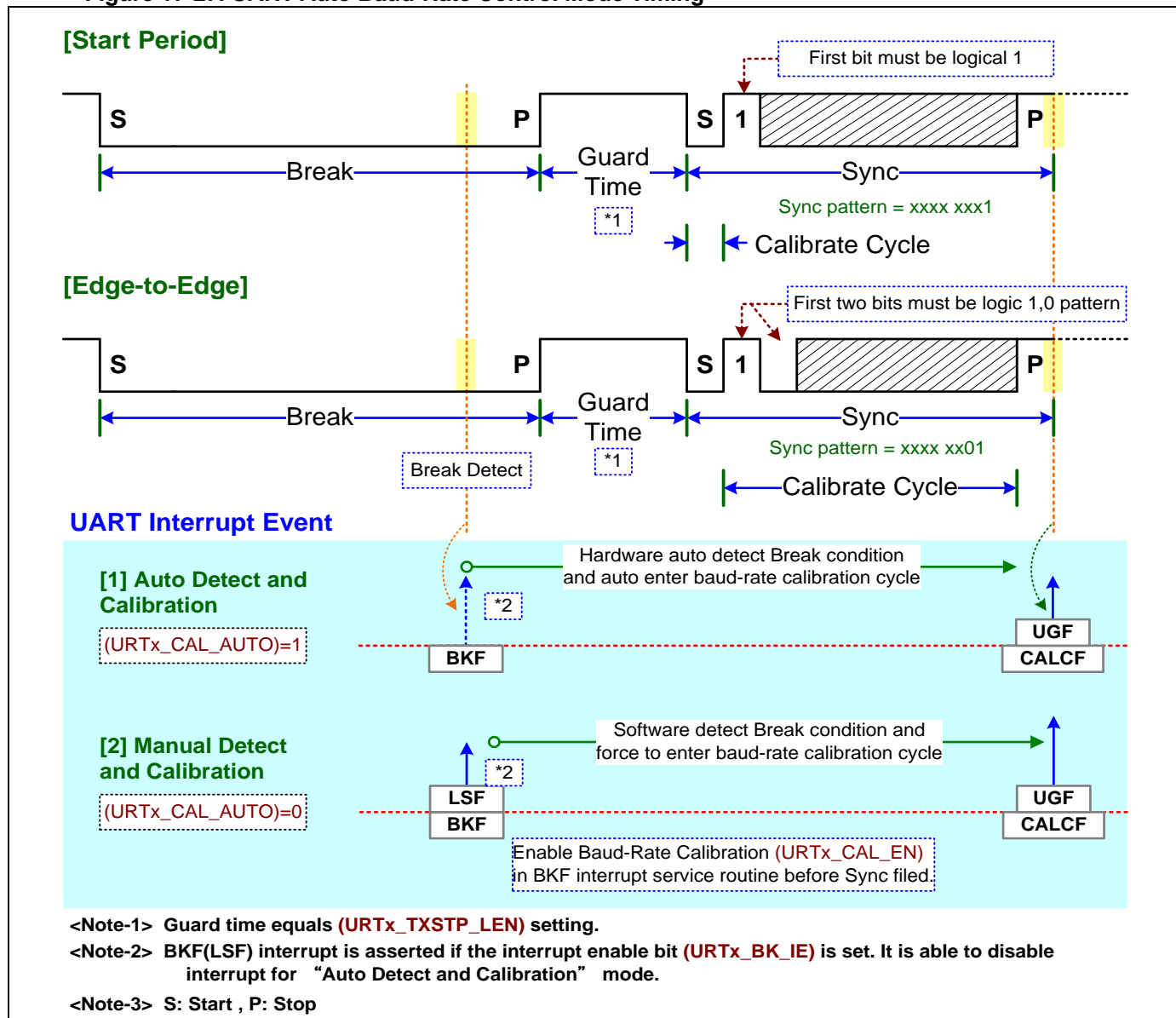
The UART module can receive a Break condition and activates the BKF flag (**URT_x_BKF**) and asserts the interrupt if the related interrupt enable register of **URT_x_BK_IE** is enabled. User can enable the baud-rate calibration by setting **URT_x_CAL_EN** register and the chip will asserted the CALCF flag (**URT_x_CALCF**) when the calibration is completed. The UART baud-rate calibration is able to support Baud-Rate timer in separated and combined modes.

After the calibration is completed, normally the chip will update the Baud-Rate timer preload value in **URTx_PSR** and **URTx_RLR** registers by calibration result. If the calibration is failure by Baud-Rate timer counting overflow or underflow, the chip will not update the Baud-Rate timer preload value. User can get the calibration overflow or underflow condition in **URTx_CALOVF** and **URTx_CALUDF** register flags after calibration completed.

Also user can set **URTx_CAL_AUTO** register to enable hardware auto baud-rate calibration. When the auto baud-rate calibration function is enabled and receives a Break condition, chip will automatically enter baud-rate calibration process during Sync character received cycle.

The UART is supported two calibration modes of Start mode and Edge mode by setting **URTx_CAL_MDS** register. The Start mode calibrates the baud-rate by measuring the Start bit only and the Sync character must be (xxxx xxx1)₂. The Edge mode calibrates the baud-rate by measuring the start falling edge to next falling edge and the Sync character must be (xxxx xx01)₂. (x = 0 or 1)

Figure 17-27. UART Auto Baud-Rate Control Mode Timing



17.15.3. UART BR Timeout Signal Output

The BR timer can output the timeout signal to do as a clock signal to external port of **URTx_BRO** or internal other module. User can set the initial state of timeout signal by setting **URTx_BRO_STA** register. Specially, the initial state register can be written only when the register of **URTx_BRO_LCK** is written "1" simultaneously.

17.15.4. UART Baud-Rate Setting Examples

The following table is showing the examples of UART general using baud-rate setting.

Table 17-12. UART General Using Baud-Rate Setting Examples

UART Baud-Rate		CK_URTx Frequency (MHz)	URTx Register			Actual Baud-Rate
Baud-Rate (*2)	Error (%)		PSR	RLR	OS_NUM (*1)	
1200	0.00	48.000	9	249	15	1200.000
2400	0.00	48.000	9	124	15	2400.000
4800	0.00	48.000	4	124	15	4800.000
9600	0.00	48.000	4	99	9	9600.000
14400	-0.01	48.000	2	100	10	14401.440
19200	0.00	48.000	4	49	9	19200.000
28800	-0.04	48.000	6	16	13	28811.525
38400	0.00	48.000	4	24	9	38400.000
57600	-0.04	48.000	6	6	16	57623.049
115200	-0.16	48.000	1	12	15	115384.615
230400	-0.16	48.000	1	7	12	230769.231
460800	-0.16	48.000	1	3	12	461538.462
576000	0.79	48.000	1	2	13	571428.571
921600	-0.16	48.000	1	1	12	923076.923
1152000	0.79	48.000	2	0	13	1142857.143
2000000	0.00	48.000	1	0	11	2000000.000
3000000	0.00	48.000	1	0	7	3000000.000
4000000	0.00	48.000	1	0	5	4000000.000
6000000	0.00	48.000	1	0	3	6000000.000
1200	-0.01	8.000	2	201	10	1200.120
2400	-0.01	8.000	2	100	10	2400.240
4800	-0.04	8.000	6	16	13	4801.921
9600	-0.04	8.000	6	6	16	9603.842
14400	0.08	8.000	0	138	3	14388.489
19200	-0.16	8.000	1	12	15	19230.769
28800	0.44	8.000	2	2	30	28673.835
38400	-0.16	8.000	0	12	15	38461.538
57600	-0.64	8.000	1	2	22	57971.014
115200	-0.64	8.000	0	2	22	115942.029
230400	0.79	8.000	0	4	6	228571.429
576000	0.79	8.000	0	0	13	571428.571
1152000	0.79	8.000	0	0	6	1142857.143

*1 : OS_NUM = URTx_RXOS_NUM or URTx_TXOS_NUM register value

*2 : Baud-Rate = $f(\text{CK_URTx}) / (\text{PSR}+1) / (\text{RLR}+1) / (\text{OS_NUM}+1)$

17.16. UART Data Receive and Sampling

The UART receiver data sampling mode is able to select the oversampling majority vote from three samples or one sample by setting **URT_x_OS_MDS** register. The oversampling number is programmable by setting **URT_x_RXOS_NUM** register. For URT4/5/6/7 modules, the **URT_x_RXOS_NUM** register is replaced by using TX/RX combined common register of **URT_x_TXOS_NUM**.

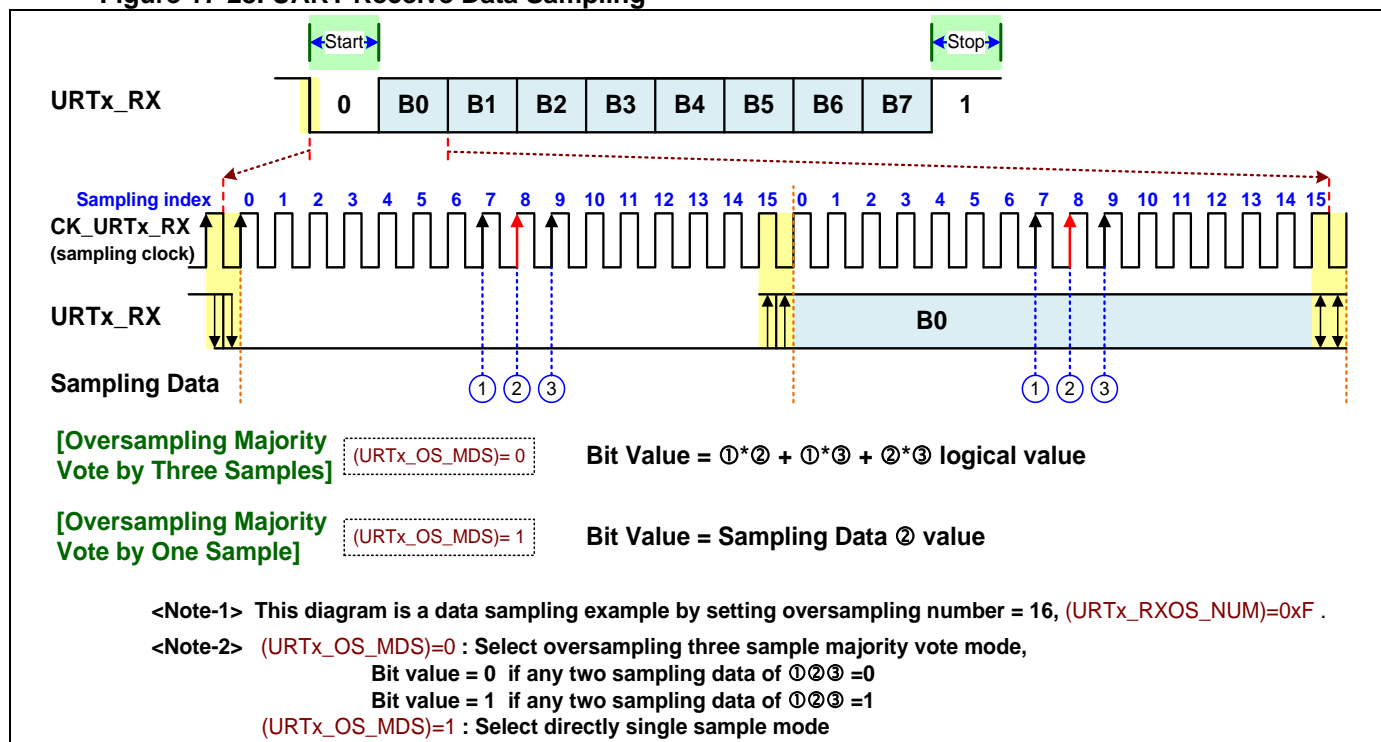
17.16.1. UART Data Sampling

User can select the valid oversampling number for data sampling in **URTx_RXOS_NUM** register.

The default oversampling majority vote is three samples in **URTx_OS_MDS** register. The UART module will decide the received bit value from the three sample value. Also user can select oversampling majority vote by one sample by setting **URTx_OS_MDS** register to 1 but it gets the worse transmission noise immunity generally. Usually this setting is used to raise the clock rate and communication speed.

The following diagram is showing the UART receive data sampling example for oversampling number=16.

Figure 17-28. UART Receive Data Sampling



The following table is showing the received bit value and noise bit status of NCF flag (**URTx_NCF**) for three samples mode. The received bit value is “0” if any two samples are “0” and is “1” if any two samples are “1”.

Table 17-13. UART Received Data Oversampling and Noise Detection

Sampling Sequence Value	Received bit		NCF status	
000	0	any two =0	0	all samples =0
001	0	any two =0	1	
010	0	any two =0	1	
011	1	any two =0	1	
100	0	any two =0	1	
101	1	any two =0	1	
110	1	any two =0	1	
111	1	any two =0	0	all samples =1

17.16.2. UART Receive Noise Character

In general the UART data receiving signal of URTx_RX can be filter by internal filter and get the valid data by oversampling majority vote. The UART module can detect the noised character bit and reflect the NCF flag. It is useful to provide the information for user to improve the UART communicated correction on application

hardware or firmware.

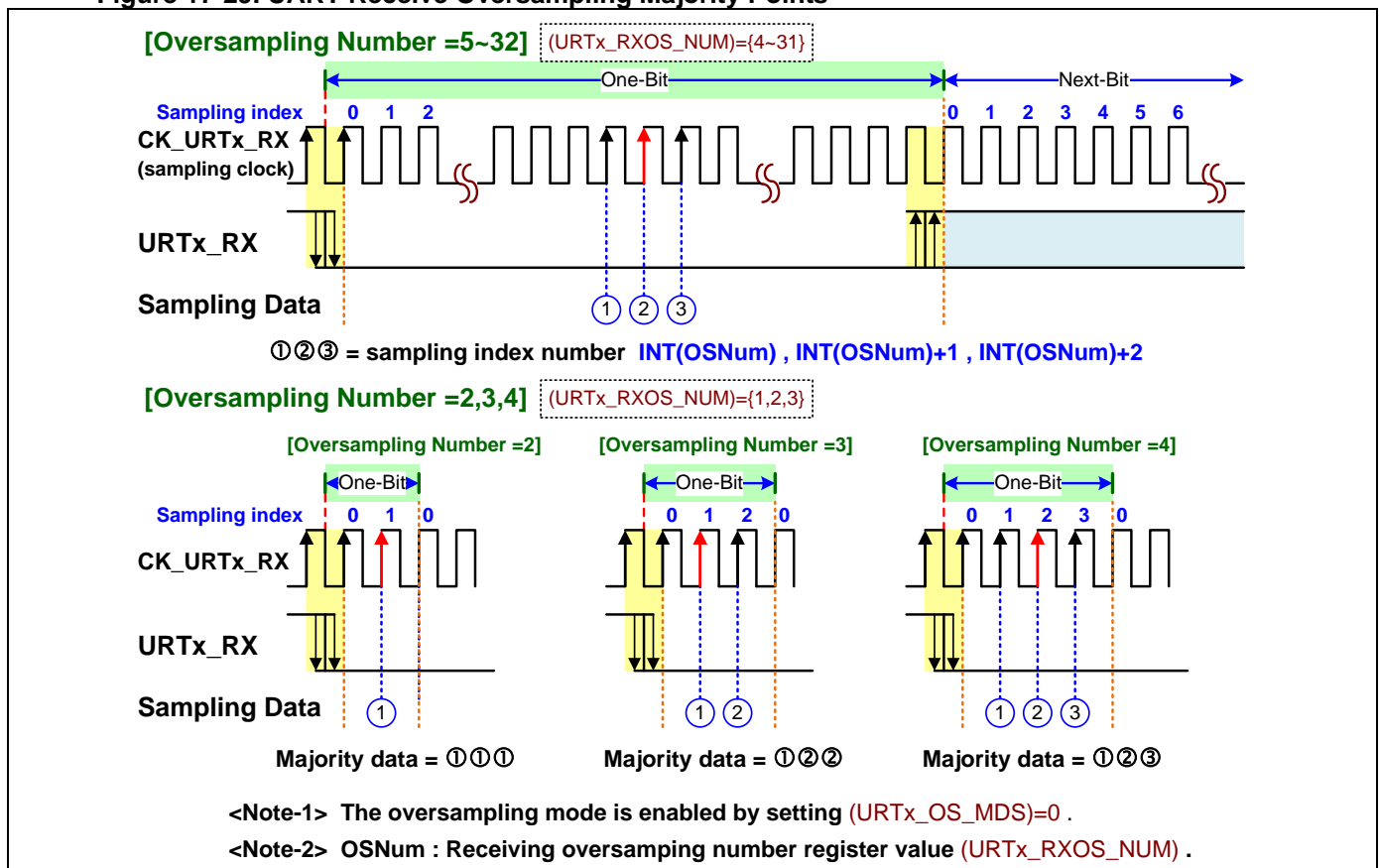
As showing in the “UART Received Data Oversampling and Noise Detection” table, the NCF flag status is active if all the three samples are not “0” or “1”.

17.16.3. UART Receive Oversampling

User can select the valid oversampling number in **URT_x_RXOS_NUM** register. When user selects three samples for oversampling majority vote, it suggests the valid value is from 8 to 32 (register value from 7 to 31). When the oversampling number is 5 ~ 32, the chip are taken from the medium three samples for data value majority vote. When the oversampling number is 2 ~ 4, the chip are taken from the fixed samples as showing in following diagram.

The following diagram is showing the UART receive oversampling majority point(s) for different oversampling number.

Figure 17-29. UART Receive Oversampling Majority Points



17.16.4. UART Receive Bit Time Error Tolerance

There is a simple method to calculate the bit time boundary of the quickest and slowest speed which can get the corrected communication.

$$\text{Valid Bit-Time : } T_Q < T_{\text{vaild}} < T_S$$

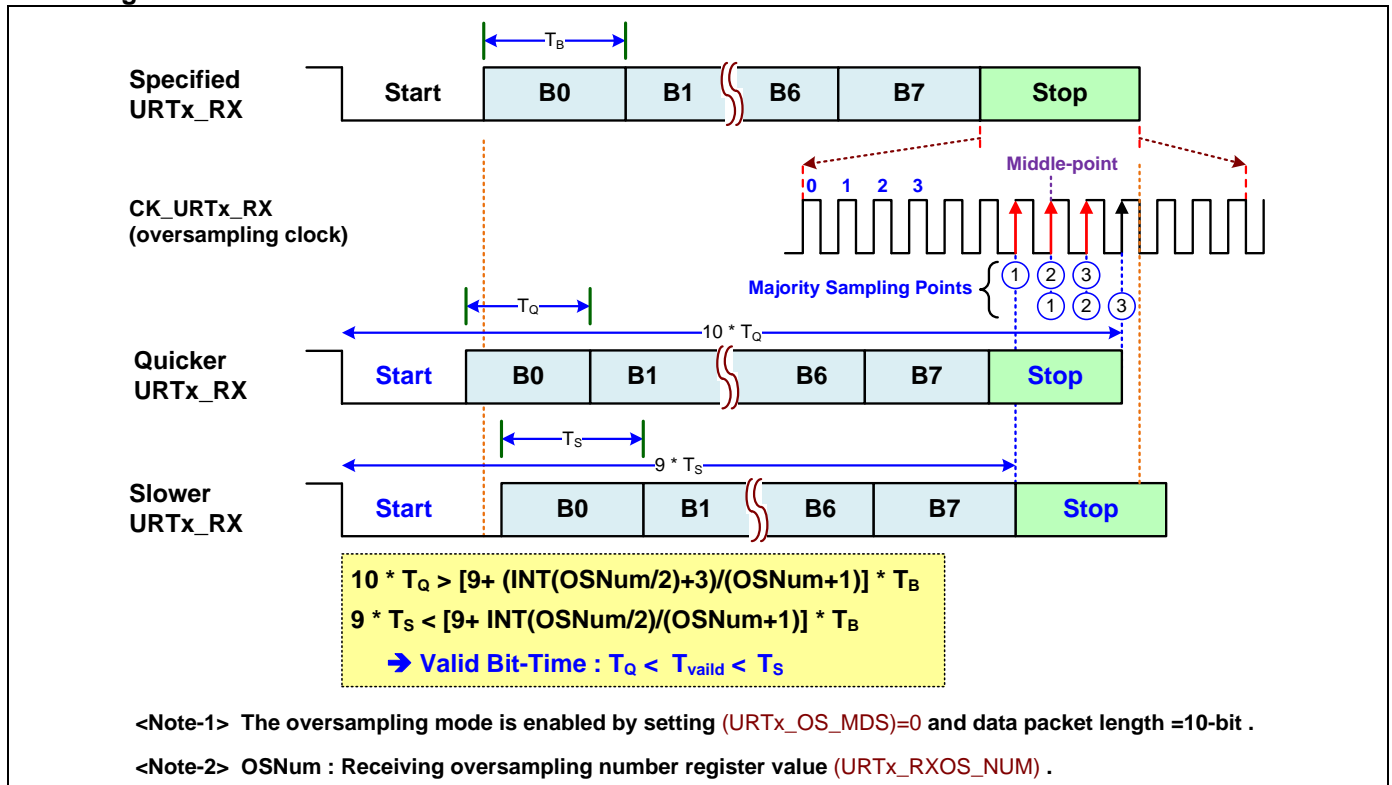
T_B is the specified bit time for data receiving. T_Q and T_S are the bit time of quickest and slowest speed for data receiving. **OSNum** is the setting oversampling number in **URT_x_RXOS_NUM** register. The following formulas are used to calculate the valid boundary time for data receiving.

$$10 * T_Q > \frac{9 + (\text{INT}(\text{OSNum}/2) + 3)}{(\text{OSNum} + 1)} * T_B$$

$$9 * T_S < \frac{9 + \text{INT}(\text{OSNum}/2)}{(\text{OSNum} + 1)} * T_B$$

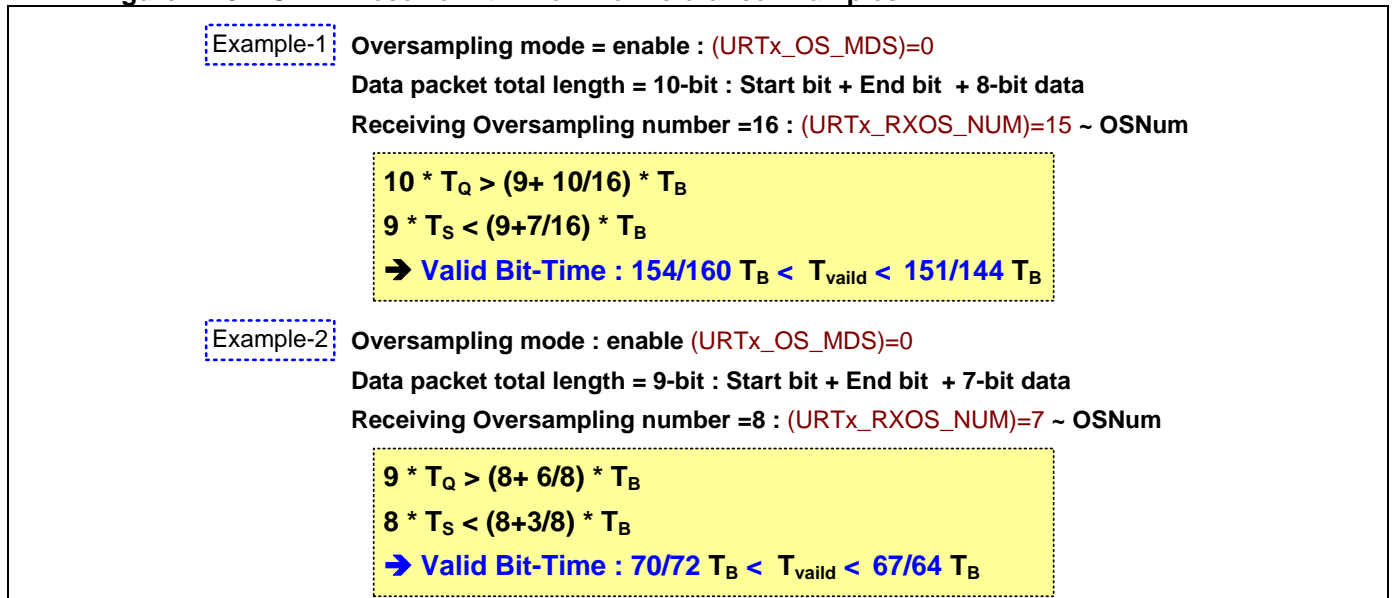
The following diagram is showing the UART receive bit time error tolerance.

Figure 17-30. UART Receive Bit Time Error Tolerance



The following diagram is showing the calculation example of UART receive bit time error tolerance for previous diagram.

Figure 17-31. UART Receive Bit Time Error Tolerance Examples



17.17. UART TMO Timeout Control

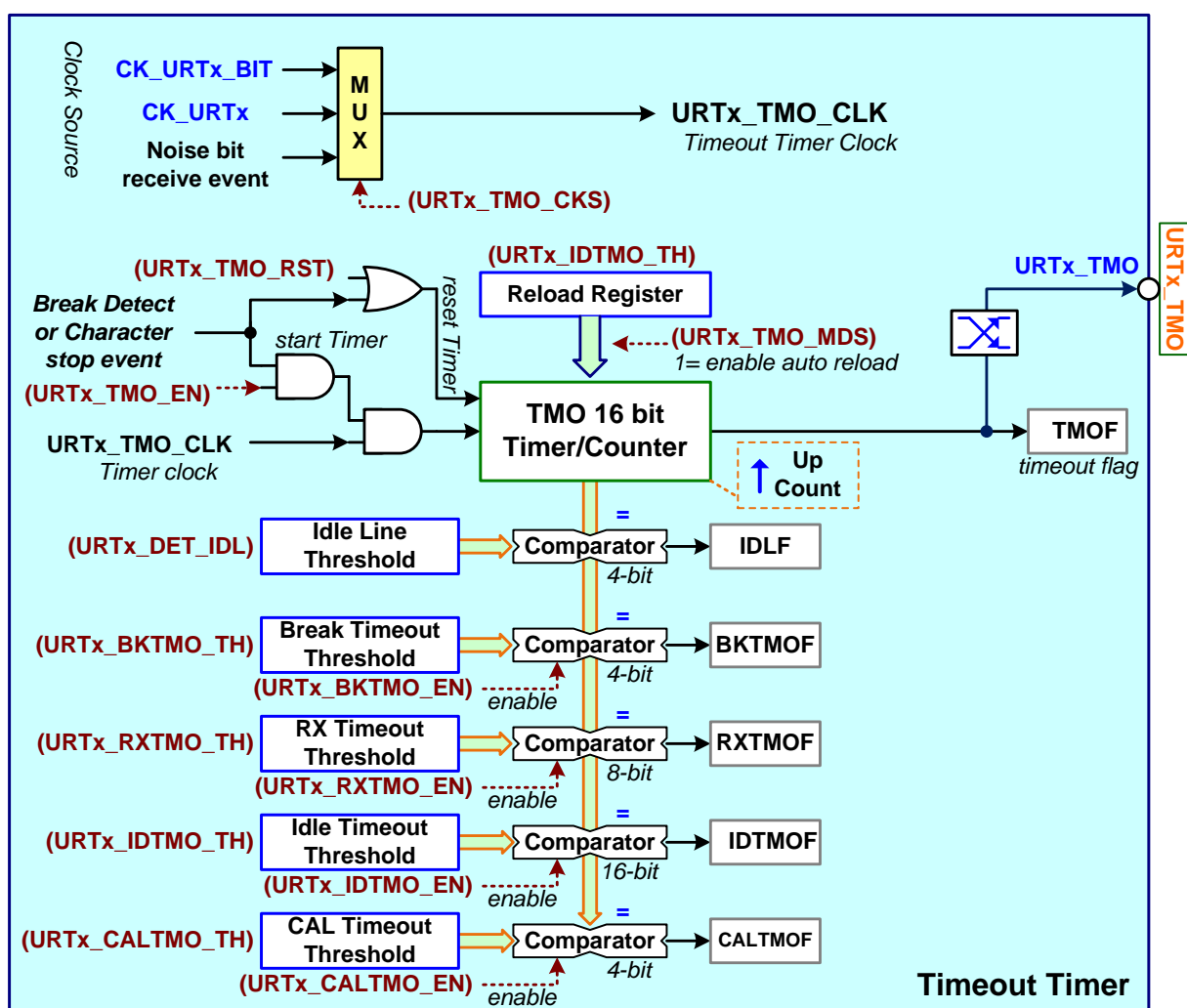
The module provides one 16-bit timeout timer (TMO) for UART access time-out control. It can configure as an UART timeout timer or a general using timer by setting **URTx_TMO_MDS** register and enable by setting **URTx_TMO_EN** register. When the TMO timer is configured as a general using timer, the **URTx_IDTMO_TH** register is used to do as the reload register of the timer.

The TMO timer can use to detect Idle Line condition, Break Timeout, RX Timeout, Idle Timeout and Baud-Rate Calibration Timeout. There are independent enable registers bit of **URTx_BKTMO_EN**, **URTx_RXTMO_EN**, **URTx_IDTMO_EN** and **URTx_CALTMO_EN** for the detection of Break Timeout, RX Timeout, Idle Timeout and Baud-Rate Calibration Timeout.

User can set the **URTx_TMO_CKS** register select the TMO timer clock source. Specially, the timer is able to calculate the communication noise condition by input the received noise bit. See the “UART Received Data Oversampling and Noise Detection” table about the noise bit definition.

A timeout timer reset enable bit of **URTx_TMO_RST** is enabled to force reset the TMO timer. Also user can get the real time counter value of TMO timer by reading **URTx_TMO_CNT** register.

Figure 17-32. UART TMO Timeout Timer Control Block



<Note> 1: x = UART module index

17.17.1. UART TMO Timeout Timer On/Off Control

When the TMO timer is configured as a general using timer, the timer enabled function is no effect by **URTx_EN** register.

The following table is showing the UART TMO Timeout Timer On/Off Control.

Table 17-14. UART TMO Timeout Timer On/Off Control

TMO Timer Function	URTx Register			Timer On/Off
	TMO_MDS	EN	TMO_EN	
URTx Timeout Timer	0	0	x	Timer Off
		1	0	Timer Off
			1	Timer On
General Using Timer	1	x	0	Timer Off
			1	Timer On

<Sign> x: Don't care

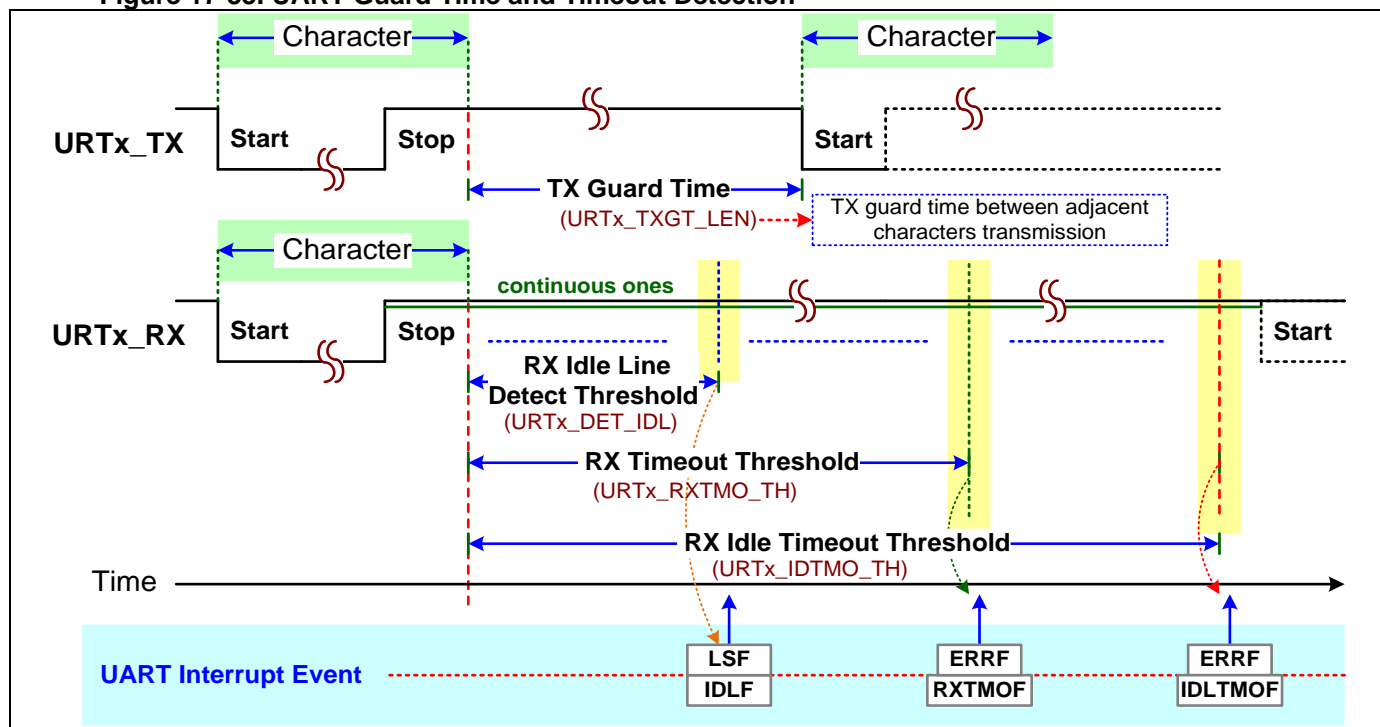
17.17.2. UART Guard Time and Timeout Detection

For data transmission, the **URTx_TXGT_LEN** can use to set the guard time between two transmitted characters.

For data receiving, user can define the bit time of Idle-Line detection threshold from last Stop bit by setting **URTx_DET_IDL** register. User can select idle line detect management mode by setting **URTx_IDL_MDS** register. When selects 'No' and the idle line has detected, chip will load. When selects 'Load' and the idle line has detected, the chip will load shadow buffer into **URTx_RDAT** data register even though it is not over the receive threshold **URTO_RX_TH** if shadow buffer is not empty.

The **URTx_IDTMO_TH** register is used to define the data transfer timeout threshold for *SmartCard* communication.

Figure 17-33. UART Guard Time and Timeout Detection

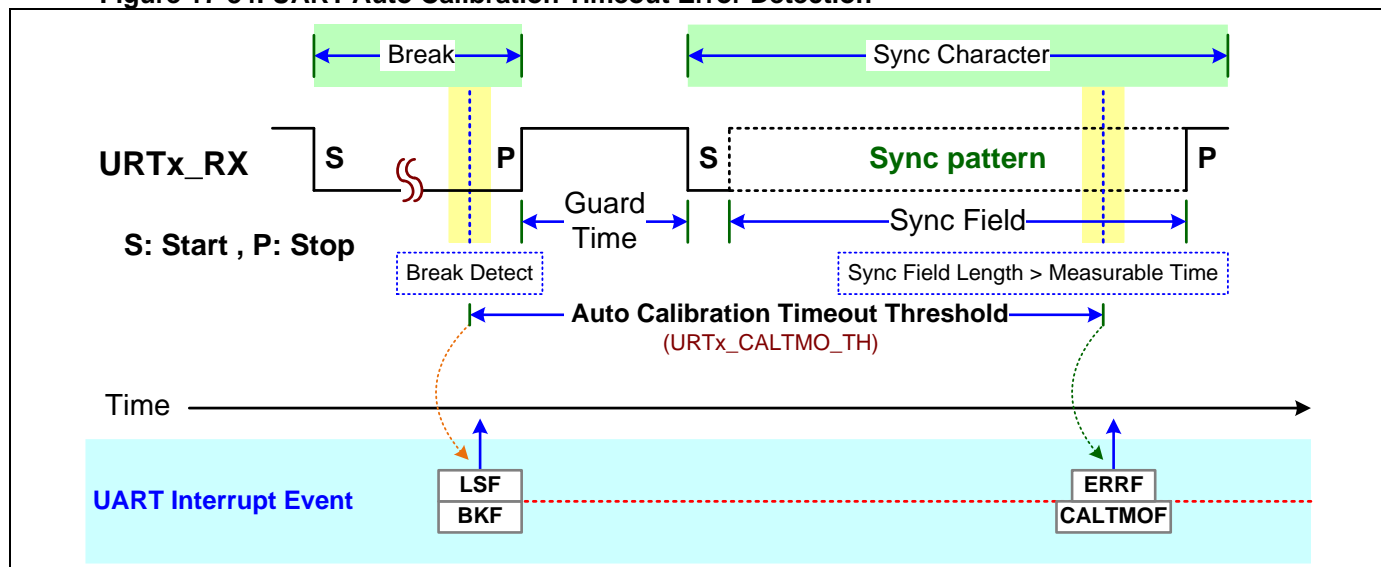


The **URT_x_RXTMO_TH** register is used to define the RX buffer data aging timeout threshold. It is useful for user to control the received data flow. When the shadow buffer receives the data number which has not arrived the RX threshold and passes a long time, chip can detect this condition and activate the RXTMOF flag (**URT_x_RXTMOF**) if user has configured the **URT_x_RXTMO_TH** register. At the time it will force the shadow buffer data to load into the RX data register (**URT_x_RDAT**) and activate the RXF flag (**URT_x_RXF**) if the RX data register is empty. User can get the data byte number from **URT_x_RNUM** register.

17.17.3. UART Auto Calibration Timeout

When the UART module processes the baud-rate calibration, the **URT_x_CALTMO_TH** register is able to define the calibration timeout threshold for unpredictable calibration error condition.

Figure 17-34. UART Auto Calibration Timeout Error Detection



The following table is showing the UART baud-rate calibration timeout condition for baud-rate timer separated mode and combined mode.

Table 17-15. UART Calibration Timeout Condition

Baud-Rate Generator Mode	Register	Calibration Timeout Condition
BR_MDS	CALTMO_TH	
0 (Separated mode)	0	RLR overflow 0xFF
	1 ~ 15	RLR overflow (CALTMO_TH * 0x10)
1 (Combined mode)	0	[PSR : RLR] overflow 0xFFFF
	1 ~ 15	[PSR : RLR] overflow (CALTMO_TH * 0x100)

<Note> PSR/RLR/CALTMO_TH ~ URT_x registers

The following table is showing the UART baud-rate calibration timeout time for baud-rate timer separated mode and combined mode.

Table 17-16. UART Calibration Timeout Time

Baud-Rate Generator Mode	Calibration Register		Calibration Timeout Time
BR_MDS	CAL_MDS	CALTMO_TH	
0 (Separated mode)	0 (Start mode)	0	$(PSR * RXOS_NUM) * 0xFF$
		1 ~ 15	$(PSR * RXOS_NUM) * (CALTMO_TH * 0x10)$
	1 (Edge mode)	0	$(PSR * RXOS_NUM) * 0xFF * 2$
		1 ~ 15	$(PSR * RXOS_NUM) * (CALTMO_TH * 0x10) * 2$
1 (Combined mode)	0 (Start mode)	0	$RXOS_NUM * 0xFFFF$
		1 ~ 15	$RXOS_NUM * (CALTMO_TH * 0x100)$
	1 (Edge mode)	0	$RXOS_NUM * 0xFFFF * 2$
		1 ~ 15	$RXOS_NUM * (CALTMO_TH * 0x100) * 2$

<Note> PSR/RXOS_NUM/CALTMO_TH ~ URTx registers

17.17.4. UART Break Condition Timeout

The **URT_x BKTMO_TH** register is used to define the timeout threshold of Break condition for the error of too long Break condition. See the descriptions of “UART Break and Parity/Frame Error Detection” section for more information.

17.17.5. UART TMO Timeout Signal Output

The TMO timeout timer can output the timeout signal to do as a clock signal to external port of **URT_x TMO** or internal other module. User can set the initial state of timeout signal by setting **URT_x TMO_STA** register. Specially, the initial state register can be written only when the register of **URT_x TMO_LCK** is written “1” simultaneously.

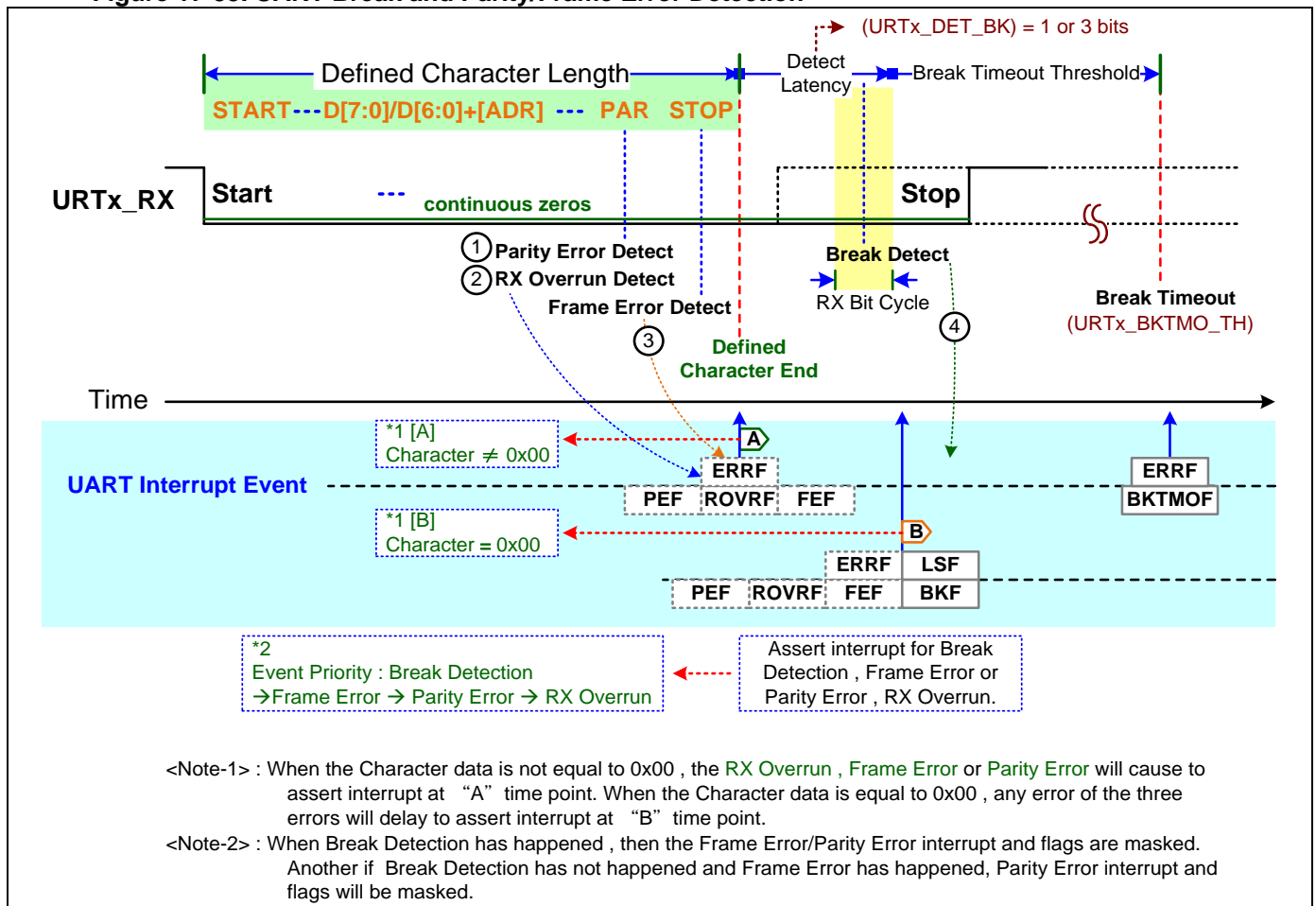
17.18. UART Error Management

The UART module can detect and manage some general communication errors. The timeout timer (TMO) is used to detect the time-out error for Break condition, baud-rate calibration, shadow buffer aging and idle condition. See the “UART TMO Timeout Control” section for more descriptions.

17.18.1. UART Break and Parity/Frame Error Detection

When UART module detects the parity error or data receiving overrun at the PAR bit and frame error at Stop bit, the related event flags of PEF, ROVRF and FEF will be delayed to activate after Stop bit. If the received character is not equal to 0x00, these flags will be activated at the end of Stop bit. If the received character is equal to 0x00, these flags will be activated at the detected time of Break condition. In the condition, the interrupts and flags of frame error and parity error are masked if Break condition is detected. Another if Break condition has not happened and frame error is detected, parity error interrupt and flag will be masked.

Figure 17-35. UART Break and Parity/Frame Error Detection



17.18.2. UART TX Error Detect and Resend Control

The UART module is able to detect the transmission TX error of Stop bit pull-low for SmartCard communication or transmitted data read-back checking for Lin communication by setting **URTx_TXE_MDS** register. When detects the TX error, the TXEF flag (**URTx_TXEF**) will be activated.

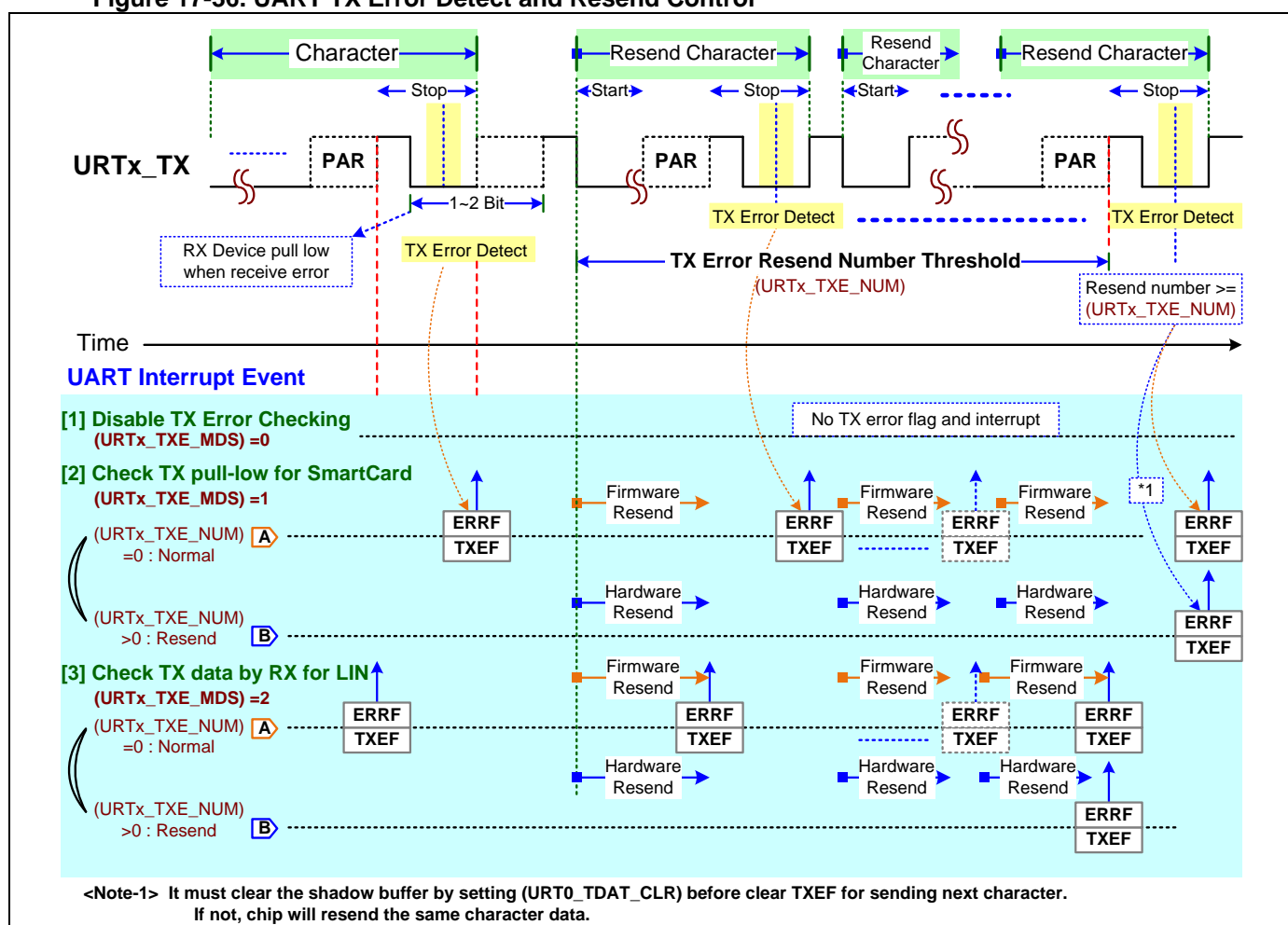
For **SmartCard** communication, user can set the register of **URTx_TXE_MDS** to “CHK_Low” and the chip can detect the **URTx_TX** signal being pulled low at the Stop bit by external receiver to indicate communication error.

For **LIN** communication, user can set the register of **URTx_TXE_MDS** to “CHK_TX” and the chip can read back the transmitted data from internal receiving path and check the data whether is wrong by application hardware connection or others.

User can enable the Auto Resend function by setting the **URTx_TXE_MDS** register to “CHK_Low” or “CHK_TX” and set the resend number of last data when detects the TX error by setting **URTx_TXE_NUM** register. When the Auto Resend function is enabled and the value of **URTx_TXE_NUM** register is >0, the TXEF flag is masked until the error is still happened at last time resent data.

*Notify: The **URTx_TX** pin needs to set open-drain mode when enables the TX error detect function.*

Figure 17-36. UART TX Error Detect and Resend Control



17.18.3. UART RX Parity Error Detect and Retry Control (for SmartCard)

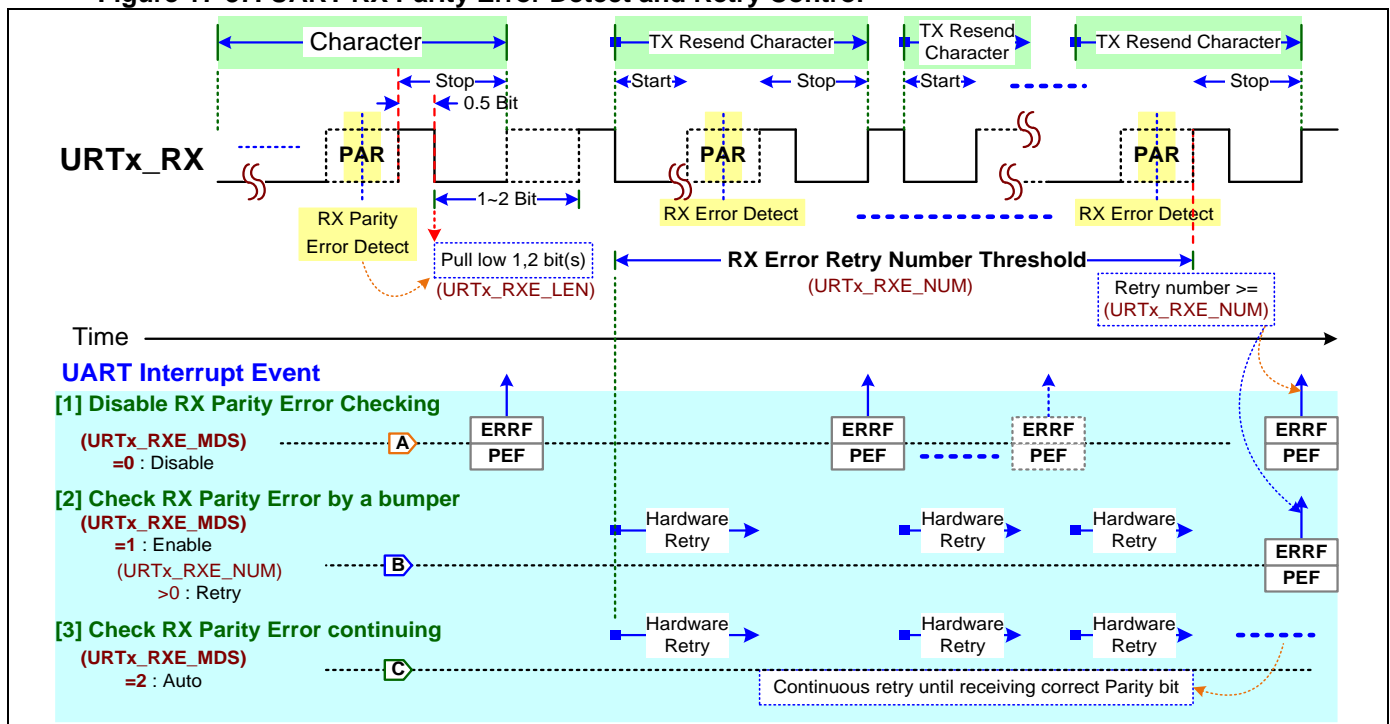
The UART module can enable to detect the receiving RX parity error at the PAR bit and select the detected mode by setting **URTx_RXE_MDS** register. When detects the RX parity error, the PEF flag (**URTx_PEF**) will be activated. The parity error detection function is only supported Stop bit length = 1 bit.

User can enable the Auto Retry function by setting the **URTx_RXE_MDS** register to “Enable” or “Auto” and set the retried number of last data when detects the RX parity error by setting **URTx_RXE_NUM** register. When the Auto Retry function is enabled, the chip will pull low the input **URTx_RX** signal during Stop bit cycle when detects the RX parity error. If **URTx_RXE_MDS** register is set to “Auto” mode, the chip will do the Retry process continuously and the PEF flag is always masked. At the time, chip will not assert the interrupt and retry to receive new data. If **URTx_RXE_MDS** register is set to “Enable” mode and the value of **URTx_RXE_NUM** register is >0, the chip will do the Retry process and the TXEF flag is masked until the error is still happened at last time retried data.

*Notify: The **URTx_RX** pin needs to set open-drain mode when enables the TX error detect function.*

User can set the bit time length to pull low on RX line when RX parity error has detected by setting **URTx_RXE_LEN** register.

Figure 17-37. UART RX Parity Error Detect and Retry Control



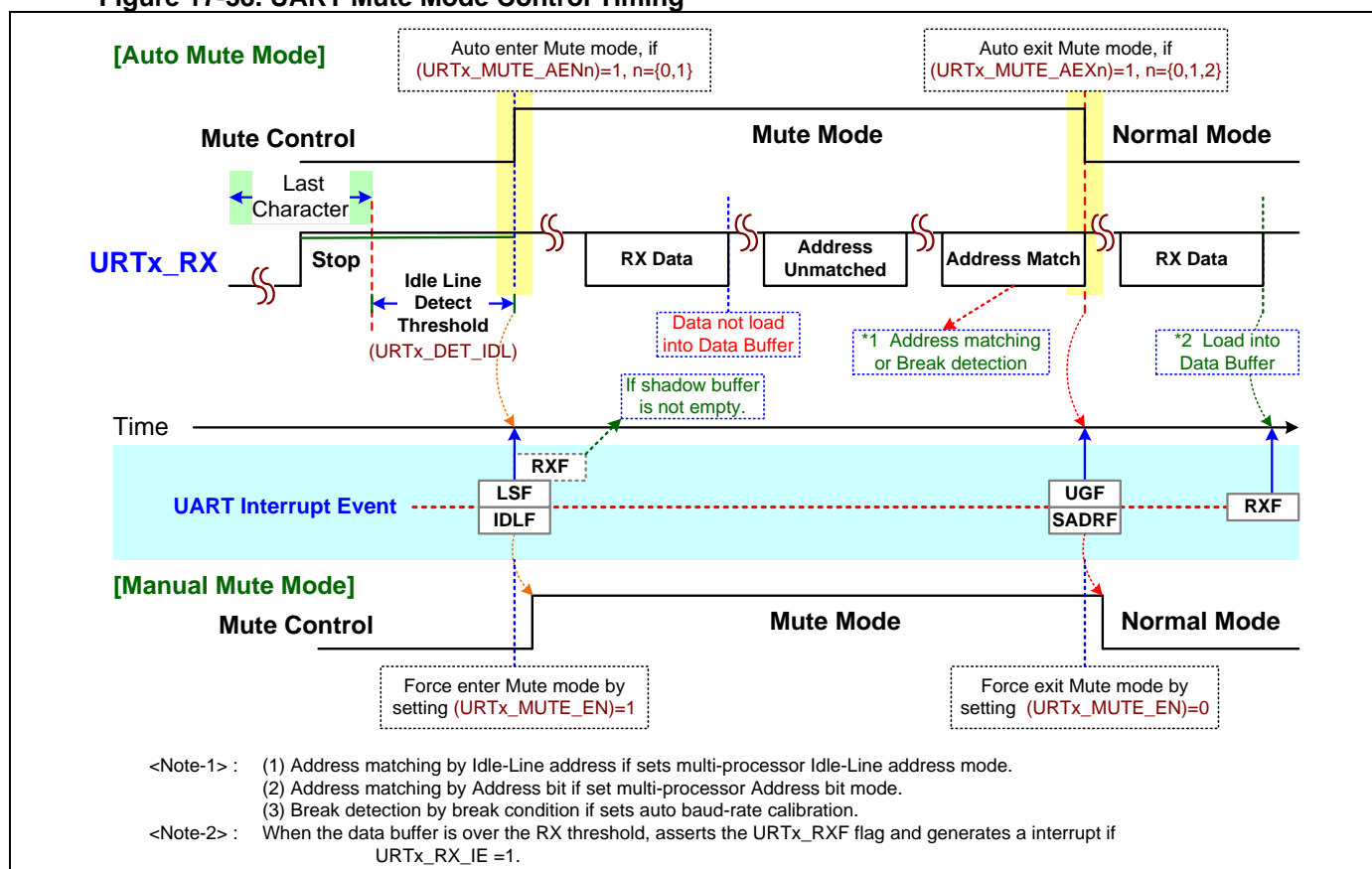
17.19. UART Mute Mode Control

The UART module is support a mute mode to disable receiving data character but the shift buffer is still operation for status detection. When the UART is entering mute mode, the RX shadow buffer is never load into data from shift buffer. The mute mode is useful for multi-processor communication.

The mute mode can be directly forced to enter or exit by the **URT_x_MUTE_EN** register setting. So user can manual to control the mute mode entering and exiting.

User can configure the trigger events of multi-processor slave address unmatched condition and idle line detection to automatically enter into mute mode or exit from the mute mode by setting **URT_x_MUTE_AEN_n** register. Also user can configure the trigger events of multi-processor slave address matched condition, Break condition detection and idle line detection to automatically exit from the mute mode by setting **URT_x_MUTE_AEX_n** register. (n=register sequence number)

Figure 17-38. UART Mute Mode Control Timing



The following table is showing the UART mute mode control for multi-processor address matched or unmatched conditions. The **URT_x_MUTE_EN** register will be set by hardware for these conditions.

Table 17-17. UART Multi-Processor Address Matched vs Mute Mode Control

Address Matched Condition	Register Setting		Affected Register
	MUTE_AEN0	MUTE_AEX0	MUTE_EN
Address Matched	0	x	0
	1	0	1
		1	0
Address Unmatched	0	x	0
	1	x	1

<Note> The RX shift register data will not be loaded to RX shadow buffer if MUTE_EN = 1.

17.20. UART Synchronous Mode

17.20.1. UART Synchronous Mode Configure

The synchronous communication supports one line SYNC and two lines SPI master/slave mode. The UART module can be configured to support synchronous mode timing by setting the **URTx_MDS** register to "SYNC". Refer the section of "[UART Operation Mode Setting](#)" for the synchronous mode setting.

As same as the data transaction of UART asynchronous mode, user needs to enable the data transmission and receiving functions by setting **URTx_TX_EN** and **URTx_RX_EN** registers. Also user can write and read the transmission and received data from the TX and RX data register (**URTx_TDAT** and **URTx_RDAT**). Refer the sections of "[UART Transmit](#)" and "[UART Receive](#)" for more information.

For synchronous two lines mode, user can select SPI master or slave mode by setting **URTx_SYNC_MDS** register. Specially, the operation of data receiving is as same as UART asynchronous mode for SPI slave mode but SPI master mode is not. For SPI master mode, user needs to write the request receiving data number to the TX data register (**URTx_TDAT**), then user can get the same data number from the RX data register (**URTx_RDAT**).

17.20.2. UART Synchronous Mode Timing

For the synchronous mode, user can set the clock mode by setting **URTx_CPOL** and **URTx_CPHA** registers for **URTx_CLK** signal. Also use can directly control the output signal of **URTx_NSS** by setting **URTx_SWEN** and **URTx_SWO** registers for SPI master communication. User can set the synchronous mode clock source and frequency by setting **URTx_TX_CKS** and **URTx_TXOS_NUM** registers. Refer the section of "[UART Clock Control](#)" for more information about the SPI clock setting.

The following table is showing the clock mode setting for UART synchronous mode configuration.

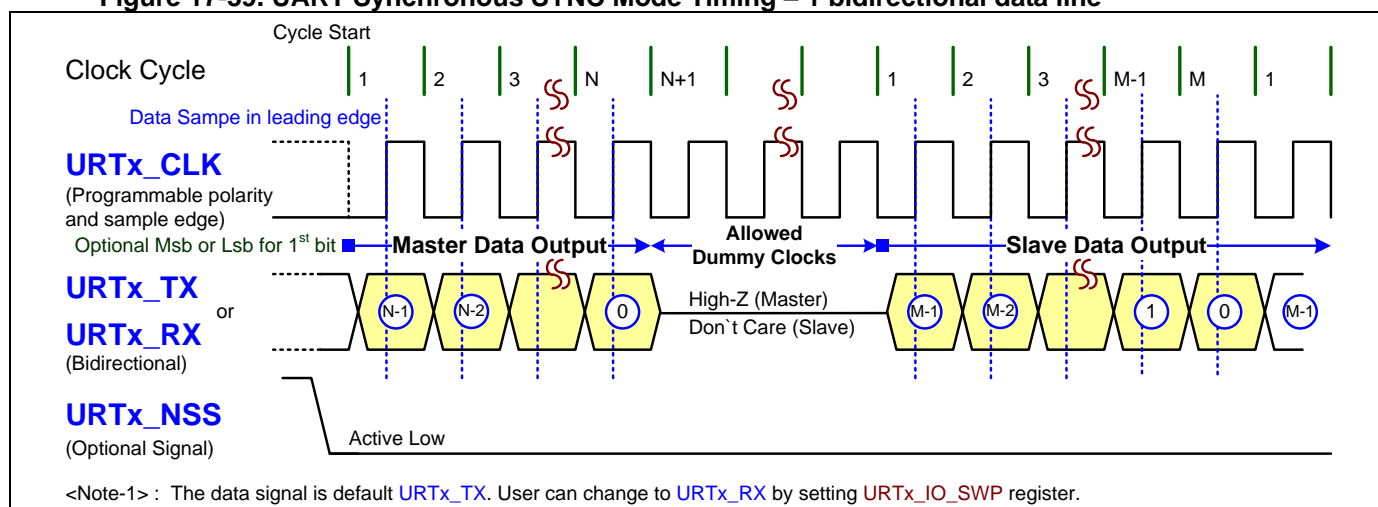
Table 17-18. UART Synchronous Clock Mode Table

Clock Mode	URTx Register		Function Description
	CPOL	CPHA	
0	0	0	UART clock low level in idle state and data sampling on leading edge.
1	0	1	UART clock low level in idle state and data sampling on trailing edge.
2	1	0	UART clock high level in idle state and data sampling on leading edge.
3	1	1	UART clock high level in idle state and data sampling on trailing edge.

The synchronous one line SYNC mode and two lines SPI master/slave mode are different by setting **URTx_DAT_LINE** to 1 or 2. Refer the previous figure of "UART Synchronous Mode Timing" for the two lines SPI master/slave mode.

The following diagram is showing the one line synchronous SYNC mode timing.

Figure 17-39. UART Synchronous SYNC Mode Timing – 1 bidirectional data line



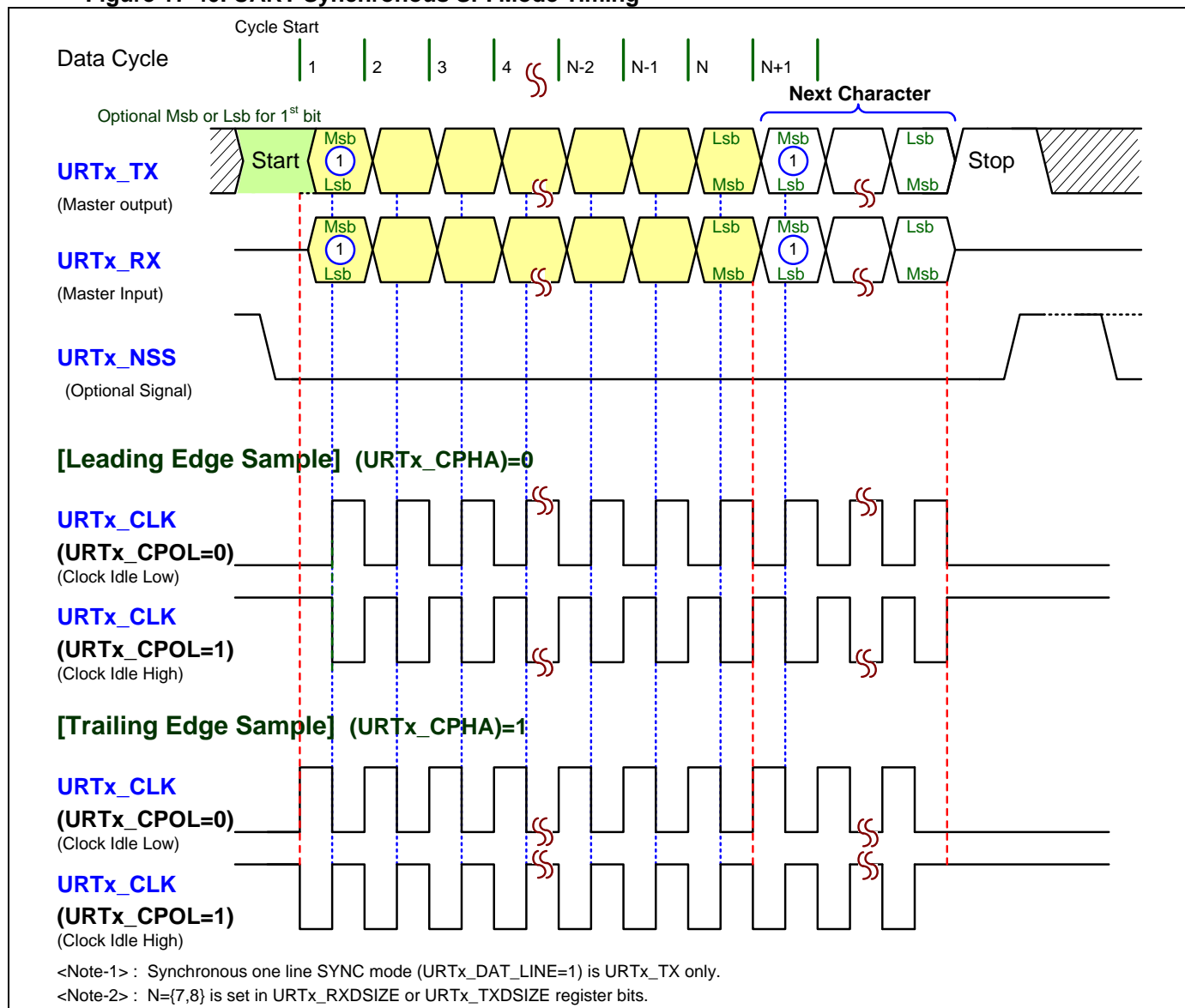
For the one line SYNC mode, the clock output is from **URTx_CLK** and the data signal is default **URTx_TX**. User can change the data signal to **URTx_RX** by setting **URTx_IO_SWP** register. For data transaction, user needs to enable the **URTx_TX_EN** register and change the data transferred direction to receiving or transmission by setting **URTx_RX_EN** register. For this mode, the TCF flag (**URTx_TCF**) can denote that the previous data transmission is complete.

For the synchronous SPI master and slave mode, the synchronous mode Start bit is high level.

During the SPI bus idle state, user can configure the **URTx_TX** output state to 'tristate' or 'driving' by setting **URTx_DOUT_MDS** register. When the register is selected 'driving', there are three states of 'Output 0', 'Output 1' and 'Last data bit' those user can select in **URTx_DOUT_IDL** register.

The following diagram is showing the two lines synchronous SPI master and slave mode timing.

Figure 17-40. UART Synchronous SPI Mode Timing



17.20.3. UART Synchronous Mode NSS Control

The UART synchronous mode is able to support the SPI communication with **NSS** signal for SPI master or slave mode.

For master mode, user can set the NSS output control mode of **URTx_NSS** signal for hardware control or software directly control by setting **URTx_NSS_SWEN** register. The **URTx_NSS_INV** register is used to inverse the **URTx_NSS** output signal.

● Synchronous Mode Software NSS Control

When enables software NSS control by enable setting in **URTx_NSS_SWEN** register, user can directly control **URTx_NSS** signal output level in **URTx_NSS_SWO** register for support the SPI communication with **NSS** signal.

● Synchronous Mode Hardware NSS Control

When enables hardware NSS control by disable setting in **URTx_NSS_SWEN** register, the chip will automatically generate **URTx_NSS** signal as the following synchronous mode NSS diagram and table.

For slave mode, user can enable the NSS input from **URTx_NSS** input by setting **URTx_NSSI_EN**. The **URTx_NSSI_INV** register is used to inverse the **URTx_NSS** input signal. Also user can directly get the input of **URTx_NSS** state by reading **URTx_NSS_SWI** register.

The **URTx_NSS** signal is keeping the last state during the idle state.

Figure 17-41. UART Synchronous Mode Hardware NSS Timing

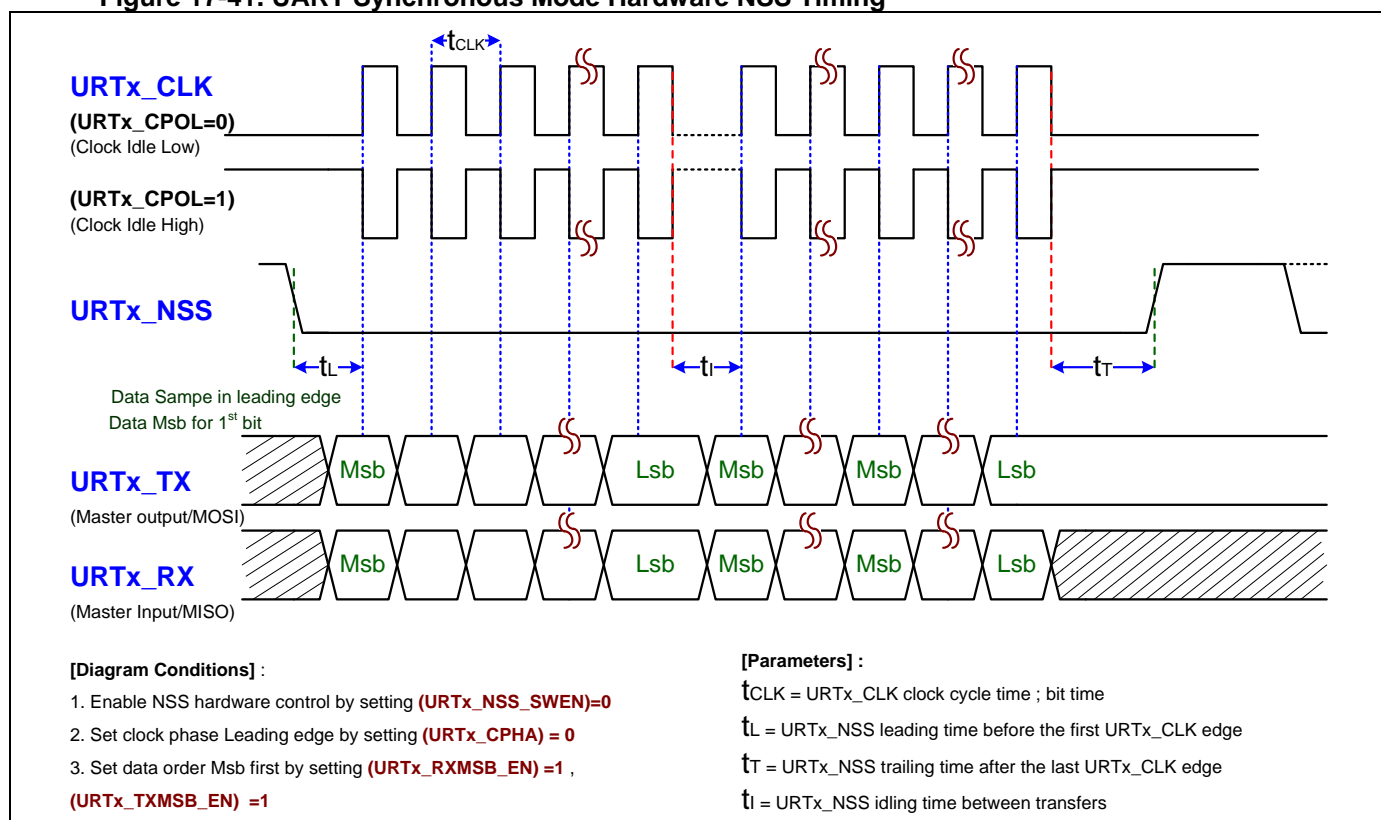


Table 17-19. UART Synchronous Mode NSS Timing Table

Clock Mode	NSS Timing			Unit
	t_L	t_T	t_i	
0 (CPOL=0, CPHA=0)	1.5	0	0.5	t_{CLK}
1 (CPOL=0, CPHA=1)	1	0.5	0.5	t_{CLK}
2 (CPOL=1, CPHA=0)	1.5	0	0.5	t_{CLK}
3 (CPOL=1, CPHA=1)	1	0.5	0.5	t_{CLK}

<Sign> t_{CLK} : UART bit time

t_L : URTx_NSS leading time before the first URTx_CLK edge (Bit time)

t_T : URTx_NSS trailing time after the last URTx_CLK edge (Bit time)

t_i : URTx_NSS idling time between transfers (Bit time), $t_i = 0$ if **URTx_NSS_PEN=0**

<Note> *1: 0.5 bit-time if Stop bit=0.5bit, 1 bit-time if Stop bit=1/1.5/2bit

*2: 1 bit-time if Stop bit=0.5bit, 1.5 bit-time if Stop bit=1/1.5/2bit

*3: 0.5 bit-time if Stop bit=0.5/1bit, 1 bit-time if Stop bit=1.5bit, 1.5 bit-time if Stop bit=2bit

17.21. UART SmartCard Clock Output

The Baud-Rate generator is able to be configuration to output the **SmartCard** clock from **CK_URTx_PSC**. User can configure the Baud-Rate generator by setting **URTx_PSR**, **URTx_RLR** and **URTx_TXOS_NUM** registers for **SmartCard** clock output. In **SmartCard** application, the data transferred line is bidirectional signal. Normally both transmitted and received clock rates are the same, so the setting of **URTx_TXOS_NUM** and **URTx_RXOS_NUM** should be identical. For URT4/5/6/7 modules, the **URTx_RXOS_NUM** register is replaced by TX/RX combined common register of **URTx_TXOS_NUM**.

17.21.1. SmartCard Clock Frequency Calculation

According to the **ISO/IEC 7816-3** standard, the descriptions of related abbreviated terms are listed in following.

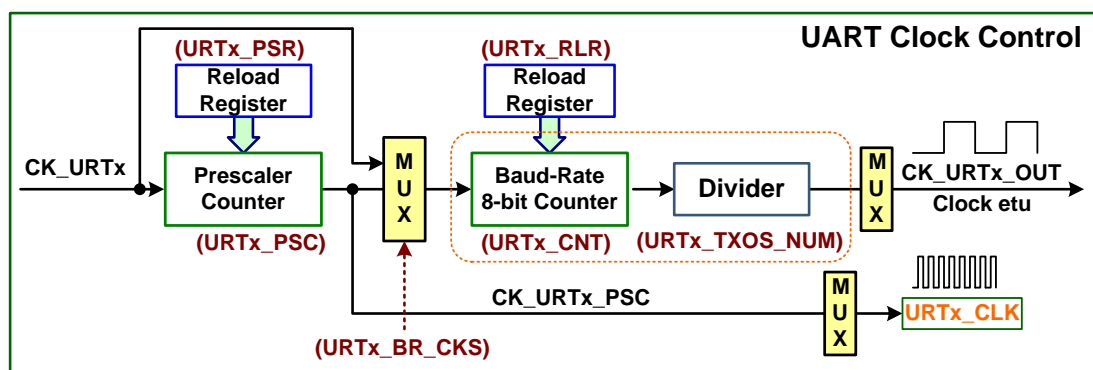
- etu** : elementary time unit (bit time)
- F** : clock rate conversion integer
- D** : baud rate adjustment integer
- f** : frequency value of the clock signal provided to the card by the interface device

In the **SmartCard** communication, the module can receive the items of “F”, “D” and “f” for external SmartCard. User can calculate these items and set the valid value to the registers of **URTx_PSR**, **URTx_RLR** and **URTx_TXOS_NUM**.

User can select the 8-bit Baud-Rate counter input clock coming from **CK_URTx** signal or the Prescaler clock output **CK_URTx_PSC** by setting **URTx_BR_CKS** register.

The following diagram is showing the calculation of UART SmartCard clock output.

Figure 17-42. UART SmartCard Clock Output



$$etu = F / (D * f)$$

$$F = (URTx_RLR + 1) * (URTx_TXOS_NUM + 1)$$

$$f = f(CK_URTx_PSC) = f(CK_URTx) / (URTx_PSR + 1)$$

$$f(CK_URTx) = (URTx_PSR + 1) * f(CK_URTx_PSC)$$

$$f(etu) = (D * f) / F = f(CK_URTx_BIT)$$

$$\textcircled{1} (URTx_BR_CKS) = 0 (CK_URTx_PSC)$$

D = fixed to 1

$$f(etu) = f(CK_URTx) / ((URTx_PSR + 1) * (URTx_RLR + 1) * (URTx_TXOS_NUM + 1))$$

$$= (f(CK_URTx_PSC)) / ((URTx_RLR + 1) * (URTx_TXOS_NUM + 1))$$

$$\textcircled{2} (URTx_BR_CKS) = 1 (CK_URTx)$$

D = $(URTx_PSR + 1) = 1 \sim 64$

$$f(etu) = f(CK_URTx) / ((URTx_RLR + 1) * (URTx_TXOS_NUM + 1))$$

$$= ((URTx_PSR + 1) * f(CK_URTx_PSC)) / ((URTx_RLR + 1) * (URTx_TXOS_NUM + 1))$$

17.21.2. UART SmartCard Clock Output Setting Examples

The following table is showing the examples of UART SmartCard clock output setting with CK_URTx = 48.0 MHz.

Table 17-20. UART SmartCard Clock Output Setting Examples

SmartCard etu		CK_URTx_PSC	URTx Register				SmartCard Parameters		
etu (*1) (us)	f _(etu) (KHz)	f (MHz)	PSR (*2)	RLR	TXOS_NUM RXOS_NUM	BR_CKS (*3)	F	f _(max) (MHz)	D
93.00	10.75	4.000	11	11	30	0	372	4	1
77.50	12.90	4.800	9	11	30	0	372	5	1
93.00	10.75	6.000	7	17	30	0	558	6	1
93.00	10.75	8.000	5	92	7	0	744	8	1
93.00	10.75	12.000	3	35	30	0	1116	12	1
93.00	10.75	16.000	2	185	7	0	1488	16	1
116.25	8.60	16.000	2	59	30	0	1860	20	1
106.67	9.38	4.800	9	63	7	0	512	5	1
112.00	8.93	6.857	6	95	7	0	768	7.5	1
106.67	9.38	9.600	4	127	7	0	1024	10	1
128.00	7.81	12.000	3	191	7	0	1536	15	1
128.00	7.81	16.000	2	127	15	0	2048	20	1
46.50	21.51	4.000	11	11	30	1	372	4	2
46.50	21.51	6.000	7	17	30	1	558	6	2
58.13	17.20	16.000	2	59	30	1	1860	20	2
53.33	18.75	4.800	9	63	7	1	512	5	2
56.00	17.86	6.857	6	95	7	1	768	7.5	2
64.00	15.63	16.000	2	127	15	1	2048	20	2
23.25	43.01	4.000	11	11	30	1	372	4	4
23.25	43.01	8.000	5	92	7	1	744	8	4
29.06	34.41	16.000	2	59	30	1	1860	20	4
32.00	31.25	12.000	3	191	7	1	1536	15	4
11.63	86.02	12.000	3	35	30	1	1116	12	8
14.53	68.82	16.000	2	59	30	1	1860	20	8
13.33	75.00	9.600	4	127	7	1	1024	10	8
16.00	62.50	12.000	3	191	7	1	1536	15	8

<Note> *1 : $etu = F / (D * f)$, $F = (RLR+1) * (TXOS_NUM+1)$

*2 : $CK_URTx_SC = CK_URTx / (PSR+1)$ ~ $CK_URTx = 48.0$ MHz

17.22. UART IrDA Control

The UART module is built an IrDA encoder and an IrDA decoder in the data interface for IrDA communication and can be enabled by setting **URTx_IR_EN** register. When enables the IrDA encoder and decoder, the over-sampling mode must set “Three” by setting **URTx_RXOS_MDS** register.

17.22.1. UART IrDA Timing

For IrDA data transmission, user can set the IrDA data pulse width by setting **URTx_IR_PW** register.

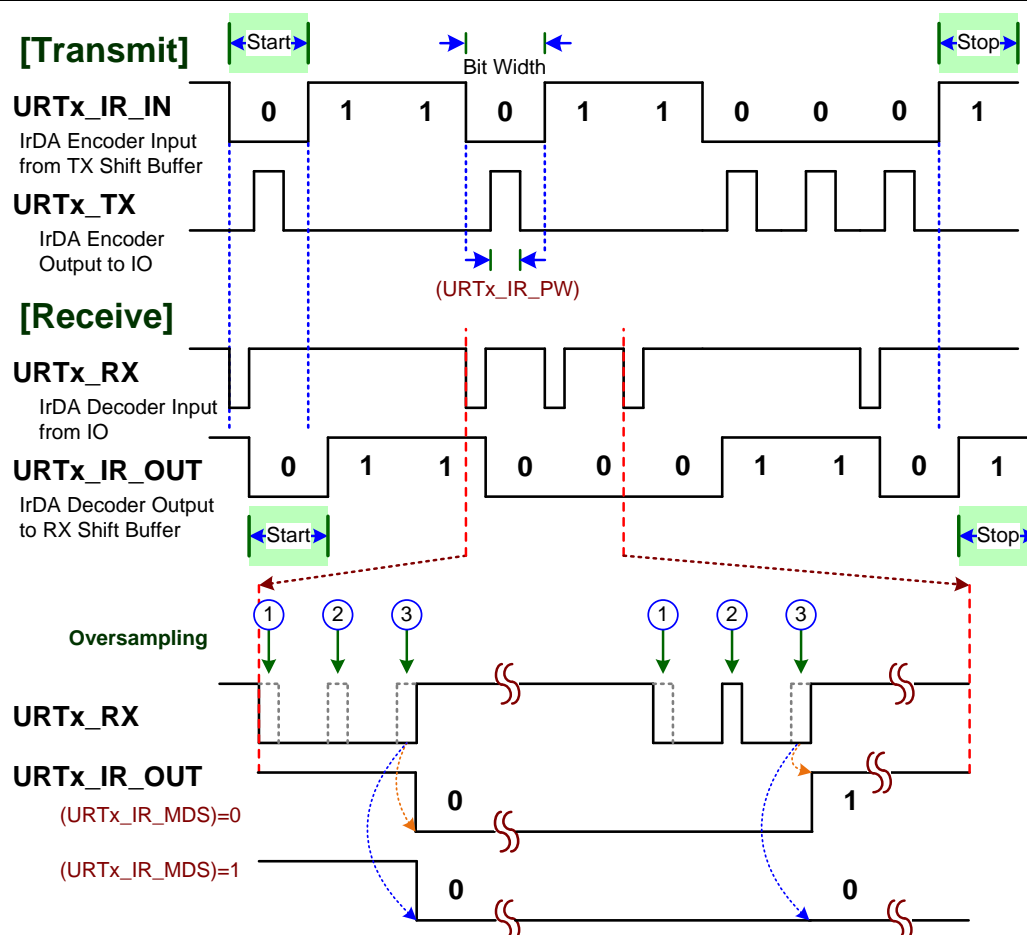
$$\text{IrDA pulse width} = (\text{URTx_IR_PW} + 1) * T_{\text{CK_URTx_TX}}$$

The data pulse width needs equal to the range of 3/16 to 4/16 of a bit time for valid IrDA communication. For example, the value of **URTx_IR_PW** register can be 2 or 3 if the value of **URTx_TXOS_NUM** register is 15 (oversampling number=16).

For IrDA data receiving, user can select the received bit sampling mode in **URTx_IR_MDS** register. When selects “Normal”, the IrDA sampling sequence value need equal 000 then output bit value 0 and others output 1. When selects “Wide”, the IrDA sampling sequence value need equal 000,001,010,100 then output bit value 0 and others output 1.

The IrDA data receiving can support “Normal” and “Wide” modes. The following figure is showing the UART IrDA timing.

Figure 17-43. UART IrDA Timing



- <Note> 1. Oversampling mode selects “Three” (**URTx_OS_MDS**)=0 :
 (**URTx_IR_MDS**)=0 : Oversampling sample ①②③ need all =0 ,
 then **URTx_IR_OUT** output 0 and output 1 for other conditions.
 (**URTx_IR_MDS**)=1 : Oversampling sample ①②③ need any two=0 ,
 then **URTx_IR_OUT** output 0 and output 1 for other conditions.
2. Oversampling mode selects “One” (**URTx_OS_MDS**)=1 :
 If the **URTx_RX** input sample is 0, then **URTx_IR_OUT** output 0
 and output 1 for input sample being 1.

17.22.2. UART IrDA Data Sampling

The following table is showing the received bit value by “Normal” and “Wide” modes for IrDA communication.

Table 17-21. UART IrDA Received Data Oversampling and Sampling Mode

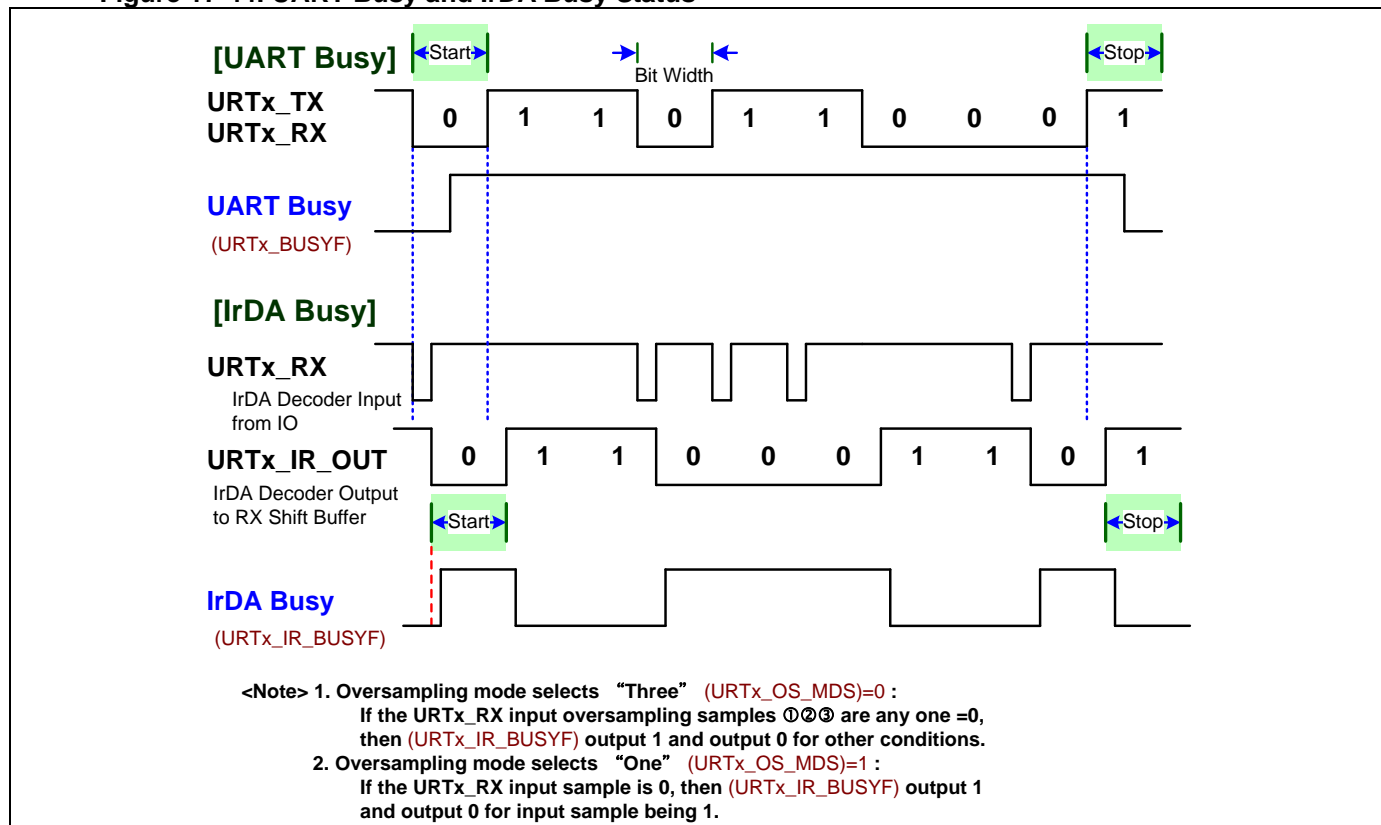
Sampling Sequence Value	Register	Received bit	Comment
	IR_MDS		
000	0 (Normal)	0	all three samples =0
001		1	
010		1	
011		1	
100		1	
101		1	
110		1	
111		1	
000	1 (Wide)	0	any two samples =0
001		0	any two samples =0
010		0	any two samples =0
011		1	any two samples =1
100		0	any two samples =0
101		1	any two samples =1
110		1	any two samples =1
111		1	any two samples =1

17.22.3. UART IrDA Busy Flag

The UART module provides two flags of UART busy flag (**URT_x_BUSYF**) and IrDA busy flag (**URT_x_IR_BUSYF**) to indicate the data communication status for software monitor.

The following diagram is showing the relation (1) between the status of UART busy flag and UART TX/RX signals (2) between the status of IrDA busy flag and UART RX signal.

Figure 17-44. UART Busy and IrDA Busy Status

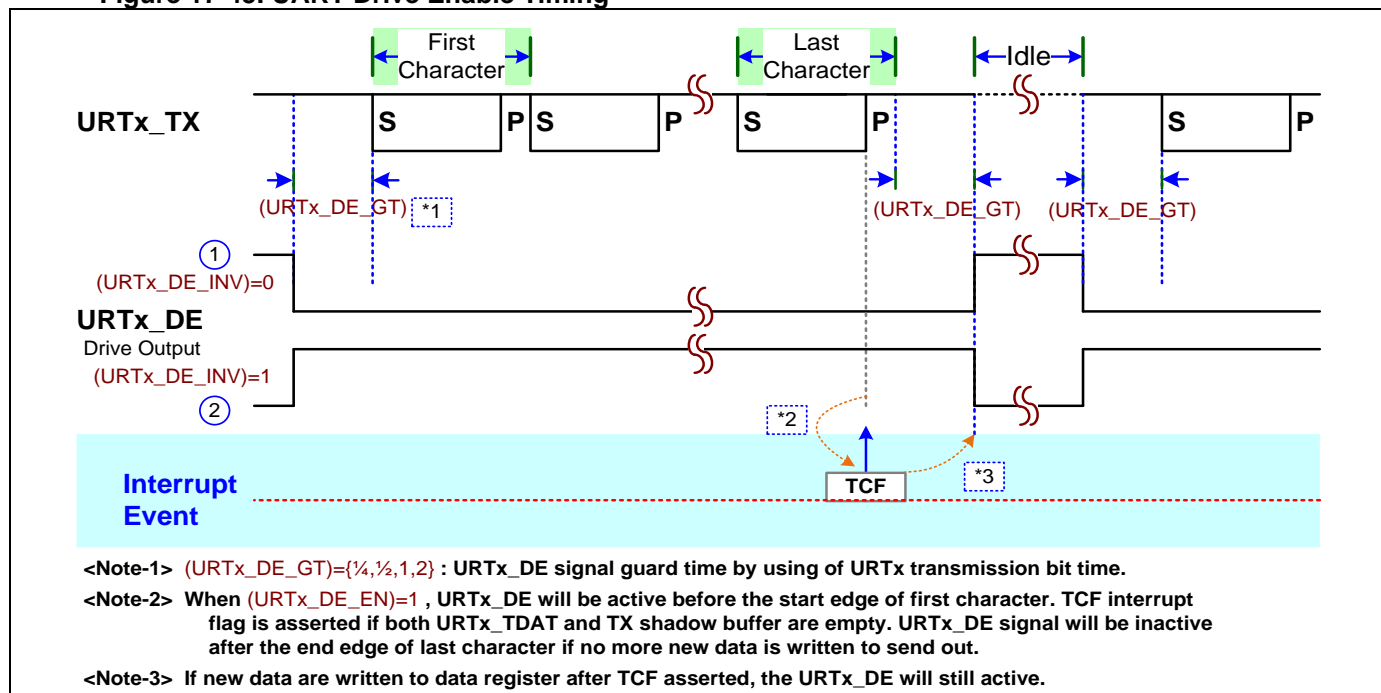


17.23. UART DE Control

The UART module provides one data enable signal of **URTx_DE** and enables in **URTx_DE_EN** register. This signal is used to indicate the data transmitted period and can output to external signal drive device. The external signal drive device can receive the UART TX signal and drive it with a signal enhanced buffer to the target of UART receiver for long distance communication.

User can set the guard time before Start bit and after Stop bit by setting the **URTx_DE_GT** register. The **URTx_DE_INV** register is used to inverse the output signal of **URTx_DE**.

Figure 17-45. UART Drive Enable Timing



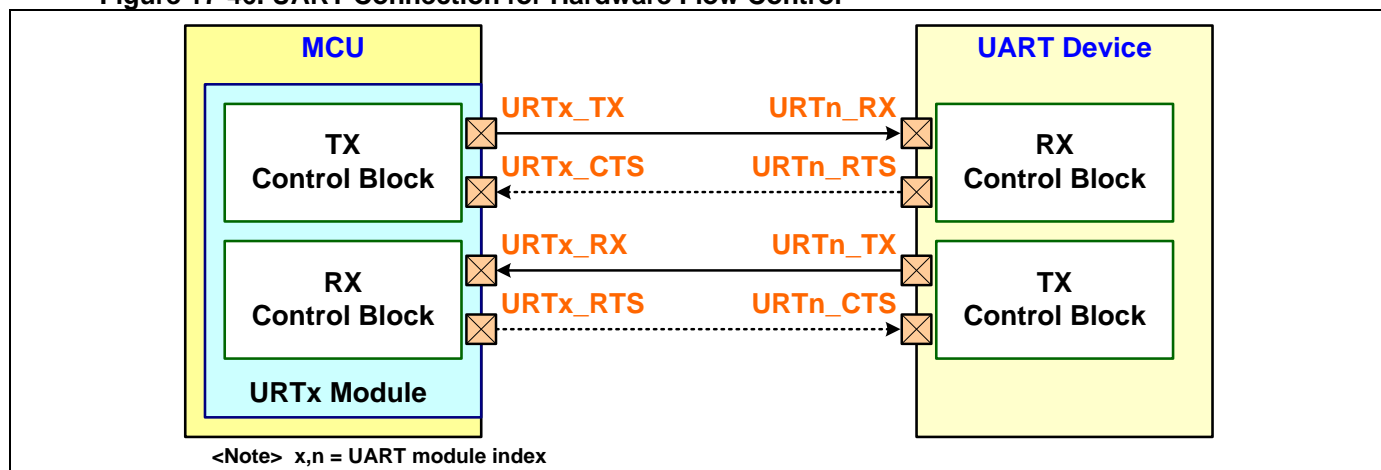
17.24. UART Hardware Flow Control

The UART supports a hardware flow control function for data transaction and provides two control signals of **URTx_CTS** (Clear to Send) and **URTx_RTS** (Request to Send) for the hardware flow control.

17.24.1. UART Connection for Hardware Flow Control

The following diagram is showing the UART connection for hardware flow control. The **URTx_TX** and **URTx_RTS** (Request to Send) signals are always as data output signals and the **URTx_RX** and **URTx_CTS** (Clear to Send) signals are always as data input signals.

Figure 17-46. UART Connection for Hardware Flow Control



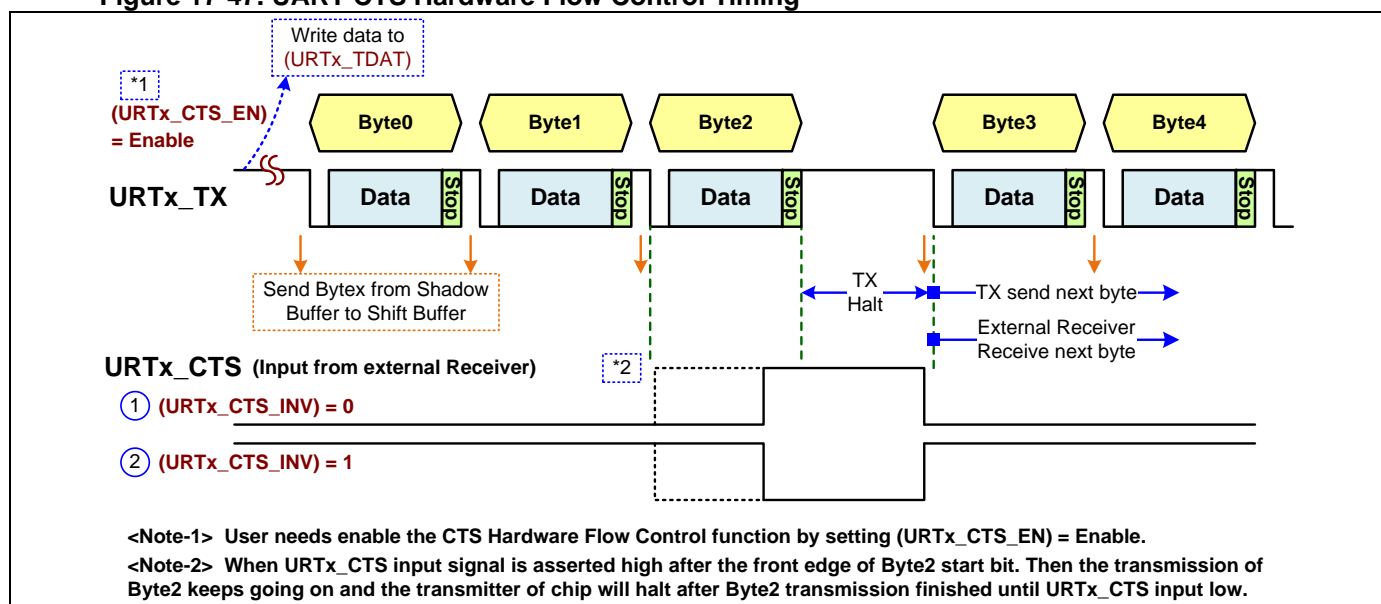
17.24.2. UART CTS Hardware Flow Control

The CTS signal is input and activates to indicate the external UART receiver which is not able to receive data anymore. The UART module must hold the data transmission until the CTS signal is inactive.

User can enable the hardware CTS input function in **URTx_CTS_EN** register. The **URTx_CTS_INV** register is used to inverse the output signal of **URTx_CTS**.

User can get the real time status of CTS line from reading **URTx_CTS** register bit.

Figure 17-47. UART CTS Hardware Flow Control Timing



17.24.3. UART RTS Hardware Flow Control

During data receiving the RX overrun condition will be happened and the ROVRF (**URT_x_ROVRF**) flag will be activated if the all data buffer of UART module is full. User needs to manage this condition.

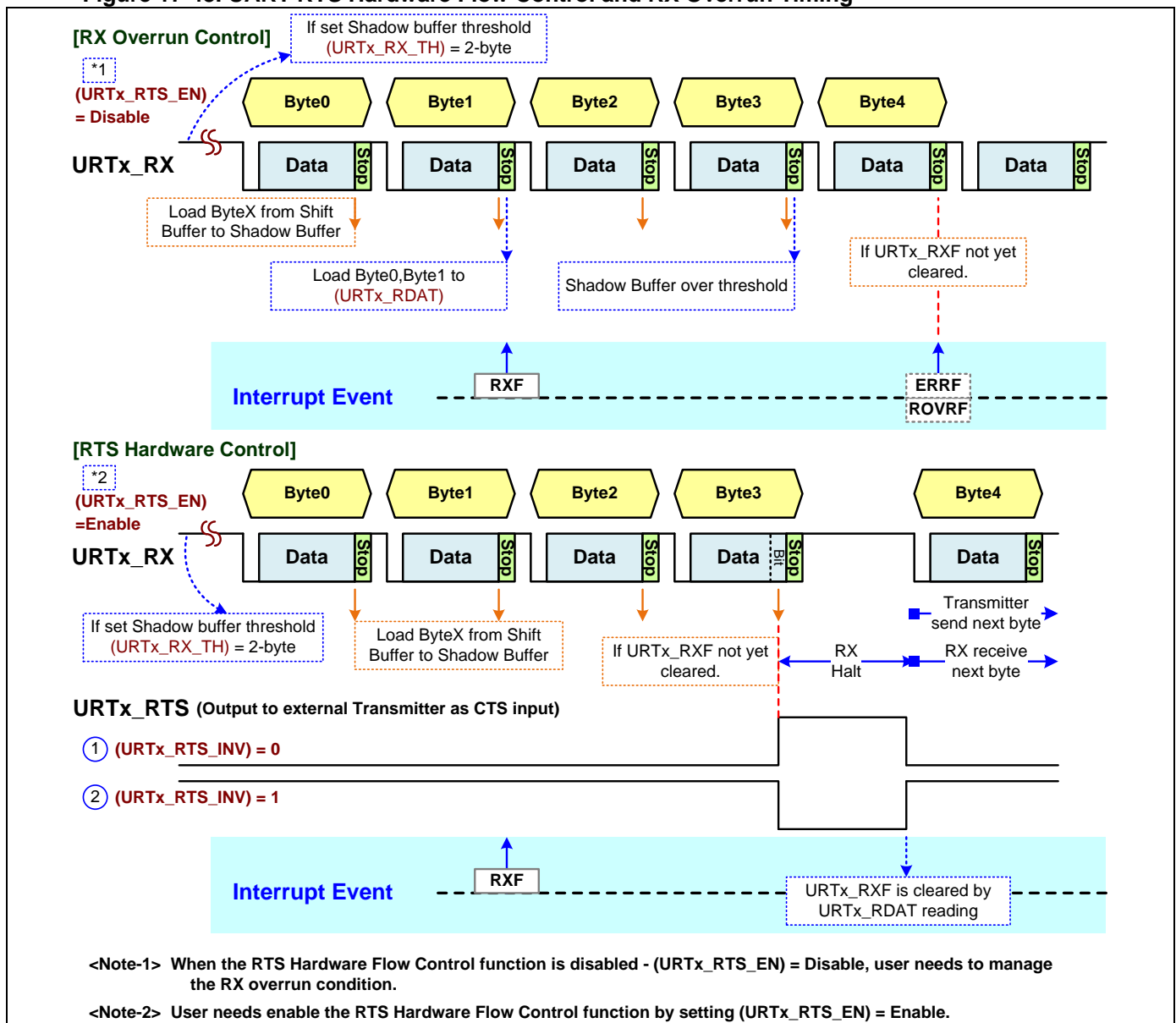
When the external transmitter supports the CTS data flow control, user can use the RTS Hardware Flow Control function to avoid RX data overrun condition by hardware auto control.

When the all data buffer of UART module is full, it can output the RTS signal and activates to indicate the external UART transmitter which does not transmit data anymore. The external UART transmitter must hold the data transmission until the data buffer of UART module is not full and the RTS signal is inactive.

User can enable the hardware RTS output function by setting **URT_x_RTS_EN** register. The **URT_x_RTS_INV** register is used to inverse the output signal of **URT_x_RTS**.

When disables the hardware RTS output function in **URT_x_RTS_EN** register, user can directly use by software and set the **URT_x_RTS** signal by setting **URT_x_RTS_OUT** register.

Figure 17-48. UART RTS Hardware Flow Control and RX Overrun Timing

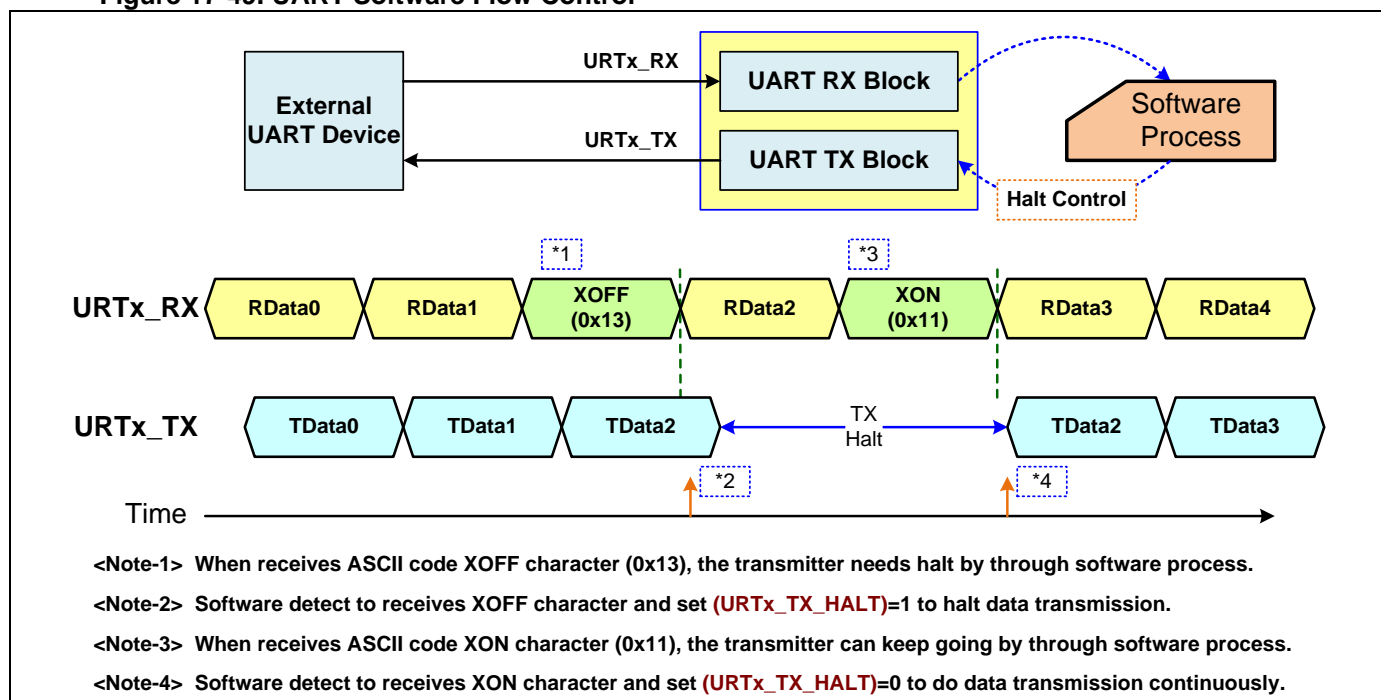


17.24.4. UART Software Flow Control

The UART module provides a software control register of **URT_x_TX_HALT** to do UART data transaction software flow control for UART ASCII code mode. When receives a **XOFF** character (0x13), user can set the register of **URT_x_TX_HALT** to halt the data transmission. When receives a **XON** character (0x11), user can clear the register of **URT_x_TX_HALT** to continue the data transmission.

For data receiving, user can send a **XOFF** character (0x13) to notify the external UART transmitter to halt data transmission if all the received data buffer of UART module is full and TX overrun will be happened. Also user can send a **XON** character (0x11) to notify the TX overrun condition is removed.

Figure 17-49. UART Software Flow Control



17.25. UART Receive Hardware Hold and Capture Control

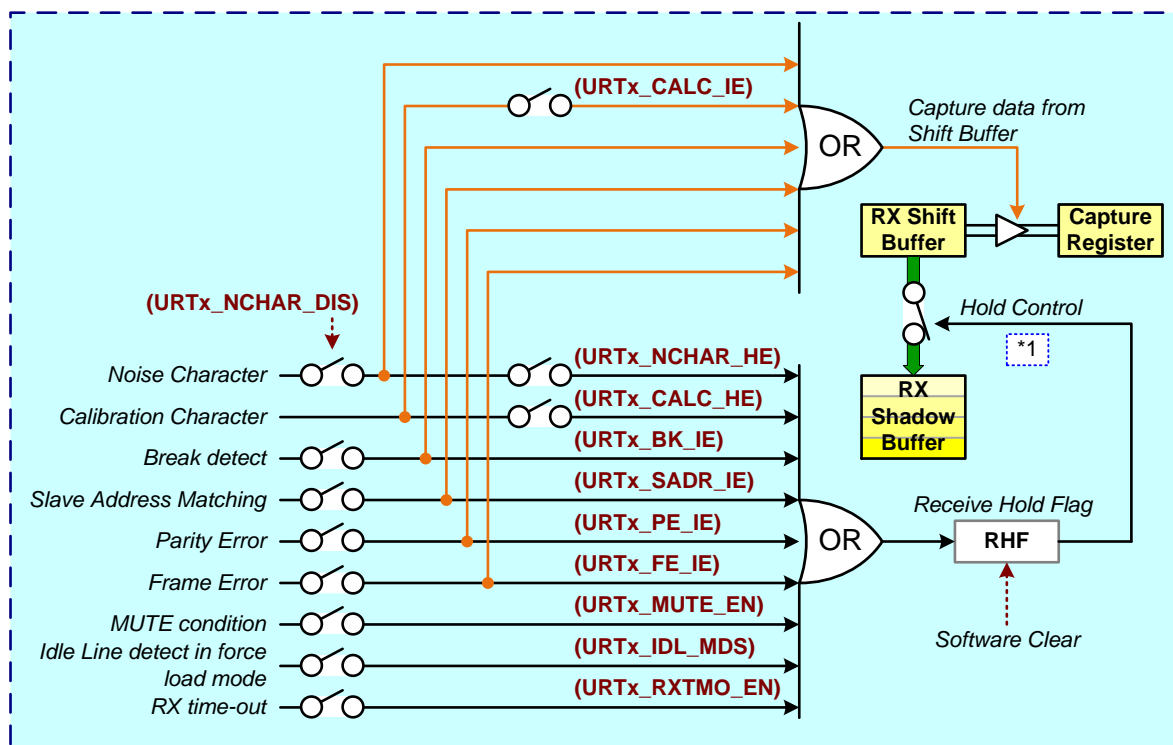
For UART data receiving, the UART module provides one capture register and support the data hold function to manage some error or special conditions.

In following diagram, all the UART events have a same control bit to enable which both capture data to capture register and hold data into shadow buffer. Of course those interrupt enable bits are also to control the event interrupt enable function. Specially, there are separated control bits for baud-rate calibration completion and noise character happened events. **URT_x_CALC_IE** and **URT_x_NCHAR_DIS** registers are used for capture data function. **URT_x_CALC_HE** and **URT_x_NCHAR_HE** registers are used for hold data function.

The capture register is used to capture the received data (**URT_x_RCAP_DAT**), Stop bit (**URT_x_RCAP_STP**), parity bit (**URT_x_RCAP_PAR**) and address bit (**URT_x_RCAP_ADR**) for these conditions of baud-rate calibration completion, noise character happened, Break detection, slave address matching, parity error and frame error. These conditions can be enabled to trigger to capture function by setting related register as showing in following diagram.

The data hold function is used to hold the received data in RX shift buffer to load into shadow for these conditions of baud-rate calibration completion, noise character happened, Break detection, slave address matching, parity error, frame error, Mute entering condition, Idle-line detection in force load mode and RX time out. These conditions can be enabled to trigger the data hold function by setting related registers as showing in following diagram. When any the data hold trigger event is happened and the related hold enable control is enabled, the RHF flag (**URT_x_RHF**) is activated and the received data is held to load into shadow buffer until all the trigger events are released.

Figure 17-50. UART Receive Hardware and Capture Hold Event



<Note-1> RX Shift Buffer data will not copy to Shadow Buffer when (URTx_RHF) is active.

The following table is showing the RHF status about shadow buffer and RX data register when multi-processor address unmatched. The RHF flag will be set by hardware for hold control. Please refer the sections of “[UART Multi-Processor](#)” and “[UART Mute Mode Control](#)” for more detail information for multi-processor mode.

Table 17-22. UART Data Buffer vs RHF Status when Multi-Processor Address Unmatched

Shadow Buffer(RX_LVL) & RDAT Register(RNUM)	Register	RHF Flag Status	Note
	MUTE_AEN0		
Both are full or Both are empty	x	0	Shadow Buffer full is defined by RX_TH
Shadow Buffer has data but not Full	0	0	RX_LVL & RNUM is not zero and RX_LVL is not equal RX_TH
	1	1	

<Note> RX_TH/RX_LVL/RNUM ~ URTx registers

17.26. UART DMA Operation

17.26.1. DMA Module Configure

When the chip supports a DMA (direct memory access) controller, user can configure the DMA setting of transferred source/destination devices, channel request arbitration and others in the DMA module before a DMA data transaction. The DMA source and the destination can be memory or peripheral.

Refer the DMA chapter for more detail information about the DMA module configuration.

17.26.2. UART DMA Control

After DMA configuration is finished, user needs to set the UART module DMA enable bits of **URTx_DMA_RXEN** and **URTx_DMA_TXEN**.

Finally, the related channel request start bit of **DMA_CHn_REQ** is necessary to be set to start the DMA transaction (n = DMA channel index). Then the transferred source/destination devices will assert the RX/TX request signal to DMA controller and the DMA controller will assert the acknowledge signal to the request source/destination devices. At the time, the data transferred connection is built for DMA transaction.

- **UART RX to DMA**

The **URT_x_DMA_RXEN** register bit is used to enable DMA data transfer from UART receiving input to DMA destination.

During the DMA transaction cycle, the received data flag of **URT_x_RXF** is masked by hardware.

- **UART TX from DMA**

The **URT_x_DMA_TXEN** register bit is used to enable DMA data transfer from DMA source to UART transmitted output.

During the DMA transaction cycle, the transmitted data flag of **URT_x_TXF** is masked by hardware. In general, the transmitted complete flag of **URT_x_TCF** will be asserted after the finished of DMA transaction.

For the UART transmission, user can set the register of **URT_x_DDTX_EN** to disable DMA TX function when hardware detects one of errors or special conditions. At the time, hardware will clear both the **URT_x_DMA_TXEN** bit and the **URT_x_DMA_RXEN** bit in this condition.

17.26.3. UART Interrupt Flag Control during DMA

During DMA operation cycle, the module's interrupt flags will control and act three types as following table. One is masked during the DMA process. Another is to disable the DMA function after the flag has asserted. At the time, hardware will disable both the **URT_x_DMA_TXEN** bit and the **URT_x_DMA_RXEN** bit in this condition. Others are normally as same as the action of not processing in DMA operation.

Table 17-23. UART Interrupt Flag Control for DMA Function

Action	Mask the Flag during DMA Process	DMA Disable after the Flag asserted (*1)	Normal Control	
Peripheral	(Data flow flags)	(Error/Detect flags)	(Other flags)	
URT _x	URT_x_TCF (*2) URT_x_RXF URT_x_TXF (*2)	URT_x_ERRF URT0_BKF(*3) URT0_IDLF(*4) URT0_CTSF(*5)	URT_x_UGF URT_x_LSF	URT0_SADRF URT0_BRTF URT0_TMOF URT0_CALCF URT0_PEF URT0_FEF URT0_NCEF URT0_ROVRF URT0_TXEF URT0_RXTMOF URT0_IDTMOF URT0_BKTMOF URT0_CALTMOF

Note-1 : When the flag is asserted, it will not force peripheral DMA disabled if the related interrupt enable bit is not enabled.

Note-2 : URT_x_TCF will be asserted after DMA TX complete.

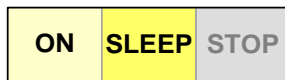
Note-3 : When the flag is asserted, it will force URT_x DMA RX disabled if the related interrupt enable bit is enabled. At the same time, the URT_x DMA TX is disabled or not by URT_x_DDTX_EN setting.

Note-4 : When the flag is asserted, it will force URT_x DMA RX disabled if the related interrupt enable bit is enabled. But URT_x DMA TX will be not.

Note-5 : When the flag is asserted, it will force URT_x DMA TX disabled if the related interrupt enable bit is enabled. But URT_x DMA RX will be not. Generally user sets CTS hardware auto control mode (URT_x_CTS_EN=1) and will not enable the URT_x_CTS_IE, URT_x will hold the transmission and not disable URT_x DMA TX when detects the CTS active.

18. SPI (Serial Peripheral Interface)

18.1. Introduction



The module can be running in ON and SLEEP modes only.

The chip provides a high-speed serial peripheral interface (SPI). SPI is a full-duplex, high-speed and synchronous communication bus with two operation modes: Master mode and Slave mode.

The SPI module builds in the shadow buffer and data register by transmit and receive independently to improve transmit and receive communication performance.

Notify: The sign of (x = module index) is using for Registers, Signals and Pins/Ports in the descriptions of this chapter. [EX]: **SPIx_EN** ~ x indicates module index number.

18.2. Features

- Support one SPI module – SPI0
- Support master and slave mode
 - Support full duplex , half duplex or simplex communication mode
 - Support data communication without NSS(slave select signal)
 - Support master data input sampling delay half of SPI clock
 - Support slave data output advance half of SPI clock
 - Support configurable idle state for SPI master standard mode data output
 - Support asynchronous clock mode for SPI slave standard mode
- Support programmable clock rate control
- Selectable 4~32-bit frame size
 - Support 4-byte data buffer and 32-bit data register for high speed communication
- Received and transmitted data are buffered with DMA capability
- Support multi-master processing capability
- Selectable clock polarity and phase
- Selectable MSB or LSB first data order
- NSS line management by hardware or software for master mode
- Configurable data transfer modes
 - Standard SPI mode (separated transmit and receive line)
 - Single/Dual/Quad/Octal SPI mode with bidirectional data transfer
 - Support data copy mode for Dual/Quad SPI modes
 - Support data lines duplicate mode for dual Quad SPI devices
- Support DTR (Double Transfer Rate) mode
- Data transmit/receive overrun detect
- Support hardware master mode failure detection and auto slave mode change

18.3. Implementation

18.3.1. Chip Implementation

The following table is showing the implementation of SPI module.

Table 18-1. SPI Implementation

Chip	SPI Max. Clock Rate		SPI Data Mode			
	Master	Slave	1/2/4-Line SPI	8-Line SPI	4-Line Duplicate	DTR
MG32F02A128/A064 MG32F02U128/U064	22MHz	16MHz	V	V	V	V
MG32F02V032	24MHz	16MHz	V	-	-	V
MG32F02N128/N064 MG32F02K128/K064	24MHz	16MHz	V	V	V	V

<Note> V: Implemented; Measure maximum clock rate at VDD>=3.3V.

18.3.2. Module's Functions

The following table is showing the implemented functions of SPI module.

Table 18-2. SPI Modules' Functions

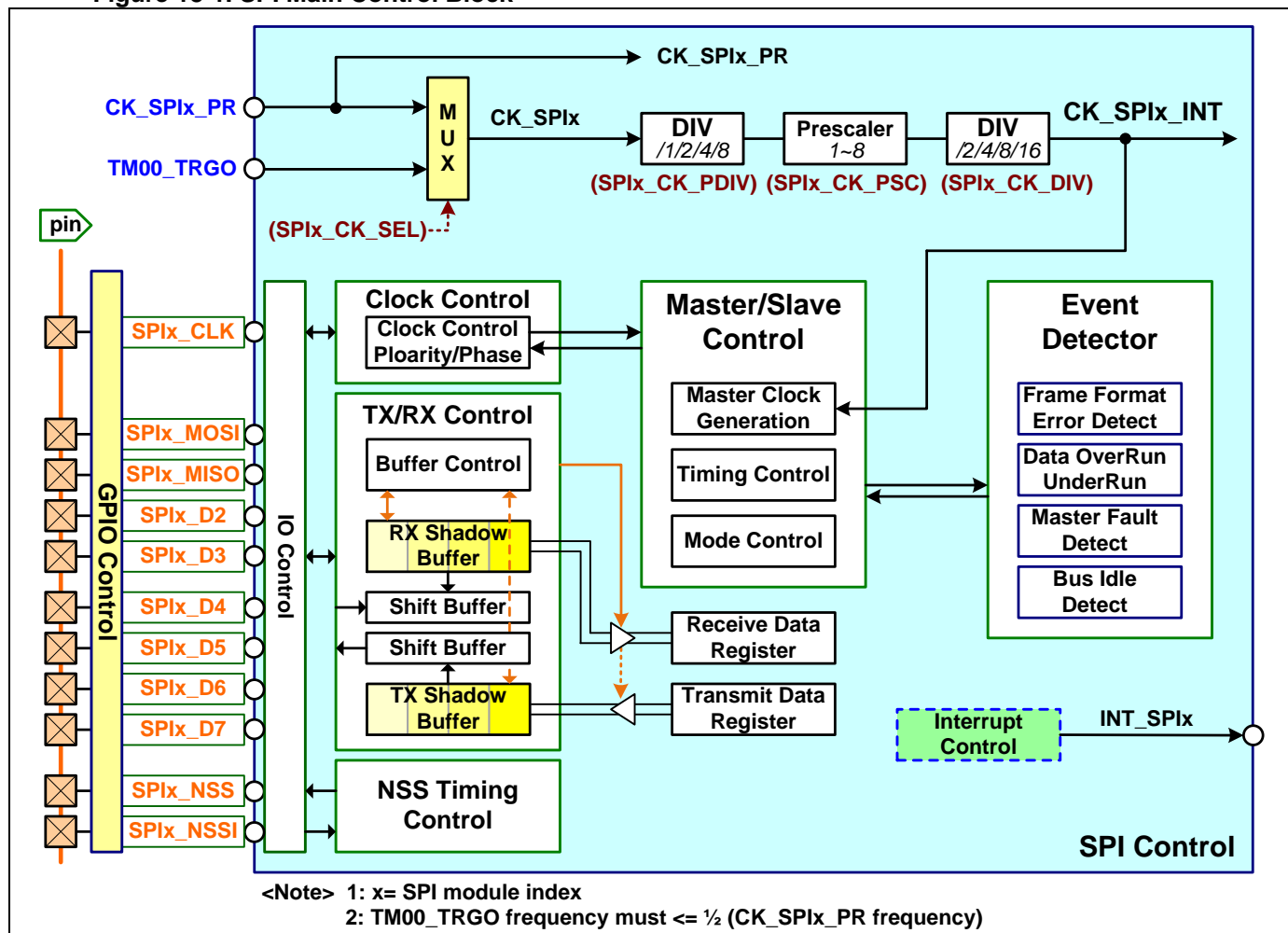
Module	SPI0			Comment
Chip	MG32F02A128 MG32F02U128 MG32F02A064 MG32F02U064	MG32F02V032	MG32F02N128 MG32F02K128 MG32F02N064 MG32F02K064	
Module Functions				
SPI Standard Mode	Master/Slave	Master/Slave	Master/Slave	2 data line full-duplex communication
SPI 1/2/4-Line Mode	yes	yes	yes	1/2/4 data lines, half-duplex communication
SPI 4-Line Duplicate	yes	-	yes	8 data lines with two duplicated 4 data lines, half-duplex communication
SPI 8-Line Mode	yes	-	yes	8 data lines, half-duplex communication
Slave asynchronous mode	yes	yes	yes	Slave mode process with asynchronous clock input
Master data sampling delay	yes	yes	yes	Master data input sampling delay 1/2 SPI clock time
Slave data output advance	-	yes	yes	Slave data output advance 1/2 SPI clock time for Mode 0,3
Data line copy mode	yes	yes	yes	all data lines with same data for 2/4 data lines mode
DTR mode	yes	yes	yes	dual transfer rate mode
CLK pin	1	1	1	clock signal
Data pins	8	4	8	MOSI(D0),MSIO(D1),D[2..7] signals for Full/Half Duplex
NSS pin	1	1	1	NSS line management by hardware or software for master mode
NSSI pin	1	1	1	extra NSS input for multi-master application
Swappable MOSI/MISO	yes	yes	yes	
Swappable D[3:0]/D[7:4]	yes		yes	
Master mode	yes	yes	yes	
Slave mode	yes	yes	yes	
Shadow Buffer	4-byte	4-byte	4-byte	RX/TX 32-bit
Msb/Lsb transfer option	yes	yes	yes	
Programmable data bit size	4~32 bits	4~32 bits	4~32 bits	4~32 bits
Programmable clock phase and polarity	yes	yes	yes	
Hardware NSS control	yes	yes	yes	control by hardware for both master and slave
NSS pulse mode	yes	yes	yes	optional pulse between two sequent data frame
Programmable NSS pulse width	yes	yes	yes	
Back-to-Back Data Transfer	yes	yes	yes	when NSS pulse disable
Mode fault detect	yes	yes	yes	master mode failure/change detect
Bus Idle detect	yes	yes	yes	slave mode bus idle detect
Receive overrun detect	yes	yes	yes	
Transmit underrun detect	yes	yes	yes	slave mode data transmit underrun
Transmit write error detect	yes	yes	yes	slave mode NSS invalid termination; Bit count error
DMA request capability	yes	yes	yes	

18.4. Control Block

The following diagram is showing the SPI Control block.

[Notify]: The **SPIx_D[7:4]** is not supported for MG32F02V032.

Figure 18-1. SPI Main Control Block



18.5. IO Lines

18.5.1. IO Signals

- **SPIx_CLK**

It is the SPI clock signal and uses as output for SPI master mode or as input for SPI slave mode.

- **SPIx_MOSI**

It is the SPI data signal and uses as output for SPI master mode or as input for SPI slave mode when is running SPI standard two-line full duplex communication mode. When is running SPI half duplex communication mode, this signal does as SPI data line-0 (SPIx_D0) and uses as bidirectional IO for both SPI master and slave modes.

- **SPIx_MISO**

It is the SPI data signal and uses as input for SPI master mode or as output for SPI slave mode when is running SPI standard two-line full duplex communication mode. When is running SPI half duplex communication mode, this signal does as SPI data line-1 (SPIx_D1) and uses as bidirectional IO for both SPI master and slave modes.

- **SPIx_D[2..3]**

These signals are the SPI data signals of line-2 ~ line-3. They can be conjunction with SPIx_MOSI (SPIx_D0) and SPIx_MISO (SPIx_D1) signals to use as bidirectional IO for both QSPI master and slave modes.

- **SPIx_D[4..7]**

These signals are the SPI data signals of line-4 ~ line-7. They can be conjunction with SPIx_MOSI

(SPIx_D0), SPIx_MISO (SPIx_D1) and SPIx_D[2..3] signals to use as bidirectional IO for both OSPI master and slave modes.

- **SPIx_NSS**

It is the SPI slave select or chip select signal. It uses as output for SPI master mode or as input for SPI slave mode.

- **SPIx_NSSI**

It is the SPI slave select or chip select input signal. It optionally uses for SPI slave mode or multi-master application.

18.5.2. IO Configure

User must configure the related IO pins in order to use the IO lines of this module. The IO operation modes, output high speed option, pull-high option, output drive strength, IO deglitch filter and input inverse selection are programmable by pin independent. Please refer the section of “[IO Mode](#)” in GPIO chapter for more detail descriptions of IO mode configuration.

Each IO signal may be mapped and selected on several IO pins by configuring the IO AFS matrix. Please refer the section of “[Alternate Function Select](#)” in GPIO chapter for more detail descriptions of IO AFS configuration. About the actual IO pin AFS information, please refer the section of “[Pin Alternate Functions Selected Table](#)” in Pin Description chapter of the chip Data Sheet.

18.6. Enabling and Clock

18.6.1. SPI Global Enable

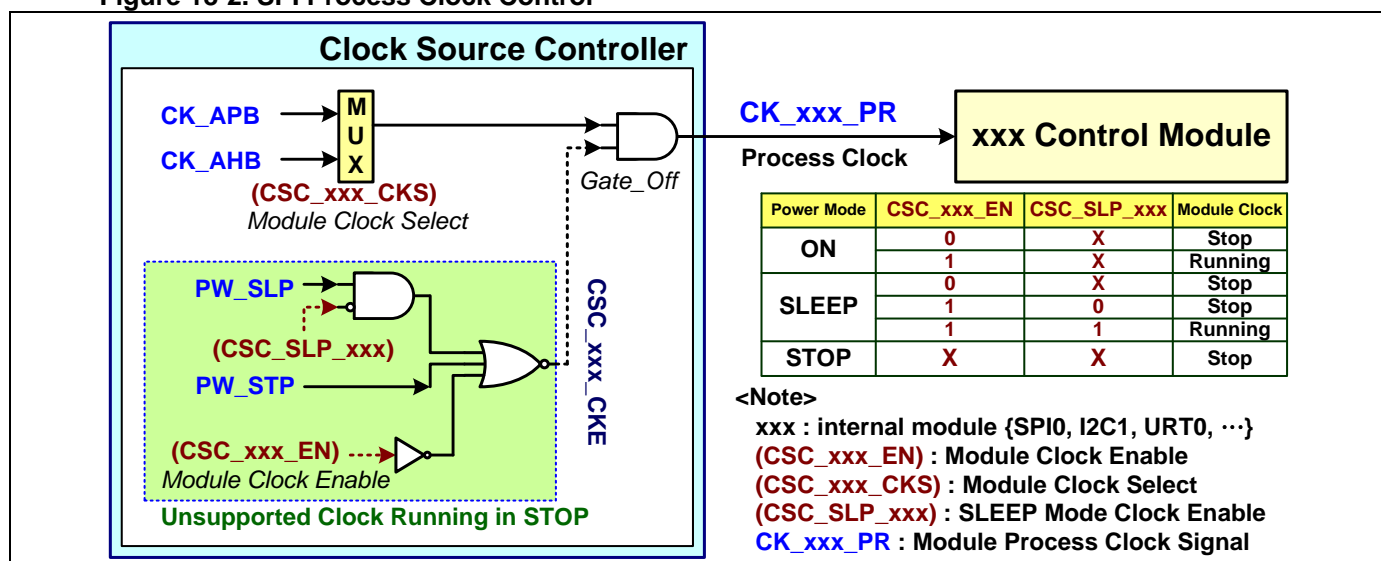
There is a global enable bit of **SPIx_EN** for all functions of this module. When this bit is disabled, all the SPI functions are not working.

18.6.2. SPI Clock Control

- **Module Process Clock**

The module process clock of **CK_SPIx_PR** is using for the interface control logic between APB bus and the module. It is coming from CSC (Clock Source Controller) module. It can be enabled in **CSC_SPIx_EN** register and select the clock source from APB clock or AHB clock in **CSC_SPIx_CKS** register. User can plan the module clock is running or not beforehand for chip entering **SLEEP** mode by setting **CSC_SLP_SPIx** register. Refer the System Clock chapter for more information.

Figure 18-2. SPI Process Clock Control



- **Module Internal Clock**

The SPI module is able to output the internal clock of **CK_SPIx_INT** as the sampling clock for received data signal and the clock source for transmitted data signal. User can select the clock source from the module process clock of **CK_SPIx_PR** or the timer trigger output signal of **TM00_TRGO** by setting **SPIx_CK_SEL** register. Also the module provides one pre-divider, one prescaler and one post-divider to generate the internal clock of **CK_SPIx_INT**. The related setting registers are **SPIx_CK_PDIV**, **SPIx_CK_PSC** and **SPIx_CK_DIV**.

[Notify]: Usually the internal clock frequency needs slow down at least 1/2 than module process clock.

18.7. Interrupt and Event

18.7.1. SPI Interrupt Control and Status

There is one signal of **INT_SPIx** to be generated in this SPI control module. **INT_SPIx** sends to EXIC External Interrupt Controller to do as an interrupt event.

These interrupt flags are using for the interrupt service routine (ISR) flow control. Generally these flags are set by hardware and software clears them when the serviced job is completed in the related ISR. Each interrupt flag has one interrupt enable bit. User can enable or disable it. There is one interrupt all enable bit of **SPIx_IEA** to enable or disable all the interrupt sources for this module.

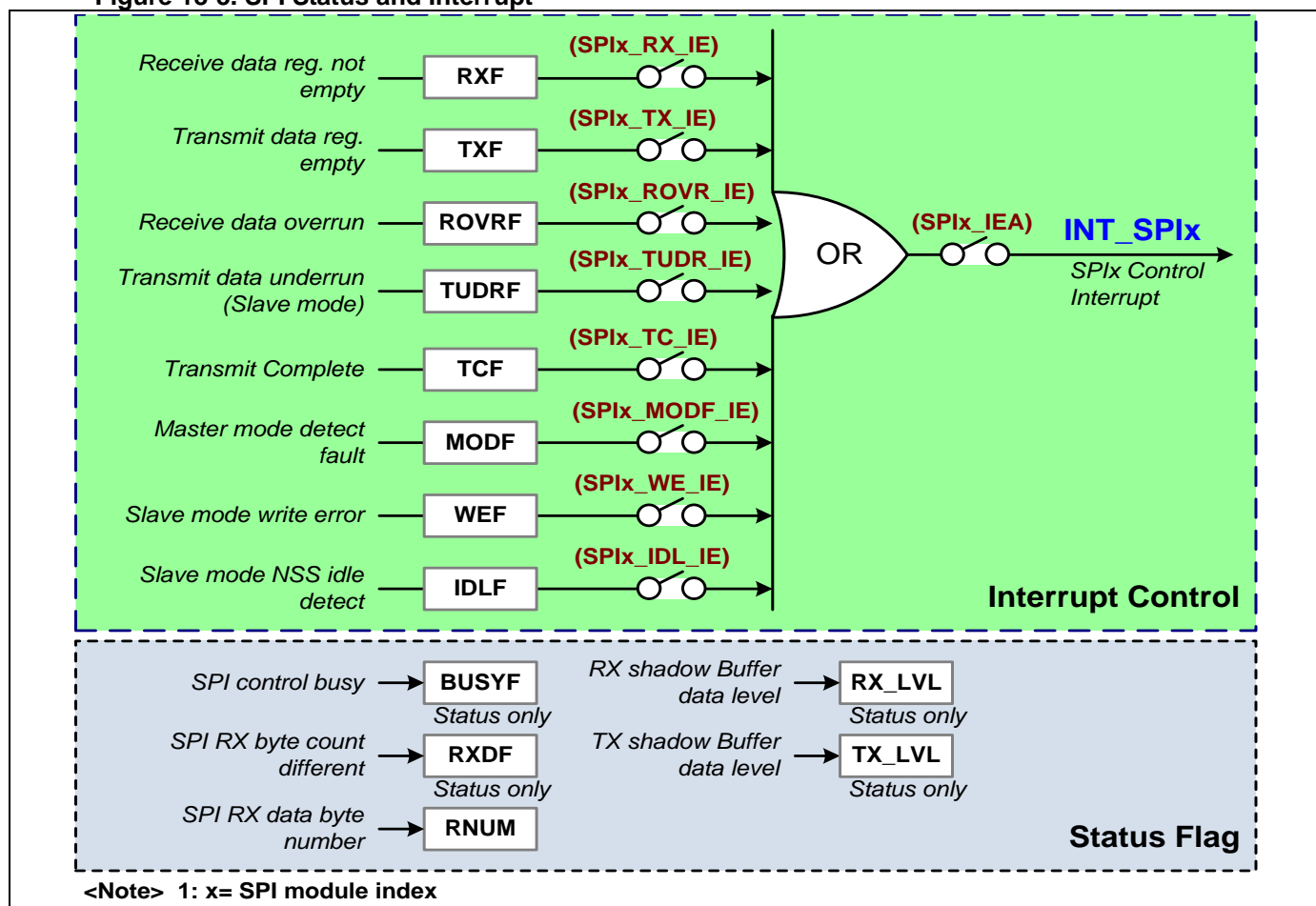
There are some status bits those are reading only to provide internal control status. One busy flag of **SPIx_BUSYF** is used to indicate the data transfer busy status. Refer the register descriptions of the related status bits for more information.

18.7.2. SPI Interrupt Flags

Generally these interrupt flags are set by hardware and clear by software writing 1. Refer the register descriptions for more information about the related interrupt flags and interrupt enable bit.

The following diagram is showing the SPI status and interrupt control block.

Figure 18-3. SPI Status and Interrupt



- **IDLF**

SPI slave mode NSS idle detect flag (**SPIx_IDLF**). There is a related interrupt enable register bit of **SPIx_IDL_IE**.

- **TCF**

SPI transmission complete flag (**SPIx_TCF**). When both shadow buffer and data register are empty and shift buffer shift out complete, then set this flag. There is a related interrupt enable register bit of **SPIx_TC_IE**.

- **RXF**

SPI receive data register not empty flag (**SPIx_RXF**). There is a related interrupt enable register bit of **SPIx_RX_IE**. When received data buffer level **SPIx_RX_LVL** is greater than or equal to the shadow buffer threshold **SPIx_RX_TH** setting, this flag is set and the data buffer content copy to data register **SPIx_RDAT**. This bit is cleared when **SPIx_RDAT** is read or this flag set to 1 by software. But this flag is not cleared when **SPIx_RDAT** is read by SWD debugging.

- **TXF**

SPI transmit data register empty flag (**SPIx_TXF**). There is a related interrupt enable register bit of **SPIx_TX_IE**. When transmitted shadow buffer is empty and the data register **SPIx_TDAT** will copy to the shadow buffer, this flag is set. This bit is cleared when **SPIx_TDAT** is written or this flag set to 1 by software.

- **MODF**

SPI mode detect fault flag (**SPIx_MODF**). For master mode, the flag will be set if the NSS signal is pull-low by external device. There is a related interrupt enable register bit of **SPIx_MODF_IE**. Refer the section of "[SPI Master Mode Change Detect](#)" for more information.

- **WEF**

SPI slave mode write error flag (**SPIx_WEF**). It will assert an error when master stop read by setting high on NSS signal before a complete data transaction. The bit size of a data transaction is defined in **SPIx_DSIZE**. There is a related interrupt enable register bit of **SPIx_WE_IE**.

- **ROVRF**

SPI receive overrun flag (**SPIx_ROVRF**). There is a related interrupt enable register bit of **SPIx_ROVR_IE**.

- **TUDRF**

SPI slave mode transmit underrun flag (**SPIx_TUDRF**). There is a related interrupt enable register bit of **SPIx_TUDRF_IE**.

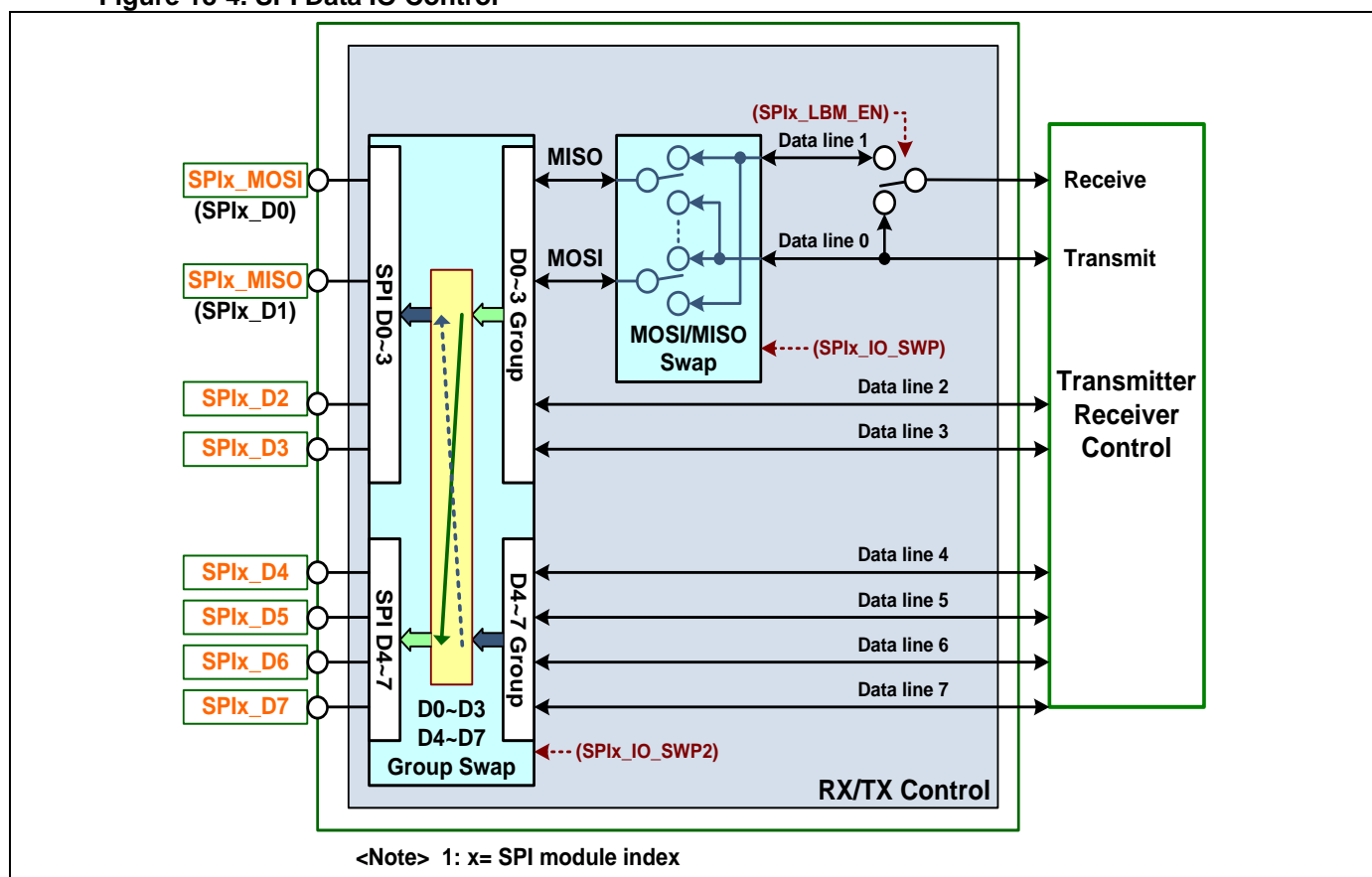
18.8. SPI Module IO Control

This module provides maximum 8 data signals – **SPIx_D[7:0]**, one clock signal – **SPIx_CLK**, one slave select input/output signal – **SPIx_NSS** and one extra slave select input signal – **SPIx_NSSI**. The **SPIx_D0** and **SPIx_D1** signals are also naming **SPIx_MOSI** and **SPIx_MISO** to do as SPI master output/slave input data signal and SPI master input/slave output data signal for standard SPI communication.

18.8.1. SPI IO Control

The **SPIx_IO_SWP** register is used to enable to swap the signals of **SPIx_MOSI** and **SPIx_MISO**. The **SPIx_IO_SWP2** register is used to enable to swap the signals of **SPIx_D[3:0]** and **SPIx_D[7:4]**. The following diagram is showing the SPI data IO control block.

Figure 18-4. SPI Data IO Control



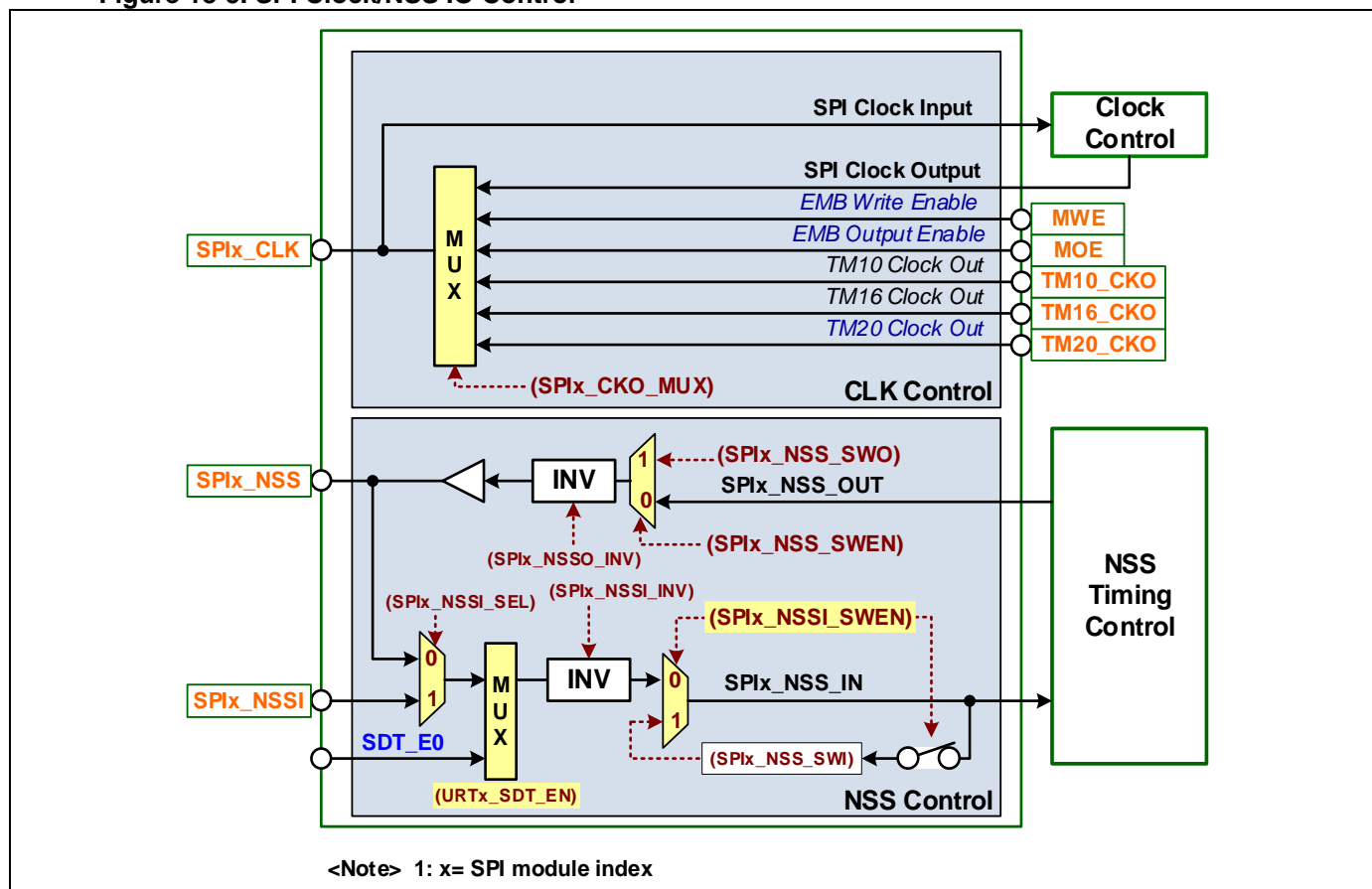
For the application, user can select the **SPIx_CLK** clock output source by setting **SPIx_CKO_MUX** register. The **MWE** and **MOE** signals are come from EMB (External Memory Bus) module. All the **TMx_CKO** signals are come from TMx timer modules.

The registers of **SPIx_NSSO_INV** and **SPIx_NSSI_INV** are used to inverse the related signal. User can select the NSS signal input from **SPIx_NSS** or **SPIx_NSSI** by setting **SPIx_NSSI_SEL** register.

The **SPIx_NSS_SWEN** register is used to enable **SPIx_NSS** output signal by software control. When it enables, user can directly control the output of **SPIx_NSS** by setting **SPIx_NSS_SWO** register. The **SPIx_NSSI_SWEN** register is used to enable **SPIx_NSS** or **SPIx_NSSI** input signal by software control. When it enables, user can directly get the input of **SPIx_NSS** or **SPIx_NSSI** by reading **SPIx_NSS_SWI** register.

The following diagram is showing the SPI clock and NSS control block.

Figure 18-5. SPI Clock/NSS IO Control



18.8.2. SPI IO Mode

Usually user can set the IO mode of the using IO pins to push-pull or open-drain for SPI output signals. Also user can set the IO mode of the using IO pins to digital input or open-drain for SPI input signals. For SPI bidirectional signal, the IO mode of the using IO pins need set to push-pull or open-drain. When selects push-pull mode, the SPI data signal will output by push-pull mode and be open-drain for idle or input state. When selects open-drain mode, the SPI data signal will always be open-drain for idle, input or output state.

18.8.3. SPI Loop Back Mode

The SPI module is built-in a loop back mode for internal self-testing by setting **SPIx_LBM_EN** register. When loop back mode enables, the received input is taken from transmitted output to replace from input pin for received input. The SPI loop back mode function is only supported for standard SPI mode.

18.9. SPI Connection for Application

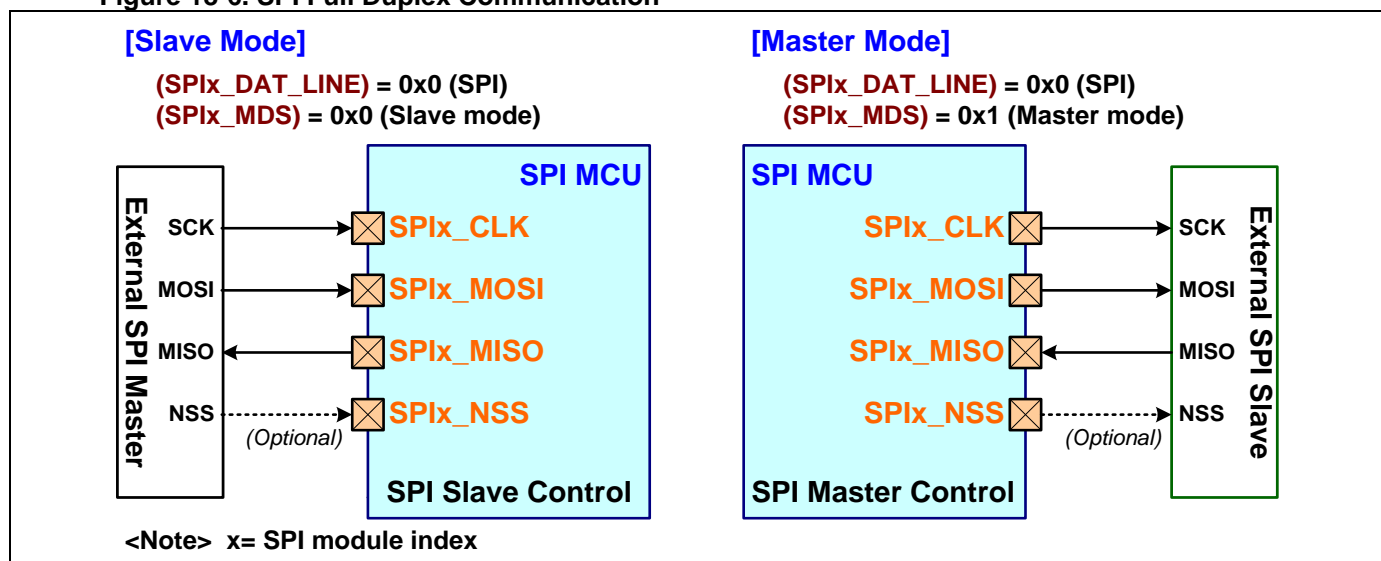
The following diagrams are showing the SPI application connections of Full Duplex Communication, Simplex Communication, Half Duplex Communication, Multi-Slave Communication, Two Master Communication and Master and Slave Swap Communication.

For the following descriptions, the **SCK** is the SPI clock signal and the **NSS** is negative SPI slave select signal.

18.9.1. Full Duplex Communication

These are four connected signals of **SCK**, **MOSI** (master output / slave input), **MISO** (master input / slave output) and optional **NSS**. The data signal of **MOSI** is unidirectional from master to slave and the data signal of **MISO** is unidirectional from slave to master. This connection is able to do full duplex communication.

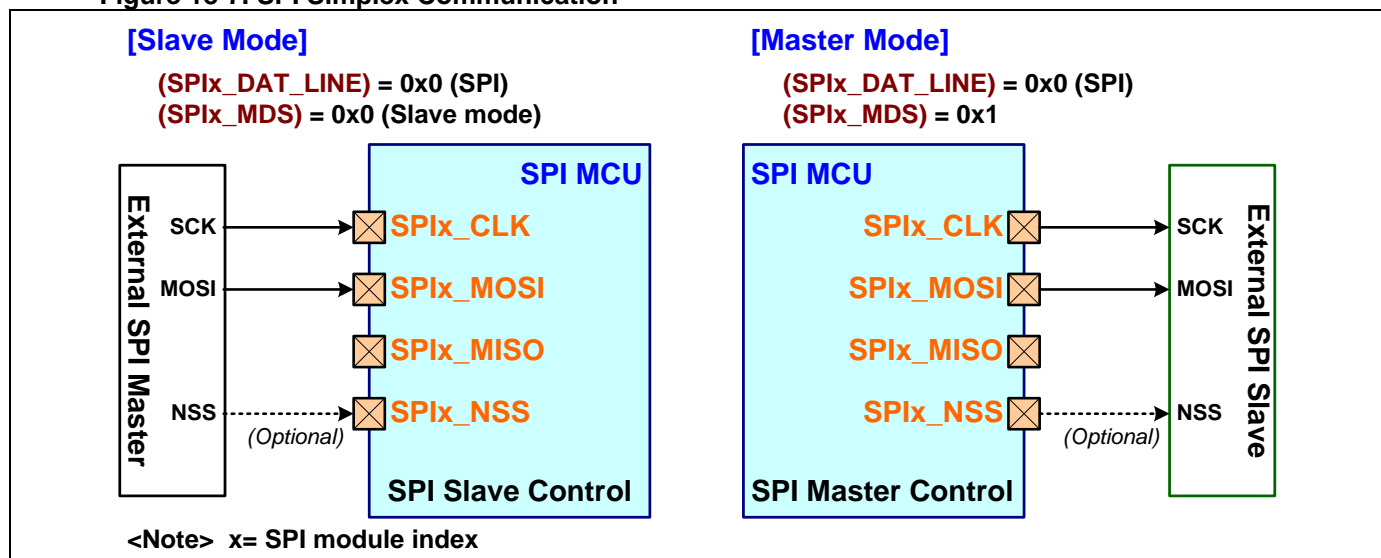
Figure 18-6. SPI Full Duplex Communication



18.9.2. Simplex Communication

These are three connected signals of **SCK**, **MOSI** (master output / slave input) and optional **NSS**. The data signal of **MOSI** is unidirectional from master to slave. This connection is able to do simplex communication only.

Figure 18-7. SPI Simplex Communication



18.9.3. Half Duplex Communication

These connections are able to do half duplex communication.

- **One Data Line**

These are three connected signals of **SCK**, **MOMI** (master output / input) or **SOSI** (slave output / input) and optional **NSS**. The data signal of **MOMI** or **SOSI** is bidirectional between master and slave.

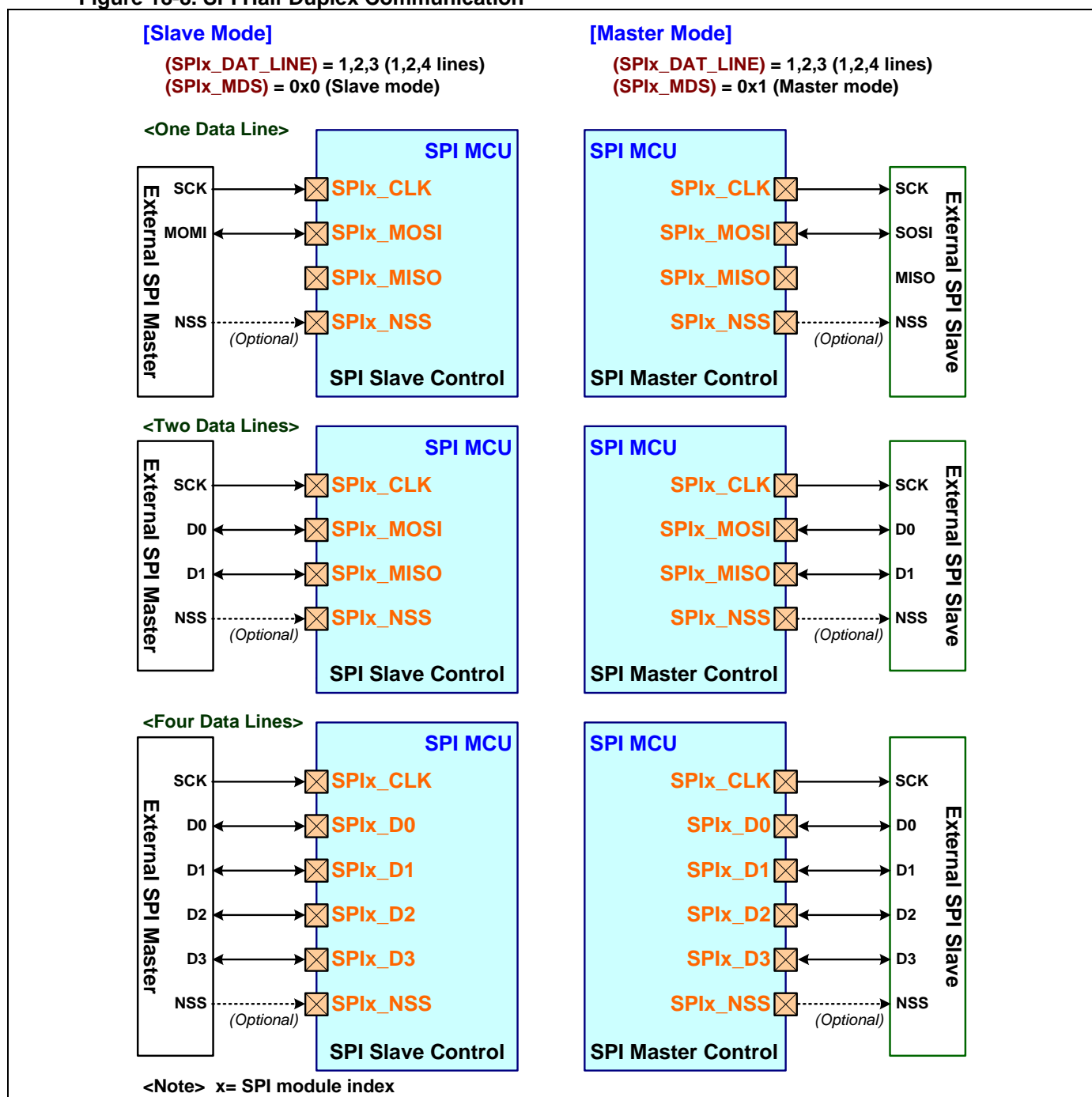
- **Two Data Lines**

These are four connected signals of **SCK**, **D[1:0]** (data signals) and optional **NSS**. The data signals of **D[1:0]** are bidirectional between master and slave.

- **Four Data Lines**

These are six connected signals of **SCK**, **D[3:0]** (data signals) and optional **NSS**. The data signals of **D[3:0]** are bidirectional between master and slave.

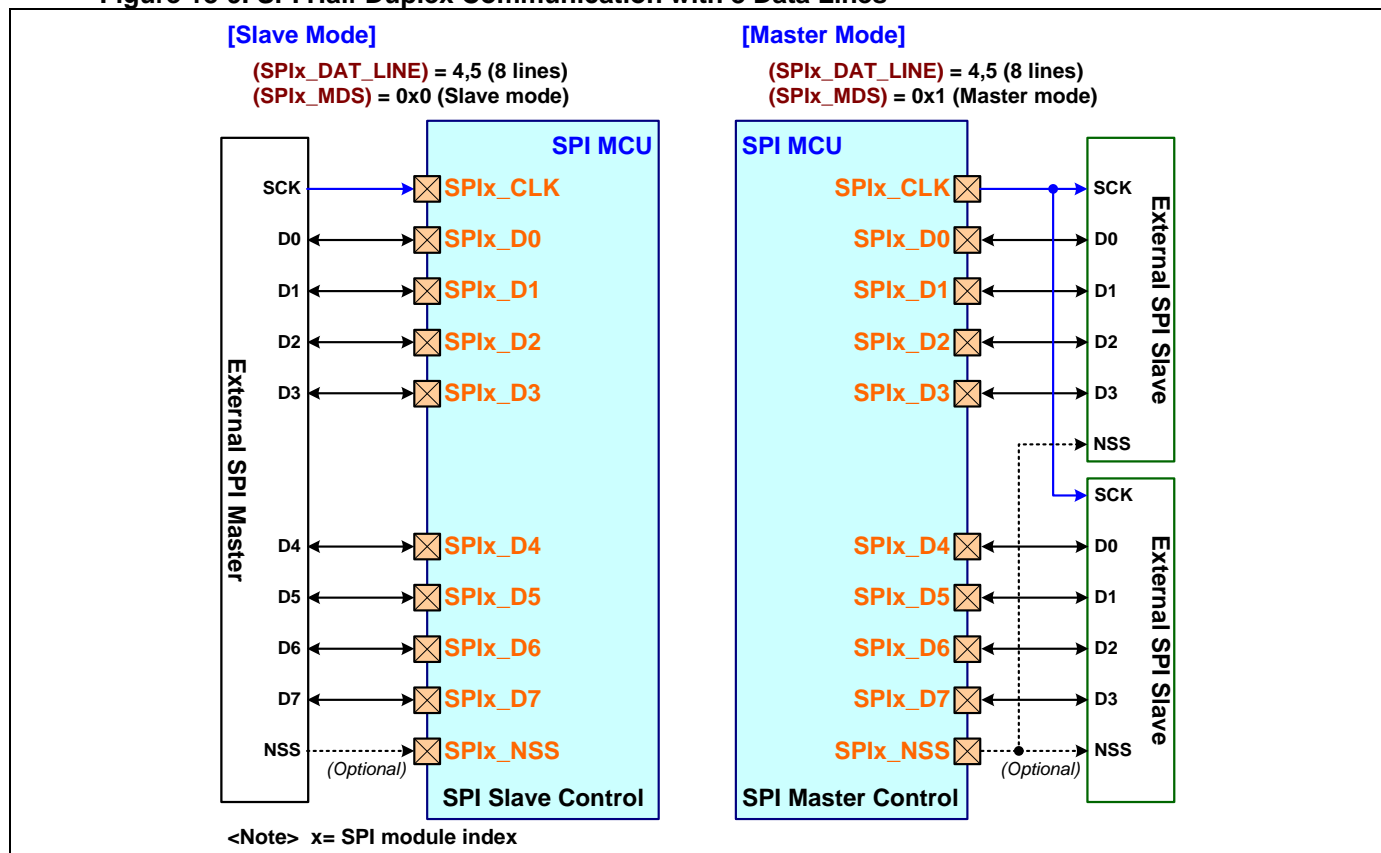
Figure 18-8. SPI Half Duplex Communication



- Eight Data Lines

These are ten connected signals of **SCK**, **D[7:0]** (data signals) and optional **NSS**. The data signals of **D[7:0]** are bidirectional between master and slave.

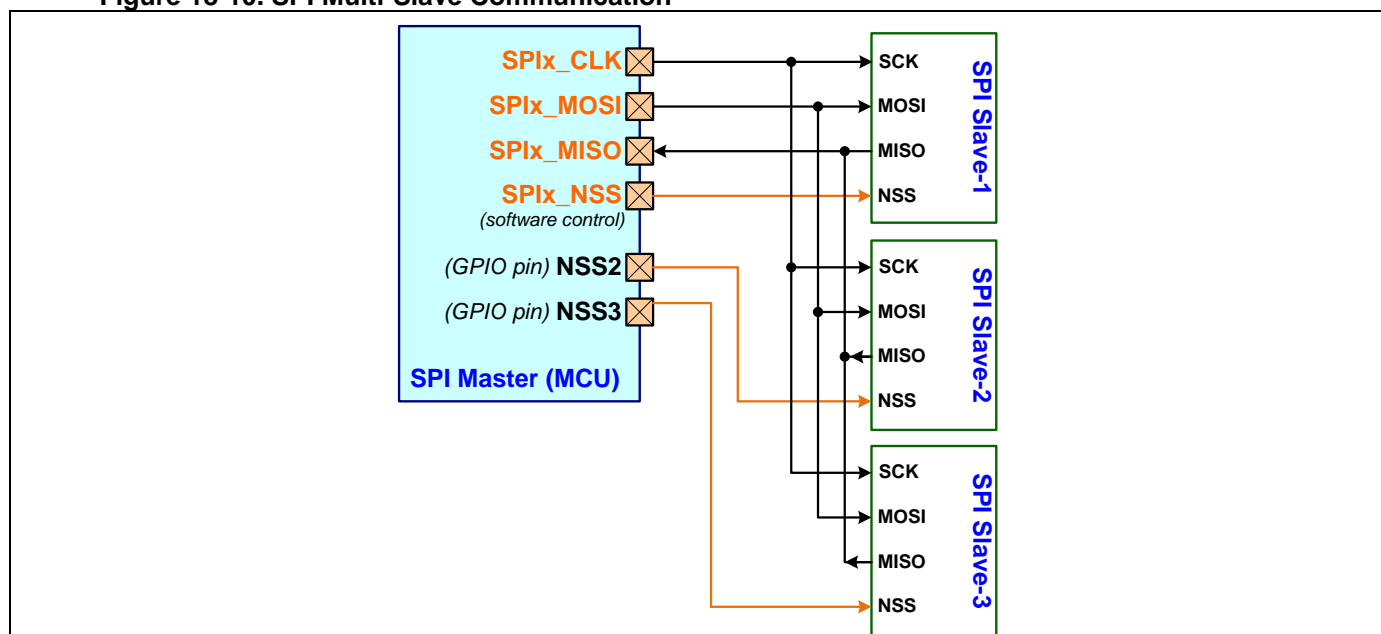
Figure 18-9. SPI Half Duplex Communication with 8 Data Lines



18.9.4. Multi-Slave Communication

This connection is one SPI master with multiple SPI slaves. According the number of slaves, user needs to add extra **NSS** signal(s) (**NSS2**, **NSS3**, ...) for the slave select control of these slaves.

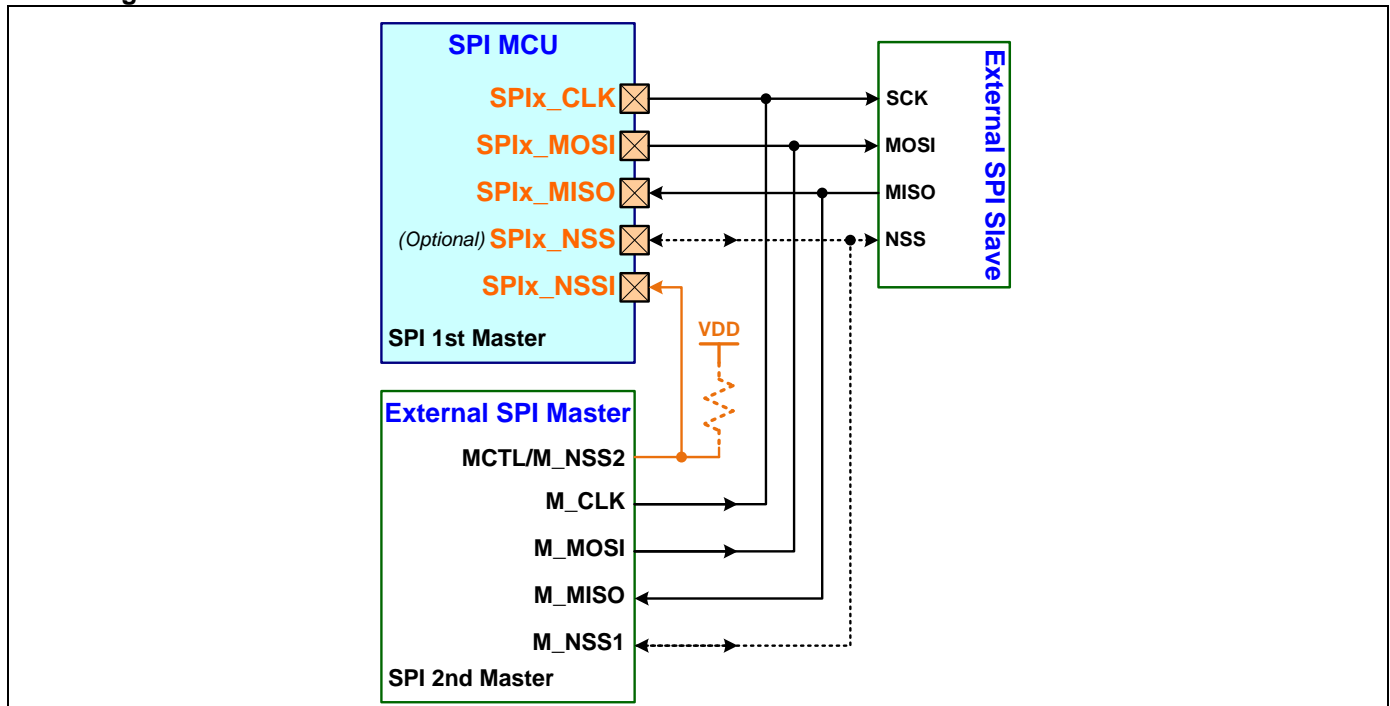
Figure 18-10. SPI Multi-Slave Communication



18.9.5. Two Master Communication

This connection is two SPI masters with one SPI slave. One extra **NSSI** signal can use to notify one 1st master to release the SPI bus by another 2nd master. When the **NSSI** is active, the 1st master needs to disable and releases the SPI bus. Then the slave is free and can be commanded by 2nd master. When the 2nd master is finished the SPI transaction, it can release the bus and set **NSSI** inactive. Then the 1st master can get the SPI bus again.

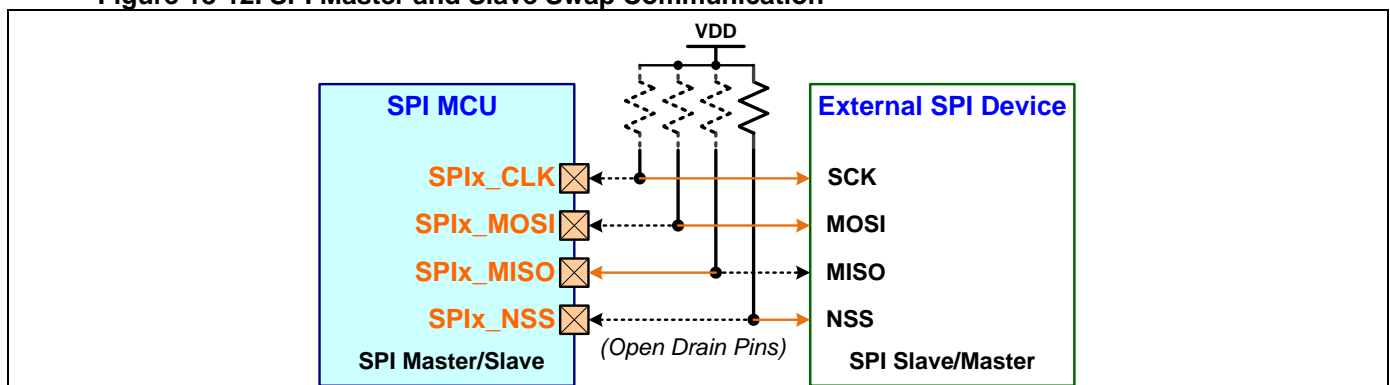
Figure 18-11. SPI Two Master Communication



- **Master and Slave Swap Communication**

This connection is two SPI devices those can do as master or slave. When one is master, another must be slave. They can swap the role of master and slave by activating and pulling the **NSS** signal to notify each other during the **NSS** signal is inactive state. Refer the "[SPI Master Mode Change Detect](#)" section for more information.

Figure 18-12. SPI Master and Slave Swap Communication



18.10. SPI Fundamental Control

18.10.1. SPI Clock

The SPI module can be set the clock mode by setting **SPIx_CPOL** and **SPIx_CPHA** registers for **SPIx_CLK** signal. The following table is showing the SPI clock mode setting for clock idle state and data sampling clock edge.

Table 18-3. SPI Clock Mode Table

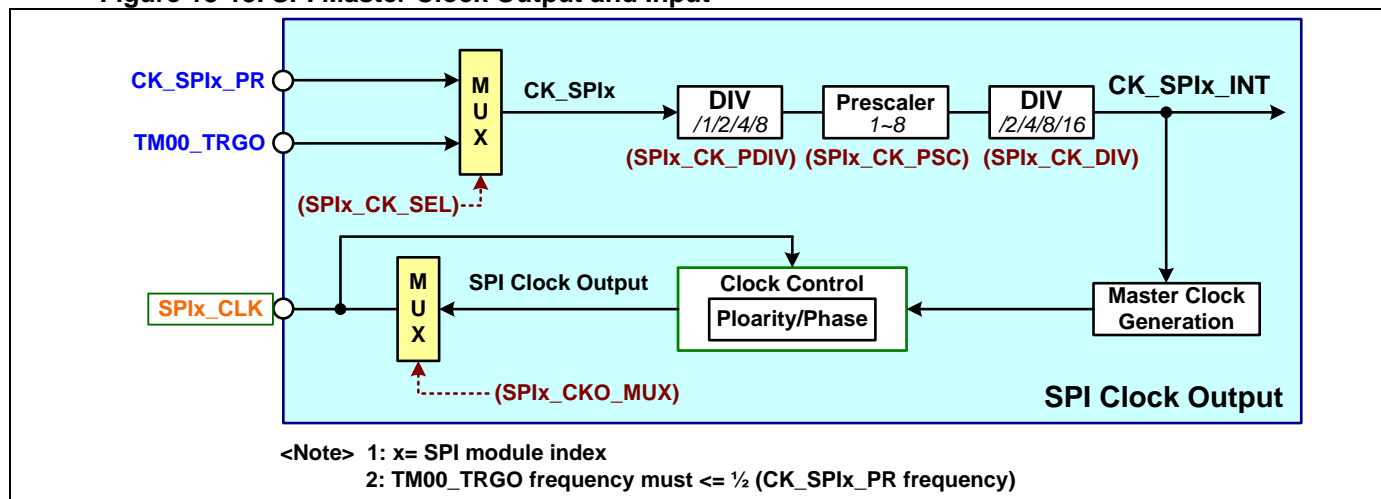
Clock Mode	SPI Register		Function Description
	SPIx_CPOL	SPIx_CPHA	
0	0	0	SPI clock low level in idle state and data sampling on leading edge.
1	0	1	SPI clock low level in idle state and data sampling on trailing edge.
2	1	0	SPI clock high level in idle state and data sampling on leading edge.
3	1	1	SPI clock high level in idle state and data sampling on trailing edge.

● SPI Clock Setting

The SPI clock frequency is divided from the module input clock CK_APB, CK_AHB or TM00_TRGO and set by **SPIx_CK_PDIV**, **SPIx_CK_PSC** and **SPIx_CK_DIV** registers. By design, the SPI clock frequency is up to 1/2 module input clock frequency for SPI master mode and is up to 1/4 module input clock frequency for SPI slave mode.

The following figure is showing the SPI master clock output and Input.

Figure 18-13. SPI Master Clock Output and Input



● SPI Clock Toggle

Usually the SPI clock is generated and toggling by hardware control for SPI master mode. By application request, user can directly toggle the SPI clock output signal **SPIx_CLK** by register control in **SPIx_CKO_TOG** register. When it enables, the **SPIx_CLK** signal will be toggled from low to high or high to low. This register is set by software and cleared by hardware.

18.10.2. SPI Transmit

When detect the TXF (**SPIx_TXF**) flag, user can write the transmission data to the TX data register (**SPIx_TDAT**) and the chip will automatically clear the TXF flag. After the time, User can write the next transmission data to the same TX data register when detects the TXF flag again. Repeat the sequence until the data transmission complete.

When the SPI is running in half duplex bi-direction communication mode, the TCF flag (**SPIx_TCF**) can denote that the previous data transmission is complete. User can change the data transfer direction to receiving by setting **SPIx_RX_EN** register and change the data line(s) to Hi-Z or GPIO data latch state by setting **SPIx_TX_DIS** register.

User can select the data output tristate or driving mode during idle state by setting **SPIx_DOUT_MDS** register for SPI master standard full duplex communication mode. When the register is disabled and data transfers during idle state, the **SPIx_MOSI** will output with tristate for master mode. When the register is enabled and data transfers during idle state, the **SPIx_MOSI** will output with driving for master mode. When the **SPIx_DOUT_MDS** register is selected 'driving', there are three states of 'Output 0', 'Output 1' and 'Last data bit' those user can select in **SPIx_DOUT_IDL** register.

By design, the SPI data must be updated to TX shift buffer before the previous clock edge of the first sampling clock edge of a frame data for SPI slave data transmission mode. One data updated control option is used to enable the SPI slave mode transmitted data directly update in **SPIx_TXUPD_EN** register. User can enable this register and the SPI data can be delayed updated to TX shift buffer before the first sampling clock edge of a frame data.

18.10.3. SPI Receive

As same as data transmission, user can read the received data from the RX data register (**SPIx_RDAT**) when detect the RXF flag (**SPIx_RXF**) and the chip will automatically clear the RXF flag. By same way, User can read the received data when every time detects the RXF flag. Repeat the sequence until the data receiving complete.

The **SPIx_RX_TH** register is used to set the data byte threshold of shadow buffer. When the data byte number of shadow buffer is equal to the threshold, hardware will copy the shadow buffer content to data register.

For last tail data control, user can check the RXDF flag (**SPIx_RXDF**) and **SPIx_RNUM** register. RXDF flag indicates received data byte number is different from previous received data byte number. **SPIx_RNUM** register indicates received data byte number when data shadow buffer last transfer to **SPIx_RDAT** register. Firmware can write an initial value for received byte number comparison.

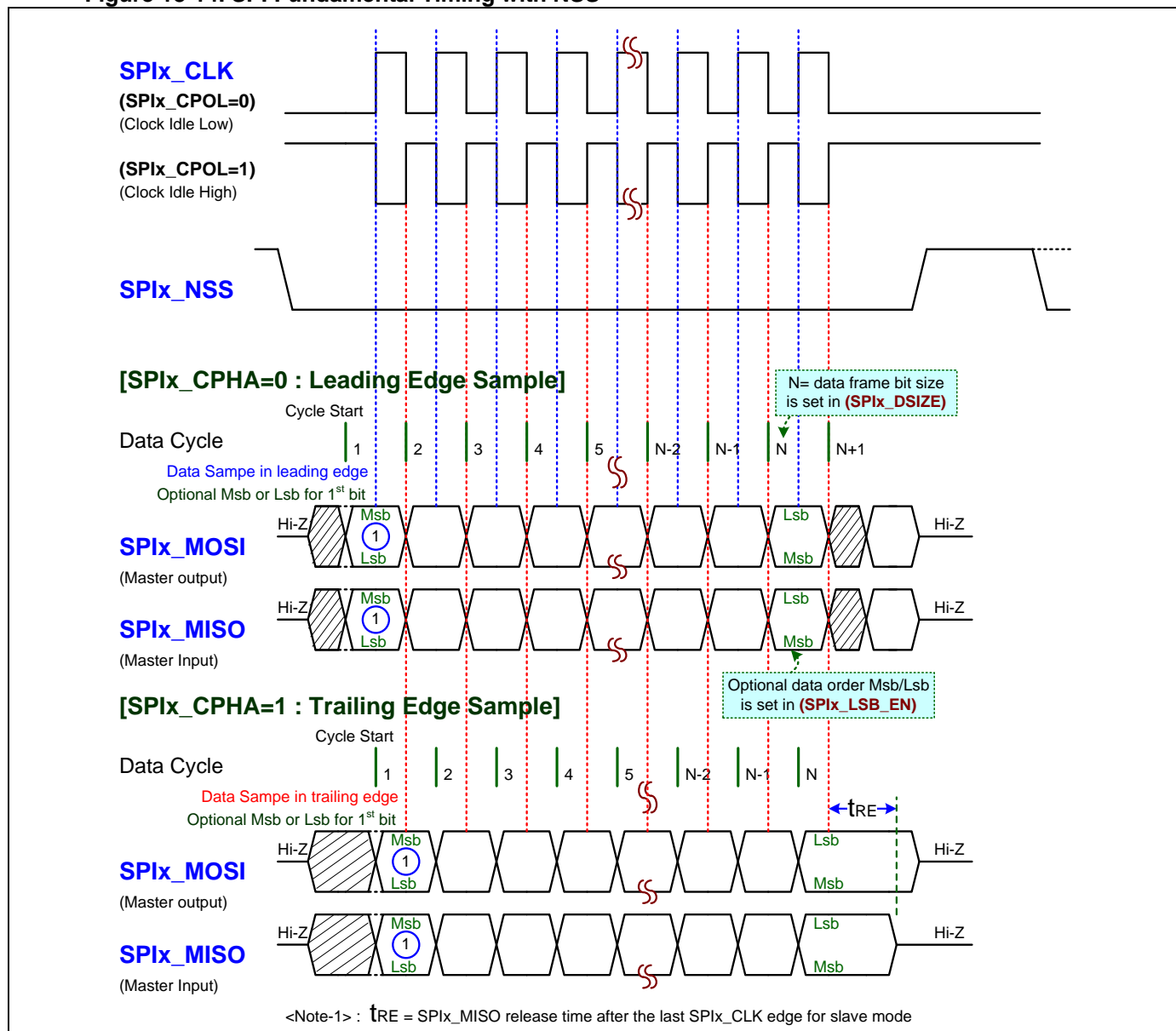
When the receiving shadow buffer has remaining content through a period bus idle after last data receiving for SPI salve mode, user can trigger to upload the aging data of shadow buffer to data register **SPIx_RDAT** and read it by setting **SPIx_RSB_TRG** register.

18.10.4. SPI Fundamental Timing

● SPI Control with NSS

The following diagram is showing the SPI fundamental timing for clock mode 0~3 with NSS signal. The data signals of **MOSI** and **MISO** are Hi-Z state for output condition during **NSS** inactive cycle.

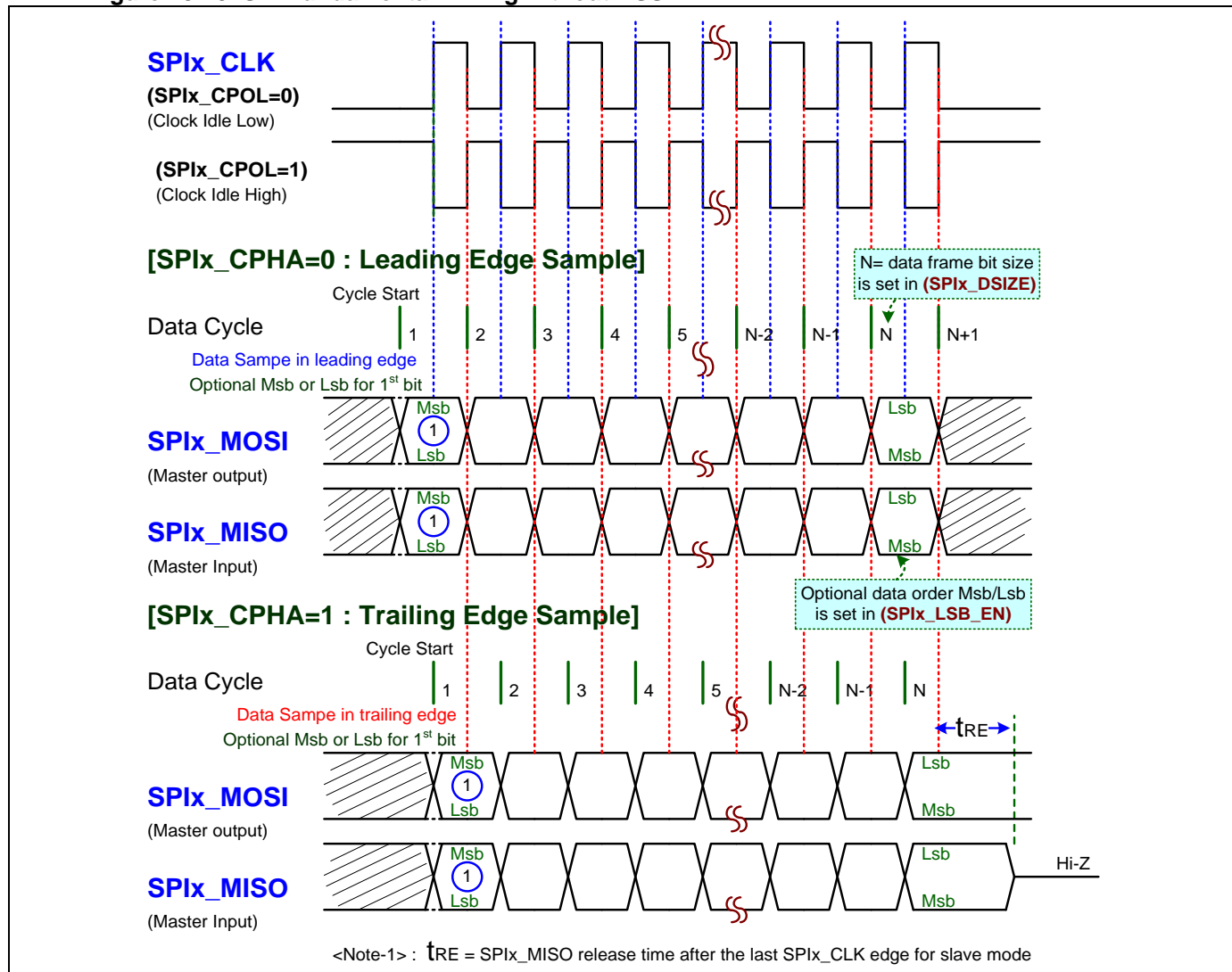
Figure 18-14. SPI Fundamental Timing with NSS



● SPI Control without NSS

The following diagram is showing the SPI fundamental timing for clock mode 0~3 without NSS signal. When does not use the NSS signal, the data signals of **MOSI** and **MISO** are unknown state or Hi-Z state during **NSS** inactive cycle.

Figure 18-15. SPI Fundamental Timing without NSS

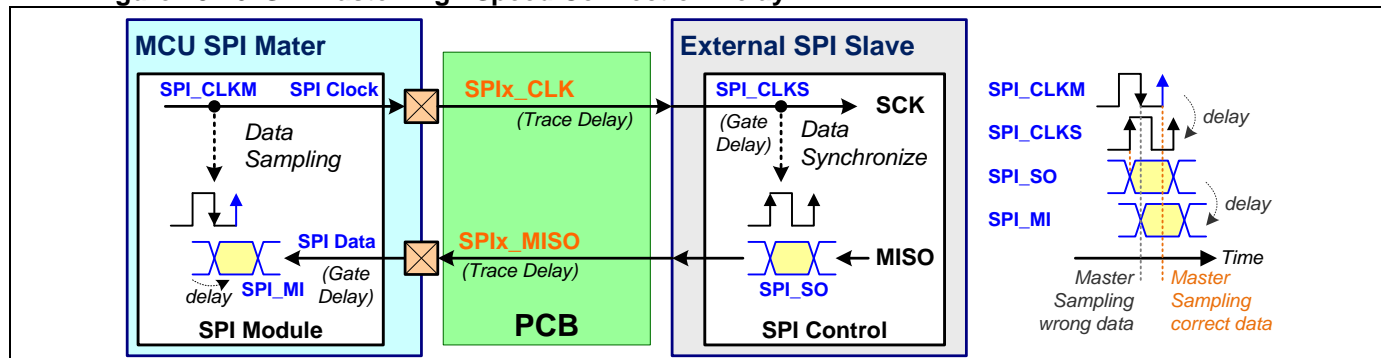


18.10.5. SPI High Speed Operation

For SPI master mode, user can select the data receive sampling edge in **SPIx_RX_CTL** register by actual system signal timing. When selects 'Normal', the SPI data sampling on leading edge or trailing edge of SPI clock is set in SPI0_CPHA register. When selects 'Next', the SPI data sampling at the next edge of the selected clock edge which is set in SPI0_CPHA register.

The following diagram is showing the SPI connection and clock/data trace path on application PCB. When the SPI slave output data is delayed over 1/2 SPI clock cycle time at the SPI master input, the SPI master must delayed 1/2 SPI clock cycle time to sample the correct input data.

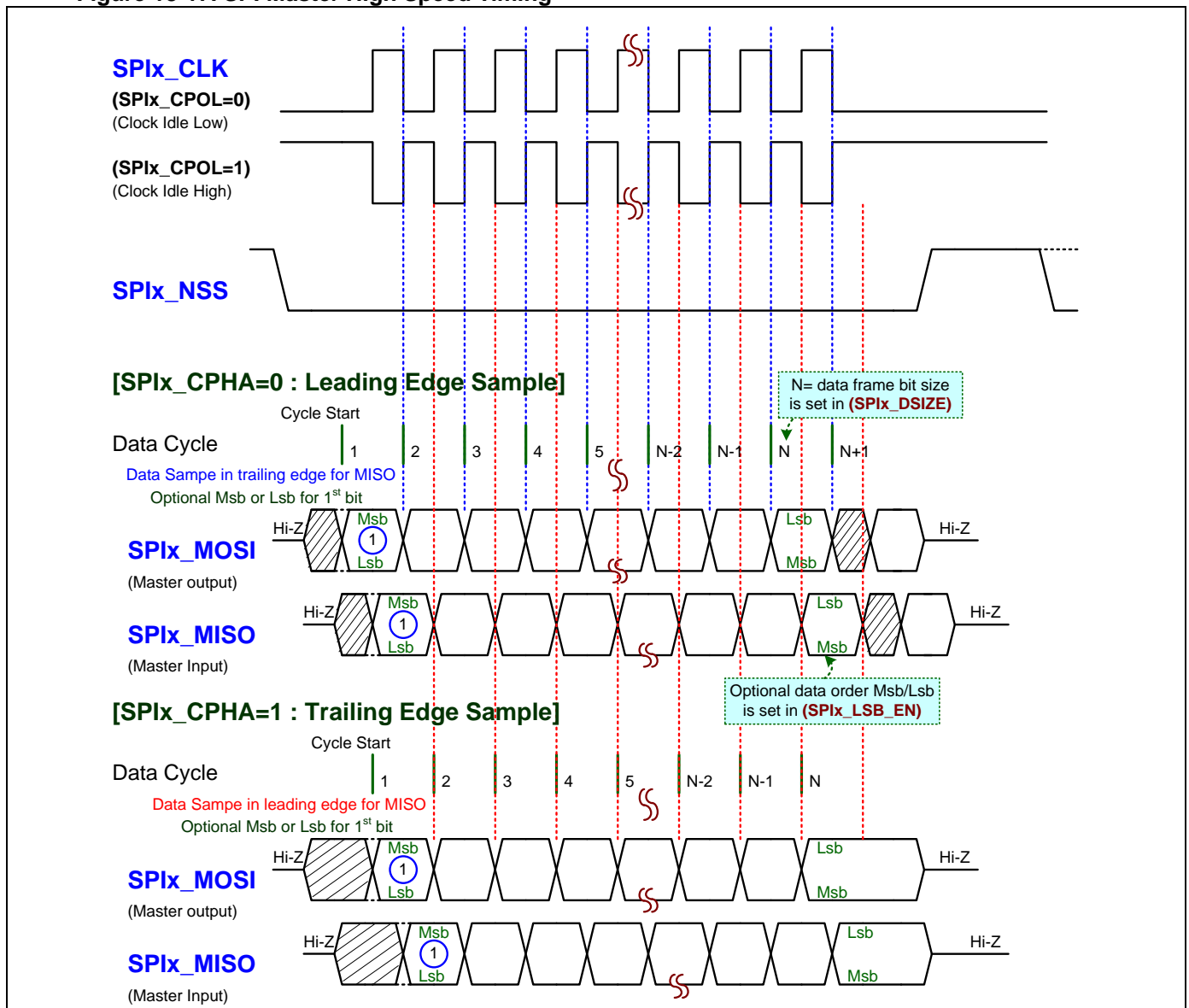
Figure 18-16. SPI Master High Speed Connection Delay



When the SPI master application is with high speed operation or long trace delay time, the SPI receiving data signal (**SPIx_MISO**) is possible to be delayed over 0.5T SPI clock to MCU SPI output clock (**SPIx_CLK**) as following diagram. The external SPI slave inputs the SPI clock which is delayed from MCU SPI master by PCB trace routing and chip gate delay. Again the external SPI slave output the SPI data which is delayed to MCU SPI master by PCB trace routing and chip gate delay. When the total of two delayed time is over 0.5T SPI clock, it will make receiving wrong that the MCU SPI master samples the delayed SPI receiving data by internal SPI clock. For SPI master mode, user can set the SPI data sampling at the next edge of the selected clock edge to perform the correct SPI data transaction by setting **SPIx_RX_CTL** register.

The following diagram is showing the SPI master mode high speed or long delay time timing.

Figure 18-17. SPI Master High Speed Timing



For SPI slave mode, the SPI receiving data (**SPIx_MISO**) output timing is normal as same as the figure of “**SPI Fundamental Timing with NSS**” in the section of “SPI Fundamental Timing”. When the SPI slave application is with high speed operation and long trace delay time, it is also possible to be delayed over 0.5T SPI clock to external SPI master device by PCB trace routing and chip gate delay. At the input point of external SPI master device, the SPI receiving data (**SPIx_MISO**) input timing is like as “**SPI Master High Speed Timing**” diagram. So the external SPI master device must be sampling the input SPI data (**SPIx_MISO**) at the next edge of the normal selected clock edge to perform the correct SPI data transaction.

Others, user can enable the high speed function for SPI slave mode and raise the SPI frequency to 1/3 module input clock (**CK_SPIx**) frequency by setting **SPIx_HS_EN** register. When this register bit is enabled and the APB clock is running up to 48MHz, the maximum SPI clock frequency is 16MHz for slave mode.

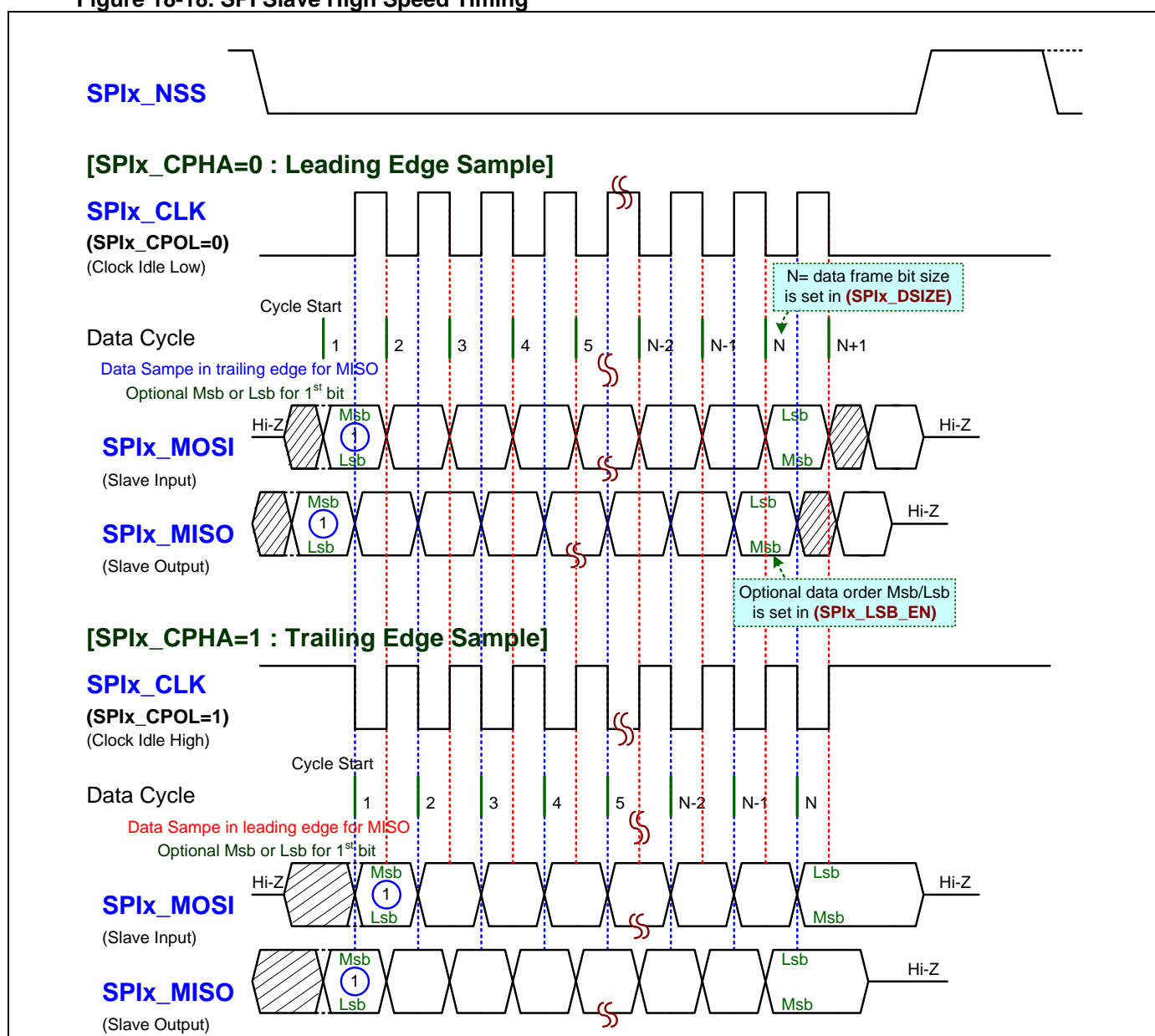
For SPI standard slave mode, there is one another high speed operation option to input clock and processes data transaction in asynchronous mode by setting **SPIx_ASYNC_EN** register. For other SPI slave modes, the **SPIx_ASYNC_EN** register must be set to disable. When this bit is enabled, the SPI shift buffer clock is directly used the SPI clock (**SPIx_CLK**) input. When this bit is disabled, the SPI clock input is synchronized by internal clock. When this function is enabled, it can raise the maximum SPI clock frequency up to 24MHz for SPI for slave mode.

In the asynchronous slave mode, the chip can support to output the SPI data at the previous half-clock edge of the selected clock edge which is set in **SPIx_CPHA** register by setting **SPIx_TX_CTL** register. It is useful for high speed operation or long trace delay time on SPI slave application. The SPI receiving data signal is possible to be delayed over 0.5T SPI clock to MCU SPI output clock as “**SPI Master High Speed Timing**”. Also the external SPI master device must be sampling the input SPI data at the next edge of the normal selected clock edge to perform the correct SPI data transaction. But if the external SPI master device cannot support the sampling delay at the next edge function, user can set the chip to output the SPI data at one advanced half-clock time to perform the correct SPI data transaction by setting **SPIx_TX_CTL** register.

[Notify]: The **SPIx_TX_CTL** register is not supported for MG32F02A128/U128/A064/U064.

The following diagram is showing the SPI slave mode high speed or long delay time timing.

Figure 18-18. SPI Slave High Speed Timing

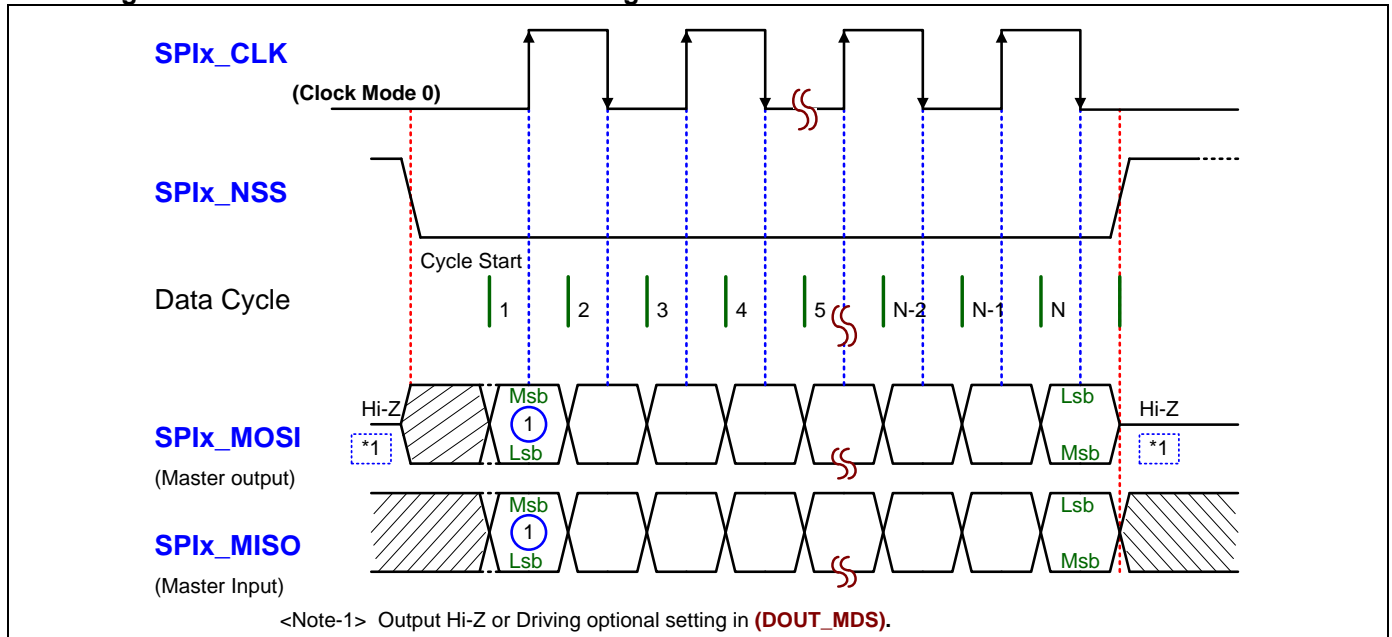


18.10.6. SPI DTR Mode

The SPI module can support the dual transfer rate (DTR) mode by enabling in **SPIx_DTR_EN** register for

SPI master clock mode-0 only. When it enables, the SPI data will transfer at both rising and falling edges of SPI clock.

Figure 18-19. SPI Master DTR Mode Timing



18.10.7. SPI NSS Mode

User can set the NSS control mode of **SPIx_NSS** or **SPIx_NSSI** signals by setting **SPIx_MDS**, **SPIx_NSSO_EN**, **SPIx_NSSI_EN** and **SPIx_NSSI_SEL** registers.

The SPI module is able to support the SPI communication with **NSS** signal or not by setting **SPIx_NSSO_EN** or **SPI0_NSSI_EN** registers. Also use can use software registers to control **NSS** signal. The **SPIx_NSS_SWEN** register is used to enable **SPIx_NSS** and **SPIx_NSSI** signal by software control.

● SPI NSS Control Setting

The following table is showing the register setting of SPI NSS mode control.

Table 18-4. SPI NSS Control Table

NSS Mode	SPI Register				Function Description
	MDS	NSSO_EN	NSSI_EN	NSSI_SEL	
Master	1	0	0	x	Master without NSS output
Master + NSS	1	1	0	x	Master with NSS output
Slave	0	x	0	x	Slave without NSS input
Slave + NSS	0	x	1	0	Slave with NSS input from SPIx_NSS pin
				1	Slave with NSS input from SPIx_NSSI pin
Master + MODF	1->0	0	1	0	Master without NSS output and with MODF function (using SPIx_NSS pin), changed Slave with NSS input
				1	Master without NSS output and with MODF function (using SPIx_NSSI pin), changed Slave with NSS input
Master + NSS +MODF	1->0	1	1	1	Master with NSS output and with MODF function (using NSSI pin), changed Slave with NSS input (Master NSS output to SPIx_NSS pin and NSS input(MODF) from SPIx_NSSI pin)
				0	Not Support

x : do not care

1->0 : Chip detect NSS or NSSI signal pull-low then change to Slave mode or disable SPI

● SPI Master Hardware NSS Control

When user uses the hardware control NSS function by setting **SPIx_NSS_SWEN** register, user can set the **SPI0_NSS_PEN** register to enable **NSS** pulse generation between two frame data transfer.

The following table is showing the SPI NSS timing table.

Table 18-5. SPI Master NSS Timing Table

Chip	Clock Mode	NSS Timing				Unit
		t _L	t _r	t _i	t _{PW}	
All Chips	0 (CPOL=0, CPHA=0)	1	0.5	0.5/2.5/3.5 (*1)	1 or 2 (*2)	t _{CLK}
	2 (CPOL=1, CPHA=0)					
	1 (CPOL=0, CPHA=1)	0.5	0.5	0.5/2/3 (*1)	1 or 2 (*2)	t _{CLK}
	3 (CPOL=1, CPHA=1)					

<Sign> t_{CLK} : SPI bit time

t_L : SPIx_NSS leading time before the first SPIx_CLK edge (Bit time)

t_r : SPIx_NSS trailing time after the last SPIx_CLK edge (Bit time)

t_i : SPIx_NSS idling time between transfers (Bit time), t_i = 0 if SPI0_NSS_PEN=0

t_{PW} : SPIx hardware NSS pulse width during idle cycle

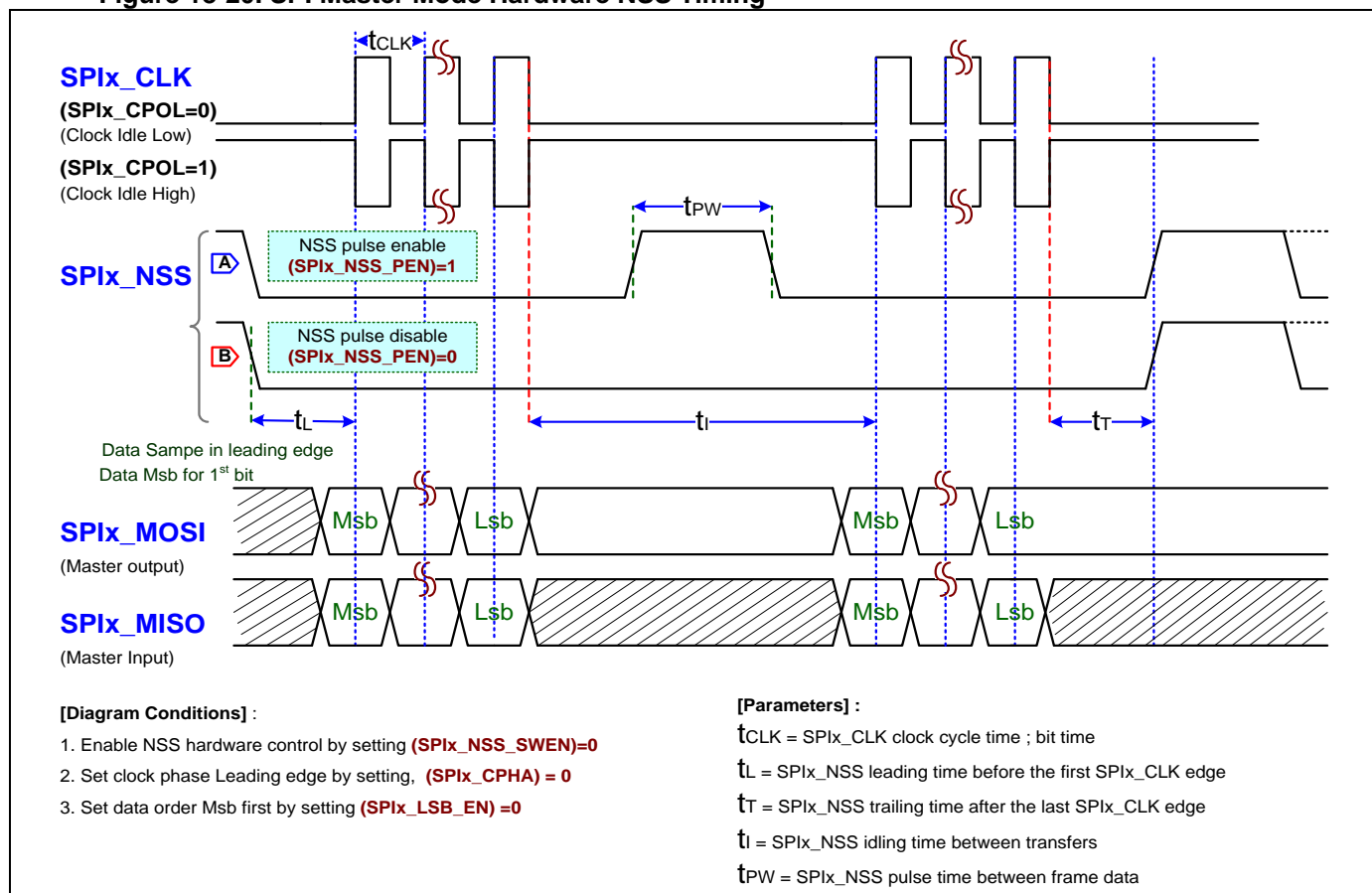
<Note> *1: t_i = 0 if SPIx_NSS_PEN=0; = 1 or 2 by SPIx_NSS_IDT setting if SPIx_NSS_PEN=1

*2: t_{PW} = 0 if SPIx_NSS_PEN=0; = 1 or 2 by SPIx_NSS_IDT setting if SPIx_NSS_PEN=1

When **SPI0_NSS_PEN** is enabled, user can set the **NSS** pulse width by setting **SPIx_NSS_IDT** register.

The following diagram is showing the hardware NSS timing diagram for SPI master mode.

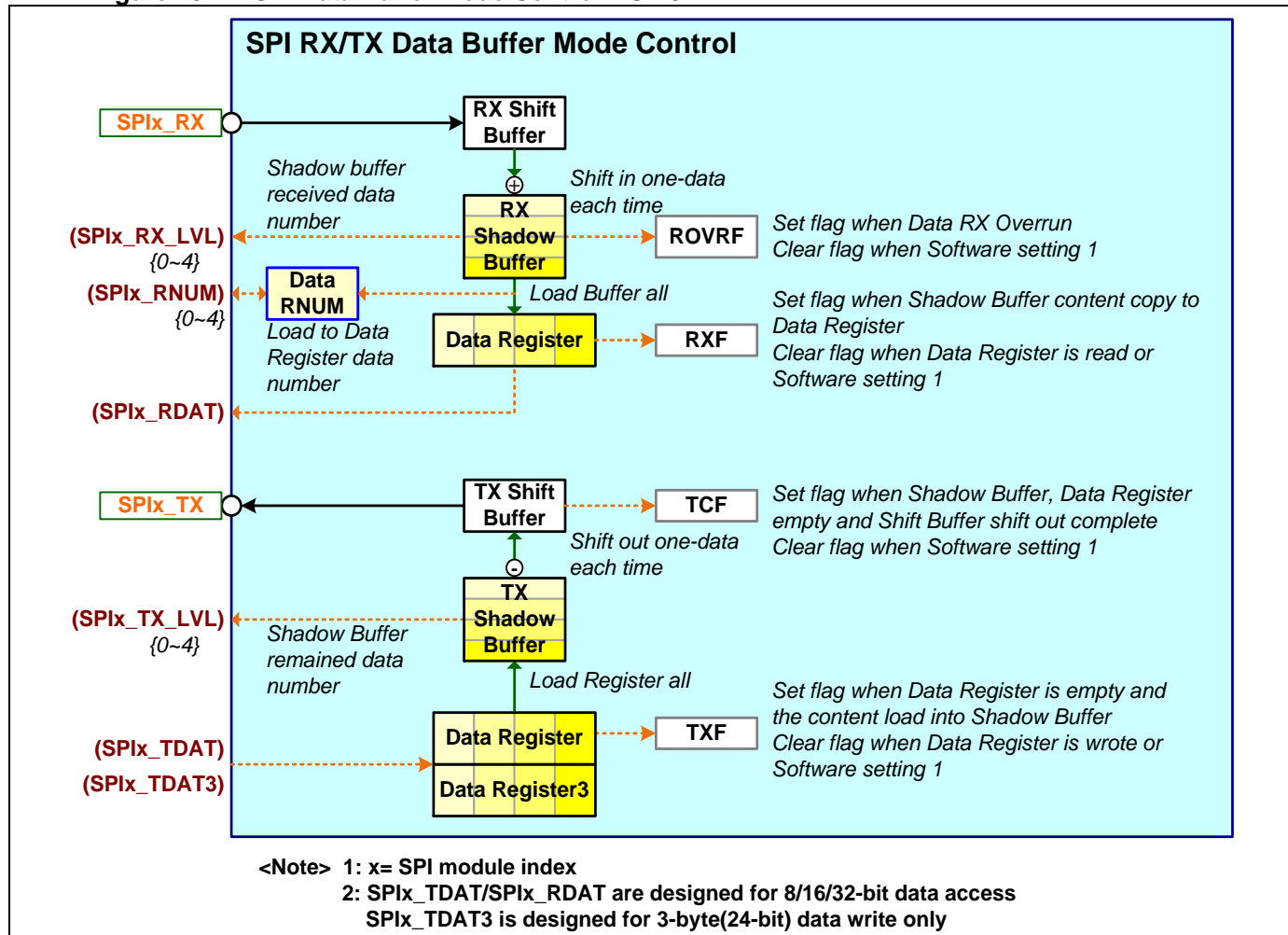
Figure 18-20. SPI Master Mode Hardware NSS Timing



18.11. SPI Data Buffer

The module implements two 32-bit shift buffers, two 32-bit shadow buffer and two 32-bit data register for data flow control and reduce the CPU overhead. User can easy use the event flags of TXF (**SPIx_TXF**) and RXF (**SPIx_RXF**) for data transfer flow control. The following diagram is showing the SPI Data Buffer mode control block.

Figure 18-21. SPI Data Buffer Mode Control – SPI0



18.11.1. SPI Data Buffer Control

The RXF flag (**SPIx_RXF**) will be activated at every time that the RX Shadow Buffer content is copied to RX data register. Also the related interrupt will be asserted if the **SPIx_RX_IE** register is enabled.

The TXF flag (**SPIx_TXF**) will be activated at every time that the TX data register content is copied to TX Shadow Buffer. Also the related interrupt will be asserted if the **SPIx_TX_IE** register is enabled. When both Shadow Buffer and TX data register are empty, the TCF flag (**SPIx_TCF**) will be activated if the Shift Buffer shifts out completely.

The data registers of **SPIx_TDAT** and **SPIx_RDAT** are designed for 8/16/32-bit data access and the register of **SPIx_TDAT3** is designed for 3-byte (24-bit) data write only. When user write one any of 8/16/32-bit data to the **SPIx_TDAT3** register, the chip will do as a 3-byte (24-bit) data to be written into.

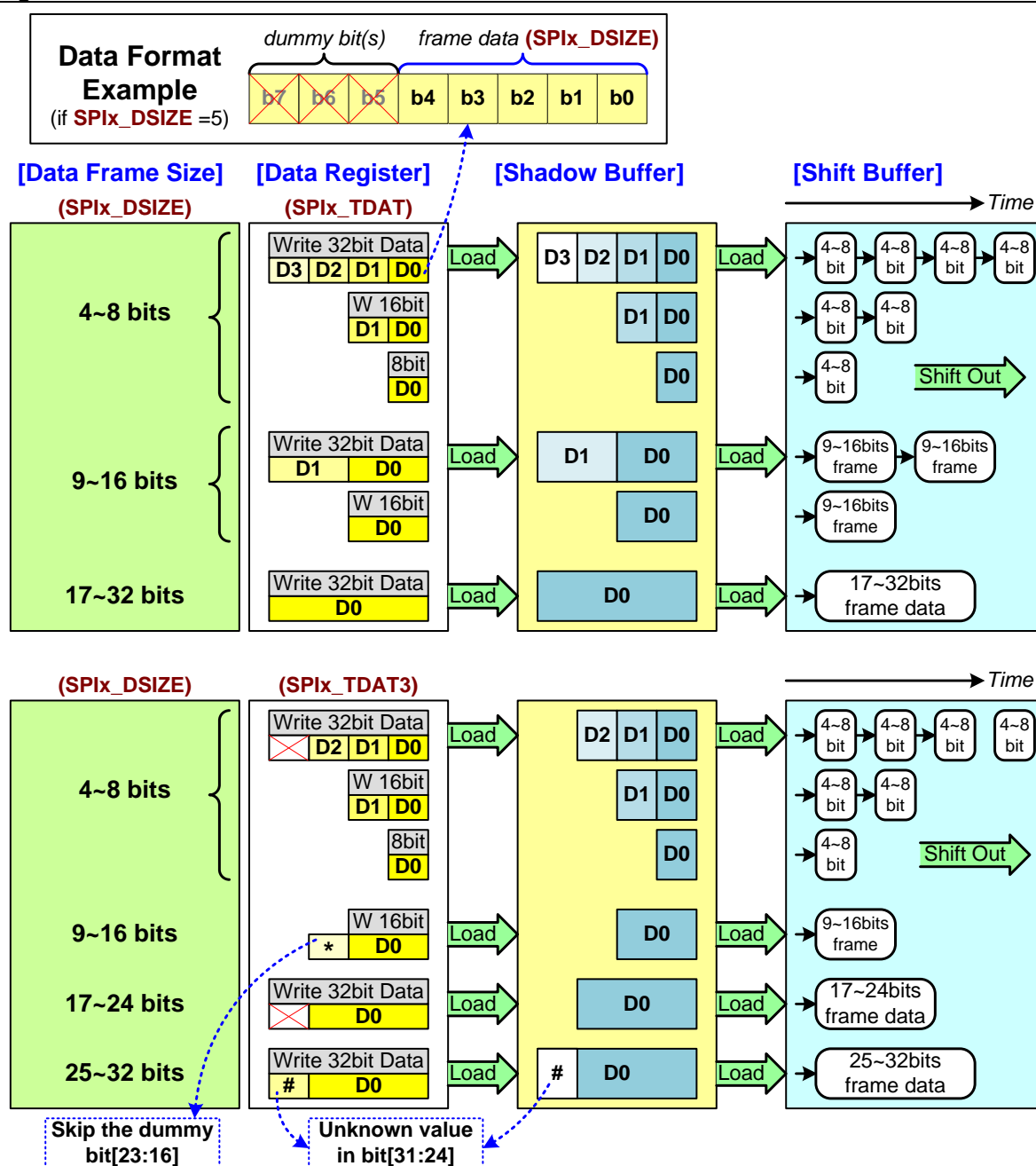
18.11.2. SPI Data Frame

User can set the data frame bit size from 4-bit to 32-bit by setting **SPIx_DSIZE** register. Also user can set **SPIx_LSB_EN** register to configure the frame data order by Lsb first or Msb first.

18.11.3. SPI Transmitted Data Frame Size

The following diagram is showing the SPI transmitted data control block for the data registers of **SPIx_TDAT** and **SPIx_TDAT3** by different frame size and written data bits.

Figure 18-22. SPI Transmitted Data Frame Size Control



- **Write 8-bit data**

When writes an 8-bit data to the data register of **SPIx_TDAT** or **SPIx_TDAT3**:

- **Frame size 4~8 bits**

The chip will copy the data to shadow buffer and send 1-times of the frame size data to the shift buffer for data transmission.

— **Frame size 9~32 bits**

The chip will copy the data with dummy data bit[31:8] to shadow buffer and send 1-times of the frame size data to the shift buffer for data transmission.

● **Write 16-bit data**

When writes a 16-bit data to the data register of **SPIx_TDAT** or **SPIx_TDAT3**:

— **Frame size 4~8 bits**

The chip will copy the data to shadow buffer and separate the data to send 2-times of the frame size data to the shift buffer for data transmission.

— **Frame size 9~16 bits**

The chip will copy the data to shadow buffer and send 1-times of the frame size data to the shift buffer for data transmission.

— **Frame size 17~32 bits**

The chip will copy the data with dummy data bit[31:16] to shadow buffer and send 1-times of the frame size data to the shift buffer for data transmission.

● **Write 32-bit data**

(1) When writes a 32-bit data to the data register of **SPIx_TDAT**:

— **Frame size 4~8 bits**

The chip will copy the data to shadow buffer and separate the data to send 4-times of the frame size data to the shift buffer for data transmission.

— **Frame size 9~16 bits**

The chip will copy the data to shadow buffer and separate the data to send 2-times of the frame size data to the shift buffer for data transmission.

— **Frame size 17~32 bits**

The chip will copy the data to shadow buffer and send 1-times of the frame size data to the shift buffer for data transmission.

(2) When writes a 32-bit data to the data register of **SPIx_TDAT3**:

— **Frame size 4~8 bits**

The chip will copy the data to shadow buffer and separate the data to send 3-times of the frame size data to the shift buffer for data transmission.

— **Frame size 9~24 bits**

The chip will copy the data to shadow buffer and send 1-times of the frame size data to the shift buffer for data transmission.

— **Frame size 25~32 bits**

The chip will copy the data with dummy data bit[31:24] to shadow buffer and send 1-times of the frame size data to the shift buffer for data transmission.

18.11.4. SPI Received Data Frame Size

● **Read 8-bit data**

When reads an 8-bit data from the data register of **SPIx_RDAT**:

— **Frame size 4~8 bits**

The chip will get 1-times of the frame size data from the shift buffer to shadow buffer and copy the shadow buffer data to data register for data receiving.

— **Frame size 9~32 bits**

The chip will get 1-times of the frame size data from the shift buffer to shadow buffer and copy the shadow buffer data with dummy data bit[31:8] to data register for data receiving.

● **Read 16-bit data**

When reads a 16-bit data from the data register of **SPIx_RDAT**:

— **Frame size 4~8 bits**

The chip will get 2-times of the frame size data from the shift buffer to shadow buffer and copy the shadow buffer data to data register for data receiving.

— **Frame size 9~16 bits**

The chip will get 1-times of the frame size data from the shift buffer to shadow buffer and copy the shadow buffer data to data register for data receiving.

— Frame size 17~32 bits

The chip will get 1-times of the frame size data from the shift buffer to shadow buffer and copy the shadow buffer data with dummy data bit[31:16] to data register for data receiving.

● Read 32-bit data

When reads a 32-bit data from the data register of **SPIx_RDAT**:

— Frame size 4~8 bits

The chip will get 4-times of the frame size data from the shift buffer to shadow buffer and copy the shadow buffer data to data register for data receiving.

— Frame size 9~16 bits

The chip will get 2-times of the frame size data from the shift buffer to shadow buffer and copy the shadow buffer data to data register for data receiving.

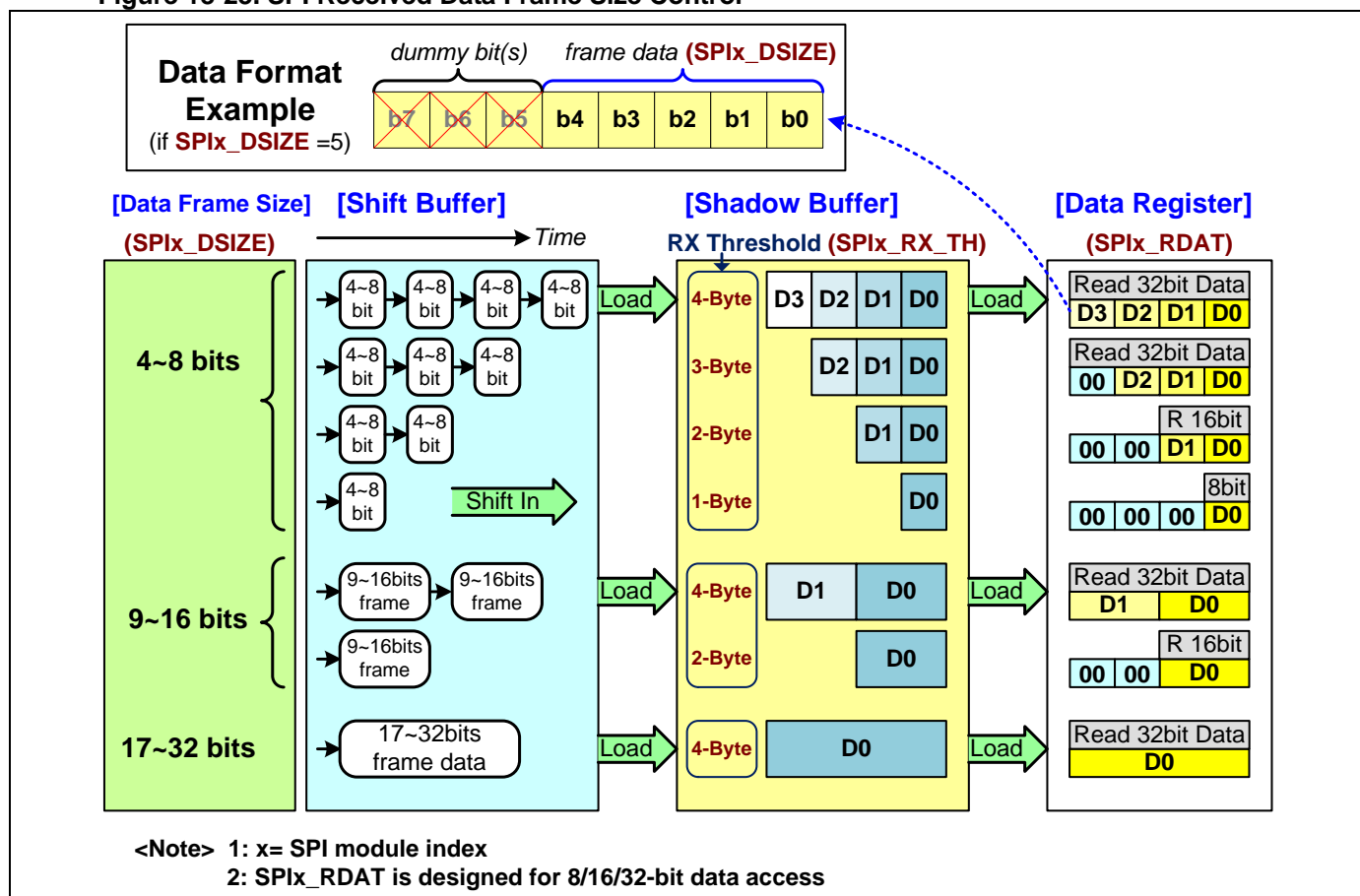
— Frame size 17~32 bits

The chip will get 1-times of the frame size data from the shift buffer to shadow buffer and copy the shadow buffer data to data register for data receiving.

[Notify]: The dummy data bits will clear to 0 in data register.

The following diagram is showing the SPI received data control block for the data register of **SPIx_RDAT** by different frame size and the data byte threshold of shadow buffer (**SPIx_RX_TH** register).

Figure 18-23. SPI Received Data Frame Size Control



18.11.5. SPI Data Buffer Clear

For firmware data control, user can force to clear the received or transmitted data buffer by setting **SPIx_RDAT_CLR** or **SPIx_TDAT_CLR** registers. These two register bit are set by software and clear by hardware.

When enables the register bit of **SPIx_RDAT_CLR**, the received data register and shadow buffer will be flushed. Also **SPIx_RXF** flag and **SPIx_RX_LVL** is cleared. When enables the register bit of **SPIx_TDAT_CLR**, the transmitted data register and shadow buffer will be flushed. Also **SPIx_TXF** flag is set and **SPIx_TX_LVL** is cleared.

18.12. SPI Data Modes

The SPI module provides several data modes and can be configured to one of the modes of standard SPI, 1-Line SPI, 2-Line SPI, 4-Line SPI, two duplicated 4-Line SPI or 8-Line SPI for flexible SPI application. Refer the [“SPI Connection for Application”](#) section for more information. The 2-Line SPI is only supported for master mode.

User can configure the data mode by setting the registers of **SPIx_DAT_LINE**, **SPIx_BDIR_OE**, **SPIx_MDS** and **SPIx_COPY_EN**. For SPI slave with NSS mode, user can enable hardware automatically to change to standard full duplex data mode before start data transaction by setting **SPIx_ADPX_EN** register. When **SPIx_NSSI_EN** is disabled, this function is no effect. When this function is enabled and NSS input is changed from inactive to active, the **SPIx_DAT_LINE** will auto be forced to 0 and change to full duplex standard SPI mode.

The following table is showing the register setting and IO pin(s) using for SPI data mode control.

Table 18-6. SPI Data Control Table

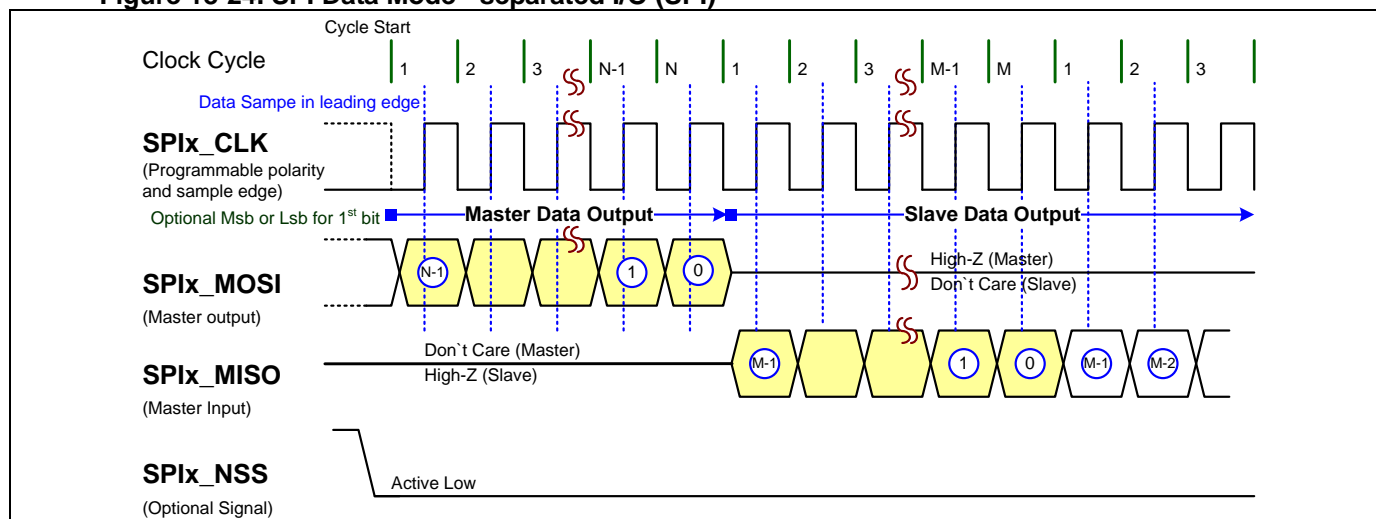
Data Mode	DPX/HPX	SPIx Register				I/O Pins				Function Descriptions
		DAT_LINE	BDIR_OE	MDS	COPY_EN	MOSI/D0	MISO/D1	D2,D3	D4~D7	
SPI	DPX	0	x	0	x	Input	Output			2 data line full-duplex communication
		0	x	1	x	Output	Input			
SPI-1 (1-Line)	HPX In	1	0	x	x	Input				1 data line half-duplex input
	HPX Out	1	1	x	x	Output				1 data line half-duplex output
SPI-2 (2-Line)	HPX In	2	0	x	0	Input	Input			2 spread data lines input
	HPX Out	2	1	x	0	Output	Output			2 spread data lines output
SPI-2C (2-Line)	HPX In	2	0	x	1	Input	Input			2 spread data lines input
	HPX Out	2	1	x	1	Output	Output			2 copied data lines output
SPI-4 (4-Line)	HPX In	3	0	x	0	Input	Input	Input		4 spread data lines input
	HPX Out	3	1	x	0	Output	Output	Output		4 spread data lines output
SPI-4C (4-Line)	HPX In	3	0	x	1	Input	Input	Input		4 spread data lines input
	HPX Out	3	1	x	1	Output	Output	Output		4 copied data lines output
SPI-4D (8-Line)	HPX In	4	0	x	0	Input	Input	Input	Input	4 spread data lines input
	HPX Out	4	1	x	0	Output	Output	Output	Output	Two duplicated 4-lines output
SPI-8 (8-Line)	HPX In	5	0	x	x	Input	Input	Input	Input	8 spread data lines input
	HPX Out	5	1	x	x	Output	Output	Output	Output	8 spread data lines output

<Note> x = 0 or 1 ; DPX/HPX = Full/Half Duplex Communication

18.12.1. SPI Data Mode – SPI

This Data Mode is able to transfer data for full duplex communication. There are two separated data signals of **SPIx_MOSI** (**SPIx_D0**) and **SPIx_MISO** (**SPIx_D1**) for SPI transmission and receiving.

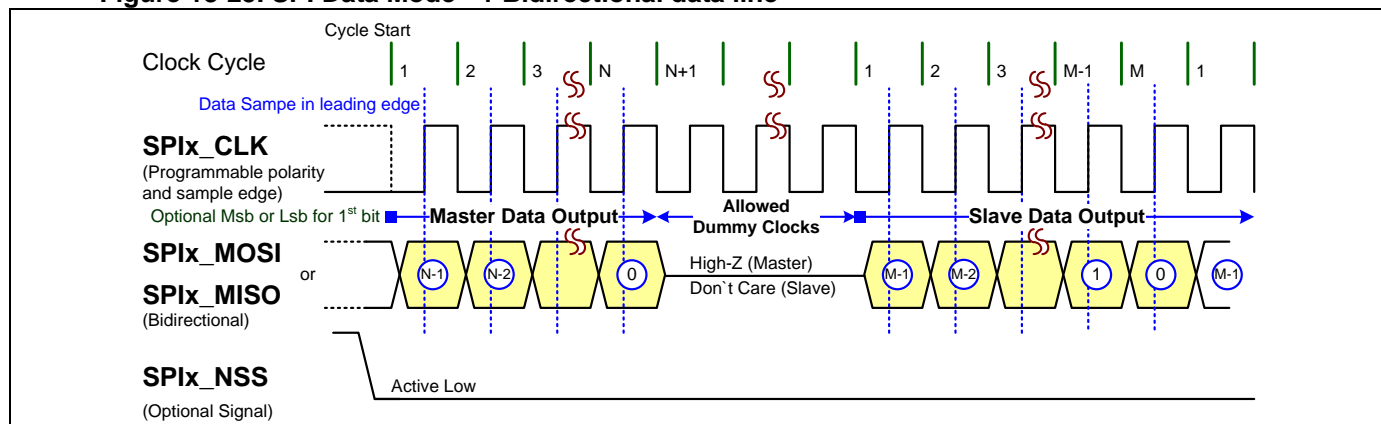
Figure 18-24. SPI Data Mode - separated I/O (SPI)



18.12.2. SPI Data Mode – SPI-1

This Data Mode is able to transfer data only for half duplex communication. There is only one bidirectional data signal of **SPIx_MOSI** (**SPIx_D0**) or **SPIx_MISO** (**SPIx_D1**) for SPI both transmission and receiving.

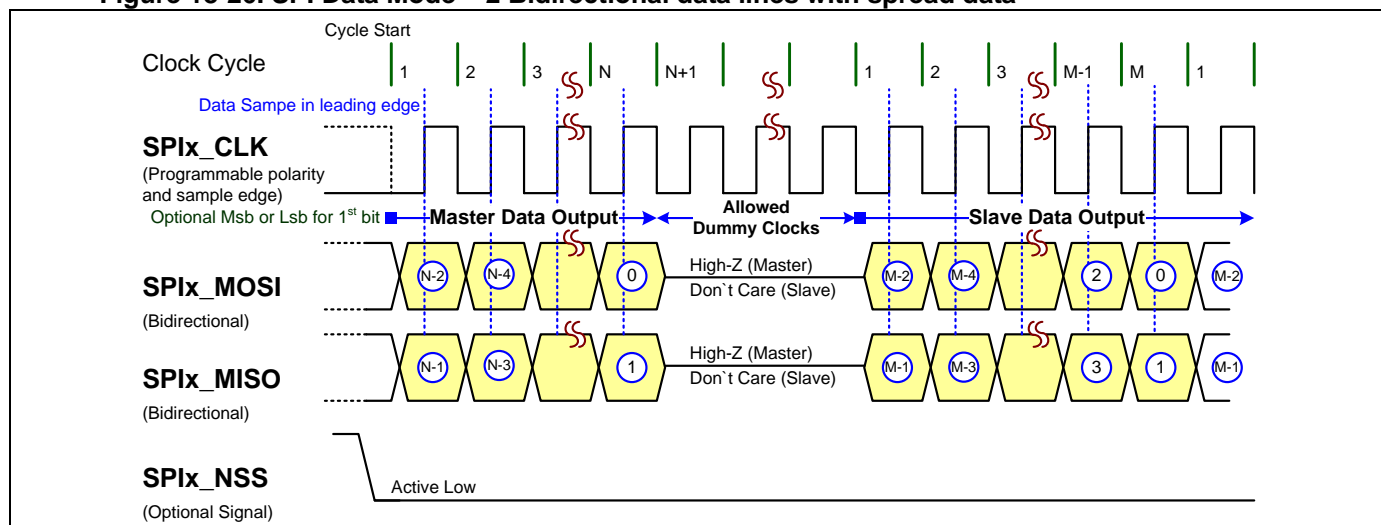
Figure 18-25. SPI Data Mode - 1 Bidirectional data line



18.12.3. SPI Data Mode – SPI-2

This Data Mode is able to transfer data only for half duplex communication. There are two bidirectional data signals of **SPIx_MOSI** (**SPIx_D0**) and **SPIx_MISO** (**SPIx_D1**) for SPI both transmission and receiving.

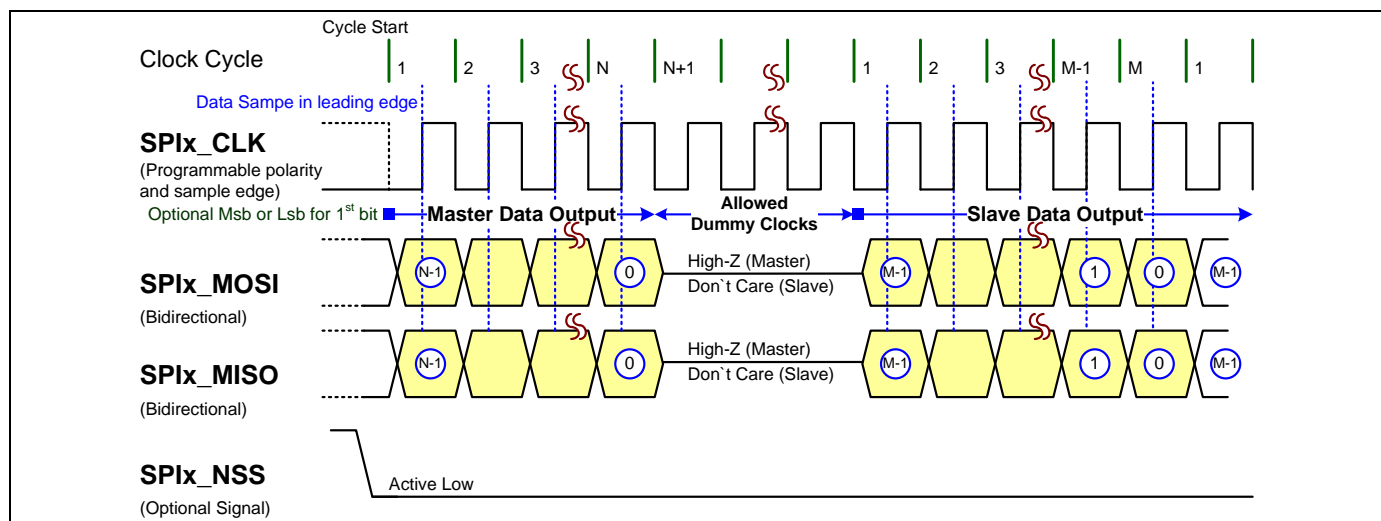
Figure 18-26. SPI Data Mode – 2 Bidirectional data lines with spread data



18.12.4. SPI Data Mode – SPI-2C

This Data Mode is able to transfer data only for half duplex communication. There are two bidirectional data signals of **SPIx_MOSI** (**SPIx_D0**) and **SPIx_MISO** (**SPIx_D1**) with copied data for SPI both transmission and receiving.

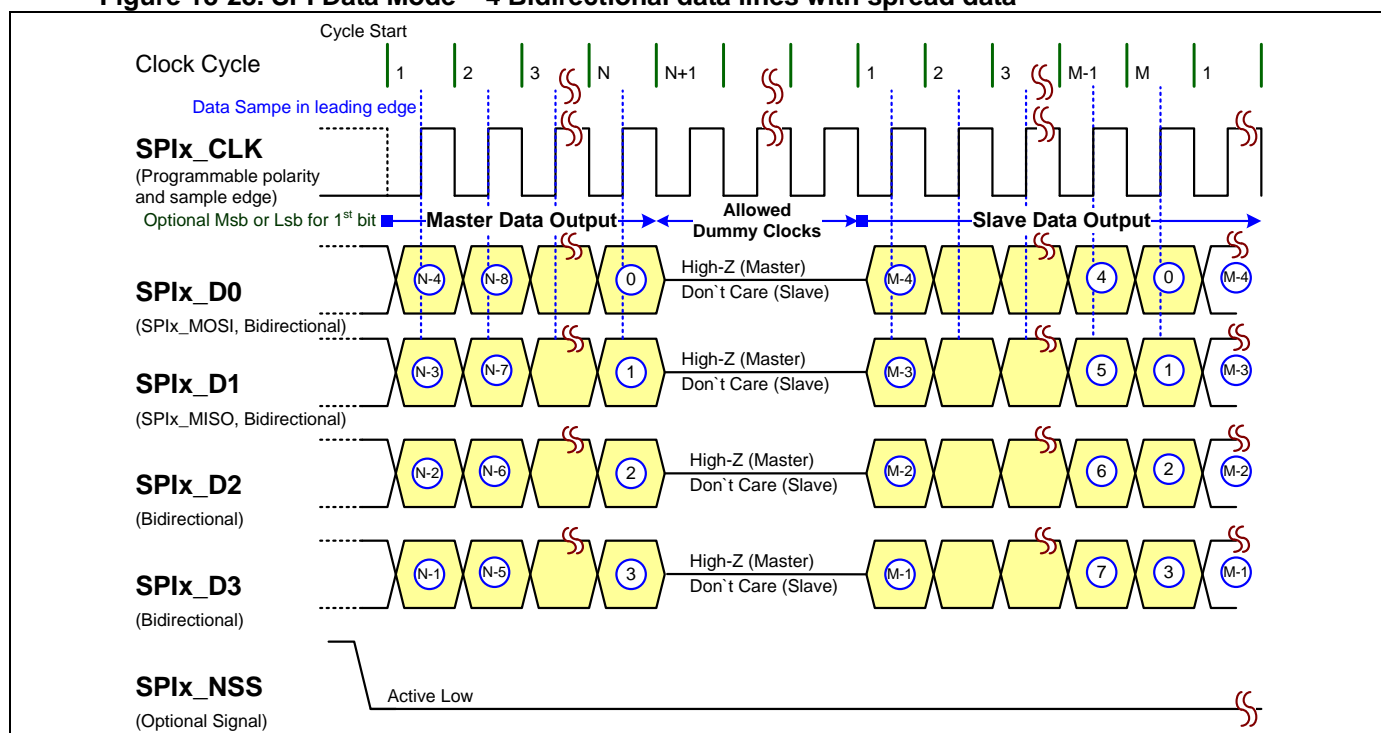
Figure 18-27. SPI Data Mode – 2 Bidirectional data lines with copied data



18.12.5. SPI Data Mode – SPI-4

This Data Mode is able to transfer data only for half duplex communication. There are four bidirectional data signals of **SPIx_D[3:0]** for SPI both transmission and receiving.

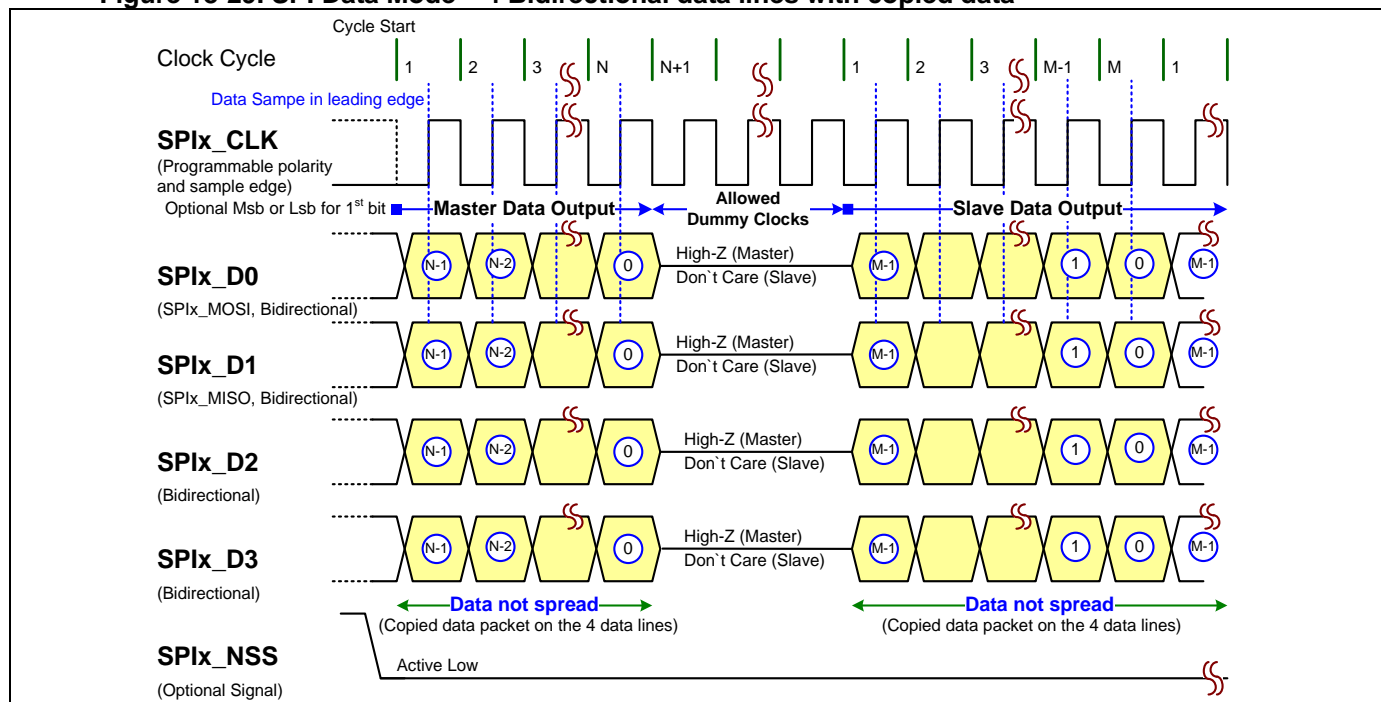
Figure 18-28. SPI Data Mode – 4 Bidirectional data lines with spread data



18.12.6. SPI Data Mode – SPI-4C

This Data Mode is able to transfer data only for half duplex communication. There are four bidirectional data signals of **SPIx_D[3:0]** with copied data for SPI both transmission and receiving.

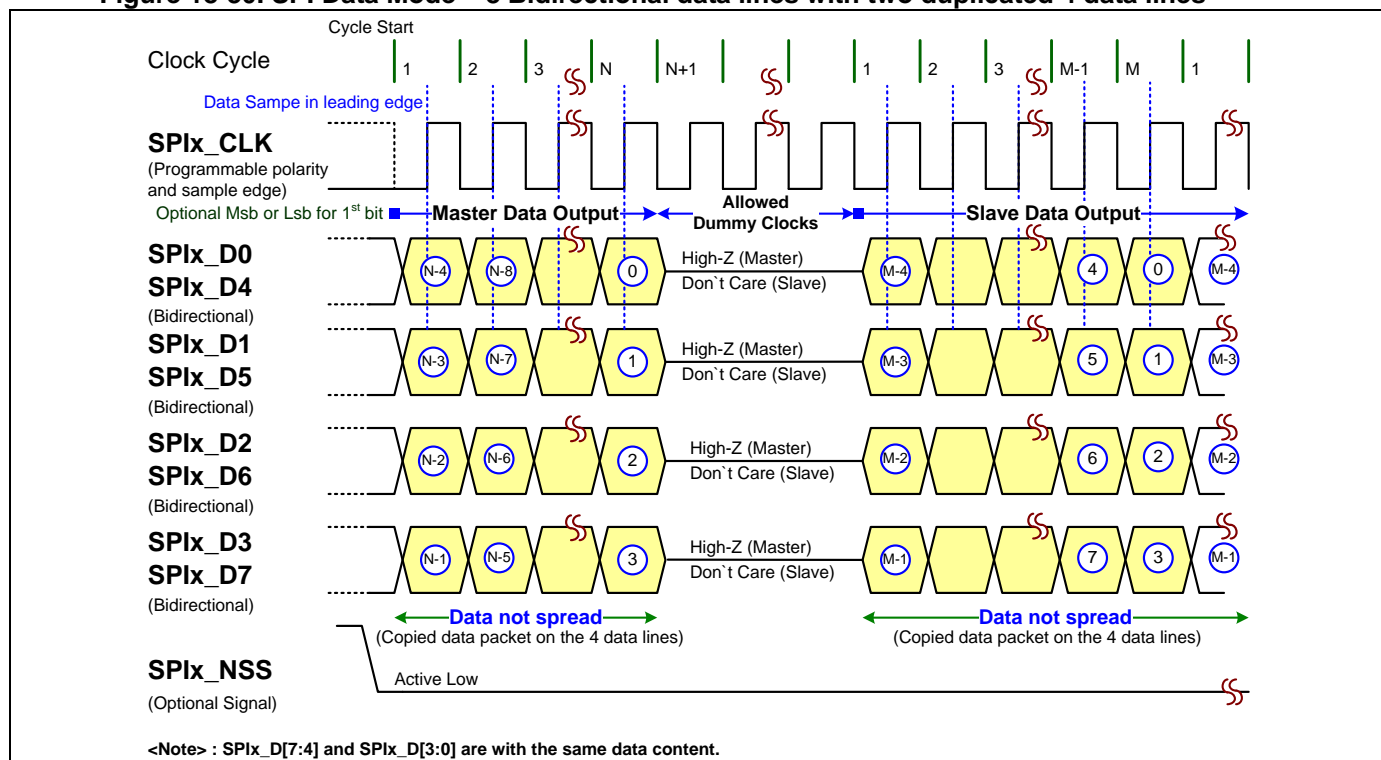
Figure 18-29. SPI Data Mode – 4 Bidirectional data lines with copied data



18.12.7. SPI Data Mode – SPI-4D

This Data Mode is able to transfer data only for half duplex communication. There are eight bidirectional data signals of **SPIx_D[7:0]** with two duplicated sets of data lines for SPI both transmission and receiving. The data signals of **SPIx_D[7:4]** and **SPIx_D[3:0]** are duplicated with the same data contents.

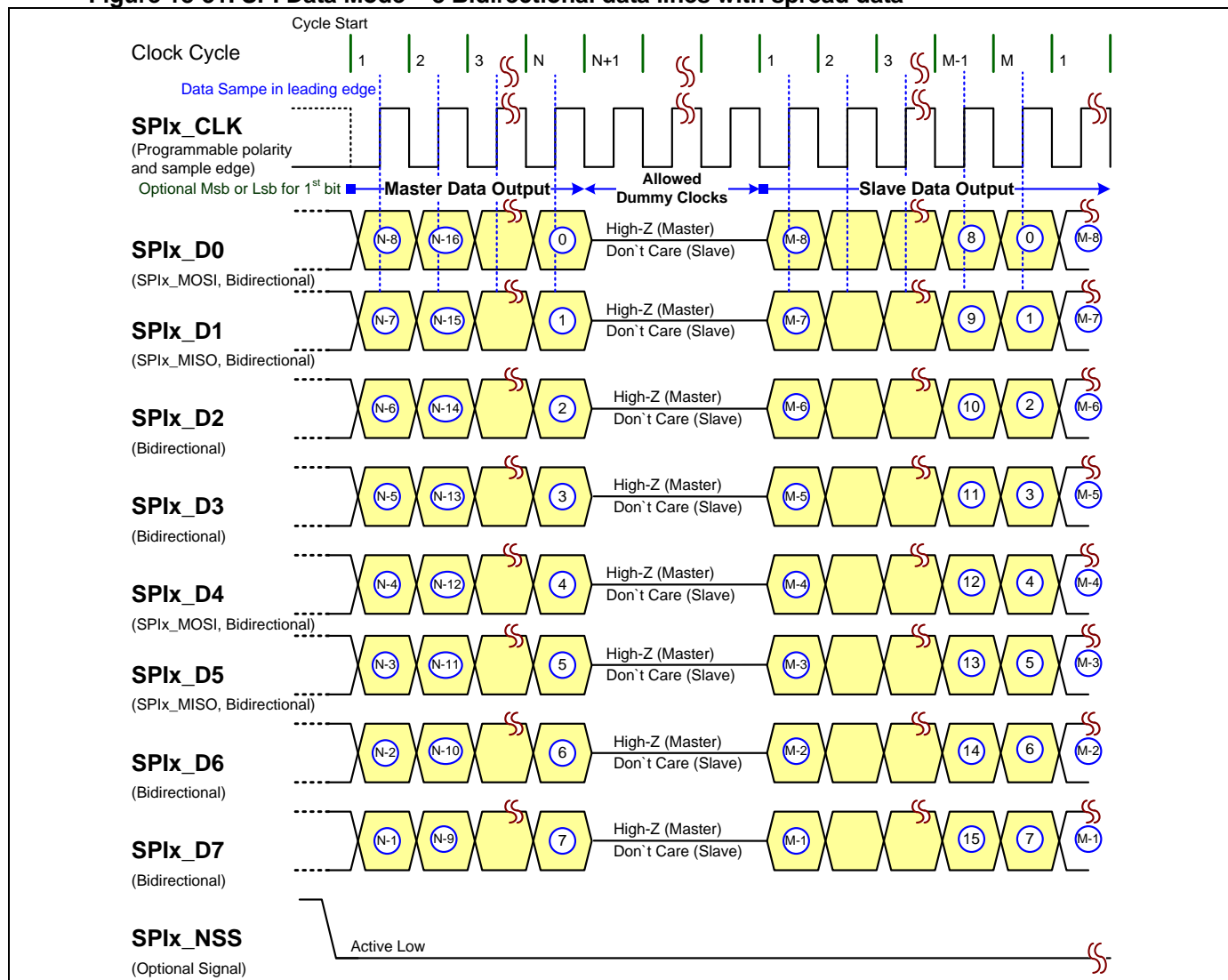
Figure 18-30. SPI Data Mode – 8 Bidirectional data lines with two duplicated 4 data lines



18.12.8. SPI Data Mode – SPI-8

This Data Mode is able to transfer data only for half duplex communication. There are eight bidirectional data signals of **SPIx_D[7:0]** for SPI both transmission and receiving.

Figure 18-31. SPI Data Mode – 8 Bidirectional data lines with spread data



18.13. SPI Supported Serial Flash

User can use this SPI module to communicate with external SPI serial flash. The following table is showing the list of general SPI flash types.

Table 18-7. SPI Flash Type List

Flash Type	Command Address	Data		Function Description
	Access Line	Access Line	DPX/HPX	
SPI (Standard)	1	1(In) / 1(Out)	DPX	2 data line full-duplex communication
SPI_D (Dual Output)	1	2	HPX	1 data line input and 2 data lines output for Slave 1 data line output and 2 data lines input for Master
SF (SPI-1)	1	1	HPX	1 data line half-duplex communication
SF2 (SPI+DPI)	1	2	HPX	transfer command by 1 data line and transfer data by 2 data lines (data spread only)
SP2 (DPI)	2	2	HPX	transfer both command and data by 2 data lines
SF4 (SPI+QPI)	1	4	HPX	transfer command by 1 data line and transfer data by 4 data lines (data spread only)
SP4 (SPI+QPI)	1 or 4	4	HPX	transfer command/address by 1 or 4 line and transfer data by 4 data lines (data spread only)
SP8 (SPI+OPI)	1 or 8	8	HPX	transfer command/address by 1 or 8 line and transfer data by 8 data lines

<Note> DPX/HPX = Full/Half Duplex Communication

User can set the registers of **SPIx_DAT_LINE**, **SPIx_BDIR_OE** and **SPIx_MDS** for the different flash control mode. Refer the "[SPI Data Modes](#)" section for more information.

The following table is showing the control registers' setting and pins' IO of SPI flash.

Table 18-8. SPI Flash Control Table

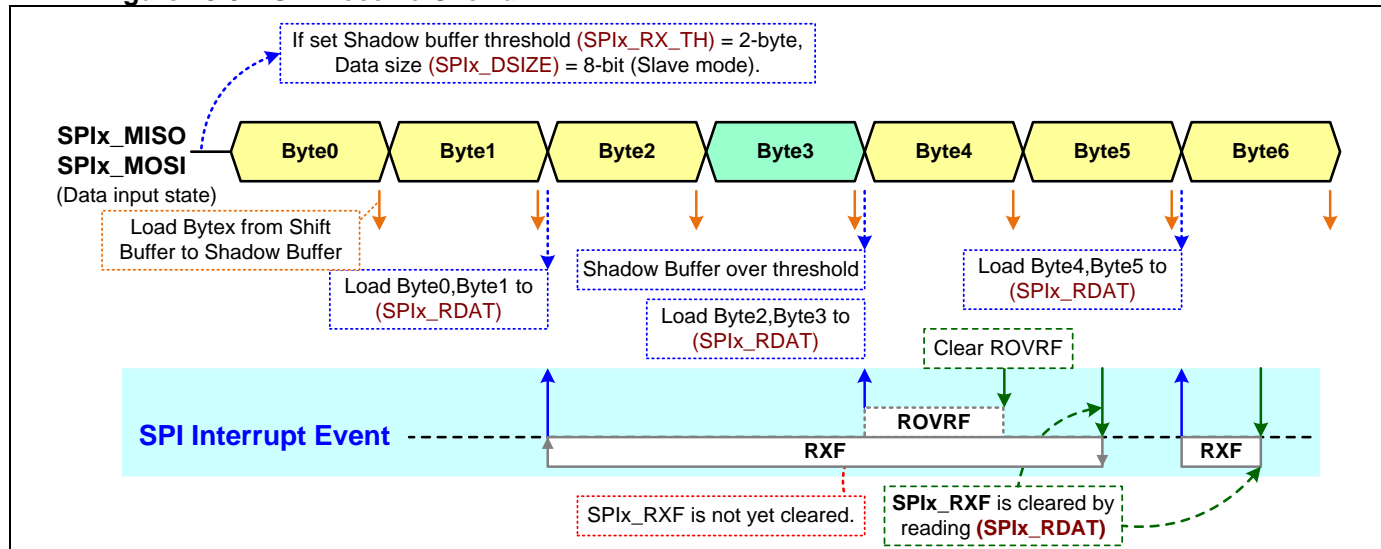
Flash Mode	DPX/HPX	SPI Register			I/O Pins				Function Description
		DAT_LINE	BDIR_OE	MDS	MOSI/D0	MISO/D1	D2,D3	D4~D7	
SPI	DPX	0	x	0	Input	Output			2 data line full-duplex communication
		0	x	1	Output	Input			
SPI-1	HPX	1	0	x	Input				1 data line half-duplex communication
		1	1	x	Output				
DPI	HPX	2	0	x	Input	Input			transfer both command and data by 2 data lines
		2	1	x	Output	Output			
QPI	HPX	3	0	x	Input	Input	Input		transfer both command and data by 4 data lines (data spread only)
		3	1	x	Output	Output	Output		
OPI	HPX	5	0	x	Input	Input	Input	Input	transfer both command and data by 8 data lines
		5	1	x	Output	Output	Output	Output	

<Note> x = 0 or 1 ; DPX/HPX = Full/Half Duplex Communication

18.14. SPI Receive Overrun

When Shadow Buffer is full and RX data register (**SPIx_RDAT**) is not empty for slave mode, the ROVRF flag (**SPIx_ROVRF**) will be activated. If the Shift Buffer is still receiving a frame data and the RXF flag is not yet clear by software, the next frame data will input and overlay the Shift Buffer content as showing the condition of Byte-4 and Byte-5 in following diagram.

Figure 18-32. SPI Receive Overrun

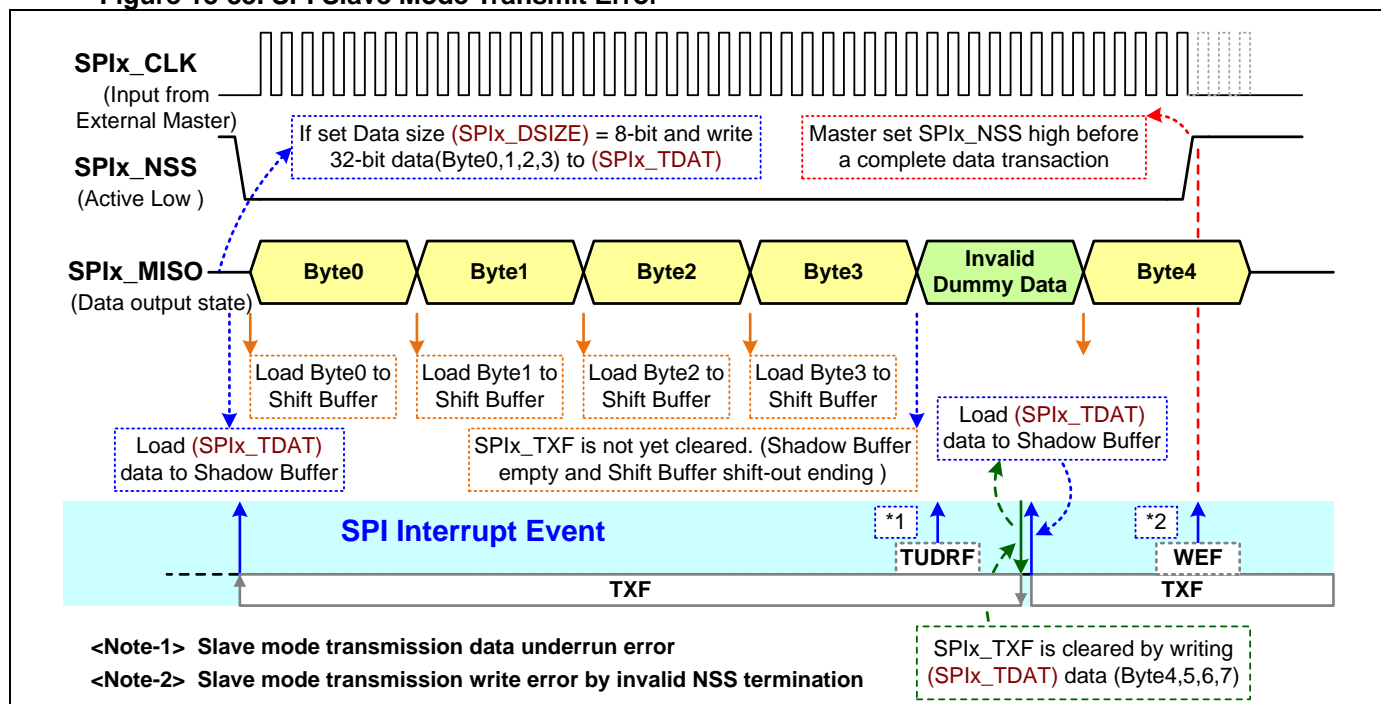


18.15. SPI Transmit Error

When both Shadow Buffer and TX data register are empty for slave mode, the TUDRF flag (**SPIx_TUDRF**) will be activated if the Shift Buffer shifts out completely and external SPI master still requests data.

When the NSS signal (**SPIx_NSS**) is inactive at the time which a data frame is not yet transferred ending by external master, the write error WEF flag (**SPIx_WEF**) will be activated.

Figure 18-33. SPI Slave Mode Transmit Error



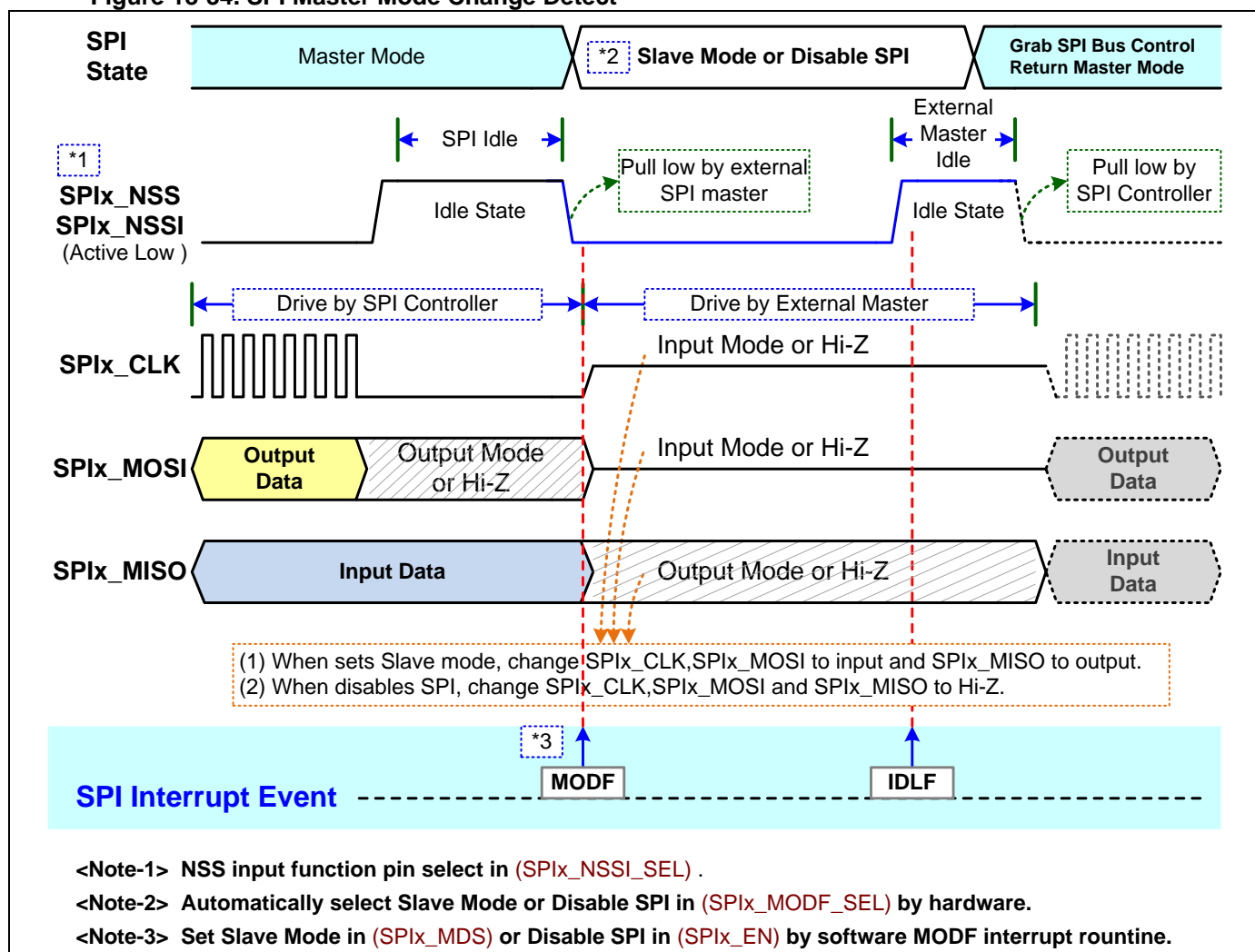
18.16. SPI Master Mode Change Detect

For multi-master communication, the SPI module can detect the state of **SPIx_NSS** or **SPIx_NSSI** signal. When the SPI module is running in master mode and is inactivated during idle state by external SPI master, the mode failure **MODF** flag (**SPIx_MODF**) will be activated and the SPI module will be disabled or switch to slave mode by setting **SPIx_MODF_SEL** register.

Then the SPI module can detect the signal of **SPIx_NSS** or **SPIx_NSSI** signal whether goes into idle state. If detects idle state, the **IDLF** flag (**SPIx_IDLF**) will be activated. So user can grab the SPI bus and set to master mode again. Others a hardware status bit is used to report the SPI bus idle status for slave with NSS mode in **SPIx_IDL_STA** register.

Refer the “[SPI Two Master Communication](#)” section for more information.

Figure 18-34. SPI Master Mode Change Detect



18.17. SPI DMA Operation

18.17.1. DMA Module Configure

When the chip supports a DMA (direct memory access) controller, user can configure the DMA setting of transferred source/destination devices, channel request arbitration and others in the DMA module before a DMA data transaction. The DMA source and the destination can be memory or peripheral.

Refer the DMA chapter for more detail information about the DMA module configuration.

18.17.2. SPI DMA Control

After DMA configuration is finished, user needs to set the SPI module DMA enable bits of **SPIx_DMA_RXEN** and **SPIx_DMA_TXEN**.

Finally, the related channel request start bit of **DMA_CHn_REQ** is necessary to be set to start the DMA transaction (n = DMA channel index). Then the transferred source/destination devices will assert the RX/TX request signal to DMA controller and the DMA controller will assert the acknowledge signal to the request source/destination devices. At the time, the data transferred connection is built for DMA transaction.

- **SPI RX to DMA**

The **SPIx_DMA_RXEN** register bit is used to enable DMA data transfer from SPI receiving input to DMA destination.

During the DMA transaction cycle, the received data flag of **SPIx_RXF** is masked by hardware.

- **SPI TX from DMA**

The **SPIx_DMA_TXEN** register bit is used to enable DMA data transfer from DMA source to SPI transmitted output.

During the DMA transaction cycle, the transmitted data flag of **SPIx_TXF** is masked by hardware. In general, the transmitted complete flag of **SPIx_TCF** will be asserted after the finished of DMA transaction.

18.17.3. SPI Interrupt Flag Control during DMA

During DMA operation cycle, the module's interrupt flags will control and act three types as following table. One is masked during the DMA process. Another is to disable the DMA function after the flag has asserted. At the time, hardware will clear both the **SPIx_DMA_TXEN** bit and the **SPIx_DMA_RXEN** bit in this condition. Others are normally as same as the action of not processing in DMA operation.

Table 18-9. SPI Interrupt Flag Control for DMA Function

Action	Mask the Flag during DMA Process	DMA Disable after the Flag asserted (*1)	Normal Control
Peripheral	(Data flow flags)	(Error/Detect flags)	(Other flags)
SPIx	SPIx_TCF (*2) SPIx_RXF SPIx_TXF (*2)	SPIx_MODF SPIx_WEF SPIx_ROVRF SPIx_TUDRF	SPIx_IDLF

Note-1 : When the flag is asserted, it will not force peripheral DMA disabled if the related interrupt enable bit is not enabled.

Note-2 : SPIx_TCF will be asserted after DMA TX complete.

19. USB (Universal Serial Bus)

19.1. Introduction



The module can be running in ON and SLEEP modes but can wakeup from STOP mode.

For MG32F02U series, the chip provides a USB (Universal Serial Bus) device with full-speed function. It is fully compliant with USB specification 2.0 and 1.1 to support various USB applications. The USB block contains an on-chip 3.3V regulator, a USB transceiver which transmits and receives differential USB signal, a USB Core to perform NRZI encoding and decoding, bit stuffing, CRC generation and checking, serial-parallel data transforming, data flow between USB data buffer and CPU.

An independent 512-byte SRAM is as USB receiving and transmission data packet buffer which can be configurable and shared for all endpoints. Also the data packet buffer is able to directly access by CPU and use for firmware whenever the USB function is enabled or disabled.

Notify: The sign of (x = module index; n = Endpoint index number) is using for Registers, Signals and Pins/Ports in the descriptions of this chapter.

19.2. Features

- USB 2.0 full-speed with 12Mbps
 - Compliant with USB specification v1.1/v2.0.
- Support USB suspend/resume and remote wake-up
- Support 8 endpoints with In and Out directions
 - Each endpoint support flexible In, Out, simultaneous In and Out operations
 - Support 7 configurable endpoints with relocated address value except endpoint-0
 - Support independently receive and transmit buffer with separated start address for each endpoint
 - Support double buffer mode for each endpoint independently
- Support control transfer for endpoint 0
- Support interrupt, bulk and isochronous transfer for all endpoints except endpoint-0
- Support USB SRAM size 512 bytes shared for all endpoints' packet buffer
- Supports USB 2.0 Link Power Management
- Received and transmitted data are buffered with DMA capability for EP3,4

19.3. Implementation

19.3.1. Chip Implementation

MG32F02U series is embedded one USB full speed device with USB 2.0 compliant and one 3.3v LDO for USB operation power. Please refer the table of "[Chip Implementation Table](#)" in the section of "Applicable Chips" for more detail information.

19.3.2. Module's Functions

The following table is showing the Endpoint functions of USB module. The EP0 is only able to configure as Control transfer type.

Table 19-1. USB Endpoint Functions

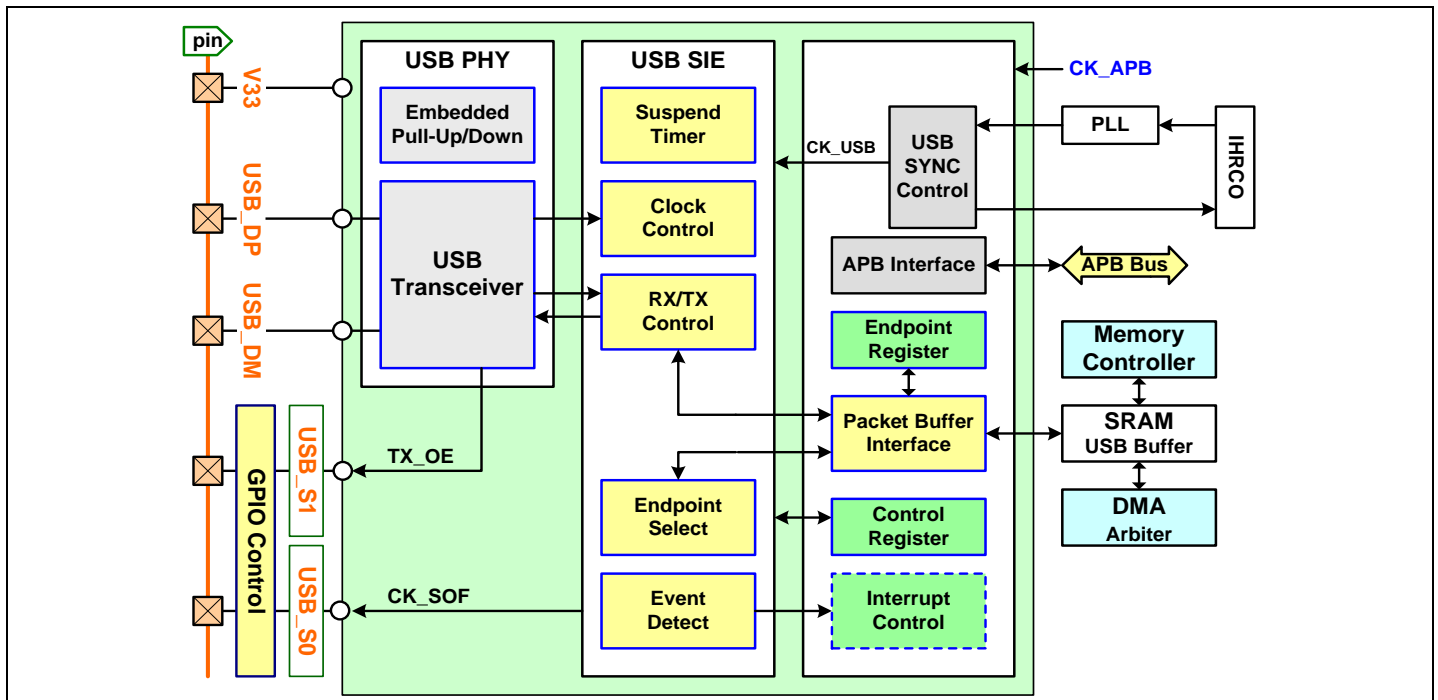
	USB Endpoint							
Function	EP0	EP1	EP2	EP3	EP4	EP5	EP6	EP7
Control Transfer	V							
Interrupt Transfer		V	V	V	V	V	V	V
Bulk Transfer		V	V	V	V	V	V	V
ISO Transfer		V	V	V	V	V	V	V
Double Buffer		V	V	V	V	V	V	V
DMA				V	V			
Programmable Endpoint Address		V	V	V	V	V	V	V
Relocated Buffer Address	V	V	V	V	V	V	V	V
Receiving Overflow Detect	V	V	V	V	V	V	V	V

<Sign> " V " ~ Support

19.4. Control Block

The following diagram is showing the USB Control block.

Figure 19-1. USB Main Block



- **USB PHY : USB Physical Interface**

This USB PHY is used as the interface to connect an external USB host. It includes the USB transceiver, configurable internal embedded pull-up and pull-down resistors in the analog front end block. The chip can send the output enable control signal of the USB transceiver to **TX_OE** signal.

Refer the section of "[USB Analog Front End](#)" for more information.

- **USB SIE : Serial Interface Engine**

This block includes the suspend timer, USB clock control block, RX/TX control block, Endpoint select block and USB event detector.

- **USB SYNC Control**

The USB SYNC control block inputs and locks the PLL output clock to generate the **CK_USB** (48MHz) clock for the USB SIE block.

Refer the section of "[USB Clock Control](#)" for more information.

- **Packet Buffer Interface**

There is one embedded independent SRAM memory zone to use as USB packet data buffer. This Packet Buffer Interface block is the interface between the USB macro and USB packet data buffer. It can be configurable for user request buffer mode and endpoints' allocation to control the SRAM memory access of transmission and reception.

Refer the section of "[USB Packet Buffer](#)" for more information.

- **Endpoint/Control/Interrupt Register**

For each endpoint EP0~7, it has one independent set of registers those contains the endpoint address, endpoint type, endpoint packet data buffer mode, endpoint packet data start address, endpoint packet data transmission and reception counter, USB event flags and interrupts.

Refer the section of "[USB Endpoint Configuration](#)" for more information.

19.5. IO Lines

This module is including of three pins, one embedded +3.3 volt power regulator output pin – **V33** and two USB communication pins – **DM** and **DP** for USB interface. For internal using, two output signals - **USB_S0** and **USB_S1** to output USB module internal signals.

19.5.1. IO Signals

- **V33**

It is the USB embedded LDO output power pin.

- **DM / DP**

They are the USB interface differential **DM** and **DP** signals.

- **CK_SOF / TX_OE**

Refer the “USB Main Block” diagram, there are two output signals of **CK_SOF** and **TX_OE** those are able to output from **USB_S0** and **USB_S1** pins. The chip can output the internal generated start of frame (SOF) signal **CK_SOF**. Also the chip can send the output enable control signal of the USB transceiver to **TX_OE** signal.

19.5.2. Bus State

The following table is showing the USB DM/DP lines’ bus states and related detection interrupts and flags.

Table 19-2. USB Bus State and Flag

USB State	Bus State	Reflected Flag	
		Interrupt	Status
J	low speed : (D+, D-) = (0,1) (differential "0") full speed : (D+, D-) = (1,0) (differential "1")		J_STA
K	low speed : (D+, D-) = (1,0) (differential "1") full speed : (D+, D-) = (0,1) (differential "0")		K_STA
SE0	(D+, D-) = (0,0)		SE0_STA
SE1	(D+, D-) = (1,1)	SE1F	SE1_STA
Reset	Keep SE0 state for more than 2.5us, host will drive 10ms.	RSTF	
Resume	Any K state event shown on the bus, either driven by the host (Host keep Resume > 20ms) or device (Remote Wakeup, Device keep Resume 1~15ms)	RSMF	
Suspend	Keep Idle for more than 3 ms, the device should go into suspend state within no more than 10ms.	SUSF	
SOP	Start Of Packet : Idle => K state	SOF	

19.6. Power and Clock

19.6.1. USB Global Enable

There is a global enable bit of **USB_EN** for all functions of this module. When this bit is disabled, all the USB functions are not working.

19.6.2. USB Power Control

The USB block contains an on-chip 3.3V power regulator with one power **V33** pin. The operation power of USB analog front-end is from the power **V33** pin. User can enable the USB embedded regulator by setting **USB_V33_EN** register. Refer the section of “[USB Application Circuit](#)” for more information.

Optionally, User can use external +3.3 volt power source to connect to the **V33** pin and turn off the USB embedded regulator by setting **USB_V33_EN** register. User also can turn off the USB embedded regulator and connect internal **VDD** to the power **V33** pin by setting **USB_V33_EN** and **USB_V33_VDD** register when the **VDD** power voltage range is +3.3 volt with +/- 10% tolerance.

User can plan the power control of analog USB macro is on or off beforehand for chip entering **SLEEP** or **STOP** mode by setting **PW_SLP_USB** or **PW_STP_USB** registers. Refer the System Power chapter for more information.

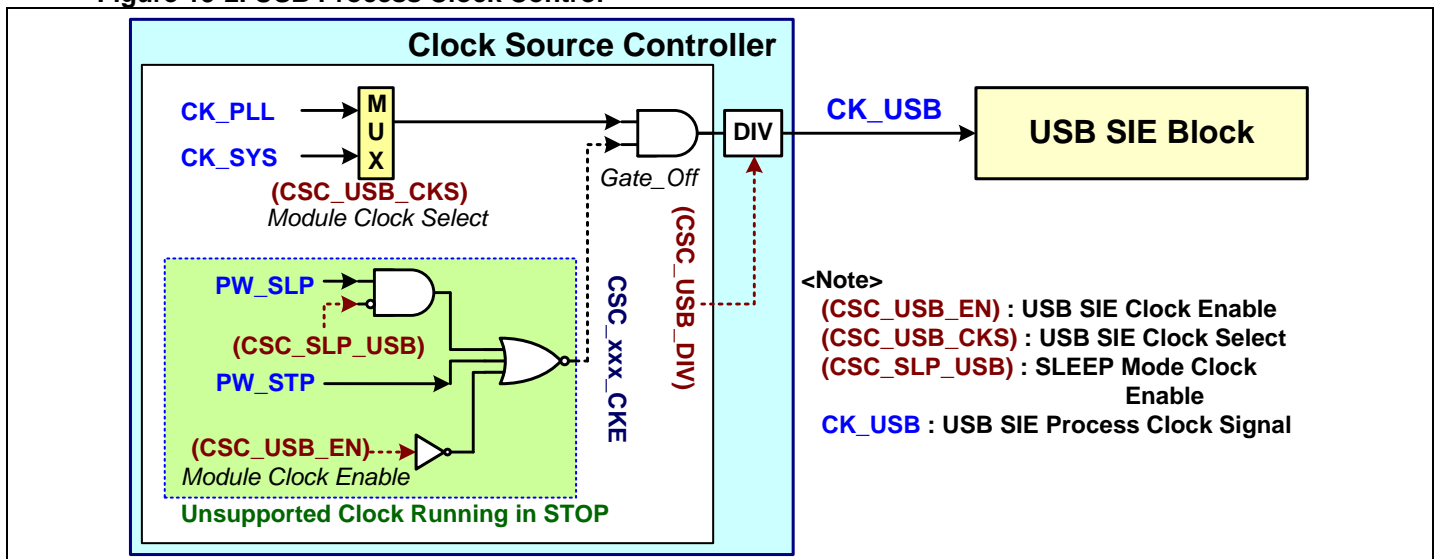
19.6.3. USB Clock Control

● Module Process Clock

The module process clock of **CK_USB_PR** is using for the interface control logic between APB bus and the module. It is coming from CSC (Clock Source Controller) module. It can be enabled in **CSC_USB_EN** register.

In **ON** mode, the process clock is running only if **CSC_USB_EN** register is enabled. User can plan the module clock is running or not beforehand for chip entering **SLEEP** mode by setting **CSC_SLP_USB** register. In **STOP** mode, the process clock is always stopping. Refer the System Clock chapter for more information.

Figure 19-2. USB Process Clock Control

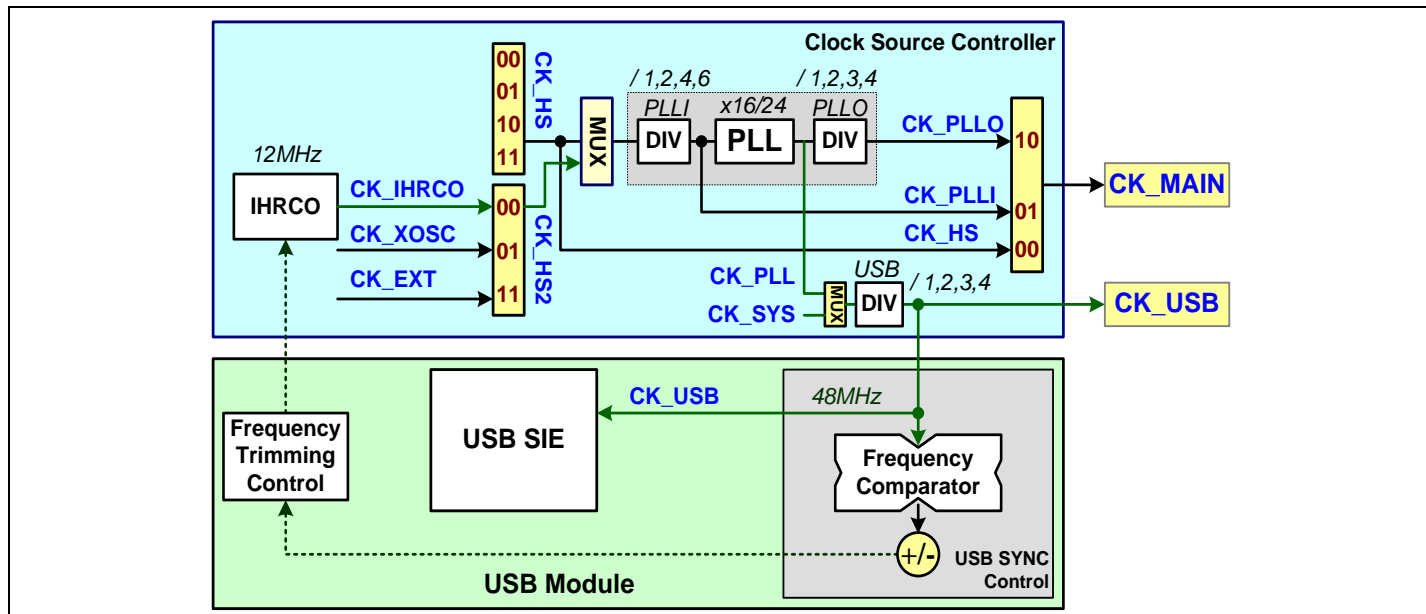


- **Module Internal Clock**

The **CK_USB** is using for the USB SIE block and the frequency is general 48MHz. It is also coming from CSC (Clock Source Controller) module. It also can be enabled in **CSC_USB_EN** register and select the clock source from **CK_PLL** clock or **CK_SYS** clock in **CSC_USB_CKS** register.

The following diagram is showing the USB SYNC control block.

Figure 19-3. USB SYNC Control Block



When it selects **CK_PLL** clock source, the **CK_USB** can generate and fine-tune from the USB SYNC control block. Also the USB SYNC control block inputs the internal PLL clock output **CK_PLL** and locks to generate the **CK_USB** 48MHz clock frequency.

When the USB module is using, the internal PLL needs to generate a 48MHz clock to input for USB SIE and others blocks. The internal PLL input clock source needs coming from IHRCO clock **CK_IHRCO** and **CK_HS2** through the MUXs selection. Usually in the condition, user can select the clock **CK_HS** as system main clock **CK_MAIN** for CPU system clock and select the clock **CK_HS2** as PLL input clock for generation USB SIE clock. Refer the section of “High Speed Clock and Low Speed Clock” in System Clock chapter for more information.

There is another high speed clock **CK_HS2** which is obtained from one of these three clock sources of **IHRCO**, **XOSC** and **EXTCK** through the high speed clock selector by setting **CSC_HS2_SEL** register. Usually this clock is using for USB application, user can select the clock **CK_HS** as system main clock **CK_MAIN** for CPU system clock and select the clock **CK_HS2** as PLL input clock **CK_PLLIX** for generation USB 48MHz clock.

19.6.4. USB Reset Control

As same as other modules, the chip will auto reset USB all blocks and registers when the USB module is receiving any system reset events of Warm reset, Cold reset and POR reset. Also there is one independent software reset enable bit **RST_USB_EN** to reset the USB module independently. It is like as which the module is receiving the Warm reset signal but only resets the USB module.

When the USB module is receiving the software reset by enabling **RST_USB_EN** register, there are two reset control levels to reset the USB module for user selection by setting **RST_USB_RCTL** register. When this register is selected 'All', chip will auto reset USB all blocks and registers. When this register is selected 'LV1', chip will reset all blocks and registers except **USB_EN** (USB module enable bit), **USB_XTR_EN** (USB transceiver enable bit), **USB_V33_EN** (USB V33 regulator enable bit) and **USB_DPU_EN** (USB DP line pull-up resistor enable bit) registers control.

19.7. Interrupt and Event

There is one signal of **INT_USB** to be generated in this USB control module. **INT_USB** sends to EXIC External Interrupt Controller to do as an interrupt event.

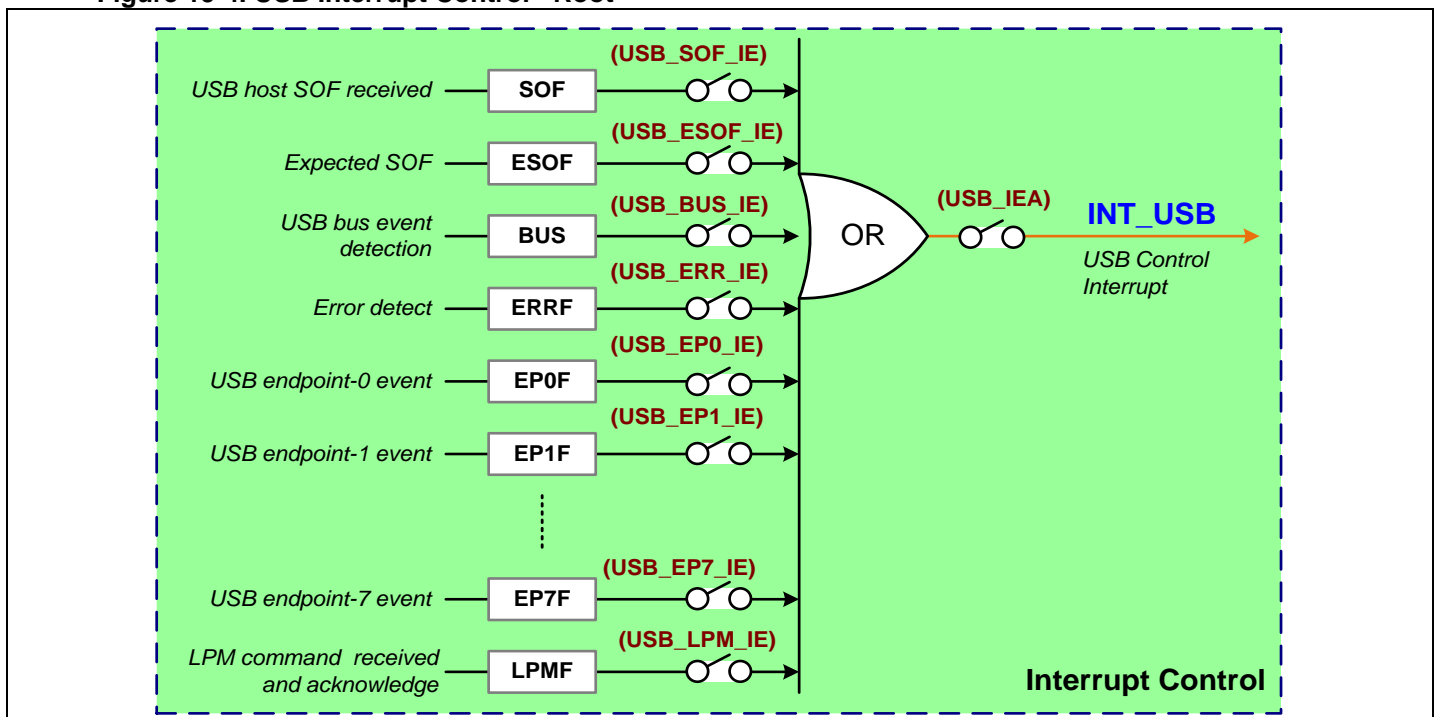
19.7.1. USB Interrupt Control and Status

These interrupt flags are using for the interrupt service routine (ISR) flow control. Generally these flags are set by hardware and software clears them when the serviced job is completed in the related ISR. Each interrupt flag has one interrupt enable bit. User can enable or disable it. There is one interrupt all enable bit of **USB_IEA** to enable or disable all the interrupt sources for this module.

❖ USB Interrupt Root

Generally these interrupt flags are set by hardware and clear by software writing 1. Refer the register descriptions for more information about the related interrupt flags and interrupt enable bit.

Figure 19-4. USB Interrupt Control - Root



● BUSF

USB bus event global interrupt flag (**USB_BUSF**). These USB bus events are including of USB bus reset, resume, suspend, SE1 state. There is a related interrupt enable register bit of **USB_BUS_IE**. It indicates any subrange flag of **USB_RWKF**, **USB_RSTF**, **USB_RSMF**, **USB_SUSF** or **USB_SE1F** is asserted when this flag is set. This flag is cleared by hardware when all the subrange interrupt flags are cleared.

● ERRF

USB bus error global interrupt flag (**USB_ERRF**). There is a related interrupt enable register bit of **USB_ERR_IE**. It indicates any of USB suspend, resume, reset and remote-wakeup events. It indicates any subrange flag of **USB_OVRF**, **USB_SETUPF**, **USB_NORSF**, **USB_BSTF** or **USB_CRCF** is asserted when this flag is set. This flag is cleared by hardware when all the subrange interrupt flags are cleared.

● SOF

USB host SOF received interrupt flag (**USB_SOF**). There is a related interrupt enable register bit of **USB_SOF_IE**. It indicates that hardware has detected a host SOF.

● ESOF

USB expected start of frame interrupt flag (**USB_ESOF**). There is a related interrupt enable register bit of **USB_ESOF_IE**. This bit is set by hardware when detected an expected SOF. It will be generated by hardware expecting to detect a host SOF, even if the real host SOF is missing or corrupted.

● LPMF

USB LPM L1 state request interrupt flag (**USB_LPMF**). There is a related interrupt enable register bit of

USB_LPM_IE. This bit is set by the hardware when LPM command entering the L1 state is successfully received and acknowledged.

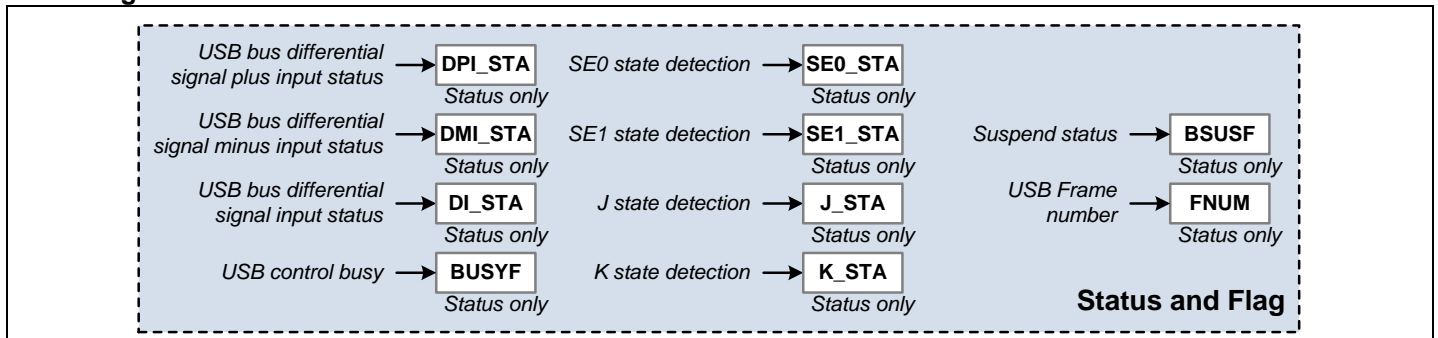
- **EP0F ~ EP7F**

USB endpoint-0 ~ 7 event global interrupt flag (**USB_ERRF**). There are related interrupt enable register bits of **USB_EP0_IE ~ USB_EP7_IE**. This flag is cleared by hardware when all the subrange interrupt flags are cleared.

❖ USB Status Flag

There are some status bits those are reading only to provide internal control status. One busy flag of **USB_BUSYF** is used to indicate the data transfer busy status. Refer the register descriptions of the related status bits for more information.

Figure 19-5. USB Global Event Status



- **BUSYF**

USB data transfer busy flag (**USB_BUSYF**).

- **BSUSF**

USB current bus suspend state (**USB_BSUSF**).

- **DI_STA**

USB differential input state (**USB_DI_STA**).

- **DPI_STA / DMI_STA**

USB bus differential signal plus input line DP/DM status (**USB_DPI_STA / USB_DMI_STA**).

- **SE0_STA / SE1_STA / J_STA / K_STA**

USB SE0/SE1/J/K state. (**USB_SE0_STA / USB_SE1_STA / USB_J_STA / USB_K_STA**).

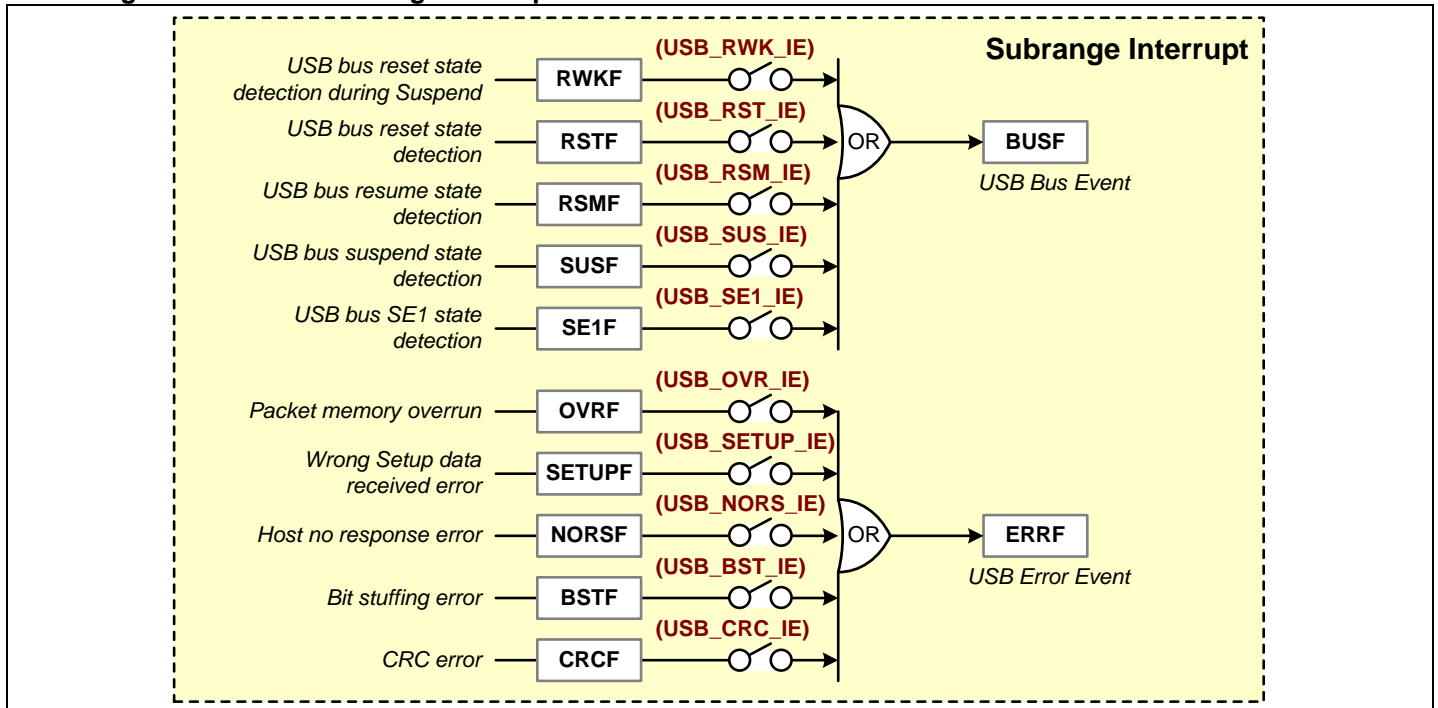
- **FNUM**

USB frame number (**USB_FNUM**). It contains the 11-bits frame number contained in the last received SOF packet. The frame number is incremented for every frame sent by the host.

19.7.2. USB Subrange Interrupt – Bus and Error

There are two independent subrange interrupt flags for the interrupt flags of **USB_BUSF** and **USB_ERRF**. The following diagram is showing the USB Bus and Error subrange interrupt control blocks.

Figure 19-6. USB Subrange Interrupt Control – Bus and Error



- **RWKF**

USB reset wakeup interrupt flag (**USB_RWKF**). During suspend, this flag is set by hardware when the function detects the USB bus reset. There is a related interrupt enable register bit of **USB_RWK_IE**.

- **RSTF**

USB bus reset detected interrupt flag (**USB_RSTF**). This flag is set by hardware when the chip detects the reset state on USB bus. It would be cleared by firmware in the USB reset interrupt-service routine. There is a related interrupt enable register bit of **USB_RST_IE**.

- **RSMF**

USB bus resume state detected interrupt flag (**USB_RSMF**). This flag is set by hardware when the chip detects the resume state on the USB bus. It would be cleared by software in the USB resume interrupt-service routine. There is a related interrupt enable register bit of **USB_RSM_IE**.

- **SUSF**

USB bus suspend state detected interrupt flag (**USB_SUSF**). This flag is set by hardware when the chip detects the suspend state on the USB bus. In the USB suspend interrupt-service routine, software should clear this bit before enter the suspend mode. There is a related interrupt enable register bit of **USB_SUS_IE**.

- **SE1F**

USB SE1 state detect interrupt flag (**USB_SE1F**). This flag is set by hardware when USB bus is kept SE1 state more than 1 USB bit time. There is a related interrupt enable register bit of **USB_SE1_IE**.

- **OVRF**

USB packet memory overrun detects interrupt flag (**USB_OVRF**). There is a related interrupt enable register bit of **USB_OVRF_IE**.

- **SETUPF**

USB wrong Setup data received error flag (**USB_SETUPF**). This flag will be asserted if the Setup data is not 8-byte or is not DATA0. Normally the Setup DATA0 is received 8-byte. There is a related interrupt enable register bit of **USB_SETUP_IE**.

- **NORSF**

USB host no response error flag (**USB_NORSF**). This flag will be asserted if USB host response timeout after data packet is sent. There is a related interrupt enable register bit of **USB_NORS_IE**.

- **BSTF**

USB Bit Stuffing error flag (**USB_BSTF**). A bit stuffing error was detected anywhere in the PID, data, and/or CRC. There is a related interrupt enable register bit of **USB_BST_IE**.

- **CRCF**

USB cyclic redundancy check error flag (**USB_CRCF**). One of the received CRCs, either in the token or in the data, was wrong. There is a related interrupt enable register bit of **USB_CRC_IE**.

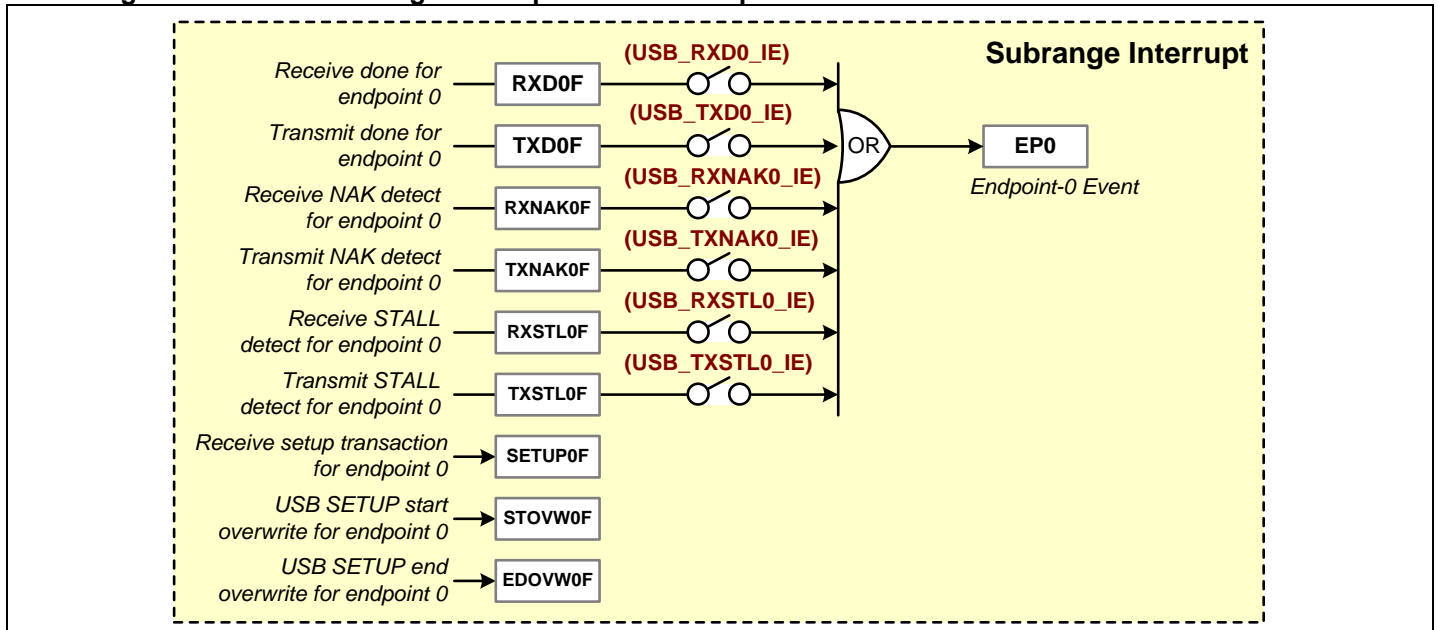
19.7.3. USB Subrange Interrupt –Endpoint

There are independent endpoint subrange interrupt flags for the interrupt flags of **USB_EPn**. The SETUP0F, STOVW0F and EDOVW0F flags are supported for USB endpoint 0 but not supported for other endpoints. The ISOOVWnF and ISOTXEnF flags are supported for USB endpoint 1~7 but not supported for endpoint 0.

❖ USB Endpoint 0

The following diagram is showing the subrange interrupt control block of USB endpoint 0.

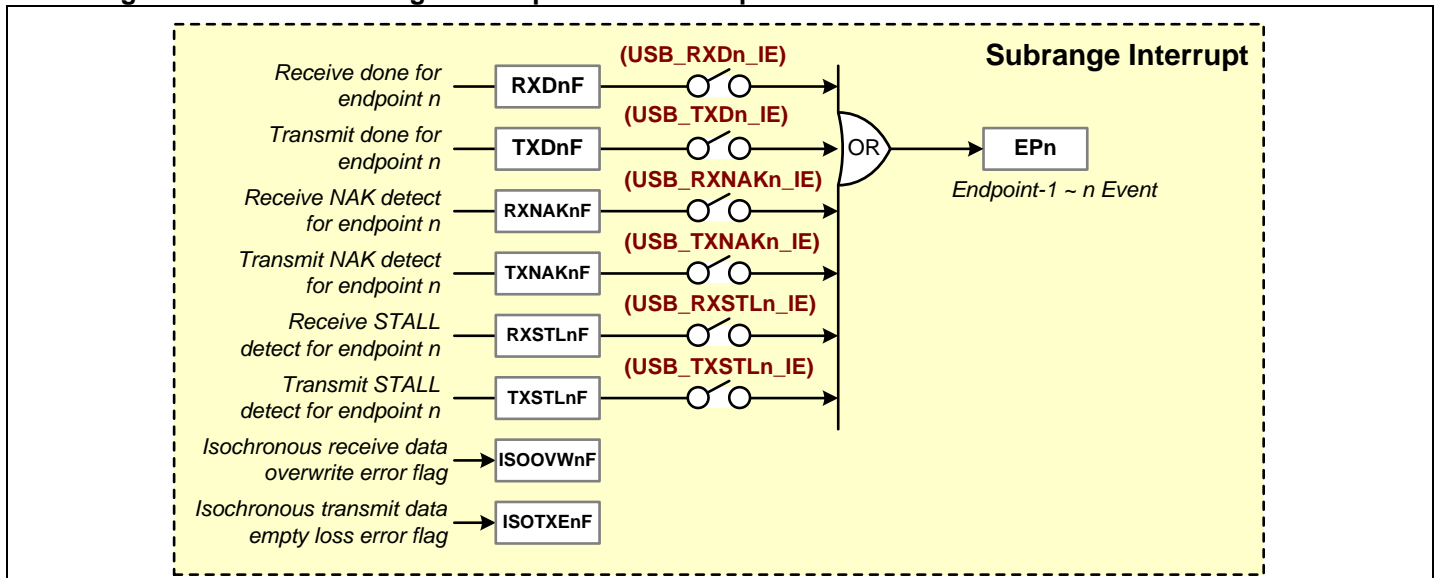
Figure 19-7. USB Subrange Interrupt Control – Endpoint 0



❖ USB Endpoint 1~7

The following diagram is showing the subrange interrupt control block of USB endpoint 1~7.

Figure 19-8. USB Subrange Interrupt Control – Endpoint 1~7



● RXDnF

USB endpoint receiving done flag (**USB_RXDnF**). There is a related interrupt enable register bit of **USB_RXDn_IE**. This flag is set by the hardware when an OUT/SETUP transaction is successfully completed on this endpoint. The type of occurred transaction, OUT or SETUP, can be determined from the SETUP bit. A transaction ended with a NAK or STALL handshake does not set this bit. (n=0~7)

- **TXDnF**

USB endpoint transmission done flag (**USB_TXDnF**). There is a related interrupt enable register bit of **USB_TXDn_IE**. This bit is set by the hardware when an IN transaction is successfully completed on this endpoint. A transaction ended with a NAK or STALL handshake does not set this bit. (n=0~7)

- **RXNAKnF**

USB endpoint receiving NAK event flag (**USB_RXNAKnF**). There is a related interrupt enable register bit of **USB_RXNAKn_IE**. This bit is set by hardware when detected a receive done on the NAK packet for OUT transaction of the selected endpoint by USB_NAKEP_SEL. (n=0~7)

- **TXNAKnF**

USB endpoint transmission NAK event flag (**USB_TXNAKnF**). There is a related interrupt enable register bit of **USB_TXNAKn_IE**. This bit is set by hardware when detected a transmit done on the NAK packet for IN transaction of the selected endpoint by USB_NAKEP_SEL. (n=0~7)

- **RXSTLnF**

USB endpoint receiving STALL event flag (**USB_RXSTLnF**). There is a related interrupt enable register bit of **USB_RXSTLn_IE**. (n=0~7)

- **TXSTLnF**

USB endpoint transmission STALL event flag (**USB_TXSTLnF**). There is a related interrupt enable register bit of **USB_TXSTLn_IE**. (n=0~7)

- **SETUP0F**

USB received setup transaction flag (**USB_SETUP0F**). This bit is set by hardware when a valid SETUP transaction has been received. Clear this bit upon detection of a SETUP transaction or the firmware is ready to handle the data/status stage of control transfer. This bit is only used for control endpoint.

- **STOVW0F**

USB SETUP start overwrites flag (**USB_STOVW0F**). This bit is set by hardware upon receipt of a SETUP token for the control endpoint to indicate that the receive FIFO is being overwritten with new SETUP data. This bit is used only for control endpoint.

- **EDOVW0F**

USB SETUP end overwrite flag (**USB_EDOVW0F**). This flag is set by hardware during the handshake phase of a SETUP transaction. This bit is cleared by firmware to read the FIFO data. This bit is only used for control endpoint.

- **ISOOVWnF**

USB isochronous receive data overwrite error flag (**USB_ISOOVWnF**). This bit is set by hardware as a USB memory access conflict happen when USB host send the next data but firmware has not yet read out the last data in time. Firmware can use this bit to make sure whether the data had been overwritten or not. (n=1~7)

- **ISOTXEnF**

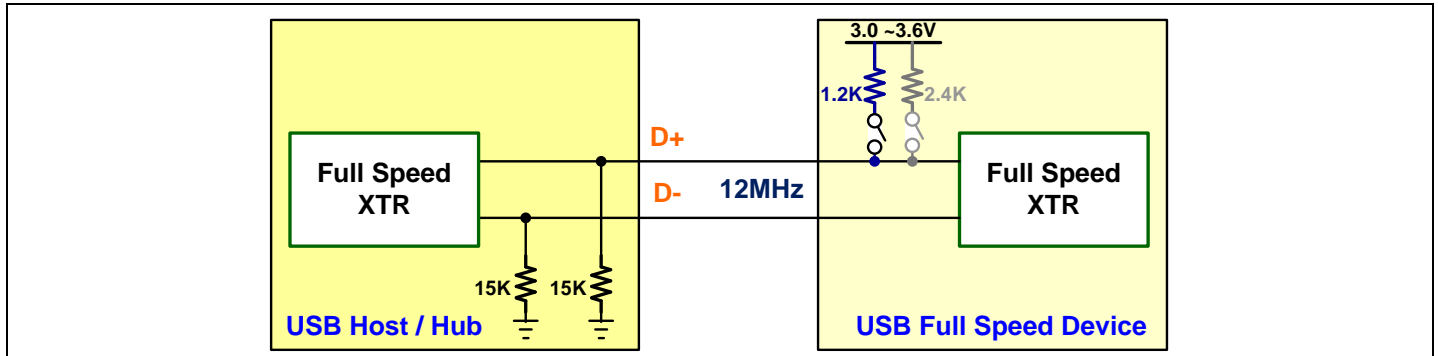
USB isochronous transmit data empty loss error flag (**USB_ISOTXEnF**). This bit is set by hardware as a USB memory access empty loss happen when USB host request the next data but firmware has not yet write the data in time. Firmware can use this bit to make sure whether the data had been empty or not. (n=1~7)

19.8. USB Connection for Application

The following diagrams are showing the USB application connections for full speed mode communication. For MG32F02U series, the chip is embedded a pull-up about 1.2K ohm resistor on **DP** line for the USB full speed mode connection.

Refer the section of “[USB Analog Front End](#)” for more information.

Figure 19-9. USB Bus Connection



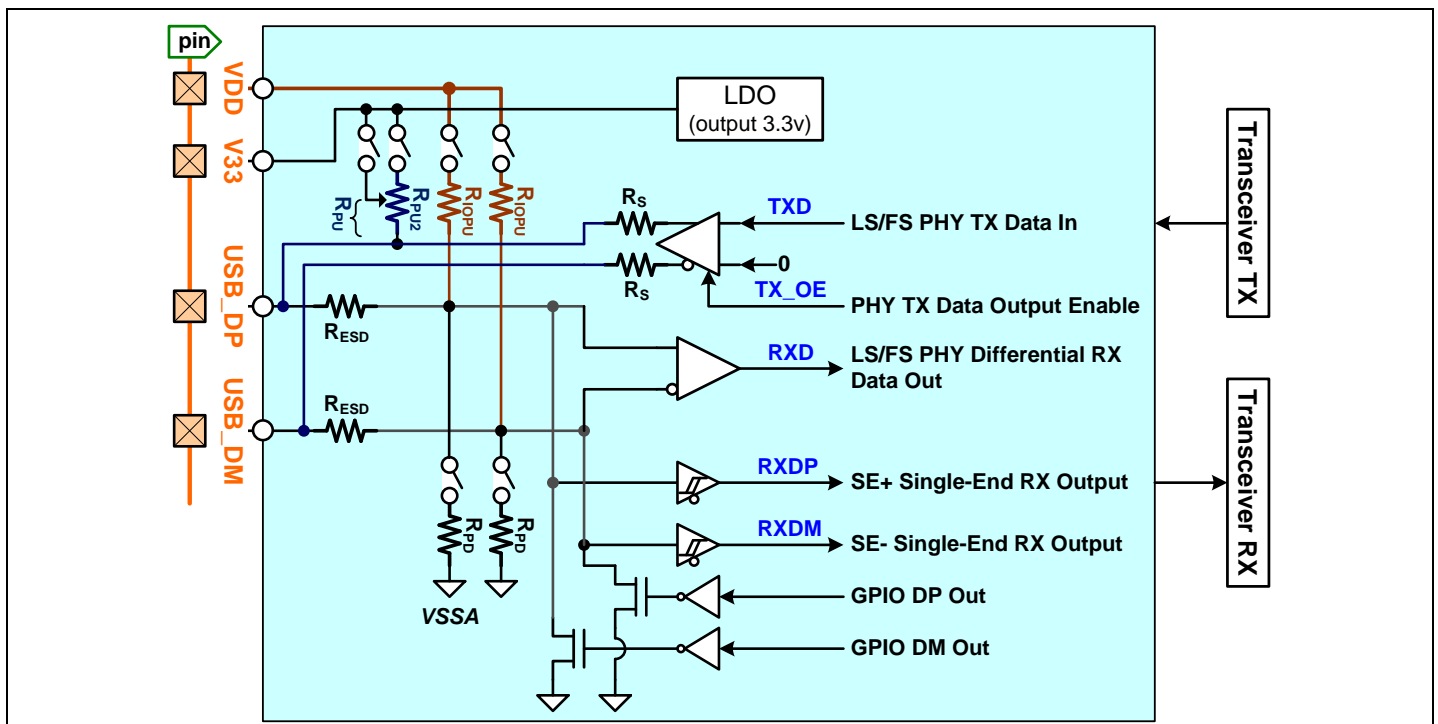
19.9. USB Fundamental Control

19.9.1. USB Analog Front End

This USB analog front end is used to interface and connect an external USB host. It includes the configurable internal embedded pull-up and pull-down resistors, USB DP/DM differential signal to single-end signal transferred devices and an embedded +3.3V USB power regulator. Refer the section of “[USB Power Control](#)” for more information about the embedded USB power regulator.

The following diagram is showing the USB analog front-end block diagram.

Figure 19-10. USB Analog Front End



The chip is embedded a pull-up about 1.2K ohm resistor (**R_{PU}**) on **DP** line for the USB full speed mode connection. User can enable this pull-up resistor by setting **USB_DPU_EN** register.

The chip can automatically disable this pull-up resistor when USB **DP** signal output low during USB transmission by setting **USB_DPU_AUTO** register. Another option for the pull-up resistor, user can enable to auto select 1.2Kohm (**R_{PU}**) or 2.4Kohm (**R_{PU2}**) by USB bus state by setting ‘Switch’ in **USB_DPU_MDS** register. The chip will

select 1.2Kohm if bus idle, SE0-state or after sequential 7-bit J-state. And the chip will select 2.4Kohm after 0.5-bit K-state. Also user can always enable DP pull high 1.2Kohm resistor by setting 'Fix' in **USB_DPU_MDS** register.

[Notify]: The USB DPU AUTO and USB DPU MDS registers is no effect if USB DPU EN is disabled.

Also the chip is embedded two pull-down about 750K~1M ohm resistors (**R_{PD}**) on **DP** and **DM** lines. User can enable this pull-down resistor by setting **USB_DP_PD** and **USB_DM_PD** registers independently.

When the USB macro is in use, the **PD4**, **PD5** and **PD6** pins are used as USB **DM** pin, USB **DP** pin and USB embedded LDO output power pin. The GPIO **PD4**, **PD5** and **PD6** must set IO mode to AIO mode.

When these pins are used as GPIO or other AFS functions, the USB LDO and USB transceiver must be turned off by setting **USB_V33_EN**, **USB_XTR_EN** and **USB_EN** registers. Also, user can enable or disable the GPIO pull-up resistor (**R_{IOPU}**) but disable all the USB pull-up and pull-down resistors.

Refer the section of "[IO Lines](#)" for more information about the USB IO signals.

[Notify]: The power voltage of USB pull-up resistor is connect to V33. The power voltage of GPIO pull-up resistors is connecting to VDD.

19.9.2. USB Endpoint Configuration

For MG32F02U series, the chip can support maximum 8 configurable endpoints of EP0~ EP7. Each endpoint has one independent set of registers those contains the endpoint address, endpoint type, endpoint packet data buffer mode, endpoint packet data start address, endpoint packet data transmission and reception counter, USB event flags and interrupts.

Each EP_n can be flexibly configured to "In", "Out" or "Simultaneous In and Out" operations by setting **USB_RXEN_n** and **USB_TXEN_n** registers. (n = USB endpoint index)

The USB module can support Control, Interrupt, Bulk and Isochronous transfer endpoint types. The EP0 endpoint is only able to configure as Control transfer type. The EP1~7 endpoints can configure as Interrupt/Bulk or Isochronous transfer type by setting **USB_RXTYPE_n** and **USB_TXTYPE_n** registers.

The endpoints of EP1~ EP7 are able to configure relocated endpoint address by setting **USB_EPADR_n** register. The chip default endpoint addresses of EP1~ EP7 are 1~7 and the EP0 endpoint address is fixed to 0.

19.9.3. USB Packet Buffer

For MG32F02U series, the chip embedded an independent 512-byte SRAM to do as USB receiving and transmission data packet buffer. Also the data packet buffer is able to directly access by CPU and use for firmware whenever the USB function is enabled or disabled.

This SRAM buffer can be configurable as the data packet buffer and shared for all available endpoints. The data packet buffer is support independently receive and transmit buffer with separated start address for each endpoint by setting **USB_RXADR_n** and **USB_TXADR_n** registers. Also the USB module is supported the independent **USB_RXCNT_n** and **USB_TXCNT_n** registers to set USB data receive and transmission packet size for each endpoint. Anyway, user needs plan the packet buffer about start address and packet data size for all available endpoints. Refer the "USB Packet Buffer Map Example" diagram for related information. (n = USB endpoint index)

The USB module can detect the packet buffer receiving memory up boundary independently for each EP_n. User can set the endpoint packet buffer memory size boundary by setting **USB_BLSIZE_n** and **USB_BLNUM_n** registers. When USB data receives and sends to the endpoint packet buffer, the OVRF flag will be asserted if the access SRAM memory address of packet data is over the related endpoint packet buffer up boundary.

The endpoint packet buffer up address boundary is set by how many memory blocks. The **USB_BLNUM_n** register is used to set the memory block number. The **USB_BLSIZE_n** register is used to set the memory size of one memory block. There are 2-byte or 32-byte settings for user choosing. When user selects 2-byte, the valid memory size range is from 2-byte to 62-byte. When user selects 32-byte, the valid memory size range is from 32-byte to 512-byte. The following is the formula of receiving memory size boundary.

Receiving memory size boundary = **USB_BLSIZE_n** x **USB_BLNUM_n**

19.9.4. USB Buffer Mode

The USB module is supported receive and transmission independent single buffer mode and double buffer mode for USB packet buffer access. User can set the USB buffer mode independent for each EP_n by setting **USB_DBM_n**, **USB_RXEN_n** and **USB_TXEN_n** registers. This double buffer mode can be enabled or disabled for each endpoint EP1~7 independently by setting **USB_DBM_n** register. The EP0 endpoint is not supported double buffer mode. (n = USB endpoint index)

The double buffer mode can be conjunction with the two packet buffers to do as one receive or transmission packet buffer and raise the USB data transfer efficiency. Those two packet buffers are set by the same EP_n

endpoint of **USB_RXADR_n** / **USB_RXCNT_n** and **USB_TXADR_n** / **USB_TXCNT_n** registers to define the USB Buffer-0 and Buffer-1.

For single buffer mode, the **USB_RXSEQ_n** and **USB_TXSEQ_n** registers are used as the USB receive and transmit endpoint sequence bits. Refer the section of "[USB Transaction Packet](#)" for more information. For double buffer mode, the **USB_RXSEQ_n** and **USB_TXSEQ_n** registers are used to set the initial USB buffer index for an OUT and IN transaction. Also these two bits can read back to indicate which USB buffer is operating by hardware. Value 0 is indicated USB Buffer-0 and value 1 is indicated USB Buffer-1.

The following table is showing the USB buffer mode control setting.

Table 19-3. USB Buffer Mode Control

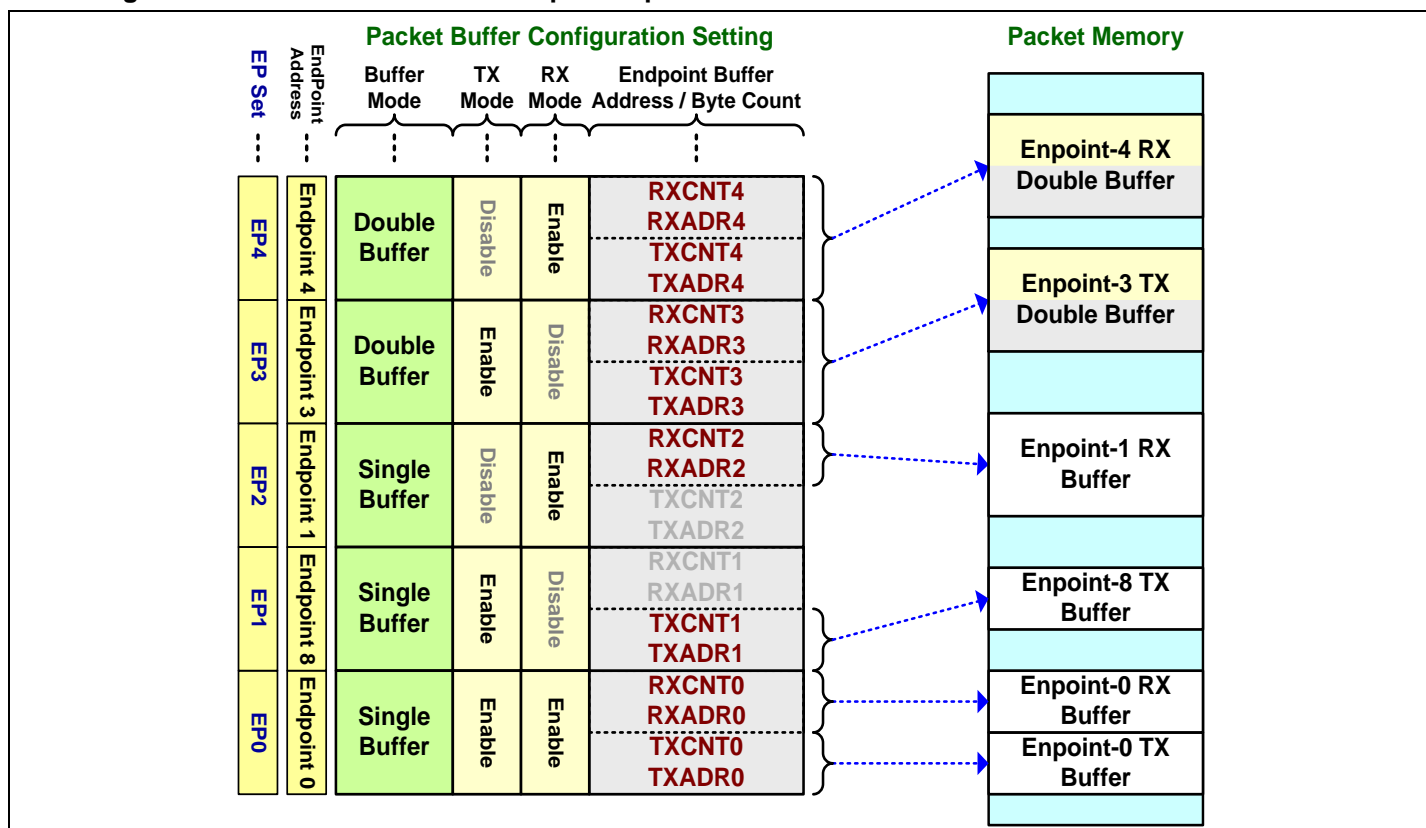
Buffer Mode	USB Register						
	DBM _n	RXEN _n	TXEN _n	RXADR _n RXCNT _n	TXADR _n TXCNT _n	RXSEQ _n	TXSEQ _n
Single Buffer RX only	0	1	0	RX Buffer	-	RX SEQ	-
Single Buffer TX only	0	0	1	-	TX Buffer	-	TX SEQ
Single Buffer RX/TX	0	1	1	RX Buffer	TX Buffer	RX SEQ	TX SEQ
Double Buffer RX	1	1	0	RX Buffer-0	RX Buffer-1	BUF IDX	-
Double Buffer TX	1	0	1	TX Buffer-0	TX Buffer-1	-	BUF IDX

<Sign> "n" ~ Endpoint Index, "-" ~ Invalid, "SEQ" ~ Sequence bit,
"BUF IDX" ~ Indication the operating USB Buffer index 0 or 1.

The following diagram is showing the example of USB packet buffer map.

- EP0 : Endpoint-0, Independent Single RX and TX Buffer
- EP1 : Endpoint-8, Single TX Buffer only
- EP2 : Endpoint-1, Single RX Buffer only
- EP3 : Endpoint-3, Double TX Buffer
- EP4 : Endpoint-4, Double RX Buffer

Figure 19-11. USB Packet Buffer Map Example



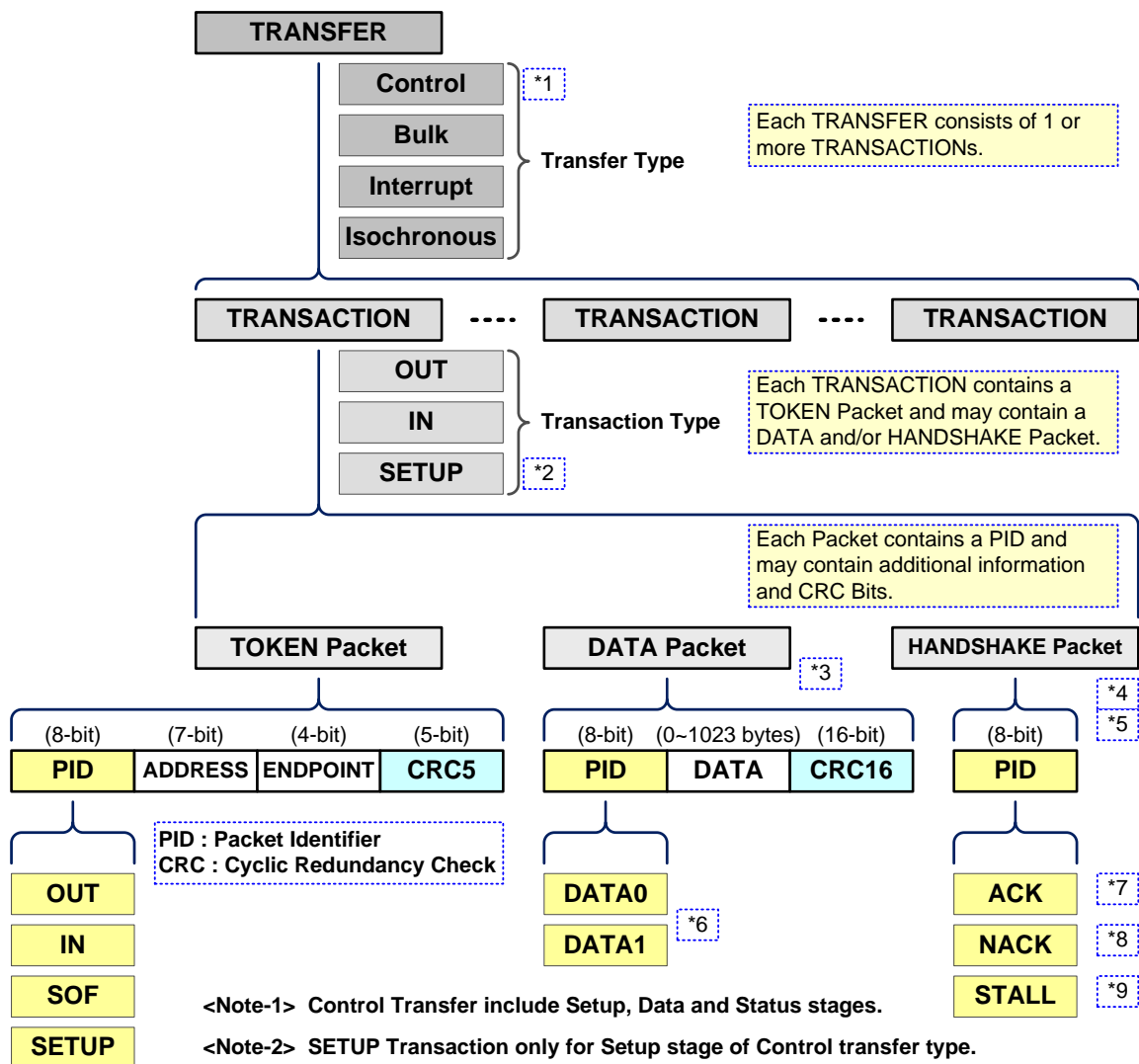
19.10. USB Operation

Before the USB data transfer between MG32F02U series (USB device) and external USB host, the chip must do USB macro initial of DM/DP/V33 pin mode setting, USB power and clock control, USB analog front end configuration, USB endpoint configuration and USB package buffer mode setting. Refer the sections of “[USB Power and Clock Control](#)” and “[USB Fundamental Control](#)” for more information about the USB macro initial.

19.10.1. USB Transfer Construction

Each **TRANSFER** consists of 1 or more **TRANSACTION**s. Each **TRANSACTION** contains a **TOKEN** packet and may contain a **DATA** and/or **HANDSHAKE** packet. Each packet contains a **PID** and may contain additional information and **CRC** bits. The following diagram is showing the USB transfer construction diagram.

Figure 19-12. USB Transfer Construction (Full Speed)



<Note-1> Control Transfer include Setup, Data and Status stages.

<Note-2> SETUP Transaction only for Setup stage of Control transfer type.

<Note-3> A NAK or STALL Handshake packet is sent instead of the DATA if the addressed endpoint is not valid for IN Transaction.

<Note-4> No HANDSHAKE Packet for Isochronous transfer type.

<Note-5> A NAK or STALL Handshake packet is sent instead of the ACK if the addressed endpoint is not valid for OUT Transaction.

<Note-6> Isochronous transfer type does not support data toggle sequence and only use DATA0 PID for data packet.

<Note-7> ACK : a host or device has received data without error

<Note-8> NACK : the device is busy or has no data to return

<Note-9> STALL : an unsupported control request, control request failed, or endpoint failed

19.10.2. USB Transfer Type

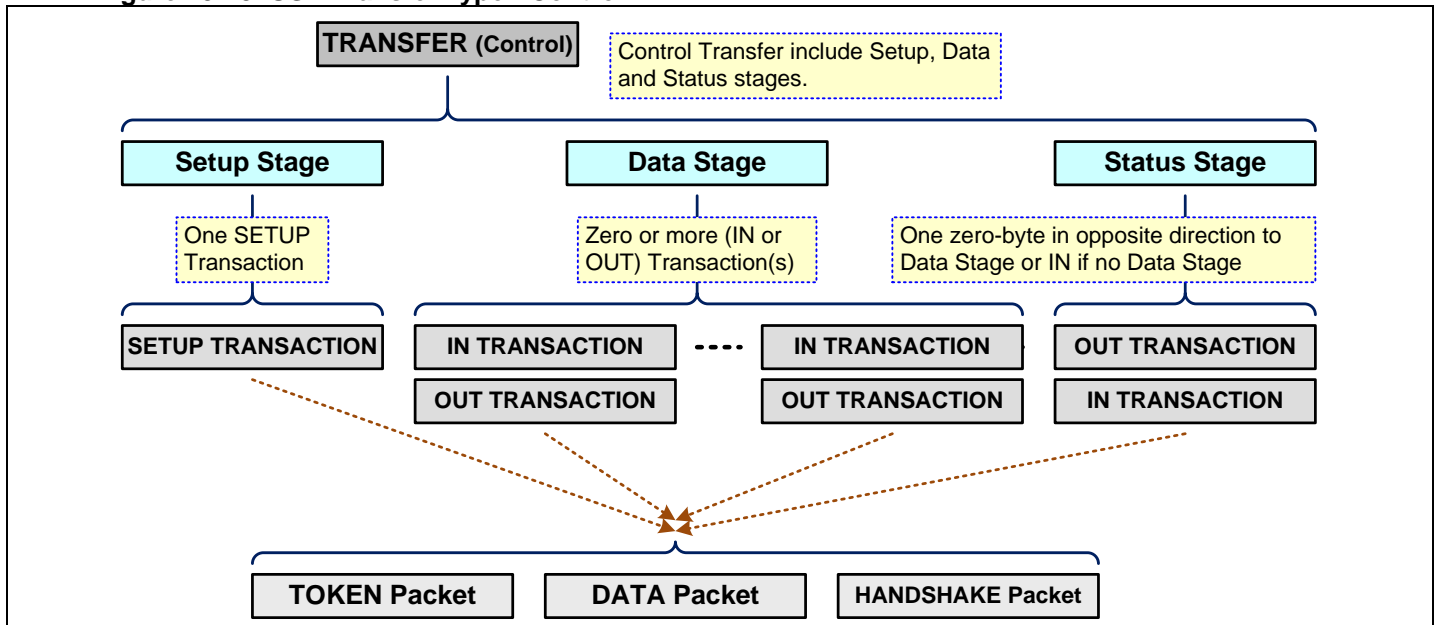
For MG32F02U series, the chip can support **Control**, **Interrupt**, **Bulk** and **Isochronous** transfer endpoint types. The EP0 endpoint is only able to configure as Control transfer type. The EP1~7 endpoints can configure as Interrupt/Bulk or Isochronous transfer type. During power-on enumeration state, each used endpoint must define the transfer type for USB host.

- **Control transfer type**

It is including of **Setup**, **Data** and **Status** stages those are separately combined of **SETUP**, **IN** and **OUT** transaction(s). Each transaction may be including of **TOKEN**, **DATA** and **HANDSHAKE** packets. The following diagram is showing the USB Control transfer type data structure.

For **Setup** stage, the PID of **DATA** packet of **SETUP** transaction is always **DATA0**. Then **Data** stage, the PID of first **DATA** packet of first **IN** or **OUT** transaction is always **DATA1**. Following **IN** or **OUT** transaction(s), the PID of **DATA** packet will be toggled between **DATA0** and **DATA1** continuously. Last **Status** stage, the PID of **DATA** packet of **IN** or **OUT** transaction is always **DATA1**.

Figure 19-13. USB Transfer Type - Control

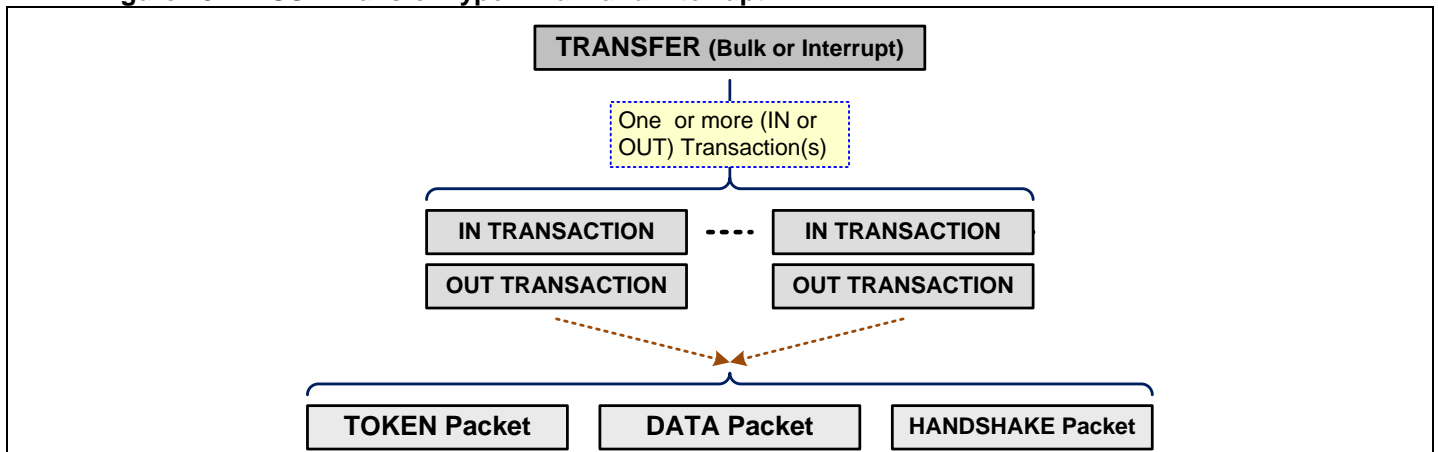


- **Bulk and Interrupt transfer type**

It is only combined one type transaction of **IN** or **OUT**. Each transaction is including of **TOKEN**, **DATA** and **HANDSHAKE** packets. The following diagram is showing the USB Bulk and Interrupt transfer type data structure.

Usually, the PID of first **DATA** packet of first **IN** or **OUT** transaction is **DATA0**. Following **IN** or **OUT** transaction(s), the PID of **DATA** packet will be toggled between **DATA0** and **DATA1** continuously.

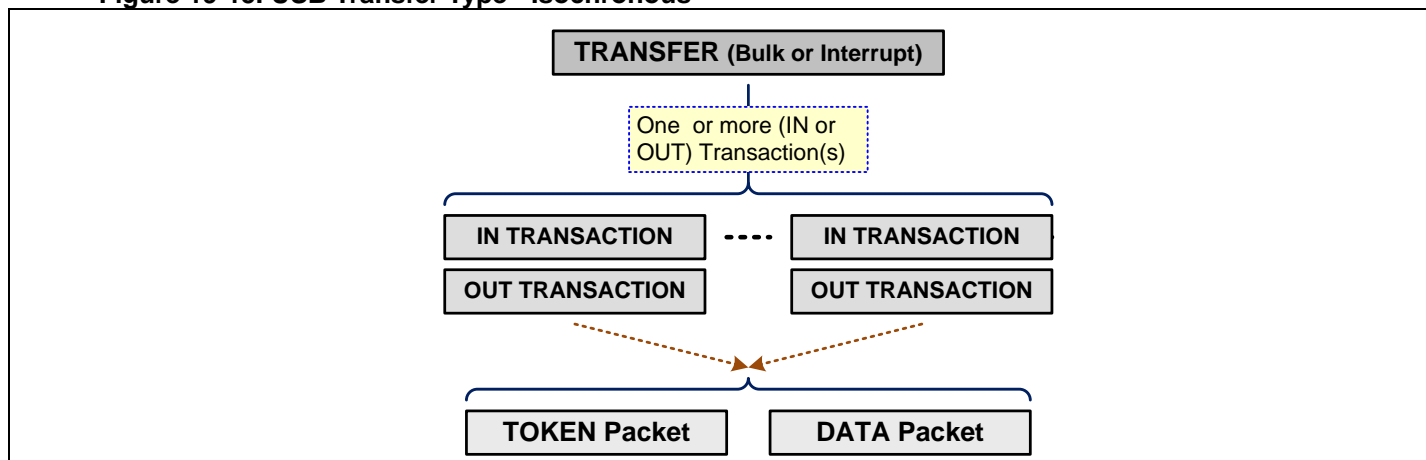
Figure 19-14. USB Transfer Type – Bulk and Interrupt



- **Isochronous transfer type**

It is only combined one type transaction of **IN** or **OUT**. Each transaction is including of **TOKEN** and **DATA** packets. The following diagram is showing the USB Isochronous transfer type data structure.

Figure 19-15. USB Transfer Type - Isochronous



The **Isochronous** transfer type is usually used to transfer USB packet data for a fixed and accurate time request. It does not support the **HANDSHAKE** packet. It also does not support **DATA0** and **DATA1** PID sequencing toggle in **DATA** packets and always use **DATA0** PID.

For the USB data transfer bandwidth limitation, it does not support to retry transmission for failed transactions and not have an expected **HANDSHAKE** packet to feedback **NAK** state.

19.10.3. USB Transaction Packet

For the Control, Bulk/Interrupt and Isochronous transfer types, each type may be combined of **SETUP**, **IN** and **OUT** transaction(s). Each transaction may be including of **TOKEN**, **DATA** and **HANDSHAKE** packets. User can get the USB device address in **USB_ADR** register. During bus enumeration, it is written with a unique value assigned by the host.

- **IN Transaction**

For **IN** transaction, user must prepare the USB host request packet data to the packet buffer. After the request packet data have prepared on the packet buffer, user needs to set **USB_TXMCn** bit to notify the chip that the packet data are ready.

When the USB module detects an **IN** transaction from the receiving **PID** of **TOKEN** packet and the **USB_TXMCn** bit is set, the chip will send the USB packet data from USB packet buffer to external USB host. The chip will clear this bit after all the USB packet data transmission and the memory release operation have been finished.

Sometimes the **USB_TXMCn** bit is not set as the request packet data have not yet ready, the chip will send **NAK** or **STALL** according to the setting of **USB_TXSTLn** register. When the **USB_TXSTLn** bit is set, the transmit endpoint will respond with a **STALL** handshake to a valid **IN** token. When this bit is clear, the transmit endpoint will **NAK**. The **USB_TXMCn** and **USB_TXSTLn** bits can be written by firmware if the **USB_TXSA_LCKn** bit is set when written along with the new value.

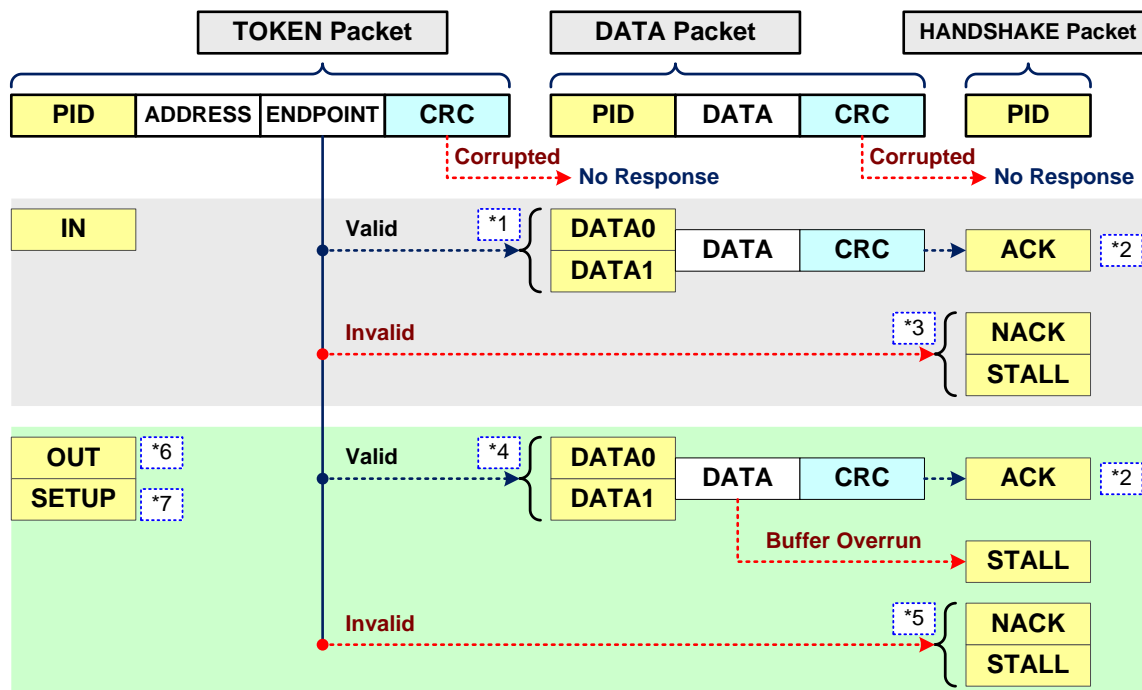
For **IN** transaction, user can set the USB transmit endpoint sequence bit to decide sending **DATA0** or **DATA1** in the next **PID** and toggled on following valid **ACK** handshake of **IN** transaction by setting **USB_TXSEQn** register. This bit can be written by firmware if the **USB_TXS_LCKn** bit is set when written along with the new **USB_TXSEQn** value.

When MG32F02U (USB device) receives an invalid endpoint for **IN** transaction, a **NAK** or **STALL** handshake packet is sent instead of the data packet. User can configure to send **NAK** or **STALL** by setting **USB_TXSTLn** register.

For **Control** endpoint (endpoint 0) **IN/OUT** transaction, user can set **No-response** by setting **USB_NORS_EN** register. When this bit is disabled, the chip (USB device) will send **ACK/NAK/STALL** packet in **IN/OUT** transaction. When this bit is enabled, the chip will be just only response **ACK** packet with **SETUP** transaction but no response with EP0 **IN/OUT** transaction. This bit will be clear by hardware when device receive an **SETUP** token.

The following diagram is showing the USB transaction packet structure and flow.

Figure 19-16. USB Transaction Packet Structure and Flow



<Note-1> Starts sending a DATA0 or DATA1 PID according to (USB_TXSEQn).

<Note-2> The USB host receives the NACK and will retry the transaction.

<Note-3> A NAK or STALL handshake packet is sent instead of the data packet, according to (USB_TXSTLn) and (USB_NORS_EN) registers.

<Note-4> The expected data toggle bit value is set in (USB_RXSEQn) for next received data.

<Note-5> A NAK or STALL handshake packet is sent instead of the data packet, according to (USB_RXSTLn) and (USB_NORS_EN) registers.

<Note-6> Check (USB_SETUPnF) bit to distinguish OUT or SETUP transaction.

<Note-7> SETUP transaction of Control endpoint is very similar to OUT except that the values of (USB_TXSEQn) and (USB_RXSEQn) bits are set to 1 and 0.

● OUT/SETUP Transaction

When the USB module detects an **OUT** or **SETUP** transaction from the receiving PID of **TOKEN** packet and the **USB_RXMCn** bit is set, the chip will load the USB packet data from external USB host to USB packet buffer. The **USB_RXMCn** bit is set to indicate the USB packet buffer is empty and ready to receive USB packet data. The chip will clear this bit after all the USB packet data reception and the memory release operation have been finished.

User can check the **USB_SETUPnF** flag to distinguish **OUT** or **SETUP** transaction. The **SETUP** transaction of **Control** endpoint is very similar to **OUT** transaction except that the values of **USB_TXSEQn** and **USB_RXSEQn** bits are set to 1 and 0.

For **OUT** or **SETUP** transaction, user can get the USB host sending packet data from the packet buffer. After the packet data have been taken out of the packet buffer, user needs to set **USB_RXMCn** bit to notify the chip that the USB packet buffer is empty and ready to receive next USB packet data.

Sometimes the **USB_RXMCn** bit is not set as the input packet data have not yet taken out, the chip will send **NAK** or **STALL** according to the setting of **USB_RXSTLn** register. When the **USB_RXSTLn** bit is set, the receive endpoint will respond with a **STALL** handshake to a valid **OUT** token. When this bit is clear, the receive endpoint will **NAK**. This bit does not affect the reception of **SETUP** tokens by a **Control** endpoint (endpoint 0). The **USB_TXMCn** and **USB_TXSTLn** bits can be written by firmware if the **USB_RXSA_LCKn** bit is set when written along with the new value.

For **OUT** or **SETUP** transaction, user can set the USB receive endpoint sequence bit to detect expected **DATA0** or **DATA1** in the next PID and toggled on following valid **ACK** handshake of **OUT** or **SETUP** transaction by setting **USB_RXSEQn** register. This bit can be written by firmware if the **USB_RXS_LCKn** bit is set when written along with the new **USB_RXSEQn** value.

When MG32F02U (USB device) receives an invalid endpoint for **OUT** or **SETUP** transaction, a **NAK** or **STALL** handshake packet is sent instead of the data packet. User can configure to send **NAK** or **STALL** by setting

USB_RXSTLn register. If the external USB host receives the **NACK** in **HANDSHAKE** packet, USB host will retry the transaction.

For **Control** endpoint (endpoint 0) **IN/OUT** transaction, user can set **No-response** by setting **USB_NORS_EN** register. When this bit is disabled, the chip (USB device) will send **ACK/NAK/STALL** packet in **IN/OUT** transaction. When this bit is enabled, the chip will be just only response **ACK** packet with **SETUP** transaction but no response with EP0 IN/OUT transaction. This bit will be clear by hardware when device receive an **SETUP** token.

19.10.4. USB Bus Reset and Endpoint Reset

The chip will monitor the USB bus to detect the bus **RESET** event. When the USB bus is kept **SE0** state more 10us by USB host, the chip will be detection and receiving a bus **RESET** event. The USB device must be reset the USB macro about related settings and state flags of all endpoint. And user needs to initial the endpoint 0 (fixed EP0) and prepare to receive the USB host next commands. When the chip is in **SUSPEND** (L2) mode and receives bus **RESET** event, the USB device must be waked up and going to do the previous USB macro reset process. Refer the section of "[USB Suspend and Resume](#)" about more information about USB resume and wake up.

When the USB device is receiving the "ENDPOINT_HALT" request by "Set Feature" USB command, the USB device can set the **USB_RXSTLn** and **USB_TXSTLn** bits for related endpoint to halt the endpoint operation. Also the USB host can recover the endpoint by sending the "ENDPOINT_HALT" request in "Clear Feature" USB command, the USB device needs to clear the **USB_RXSTLn** and **USB_TXSTLn** bits for related endpoint. Then user needs to reset the endpoint by setting **USB_RXRSTn** and **USB_TXRSTn** bits to recover the endpoint operation. The **USB_RXRSTn** and **USB_TXRSTn** bits are used to reset the receiving and transmission block of related endpoint. Also that will clear the status and flags of related endpoint.

19.10.5. USB Suspend and Resume

For low power consumption request, the **SUSPEND** and **RESUME** modes are supported to stop and wakeup the USB device.

In **SUSPEND** mode, the USB bus offered average current consumption is required and must be under 2.5mA. The MG32F02U (USB device) will be entering **SUSPEND** mode when the external USB host has stopped any communication and kept idle state on the USB bus for more than 3 ms time. For the condition, the chip will detect the **SUSPEND** event and assert the SUSF flag (**USB_SUSF**). User can configure the **SUSPEND** event detection mode by setting **USB_SUS_MDS** register. When selects 'Both', it will assert SUSF flag if the chip detects J state or SE1 state. When selects 'JS', it will assert SUSF flag only if the chip detects J state.

When the chip is entering **SUSPEND** mode, the USB device can be waked up by the USB host sending **RESUME** event or **RESET** event. The **RESUME** event is that the USB host keeps K state time more than 20ms. The USB device must be able to detect the bus state event if it is entering **SUSPEND** mode. Also it can be recovered by the USB device itself from some others' event or detection.

Refer the table of "USB Bus State and Flag" in the section of "Bus State" for more information about USB bus state.

19.10.6. USB Remote Wakeup

For application request, it is allowed that the USB device to wakeup the USB host by the **Remote Wakeup** process. By the process, the USB device must send the **RESUME** event and keep the state between 1 ms and 15 ms time. When the USB host detects the **RESUME** event from a USB device, the USB host will keep the state more than 20 ms time.

User can trigger to start the **Remote Wakeup** process by setting **USB_RWK_TRG** register. The MG32F02U can support the **Remote Wakeup** process for hardware or software mode by setting **USB_RWK_MDS** register. When selects 'Software' the **Remote Wakeup** length will be controlled by user to set the interval of **USB_RWK_MDS** value from 1 to 0. When selects 'Hardware' the **Remote Wakeup** length will be decided by hardware setting. User can select the delay time by setting **USB_RWK_DSEL** register.

19.10.7. USB Link Power Management

This USB Link Power Management (**LPM**) which is similar to the existing suspend/resume, but has transitional latencies of tens of microseconds between power states. It defines a new, fast mechanism for transitioning the bus on a root port from an enabled state (**L0**) to a new Sleep state (**L1**). It also defines new, fast timing requirements for resuming a bus from **L1** back to **L0**, and defines the method for extending the existing USB protocol to accommodate explicit **L1** entry. The **L0** to **L1** (link Sleep) transition is actively initiated by the host using a new USB-defined transaction. This allows a device to immediately detect the intent of the host and respond immediately. The **L1** to **L0** transition is similar to the USB 2.0 resume, as it can be initiated by either the host port or the device (i.e. remote wake). However, the duration of resume signaling is redefined to be three orders of magnitude faster.

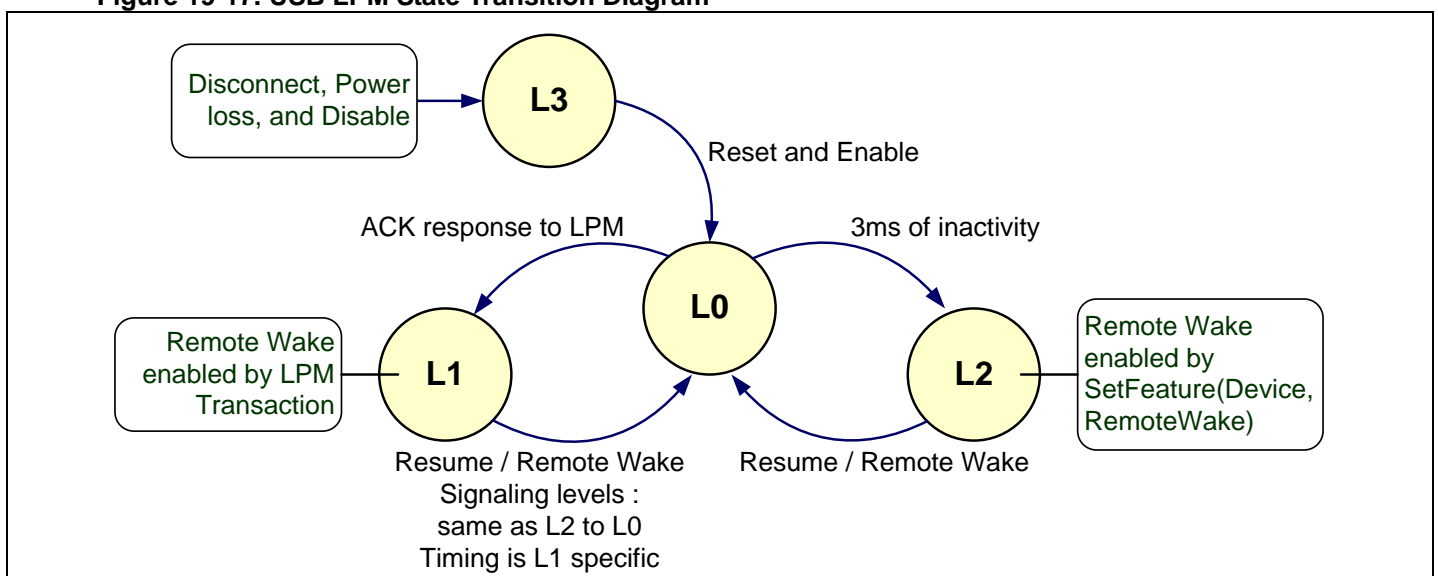
Please refer the “USB2.0 Link Power Management ECN” document for more information about USB Link Power Management. The following table is showing the definitions of USB LPM states.

Table 19-4. USB Link Power Manager States

LPM State	Description
L0 (On)	In this state, the port is enabled for propagation of transaction signaling traffic. A port in L0 is either actively transmitting or receiving data (L0-Active) or able to do so but not currently transmitting or receiving information (L0-Idle). While in this state Start-of-Frame (SOF) packets are issued by the host at a rate corresponding to the speed of the client device (1ms for full-speed).
L1 (Sleep)	L1 is similar to L2 (below) but supports finer granularity in use. When in L1, the line state is identical to L2. Entry to L1 is started by a request to a hub or host port to transition to L1. A LPM transaction is sent to the downstream device. The requested transition can only occur if the device response with an ACK handshake. Exit from L1 is via remote wake, resume signaling, reset signaling or disconnect. L1 does not impose any specific power draw requirements (from VBUS) on the attached device as L2 does. Either the host or device can initiate resume signaling when in L1. Although the signaling levels of resume are the same as L2, the duration of the signaling and transitional latencies associated with the L1 to L0 transition are much shorter.
L2 (Suspend)	This is the formalized name for USB 2.0 Suspend. Entry to L2 is nominally triggered by a command to a hub or host port to transition to suspend, at which point the port ceases repeating signaling down the port. The device discovers the suspend condition via observing 3ms of inactivity. The resultant line state is either Low or Full-speed idle. L2 also imposes power draw requirements (from VBUS) on the attached device. Exit from this state is via remote wake, resume signaling, reset signaling or disconnect.
L3 (Off)	In this state, the port is not capable of performing any data signaling. It corresponds to the powered-off, disconnected, and disabled states.

According to USB specifications, it defines an additional link power management state **L1** (Sleep). The following figure is showing the USB LPM state transition diagram.

Figure 19-17. USB LPM State Transition Diagram



❖ LPM State: L1 (Sleep)

The **L1** (Sleep) LPM state differs from the **L2** (Suspend) state in two ways: the transitional latencies into and out

of the state are much shorter (than **L2**) and there are no explicit power draw requirements on the attached device (L2 requires a suspended device to reduce its power draw from VBUS). Although there are no specification-mandated power requirements, the device won't be monitoring bus activity when its upstream port is in **L1** and should take the opportunity to conserve power whenever possible. The host will provide an indication to the device on the minimum time it will be driving resume if it (the host) initiates **L1** exit. The indicator is set by the host based on its current workload policies and will be in the range 50 to 1200 μ s. The device can use the value of this indicator to deploy its own power efficiency optimizations, based on the responsiveness indicated by the host.

L1 supplements the existing **L2** state by utilizing most of the existing suspend/resume infrastructure (signaling levels, line states while in the state, etc.) but, provides much faster transitional latencies between **L1** and **L0** (On).

The following table is showing the comparisons between the **L1** and **L2** LPM states.

Table 19-5. USB Summary Similarities/Differences between L1 and L2

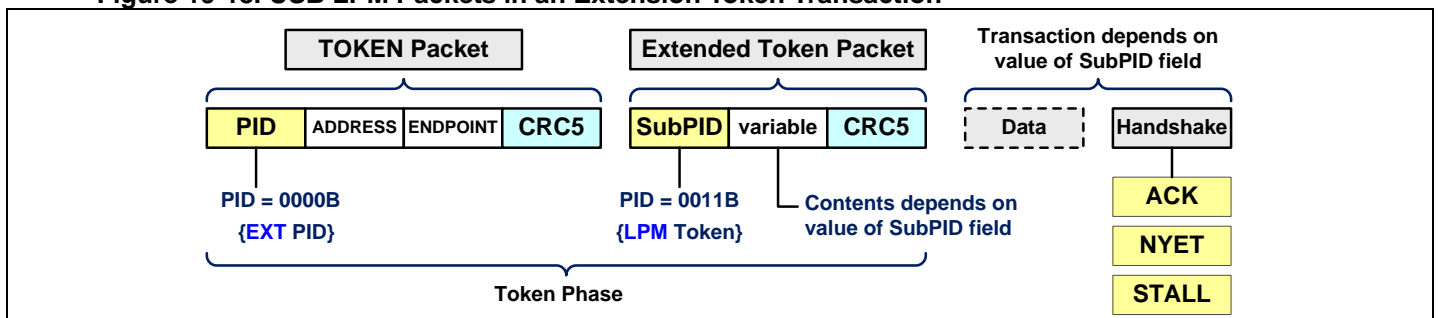
Item	L1 (Sleep)	L2 (Suspend)
Entry	Explicitly entered via LPM extended transaction	Implicitly entered via 3ms of link inactivity
Exit	Device or host-initiated via resume signaling; Remote-wake can be (optionally) enabled/disabled via the LPM transaction.	Device- or host-initiated via resume signaling; Device-initiated resumes can be (optionally) enabled/disabled by software
Signaling	Low and Full- speed idle	Low and Full-speed idle
Latencies	Entry: ~10 μ s Exit: ~70 μ s to 1ms (host-specific)	Entry: ~3ms Exit: >30ms (OS-dependent)
Link Power Consumption	~0.6mW (data line- pull-ups)	~0.6mW (data line- pull-ups)
Device Power Consumption	Device power consumption level is application and implementation specific	Device power consumption is limited to: ≤ 500 μ A or ≤ 2.5 mA
Hot Removal	Natively detected per USB2 mechanisms	Natively detected per USB 2.0 mechanisms

❖ LPM Control and Protocol

The USB module supports Link Power Management (**LPM**). User can enable **LPM** function by setting **USB_LPM_EN** register.

About the USB **LPM** Token acknowledge return state, user can set the acknowledge state in **USB_LPM_ACK** register. When this bit is set 'NYET' and the USB device is receiving the valid **LPM** Token, the return state will be NYET. When this bit is set 'ACK' and the USB device is receiving the valid **LPM** Token, the return state will be ACK. When the USB device is receiving the invalid **LPM** Token, the return state will be STALL. The following figure is showing the USB LPM packets in an Extension Token Transaction.

Figure 19-18. USB LPM Packets in an Extension Token Transaction



For LPM transaction, there are two important fields of information those are defined in LPM Token **bmAttributes** field and sent from the host to the device. The first is **HIRD** (or **BESL**) which is USB host initiated **Resume** duration. On host-initiated resume, the host drives resume signaling for a specified period of time (**HIRD**). The host can choose the duration in the **HIRD** which is a 4-bit encoded value that translates to a range of 50 microseconds to 1.2 milliseconds on a linear scale. About **HIRD**, a new value called Best Effort Service latency (**BESL**) which can replace the concept of **HIRD** and redefine to represent a range of 125 μ s to 10ms.

The second is **bRemoteWake** bit which indicates whether remote wake is enabled on the device. When the USB device is receiving a valid **LPM** Token, user can get the information of **BESL** and **bRemoteWake** in **USB_LPM_BESL** and **USB_LPM_RWK** registers.

19.11. USB DMA Operation

19.11.1. DMA Module Configure

When the chip supports a DMA (direct memory access) controller, user can configure the DMA setting of transferred source/destination devices, channel request arbitration and others in the DMA module before a DMA data transaction. The DMA source and the destination can be memory or peripheral.

Refer the DMA chapter for more detail information about the DMA module configuration.

19.11.2. USB DMA Control

The USB module is only supported DMA function for endpoint EP3 and EP4. User can enable and select the data transmission/receiving endpoint(s) independently in **USB_DMA_TXSELO** and **USB_DMA_RXSELO** registers. After DMA configuration is finished, user needs to set these DMA enable and select bits.

Finally, the related channel request start bit of **DMA_CHn_REQ** is necessary to be set to start the DMA transaction (n = DMA channel index). Then the transferred source/destination devices will assert the RX/TX request signal to DMA controller and the DMA controller will assert the acknowledge signal to the request source/destination devices. At the time, the data transferred connection is built for DMA transaction.

- **USB RX to DMA**

The **USB_DMA_RXSELO** register bit is used to enable DMA data transfer from USB receiving input to DMA destination.

During the DMA transaction cycle, the received data flag of **USB_RXDnF** is masked by hardware. (n = USB endpoint index 3 or 4)

- **USB TX from DMA**

The **USB_DMA_TXSELO** register bit is used to enable DMA data transfer from DMA source to USB transmitted output.

During the DMA transaction cycle, the transmitted data flag of **USB_TXDnF** is masked by hardware. (n = USB endpoint index 3 or 4)

19.11.3. USB Interrupt Flag Control during DMA

During DMA operation cycle, the module's interrupt flags will control and act three types as following table. One is masked during the DMA process. Another is to disable the DMA function after the flag has asserted. Others are normally as same as the action of not processing in DMA operation.

Table 19-6. USB Interrupt Flag Control for DMA Function

Action	Mask the Flag during DMA Process	DMA Disable after the Flag asserted (*1)	Normal Control		
Peripheral	(Data flow flags)	(Error/Detect flags)	(Other flags)		
USB	USB_RXDnF USB_TXDnF	USB_ERRF USB_BUSF USB_LPMF	USB_SOF USB_ESOF USB_EPnF USB_OVRF USB_SETUPF USB_NORSF USB_BTSTF USB_CRCF	USB_SUSF USB_RSMF USB_RSTF USB_RWKF USB_SE1F USB_LPMSTF USB_LPMNYF	USB_STOVWnF USB_EDOVWnF USB_SETUPnF USB_RXNAKnF USB_RXSTLnF USB_ISOOWnF USB_TXNAKnF USB_TXSTLnF USB_ISOTXEnF

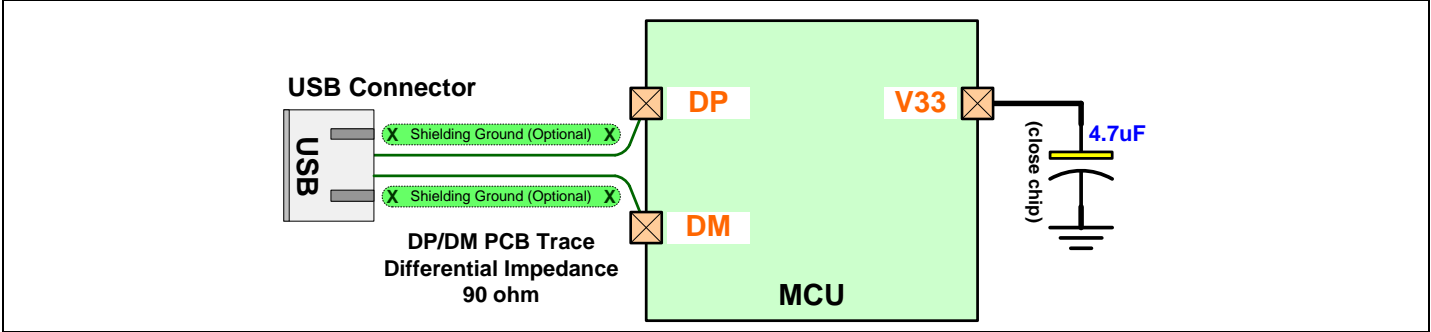
<Note> *1 : When the flag is asserted, it will not force peripheral DMA disabled if the related interrupt enable bit is not enabled.

19.12. USB Application Circuit

For the USB series chip, it implements a USB device and an internal 3.3V regulator is built-in for the USB transceiver (XCVR). One external decoupling and bypass 4.7uF capacitor is necessary on **V33** power pin for the internal 3.3V regulator output.

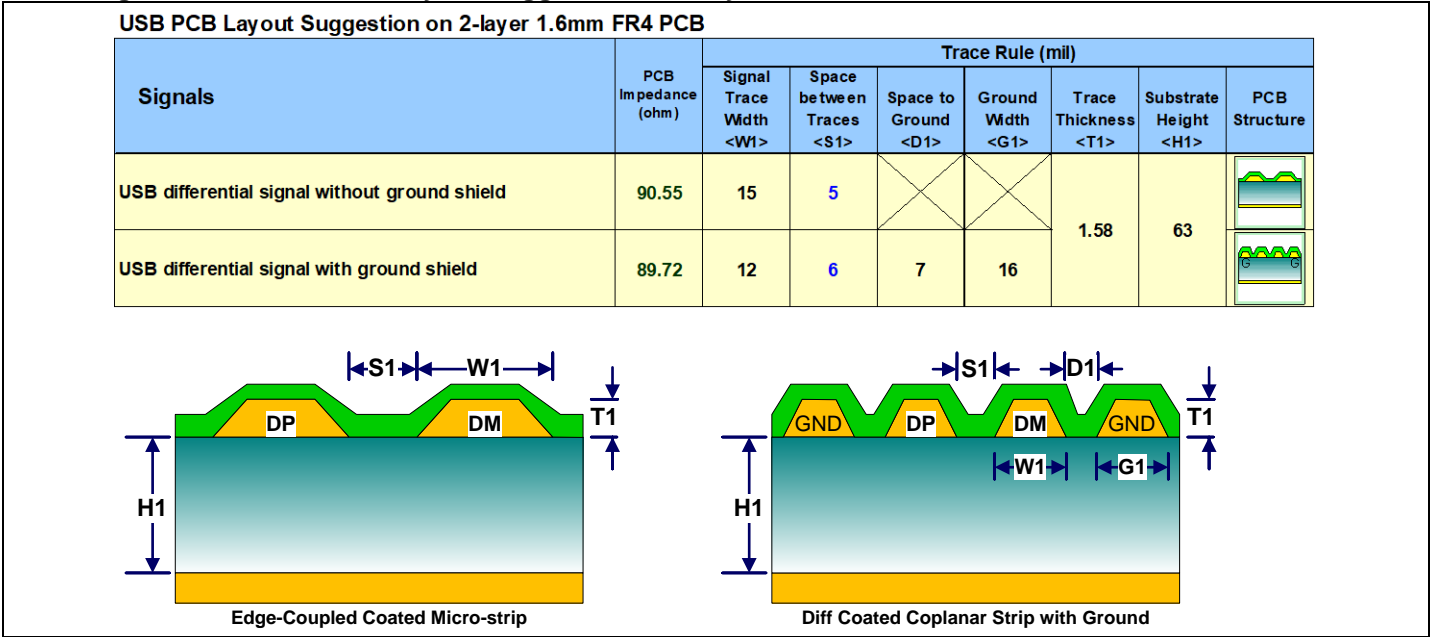
The **DP** and **DM** pins are a couple of differential signals to connect the USB host through the USB connector and cable without external serial resistors. On application PCB, the DP and DM signals' trace layout are necessary to rout with differential impedance 90 ohm to get better USB Eve-pattern for USB data transaction.

Figure 19-19. USB Application Circuit



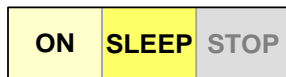
The DP and DM signals' trace layout is suggested by the structure of 'Edge-Coupled Coated Micro-strip' or 'Diff Coated Coplanar Strip with Ground' as following figure. The following table and figure are the DP and DM signals' layout suggestion reference on 2-layer 1.6mm FR4 PCB.

Figure 19-20. USB PCB Layout Suggestion on 2-layer 1.6mm FR4 PCB



20. CAN (Controller Area Network)

20.1. Introduction



The module can be running in ON and SLEEP modes only.

The chip provides a Controller Area Network (CAN) interface with bitrates up to 1 Mbit/s. It is a serial communications protocol which supports full CAN 2.0 and distributed real time control with a very high level of security. Its domain of application ranges from high speed networks to low cost multiplex wiring.

The CAN module builds in the receive message FIFOs, transmit message buffers and data register to improve transmit and receive communication performance.

Notify: The sign of (x = module index, n= RX FIFO index {0 ~ 1}, m= Message Buffer index {0 ~ 2}, z= RX Acceptance Filter index {0 ~ 5}) is using for Registers, Signals and Pins in the descriptions of this chapter.

[EX]: **CANx_EN**, **CANx_RXnF**, **CANx_TXm_REG**, **CANx_AFzR0** ~ x indicates module index number, n indicates RX FIFO, m= Message Buffer index, z= RX Acceptance Filter index.

20.2. Features

- Supports full CAN 2.0 – both 2.0A and 2.0B
 - Supports both 11-bit and 29-bit identifiers
 - Upwards compatible with the CAN FD protocol
- Supports bit rates from less than 125Kbit/s to more than 1Mbit/s
- Three transmit message buffer
 - Configurable transmit priority by request sequence or message identifier order
- Two sets of receive message FIFO
 - Three stage receive message buffer for each FIFO
 - Support one combined message FIFO from all message buffers
- Six sets of acceptance filter with identifier mask
 - Configurable Mask or List mode for each acceptance filter
 - Configurable single 32-bit filter or dual 16-bit filters for each acceptance filter
- Error manage with interrupts
- Support silent and loop-back mode for self-test operation

20.3. Implementation

20.3.1. Chip Implementation

The following table is showing the implementation of CAN module.

Table 20-1. CAN Implementation

Chip	CAN 2.0	Message Buffer		Acceptance Filter		
	FD	TX	RX	Sets	Mode	Bits
MG32F02N128/N064 MG32F02K128/K064	Upwards compatible	3 sets	FIFO x 2	6	Mask/List	32/16-bit
MG32F02A128/A064 MG32F02U128/U064 MG32F02V032	Not Implemented					

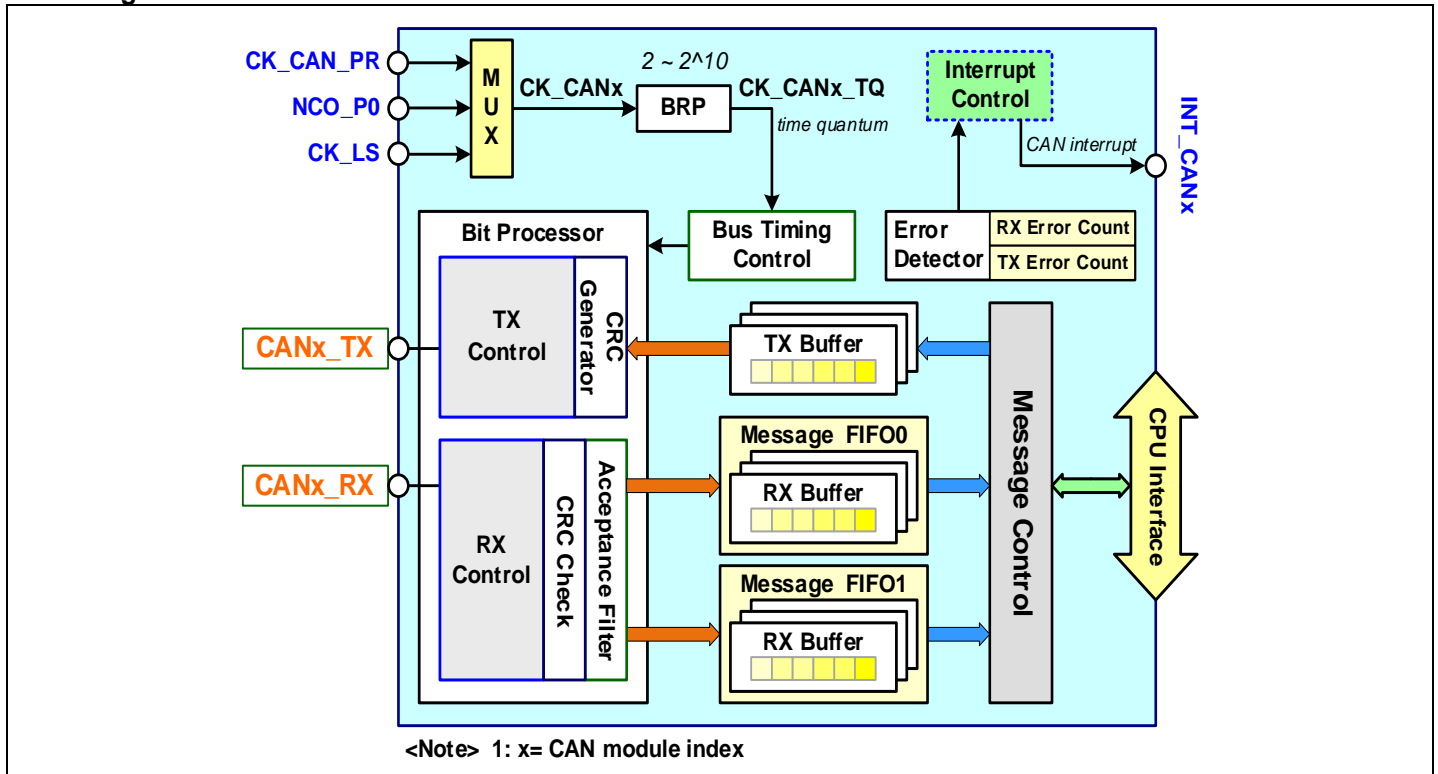
<Note> V: Implemented; FIFO: three stage receive message buffer for each FIFO

20.4. Control Block

The CAN module builds in the receive message FIFOs, transmit message buffers for CAN data communication. The configurable Acceptance Filter can filter the CAN data streams which are with the accepted CAN ID for user request. The module is including of an Error Detector to manage the CAN bus error events.

The following diagram is showing the CAN Control block.

Figure 20-1. CAN Main Control Block



20.5. IO Lines

20.5.1. IO Signals

- CANx_TX, CANx_RX

The **CANx_TX** line is used for CAN data transmission signal and **CANx_RX** is used for CAN data receiving signal. The **CANx_TX** and **CANx_RX** IO lines are used to connect the external CAN transceiver device to transfer data to other CAN devices on CAN bus.

20.5.2. IO Configure

User must configure the related IO pins in order to use the IO lines of this module. The IO operation modes, output high speed option, pull-high option, output drive strength, IO deglitch filter and input inverse selection are programmable by pin independent. Please refer the section of “[IO Mode](#)” in GPIO chapter for more detail descriptions of IO mode configuration.

Each IO signal may be mapped and selected on several IO pins by configuring the IO AFS matrix. Please refer the section of “[Alternate Function Select](#)” in GPIO chapter for more detail descriptions of IO AFS configuration. About the actual IO pin AFS information, please refer the section of “[Pin Alternate Functions Selected Table](#)” in Pin Description chapter of the chip Data Sheet.

20.6. Enabling and Clock

20.6.1. CAN Global Enable

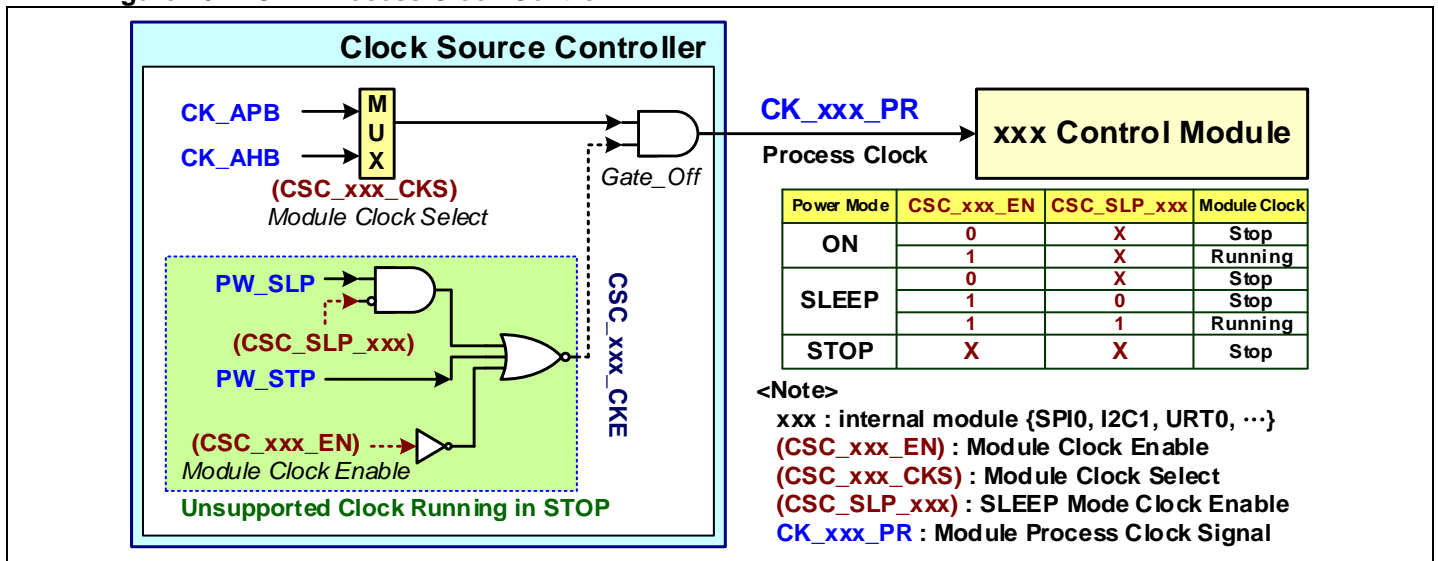
There is a global enable bit of **CANx_EN** for all functions of this module. When this bit is disabled, all the CAN functions are not working.

20.6.2. CAN Clock Control

● Module Process Clock

The module process clock of **CK_CANx_PR** is using for the interface control logic between APB bus and the module. It is coming from CSC (Clock Source Controller) module. It can be enabled in **CSC_CANx_EN** register and select the clock source from APB clock or AHB clock in **CSC_CANx_CKS** register. User can plan the module clock is running or not beforehand for chip entering **SLEEP** mode by setting **CSC_SLP_CANx** register. Refer the System Clock chapter for more information.

Figure 20-2. CAN Process Clock Control



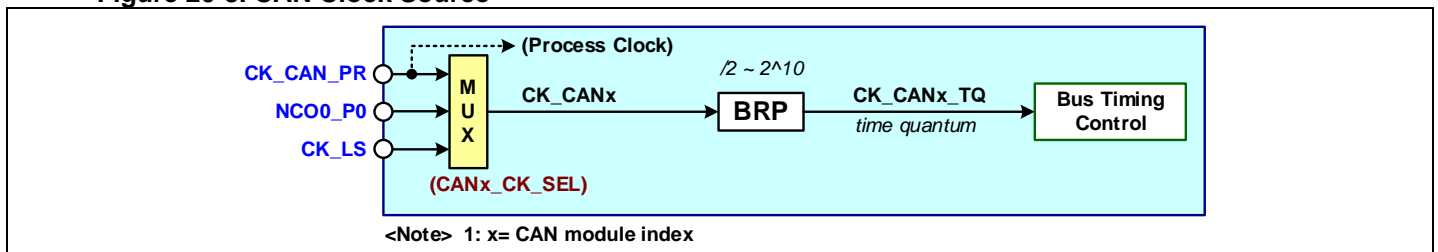
● Module Internal Clock

The CAN module is built in a Baud-Rate generator to output the time quantum clock for received data signal and the clock source for transmitted data signal. The Baud-Rate counter is implemented 10-bit counter.

User can select the internal clock source of **CK_CANx** from the module process clock of **CK_CANx_PR**, **NCO_P0** or internal ILRCO clock of **CK_ILRCO** by setting **CANx_CK_SEL** register. The CAN module is able to use the clock of **CK_CANx** as the input clock for Baud-Rate generator.

The following diagram is showing the CAN clock source block.

Figure 20-3. CAN Clock Source



20.7. Interrupt and Event

20.7.1. CAN Interrupt Control and Status

There is one signal of **INT_CANx** to be generated in this CAN control module. **INT_CANx** sends to EXIC External Interrupt Controller to do as an interrupt event.

These interrupt flags are using for the interrupt service routine (ISR) flow control. Generally these flags are set by hardware and software clears them when the serviced job is completed in the related ISR. Each interrupt flag has one interrupt enable bit. User can enable or disable it. There is one interrupt all enable bit of **CANx_IEA** to enable or disable all the interrupt sources for this module.

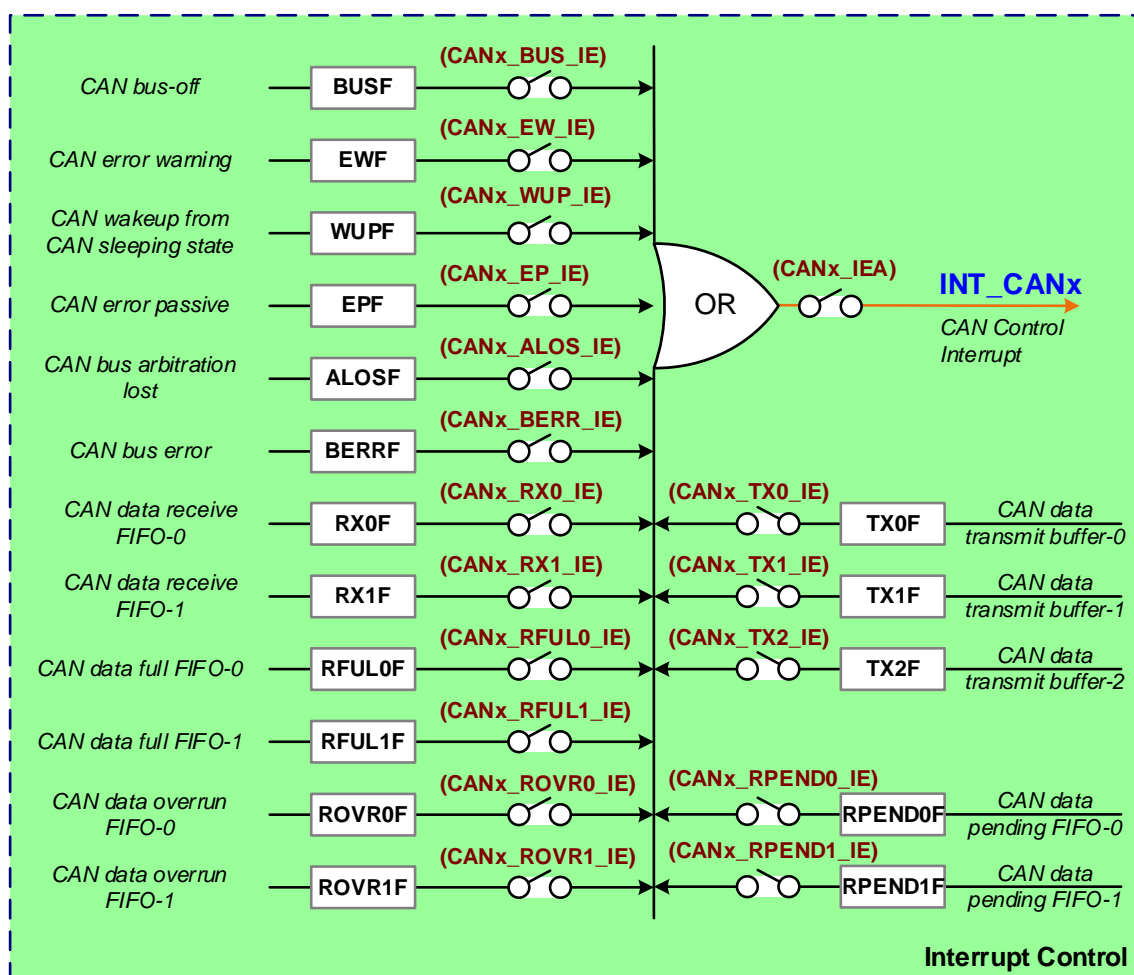
There are some status bits those are reading only to provide internal control status. Refer the register descriptions of the related status bits for more information.

20.7.2. CAN Interrupt Flags

Refer the register descriptions for more information about the related interrupt flags and interrupt enable bit.

The following diagrams are showing the CAN interrupt control block.

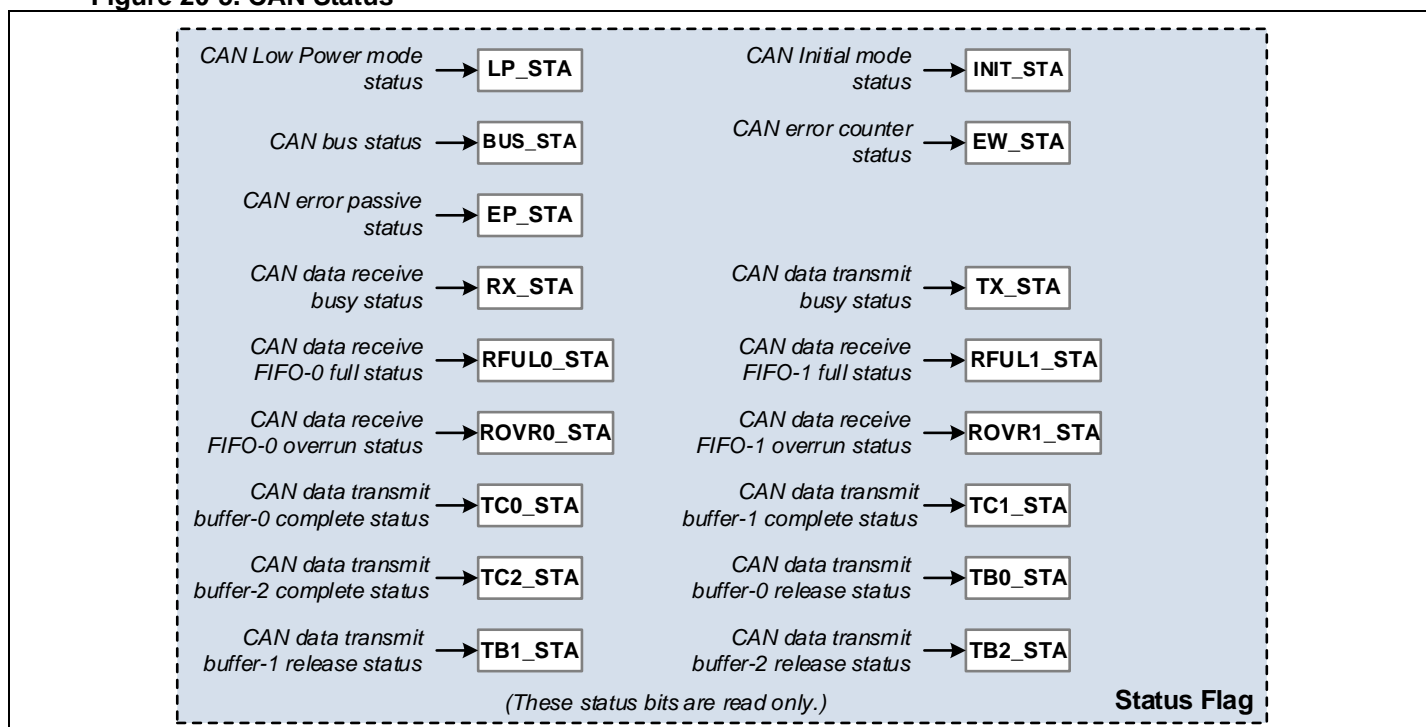
Figure 20-4. CAN Interrupt



<Note> 1: x= CAN module index

The following diagram is showing the CAN status diagram.

Figure 20-5. CAN Status



❖ Interrupt Flags

The following interrupt flag bits are able to write or read. They are set by hardware and can clear by software writing 1 except the registers of **CANx_RPEND0F** and **CANx_RPEND1F**.

● BUSF

CAN bus-off interrupt flag (**CANx_BUSF**). There is a related interrupt enable register bit of **CANx_BUS_IE**. This bit is set when happened bus-off (**CANx_BUS_STA**) is.

● EWF

CAN error warning interrupt flag (**CANx_EWF**). There is a related interrupt enable register bit of **CANx_EW_IE**. This bit is set when the error counters is reached the error warning limit (**CANx_ERR_STA**) by setting **CANx_EW_LIM** register.

● WUPF

CAN wakeup from CAN low power state interrupt flag (**CANx_WUPF**). There is a related interrupt enable register bit of **CANx_WUP_IE**. This bit is set when bus activity is detected during the CAN module is sleeping.

● EPF

CAN error passive interrupt flag (**CANx_EPF**). There is a related interrupt enable register bit of **CANx_BUS_IE**. This bit is set when the module re-enters error active state after being in error passive state or when at least one error counter exceeds the protocol defined level of 127.

● ALOSF

CAN bus arbitration lost interrupt flag (**CANx_ALOSF**). There is a related interrupt enable register bit of **CANx_ALOS_IE**. This bit is set when the module loses arbitration and becomes a receiver.

● BERRF

CAN bus error interrupt flag (**CANx_BERRF**). There is a related interrupt enable register bit of **CANx_BERR_IE**. This bit is set when the module detects an error on the CAN bus.

● RX0F, RX1F

CAN data receive message FIFO-0,1 interrupt flag (**CANx_RX0F**, **CANx_RX1F**). There are the related interrupt enable register bits of **CANx_RX0_IE**, **CANx_RX1_IE**. When received valid CAN data into any one message buffer, this flag is set after the message buffer has updated finished.

● RFUL0F, RFUL1F

CAN receive message FIFO-0,1 full interrupt flag (**CANx_RFUL0F**, **CANx_RFUL1F**). There are the related interrupt enable register bits of **CANx_RFUL0_IE**, **CANx_RFUL1_IE**. When receive FIFO is full, this bit is set.

● ROVR0F, ROVR1F

CAN receive message FIFO-0,1 overrun interrupt flag (**CANx_ROVR0F**, **CANx_ROVR1F**). There are the related interrupt enable register bits of **CANx_ROVR0_IE**, **CANx_ROVR1_IE**. When receive FIFO has not enough space and the new message is receiving, this bit is set.

- **RPEND0F, RPEND1F**

CAN receive message FIFO-0,1 overrun interrupt flag (**CANx_RPEND0F**, **CANx_RPEND1F**). There are the related interrupt enable register bits of **CANx_RPEND0_IE**, **CANx_RPEND1_IE**. When the receive data FIFO is with remained data for any message buffer, set this flag. These bits are read only. When the receive data FIFO is empty, the chip will clear this flag automatically.

- **TX0F, TX1F, TX2F**

CAN data transmit interrupt flag for TX message buffer-0,1,2 (**CANx_TX0F**, **CANx_TX1F**, **CANx_TX2F**). There is a related interrupt enable register bit of **CANx_TX0_IE**, **CANx_TX1_IE** and **CANx_TX2_IE**. When transmitted message buffer has shifted out complete through shift buffer, this flag is set.

❖ Hardware Statue

The following status bits are read only. These bits are set by hardware and cleared by hardware.

- **LP_STA**

CAN **Low Power** mode status (**CANx_LP_STA**). When this bit is set, it indicates the CAN module is entered CAN **Low Power** mode.

- **INIT_STA**

CAN **Initial** mode status (**CANx_INIT_STA**). When this bit is set, it indicates the CAN module is entered CAN **Initial** mode.

- **BUS_STA**

CAN bus status (**CANx_BUS_STA**). When this bit is set, it indicates the module is in Bus-Off state and is not involved in bus activities. When this bit is cleared, it indicates the module is involved in bus activities.

- **EW_STA**

CAN error counter status (**CANx_EW_STA**). This bit is set when the error counters was equal or large the error warning limit by setting **CANx_EW_LIM** register.

- **EP_STA**

CAN error passive status (**CANx_EP_STA**). This bit is set when the module re-enters error active state after being in error passive state or when at least one error counter exceeds the protocol defined level of 127.

- **RX_STA**

CAN data receive busy status (**CANx_RX_STA**). When this bit is set, it indicates the module is in the process of receiving a message.

- **RFUL0_STA, RFUL1_STA**

CAN data receive FIFO full status for RX message FIFO-0,1 (**CANx_RFUL0_STA**, **CANx_RFUL1_STA**). When this bit is set, it indicates the receive FIFO is full.

- **ROVR0_STA, ROVR1_STA**

CAN data receive FIFO overrun status for RX message FIFO-0,1 (**CANx_ROVR0_STA**, **CANx_ROVRL1_STA**). When receive FIFO has not enough space and the new message is receiving, this bit is set.

- **TX_STA**

CAN data transmit busy status (**CANx_TX_STA**). When this bit is set, it indicates the module is in the process of transmitting a message.

- **TC0_STA, TC1_STA, TC2_STA**

CAN data transmit complete status for TX message buffer-0,1,2 (**CANx_TC0_STA**, **CANx_TC1_STA**, **CANx_TC2_STA**). This bit is set when the last requested transmission has been successfully completed.

- **TB0_STA, TB1_STA, TB2_STA**

CAN data transmit buffer status for TX message buffer-0,1,2 (**CANx_TB0_STA**, **CANx_TB1_STA**, **CANx_TB2_STA**). When this bit is set, it indicates the transmit buffer is released. The CPU may write a message to the transmit buffer. When this bit is cleared, it indicates the transmit buffer is locked. The CPU cannot access the transmit buffer because a message is either waiting for transmission or is in the process of being transmitted.

20.8. CAN Module IO Control

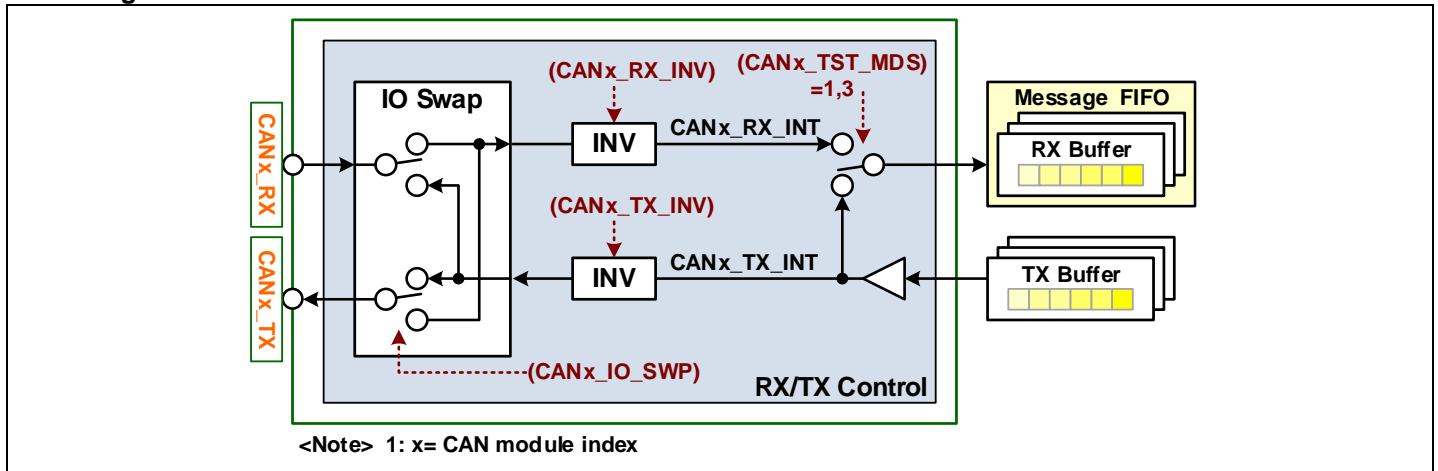
This module provides 2 data signals – **CANx_RX** and **CANx_TX** to do as CAN data receiving and transmission for CAN bus communication.

20.8.1. CAN IO Control

The **CANx_IO_SWP** register is used to enable to swap the signals of **CANx_RX** and **CANx_TX**. The registers of **CANx_RX_INV** and **CANx_TX_INV** are used to inverse the related signals of **CANx_RX** and **CANx_TX**.

The following diagram is showing the CAN module IO control block.

Figure 20-6. CAN Module IO Control



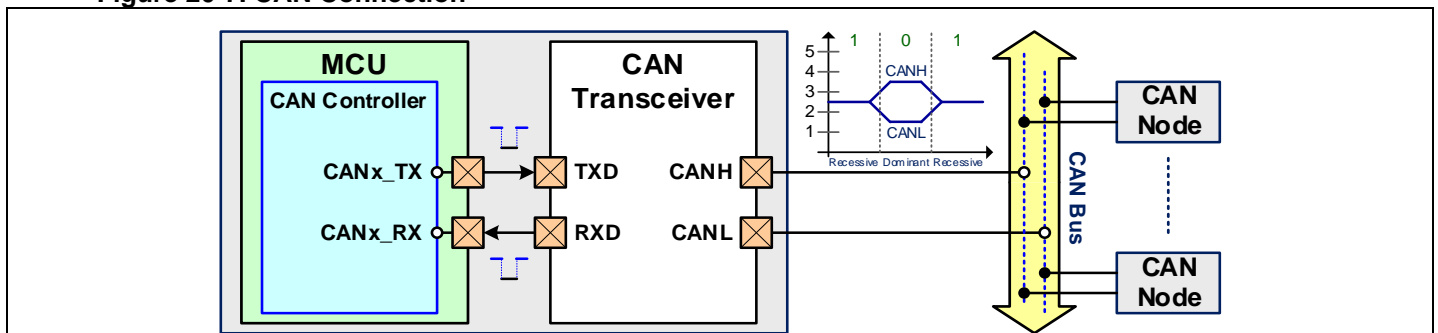
20.8.2. CAN IO Mode

Usually user can set the IO mode of the using IO pins to push-pull or open-drain for CAN output signals. Also user can set the IO mode of the using IO pins to digital input or open-drain for CAN input signals. For CAN bidirectional signal, the IO mode of the using IO pins need set to push-pull or open-drain. When selects push-pull mode, the CAN data signal will output by push-pull mode and be open-drain for idle or input state. When selects open-drain mode, the CAN data signal will always be open-drain for idle, input or output state.

20.9. CAN Connection for Application

The following diagram is showing the CAN application connection. For the following descriptions, **CANx_TX** is the CAN data transmission signal and **CANx_RX** is the CAN data receiving signal. These two signals are used to connect the external CAN transceiver device which is used the differential signals of **CANH** and **CANL** to connect to CAN bus. So the chip can transfer data to other CAN devices on CAN bus through the external CAN transceiver device.

Figure 20-7. CAN Connection



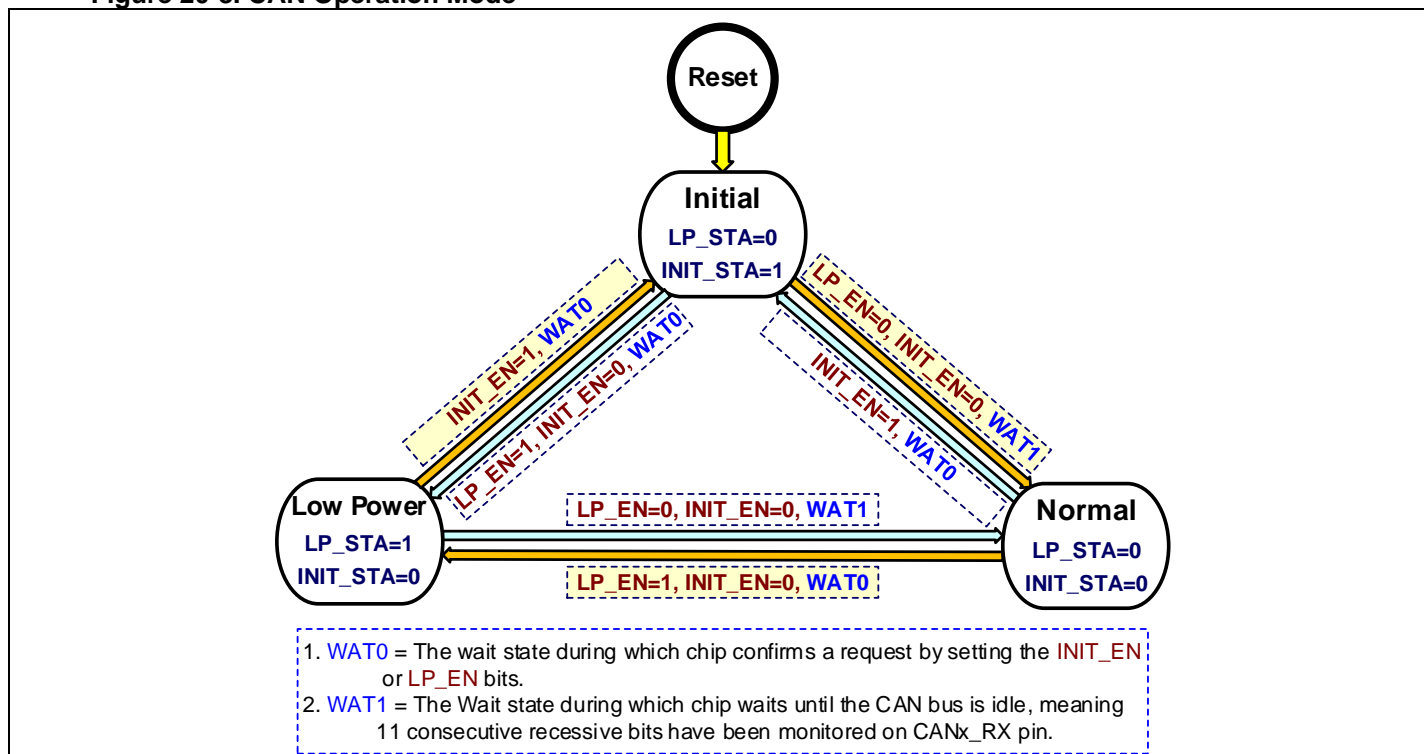
20.10. CAN Fundamental Control

20.10.1. Operation Mode

The module implements three CAN operation modes of **Initial** mode, **Normal** mode and **Low Power** mode. User can select the three mode by setting registers of **CANx_INIT_EN** and **CANx_LP_EN**.

The following diagram is showing the CAN operation mode flow diagram.

Figure 20-8. CAN Operation Mode



● Initial Mode

The module will automatically enter **Initial** mode after chip reset. Also user can set **CANx_INIT_EN** to enable and the module is forced to enter CAN **Initial** mode. To leave **Initial** mode, user can set **CANx_INIT_EN** to disable and enter **Normal** mode or **Low Power** mode by setting **CANx_LP_EN** register.

In **Initial** mode, any message currently being transmitted or received is aborted. Some CAN configure registers are only changed in this mode. User can refer the CAN register descriptions in Register Definition Guide for more information. The software initialization can be done while the module is in **Initial** mode. The filter mode configuration must be configured before entering **Normal** Mode.

When the module is entering **Initial** mode, the most part of CAN control status registers are not changed. The following table is showing the CAN control status registers action after entering **Initial** mode.

Table 20-2. CAN Initial Mode Action

Action	CAN Register							
	TXn_REQ	TBn_STA	TCn_STA	TX_STA	INIT_STA	LP_STA	BUS_STA	EW_STA
Enter Initial Mode	Clear 0	Set 1	Keep	Set 1	Set 1	Clear 0	By State	By State
	BRP	SJW	TSEG1	TSEG2	EW_LIM	TXERR_CNT	RXERR_CNT	
	RW	RW	RW	RW	RW	RW	RW	
	BERRF	ALOSF	EPF	WUPF	EWf	BUSF	TXnF	
	Keep or By State	Keep	Keep	Keep	Keep	Keep	Keep	
	RPENDnF	ROVRnF	RFULnF	RXnF	ROVRn_STA	RFULn_STA		
	Keep	Keep	Keep	Keep	Keep	Keep		

<Sign> n: register index, RW: register write access enable

● Normal Mode

When both **CANx_INIT_EN** and **CANx_LP_EN** registers are disabled, the module is in **Normal** mode.

After the software initialization is complete, user must request the module to enter **Normal** mode to start CAN data receiving and transmission on the CAN bus. When the module is entering **Normal** mode, it will be synchronized on the CAN bus and wait for the occurrence of a sequence of 11 consecutive recessive bits.

● Low Power Mode

Also user can enable **Low Power** mode to reduce power consumption by setting **CANx_INIT_EN** bit to disable and **CANx_LP_EN** bit to enable. If **CANx_INIT_EN** is enabled, the **CANx_LP_EN** bit is invalid to set 'Enable'.

In **Low Power** mode, the module enters its sleep mode provided no CAN interrupt is pending and there is no bus activity. When the module has entered **Low Power** mode, there are only **CANx_EN**, **CANx_INIT_EN** and **CANx_LP_EN** bits which can be access.

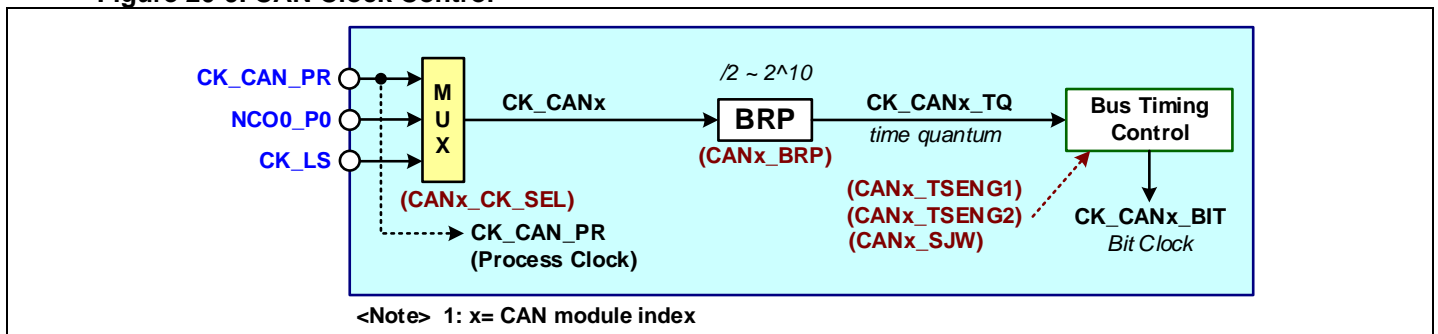
If there is bus activity or an interrupt is pending, the wake-up procedure is executed automatically. There are two modes of 'Soft' and 'Auto' to select for CAN wakeup to **Normal** mode by setting **CANx_WUP_MDS** register. When selects 'Soft', the chip is left CAN **Low Power** mode on software request by cleared **CAN0_LP_EN** bit. When selects 'Auto', the chip will be left CAN **Low Power** mode automatically by RX signal active detection.

20.10.2. CAN Clock

User can select the internal clock source of **CK_CANx** from the module process clock of **CK_CANx_PR**, **NCO_P0** or internal ILRCO clock of **CK_ILRCO** by setting **CANx_CK_SEL** register. The CAN module is able to use the clock of **CK_CANx** as the input clock for Baud-Rate generator.

The following diagram is showing the CAN clock control block.

Figure 20-9. CAN Clock Control



The CAN module is built in a Baud-Rate generator to output the internal clock **CK_CANx_TQ** as the time quantum clock for received data signal and the clock source for transmitted data signal. User can configure the Baud-Rate generator by setting **CANx_BRP** register. The Baud-Rate counter is implemented 10-bit counter.

User can calculate time quantum clock frequency by following formula.

$$CK_CANx_TQ = \frac{CK_CANx}{(CANx_BRP+1)}$$

20.10.3. CAN Bit Timing

The CAN module is built in a Bus Timing Controller which can input the time quantum clock **CK_CANx_TQ** to detect or generate the CAN bus bit timing for received or transmitted data. User can configure the CAN bit timing frame by setting **CANx_TENG1** and **CANx_TENG2** registers.

*[Notify]: Both **CANx_TENG1** and **CANx_TENG2** registers can only be written in **Initial** mode. In **Normal** mode, they are read only..*

The CAN specification describes the bit period as being composed of a Synchronization segment, a Propagation segment and two Phase Buffers. The Synchronization segment TSYN is fixed one TQ time. The TSEG1 equals the Propagation segment plus the first Phase Buffer and it is equal to $TQ * (CANx_TENG1 + 1)$. The TSEG2 is the second Phase Buffer and it is equal to $TQ * (CANx_TENG2 + 1)$.

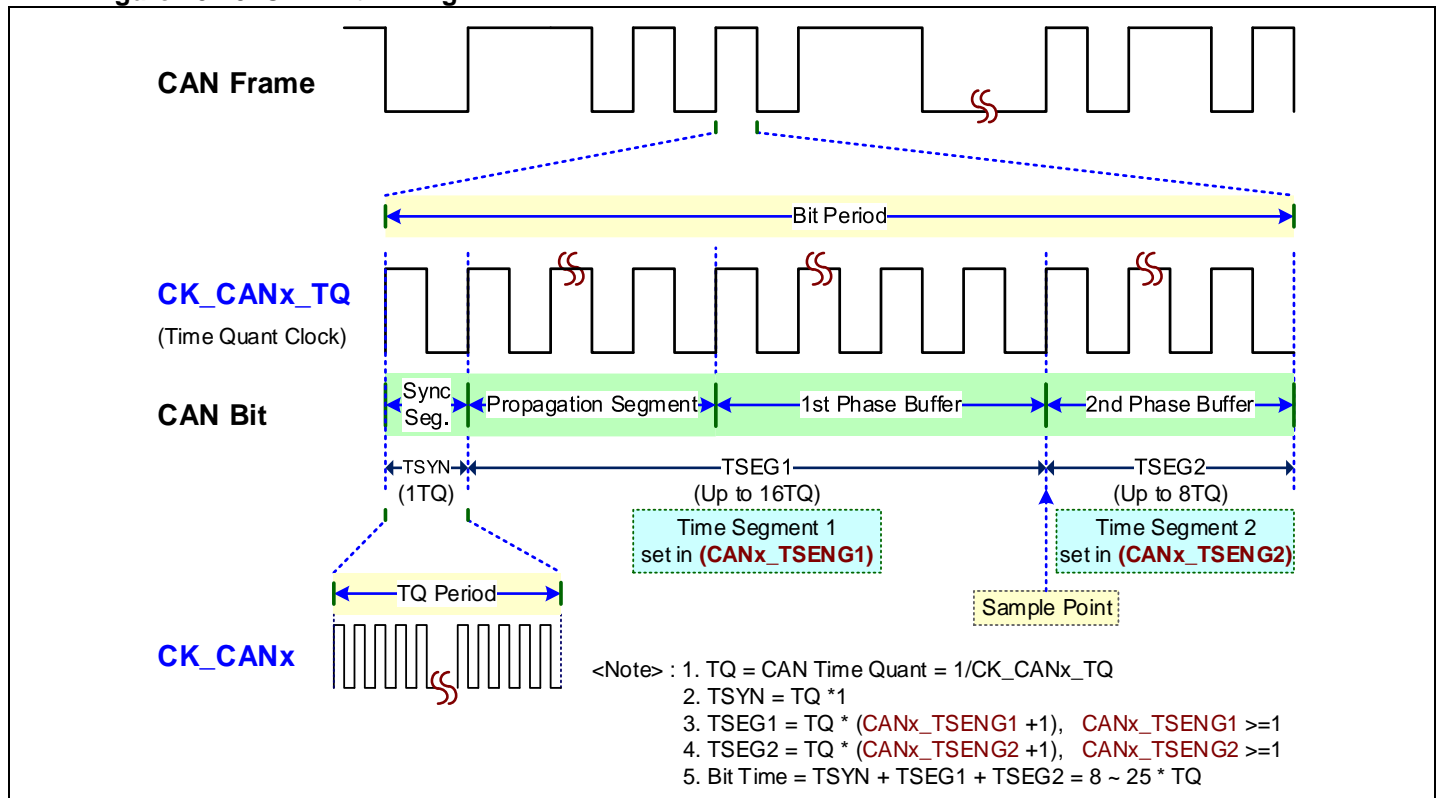
User can calculate one Bit Time by following formula. The Bit Time range is 8 to 25 of TQ time.

$$TQ = 1 / CK_CANx_TQ$$

$$Bit\ Time = TSYN + TSEG1 + TSEG2$$

The following diagram is showing the CAN Bit timing diagram.

Figure 20-10. CAN Bit Timing



20.10.5. CAN Error Detect

The CAN module can detect Bit error, Form error, Stuff error, CRC error, Acknowledge error and others. User can get the error information from the registers of **CANx_ECC_ERR**, **CANx_ECC_DIR** and **CANx_ECC_SEG**. When a bus error occurs, the module will capture the error information into these registers. These registers' value are kept until one of the register of **CANx_ECC_ERR**, **CANx_ECC_DIR** and **CANx_ECC_SEG** has been read.

User can check Bit error, Form error, Stuff error and CRC error by directly reading the error code capture register of **CANx_ECC_ERR**. The **CANx_ECC_DIR** can be read to indicate the error code capture of direction. The **CANx_ECC_SEG** can be read to indicate the error segment code.

User can check Acknowledge error when the register of **CANx_ECC_ERR** is reading 'Others', the register of **CANx_ECC_DIR** is reading 'TX' and the register of **CANx_ECC_SEG** is reading 'Acknowledge' (code = 0x19).

There is one CAN arbitration lost bit position capture register of **CANx_ECC_ALC**. This register records the bit position at which arbitration was lost. When bus arbitration lost, an arbitration lost interrupt is generated (If the related interrupt enable bit is enabled.) and the current bit position of the bit processor is captured into this register. The register value is kept until the register has been read.

● Bit Error

A node sending a bit on the bus shall also monitor the bus. A bit error is detected at that bit time, when the bit value that is monitored differs from the bit value sent.

Exceptions: a dominant bit shall not lead to a bit error when a recessive information bit is sent during arbitration, or a recessive bit is sent during ACK slot. A node sending a passive error flag and detecting a dominant bit shall not interpret this as a bit error.

● Form Error

A form error shall be detected when a fixed-form bit field contains one or more illegal bits.

Exception: A receiver monitoring a dominant bit at the last bit of EOF, or any node monitoring a dominant bit at the last bit of error delimiter or of overload delimiter, shall not interpret this as a form error.

● Stuff Error

A stuff error shall be detected at the bit time of the sixth consecutive equal bit level in a frame field that is coded by the method of bit stuffing. It shall be regarded as a form error, not as a stuff error, when a fixed stuff bit in the CRC field of an FD Frame is not at its expected value.

● CRC Error

The CRC sequence shall consist of the result of the CRC calculation of the transmitter. The receivers shall calculate the CRC in the same way as the transmitter. A CRC error shall be detected when the calculated CRC sequence does not equal the received one. In FD Frames, a mismatch between the counted stuff bits and the received stuff count shall be treated as a CRC error.

● Acknowledge Error

An acknowledge error shall be detected by a transmitter whenever it does not monitor a dominant bit during ACK slot.

The following table is showing the register definitions of and **CANx_ECC_ALC** and **CANx_ECC_SEG**.

Table 20-3. CAN Error Code Capture Value Definitions

CAN Register		CAN_ECC_ALC	CAN_ECC_SEG
Hex	Dec	Arbitration Lost Definitions	Error Segment Code Definitions
0x00	0	Arbitration lost in 1st bit of identifier (ID.28)	
0x01	1	Arbitration lost in 2nd bit of identifier (ID.27)	
0x02	2	Arbitration lost in 3rd bit of identifier (ID.26)	ID.28 to ID.21
0x03	3	Arbitration lost in 4th bit of identifier (ID.25)	Start of frame
0x04	4	Arbitration lost in 5th bit of identifier (ID.24)	SRTR bit
0x05	5	Arbitration lost in 6th bit of identifier (ID.23)	IDE bit
0x06	6	Arbitration lost in 7th bit of identifier (ID.22)	ID.20 to ID.18
0x07	7	Arbitration lost in 8th bit of identifier (ID.21)	ID.17 to ID.13
0x08	8	Arbitration lost in 9th bit of identifier (ID.20)	CRC sequence
0x09	9	Arbitration lost in 10th bit of identifier (ID.19)	Reserved bit 0
0x0A	10	Arbitration lost in 11th bit of identifier (ID.18)	Data Field
0x0B	11	Arbitration lost in SRTR bit 1	Data Length Code
0x0C	12	Arbitration lost in IDE bit	RTR bit
0x0D	13	Arbitration lost in 12th bit of identifier (ID.17)	Reserved bit 1
0x0E	14	Arbitration lost in 13th bit of identifier (ID.16)	ID.4 to ID.0

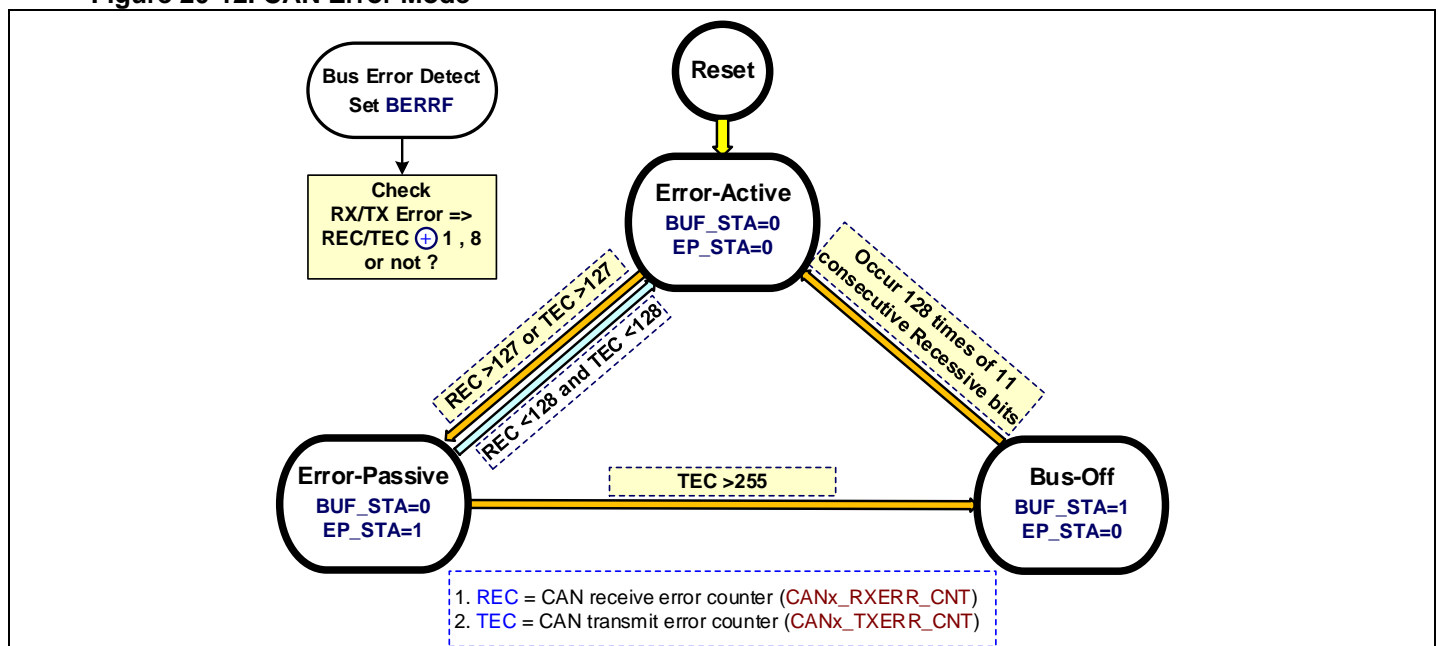
0x0F	15	Arbitration lost in 14th bit of identifier (ID.15)	ID.12 to ID.5
0x10	16	Arbitration lost in 15th bit of identifier (ID.14)	
0x11	17	Arbitration lost in 16th bit of identifier (ID.13)	Active error flag
0x12	18	Arbitration lost in 17th bit of identifier (ID.12)	Intermission
0x13	19	Arbitration lost in 18th bit of identifier (ID.11)	Tolerate dominant bits
0x14	20	Arbitration lost in 19th bit of identifier (ID.10)	
0x15	21	Arbitration lost in 20th bit of identifier (ID.9)	
0x16	22	Arbitration lost in 21st bit of identifier (ID.8)	Passive error flag
0x17	23	Arbitration lost in 22nd bit of identifier (ID.7)	Error delimiter
0x18	24	Arbitration lost in 23rd bit of identifier (ID.6)	CRC delimiter
0x19	25	Arbitration lost in 24th bit of identifier (ID.5)	Acknowledge
0x1A	26	Arbitration lost in 25th bit of identifier (ID.4)	End of frame
0x1B	27	Arbitration lost in 26th bit of identifier (ID.3)	Acknowledge delimiter
0x1C	28	Arbitration lost in 27th bit of identifier (ID.2)	Overload frame
0x1D	29	Arbitration lost in 28th bit of identifier (ID.1)	
0x1E	30	Arbitration lost in 29th bit of identifier (ID.0)	
0x1F	31	Arbitration lost in RTR bit	

<Sign> " " ~ Not defined

20.10.6. CAN Error Mode

The module implements three error modes of **Error-Active** mode, **Error-Passive** mode and **Bus-Off** mode. The following diagram is showing the CAN Error Mode flow diagram.

Figure 20-12. CAN Error Mode



- **Error-Active**
 - The state can normally take part in bus communication.
 - When an error has been detected sends an “Active Error Flag”.
(Active Error Flag : Error flag of Error frame is 6 consecutive “dominant” bits)
- **Error-Passive**
 - The state can normally take part in bus communication
 - The state will wait before initiating a further transmission after a transmission.
 - When an error has been detected sends a “Passive Error Flag”.
(Passive Error Flag: Error flag of Error frame is 6 consecutive “recessive” bit)
- **Bus-Off**
 - The state is not allowed to have any influence on the bus.

The module is in **Error-Active** if both error counters are below the **Error-Passive** limit of 128. The module will be in **Error-Passive** if at least one of the error counters is equal or over 128. When the transmit error counter is over **Bus-Off** limit value of 255, the module will be in **Bus-Off**. The module keeps in this state until the **Bus-Off** recovery sequence which consists of 128 occurrences of 11 consecutive recessive bits is received.

There are one transmit error counter register of **CANx_TXERR_CNT** and one receive error counter register of **CANx_RXERR_CNT** those are used to record the number of transmission errors and receiving errors. After hardware reset or when a Bus Off event occurs, these two counters are automatically set to '0'.

There are two receive error control modes of 'Normal' and 'Passive' by setting **CANx_RXERR_MDS** register. When this register is set 'Normal', the receive error counter register of **CANx_RXERR_CNT** register is incremented when errors are experienced in the receive bit stream. When this register is set 'Passive', the receive error counter does not increased if the error has happened in error passive state.

These transmit error counter (**TEC**) and receive error counter (**REC**) are updated by hardware according to the following rules:

- (1) When receiver detects an error: **REC + 1**

Exception: Detected "**bit error**" during the sending of an "**Active Error flag**" or an "**Overload flag**"

- (2) When receiver detects a dominant bit as the first bit after sending an Error flag: **REC + 8**

- (3) When a transmitter sends an Error flag: **TEC + 8**

Exception 1:

- Transmitter is "**Error-Passive**"
- Detects an "**Acknowledgment Error**" because of not detecting a dominant ACK
- Not detect a dominant bit while sending "**Passive Error flag**"

Exception 2:

- If **transmitter** sends an Error Flag because a "**Stuff Error**" occurred during arbitration
- Whereby the stuff bit should have been recessive
- The transmitter has been sent recessive but is monitored to be dominant.

- (4) Transmitter detects a bit error while sending an "**Active Error flag**" or an "**Overload flag**": **TEC + 8**

- (5) Receiver detects a bit error while sending an "**Active Error flag**" or an "**Overload flag**": **REC + 8**

- (6) Any node tolerates up to **7 consecutive dominant** bits after sending a "**Active Error flag**", "**Passive Error flag**" or "**Overload flag**":

- If send "**Active Error flag**" or "**Overload flag**" detecting the 14th consecutive 'dominant' bit

- After detecting the 8th consecutive 'dominant' bit following a "Passive Error flag"

- Every transmitter : **TEC + 8**
- Every receiver : **REC + 8**

- After each sequence of additional 8 consecutive 'dominant' bit

- Every transmitter : **TEC + 8**
- Every receiver : **REC + 8**

There is one CAN error warning limit register **CANx_EW_LIM** to detect user defined limit for transmit error counter or receive error counter. When the value of transmit error counter or receive error counter is reached the error warning limit, the interrupt flag of **CANx_EWF** is set if the related interrupt enable bit **CANx_EW_IE** is enabled.

[Notify]: All of the registers of **CANx_EW_LIM**, **CANx_TXERR_CNT** and **CANx_RXERR_CNT** can only be written in **Initial** mode. In **Normal** mode, they are read only.

20.11. CAN Message Frames

The module supports standard and extended Data Frames, standard and extended Remote Frames, Error Frame and Overload Frame as defined message frames in the CAN 2.0B specification

20.11.1. CAN Data Frame

● Standard Data Frame

The Data Frame begins with a Start of Frame (SOF) bit which allows synchronization for all CAN devices. The SOF is followed by the arbitration field which is consisting of 12 bits, the 11-bit identifier and one Remote Transmission Request (RTR) bit. The RTR bit is used to distinguish a data frame from a remote frame.

The arbitration field is followed the control field which is consisting of six bits, one Identifier Extension (IDE) bit, one Reserved Bit Zero (R0) and four bits of Data Length Code (DLC) bits. The DLC defines the number of bytes of data field (0-8 bytes).

The control field is followed the data field. It contains transferred data bytes which are the length defined by the DLC. After data field is the Cyclic Redundancy Check (CRC) field which is consisting of a 15-bit CRC sequence and one CRC Delimiter bit. The CRC is used to detect transmission errors.

The last field is the two-bit Acknowledge (ACK) field. The transmitting node sends a recessive bit and any other node will be sending back a dominant bit if it has received a correct reception of the frame. The ACK field will be completed with a recessive Acknowledge delimiter. The data frame is end by the End-of-Frame (EOF) with 7-bit of recessive.

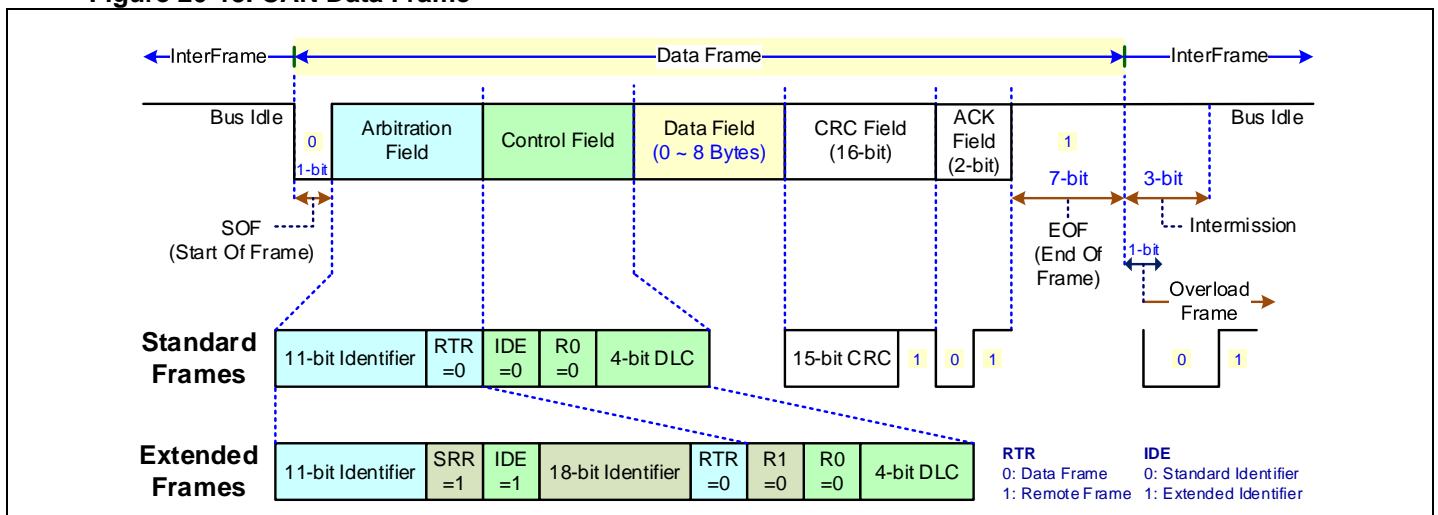
● Extended Data Frame

For the extended CAN Data Frame, the SOF bit is followed by the arbitration field which is consisting of 32 bits. It is consisting of 29-bit identifier which is combined 11 MSB bits and 18 LSB bits of the extended identifier, one Substitute Remote Request (SRR) bit, one IDE bit and one RTR bit. The IDE bit is recessive to denote an extended CAN frame.

The arbitration field is followed the control field which is consisting of six bits, two reserved dominant bits and four bits of Data Length Code (DLC) bits. The following fields of data field, CRC field, ACK field and EOF are the same structure as a standard data frame.

The following diagram is showing the CAN Data Frame diagram.

Figure 20-13. CAN Data Frame



● Interframe Space

The Interframe Space separates a preceding frame (Data Frame, Remote Frame, Error Frame, Overload Frame) from a subsequent Data Frames or Remote Frames. Overload Frames and Error Frames are not preceded by an Interframe Space and multiple Overload Frames are not separated by an Interframe Space.

Interframe Space contains the bit fields Intermission and Bus Idle. Intermission consists of three recessive bits. After the Intermission, the bus line remains in the recessive state (Bus Idle) until the next transmission starts.

During Intermission no station is allowed to start transmission of a Data Frame or Remote Frame. The only action to be taken is signaling an Overload condition. The detection of a dominant bit on the bus at the third bit of Intermission shall be interpreted as Start of Frame (SOF).

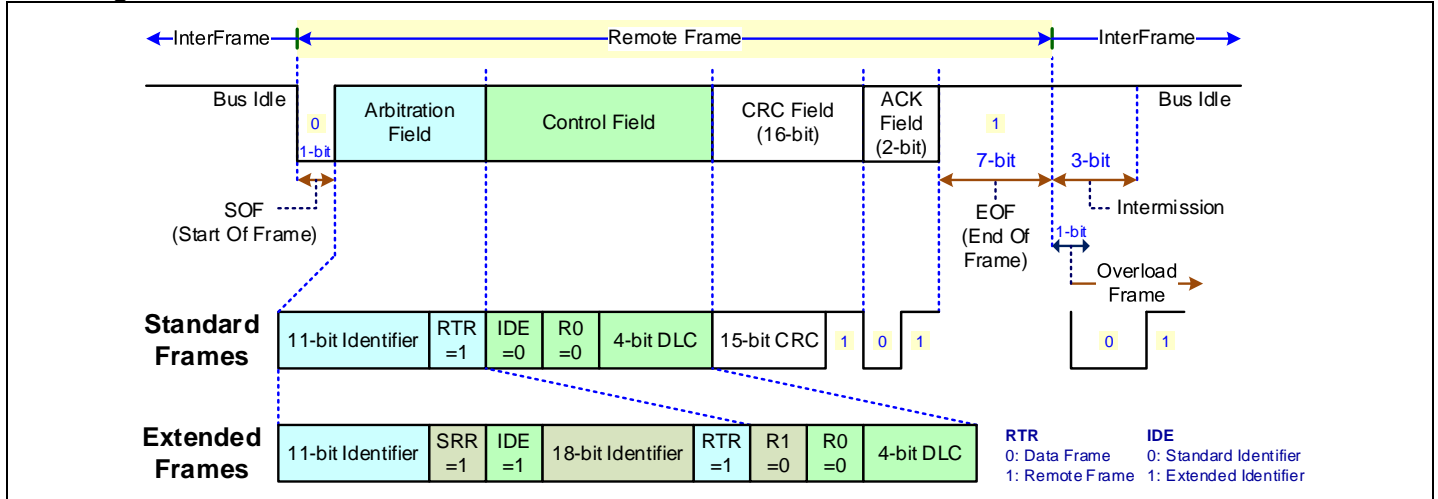
20.11.2. CAN Remote Frame

When data transmission is performed from a data source node, a destination node is possible to request data from the source. The destination node can initiate the transmission of the respective data from its source node by sending a Remote Frame. The Remote Frame and the corresponding Data Frame are named by the same identifier.

There are two differences between a remote frame and a data frame. The one, the RTR bit is with recessive. The two, there is no data field.

The following diagram is showing the CAN standard and extended Remote Frame diagram.

Figure 20-14. CAN Remote Frame



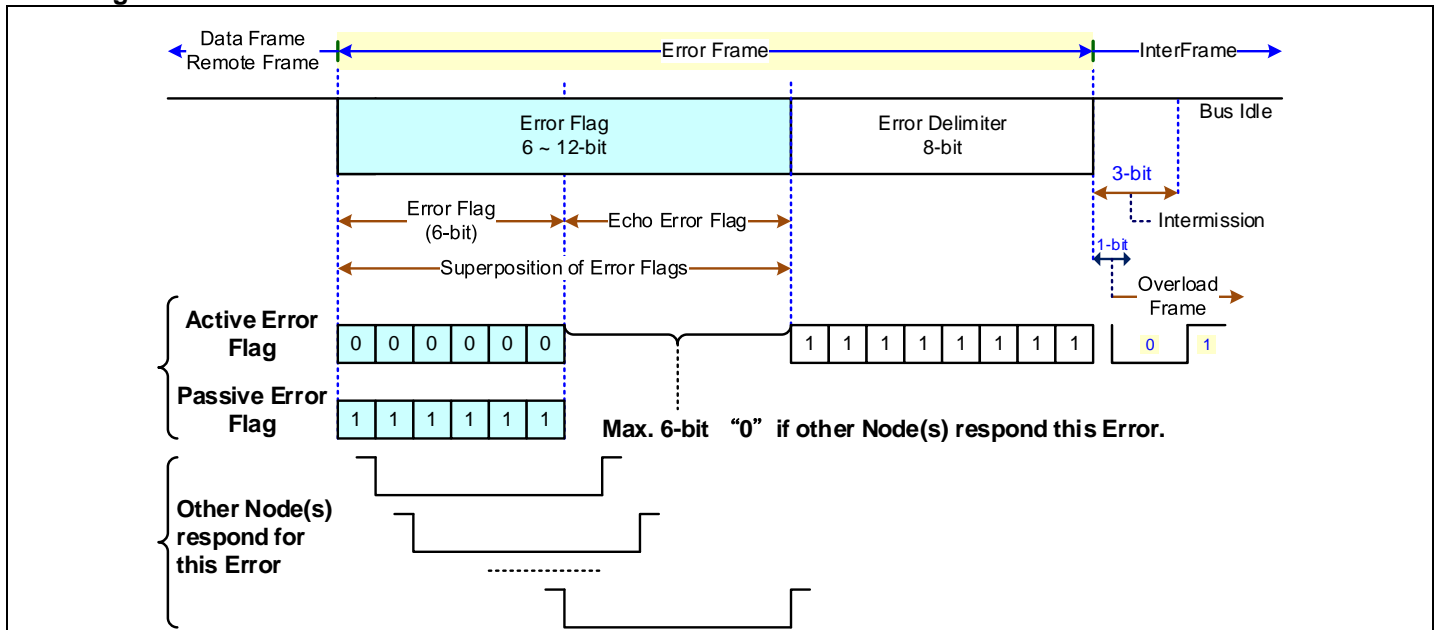
20.11.3. CAN Error Frame

The Error Frame consists of two different fields. The first field is given by the superposition of Error Flag contributed from different stations. The following second field is the Error Delimiter. An error frame is generated by any node that detects a bus error.

In order to terminate an Error Frame correctly, an error passive node may need the bus to be Bus Idle for at least 3 bit times (if there is a local error at an error passive Receiver).

The following diagram is showing the CAN Error Frame diagram.

Figure 20-15. CAN Error Frame



There are two forms of an Error Flag: an Active Error Flag and a Passive Error Flag.

1. The Active Error Flag consists of six consecutive dominant bits.

2. The Passive Error Flag consists of six consecutive recessive bits unless it is overwritten by dominant bits from

other nodes.

The Error Delimiter consists of eight recessive bits. After transmission of an Error Flag each station sends recessive bits and monitors the bus until it detects a recessive bit. Afterwards it starts transmitting seven more recessive bits.

20.11.4. CAN Overload Frame

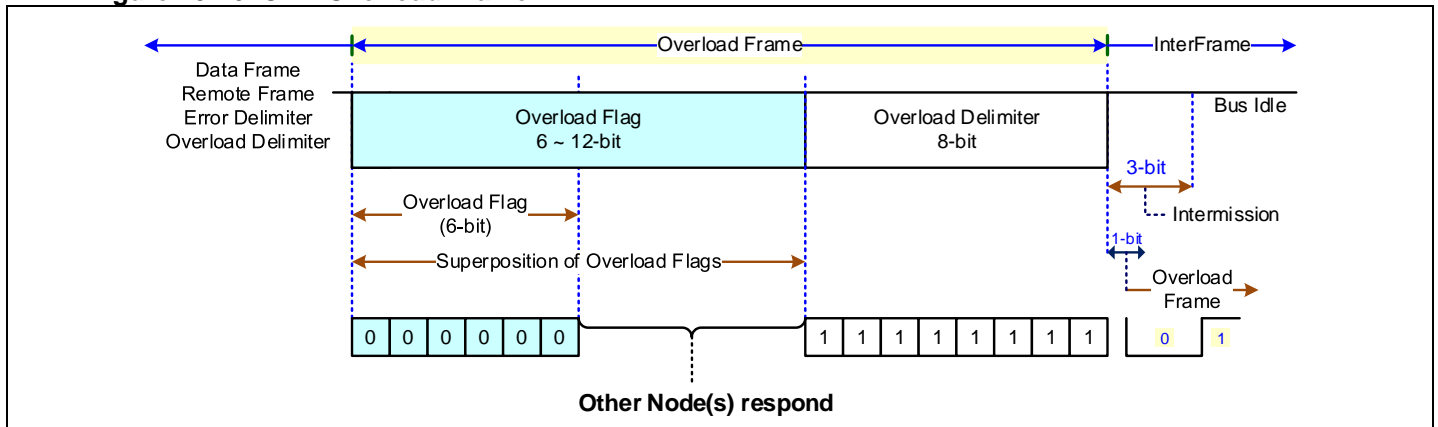
The Overload Frame contains the two bit fields Overload Flag and Overload Delimiter. Overload Flag consists of six dominant bits. Overload Delimiter consists of eight recessive bits and is of the same form as the Error Delimiter.

There are three kinds of Overload conditions, which lead to the transmission of an Overload Flag:

1. The internal conditions of a Receiver, which requires a delay of the next Data Frame or Remote Frame.
2. Detection of a dominant bit at the first or second bit of Intermission.
3. If a CAN FD node samples a dominant bit at the eighth bit (the last bit) of an Error Delimiter or Overload Delimiter, or if a CAN FD Receiver samples a dominant bit at the last bit of End-of-Frame (EOF), it will start transmitting an Overload Frame (not an Error Frame). The Error Counters will not be incremented.

The start of an Overload Frame due to Overload condition 1 is only allowed to be started at the first bit time of an expected Intermission, whereas Overload Frames due to Overload condition 2 or condition 3 start one bit after detecting the dominant bit. At most two Overload Frames may be generated to delay the next Data or Remote Frame.

Figure 20-16. CAN Overload Frame



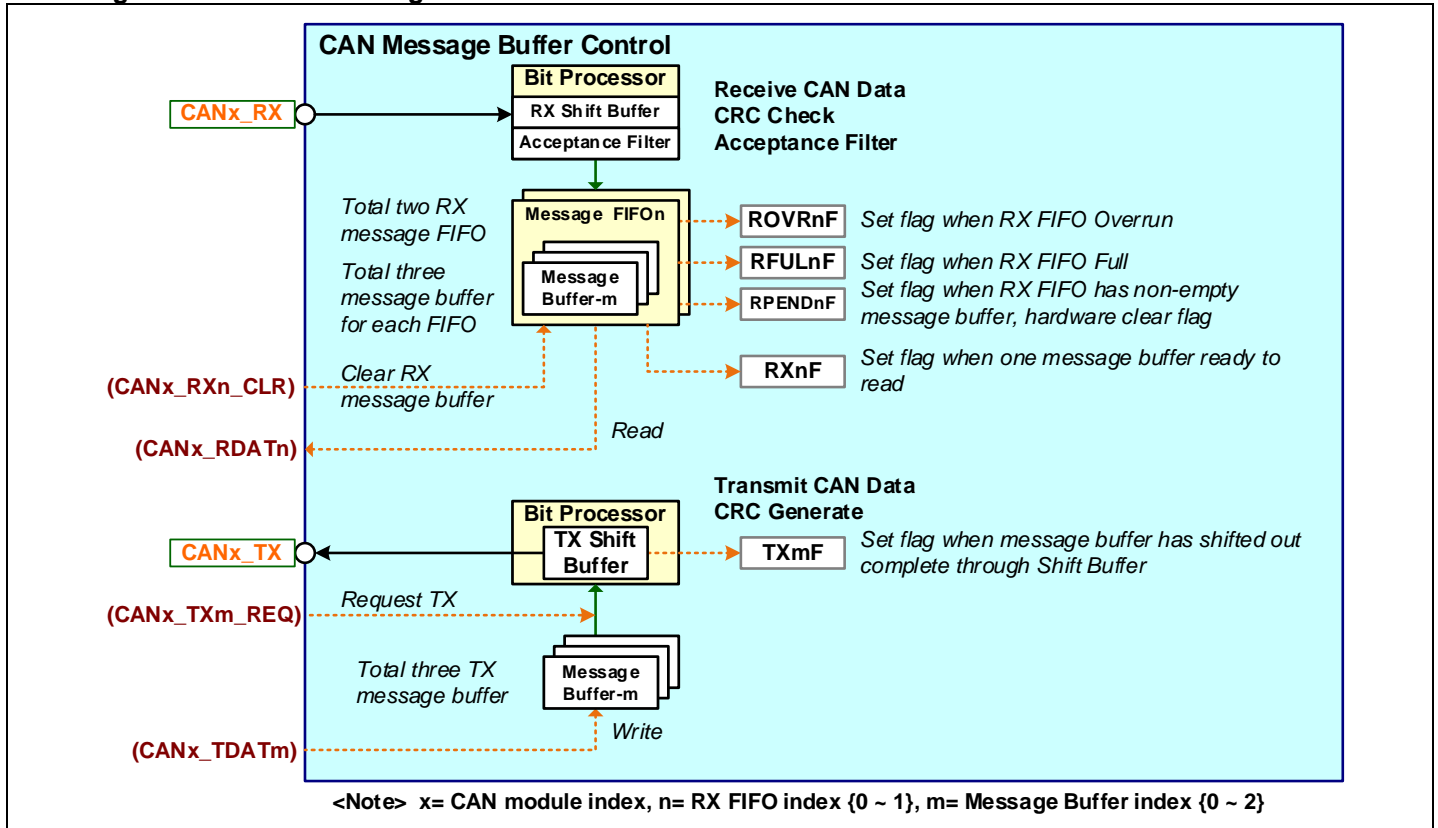
20.12. CAN Message Transfer

20.12.1. CAN Message Buffer

The module implements two receiving RX message FIFOs with separated three RX message buffer and three independent transmission TX message buffer. Each RX message FIFO and TX message buffer are implemented with four 32-bit data registers.

The following diagram is showing the CAN Message Buffer Control diagram.

Figure 20-17. CAN Message Buffer Control



User can get the CAN message data from one of RX message FIFO by reading the four 32-bit data registers of **CANx_RDATn0**, **CANx_RDATn1**, **CANx_RDATn2** and **CANx_RDATn3**. Also user can output the CAN message data to one of TX message buffer by writing the four 32-bit data registers of **CANx_TDATm0**, **CANx_TDATm1**, **CANx_TDATm2** and **CANx_TDATm3**. (n = RX message FIFO index {0 ~ 1}, m = TX message buffer index {0 ~ 2})

The following table is showing the CAN RX and TX Buffer data definitions.

Table 20-4. CAN RX and TX Buffer Data Definitions

CAN Register		Data Bit Definitions			
		Bit [31:24]	Bit [23:16]	Bit [15:8]	Bit [7:0]
RX	RDATn0	x, IDE, RTR, SID[10:6]	SID[5:0], EID[17:16]	EID[15:8]	EID[7:0]
	RDATn1	x, x, x, x, x, x, x, x	x, x, x, x, x, x, x, x	x, x, x, x, x, x, RST, CLR	x, x, x, x, DLC[3:0]
	RDATn2	Data Byte 3	Data Byte 2	Data Byte 1	Data Byte 0
	RDATn3	Data Byte 7	Data Byte 6	Data Byte 5	Data Byte 4
TX	TDATm0	x, IDE, RTR, SID[10:6]	SID[5:0], EID[17:16]	EID[15:8]	EID[7:0]
	TDATm1	x, x, x, x, x, x, x, x	x, x, x, x, x, x, x, x	x, x, x, x, x, x, STOP, REQ	x, x, x, x, DLC[3:0]
	TDATm2	Data Byte 3	Data Byte 2	Data Byte 1	Data Byte 0
	TDATm3	Data Byte 7	Data Byte 6	Data Byte 5	Data Byte 4

<Sign> SID: Standard identifier, EID: Extended identifier, n: Register index, x: Reserved bit
RST: Reset FIFO, CLR: Clear active RX buffer, STOP: STOP TX request, REQ: Transmit request

20.12.2. CAN Transmit

When user wants to transmit CAN message data to external CAN device or node, user needs to write message data to TX message buffer by writing the four 32-bit data registers of **CANx_TDATm0**, **CANx_TDATm1**, **CANx_TDATm2** and **CANx_TDATm3**. When the message data has updated and was ready on the TX message buffer, user can trigger the CAN module to send message data to CAN bus through external CAN transceiver by setting **CANx_TXm_REG** register. (m= TX message buffer index {0 ~ 2})

When detect the TXF (**CANx_TXmF**) flag if the related interrupt enable register bit of **CANx_TXm_IE** has enables, it indicates the TX message buffer has shifted out complete through shift buffer and user can update the next transmission message data to this TX message buffer. Also user can write message data to another TX message buffer which is released in idle state.

- **CAN Transmission Priority**

For TX message buffer control, user can select message buffer process priority modes of 'ID' or 'SEQ' by setting **CANx_TX_PRI** register. When selects 'ID', the message buffer is transmission by the priority of the message identifier. The same ID number is the greater priority. When selects 'SEQ', the message buffer is transmission in the request sequence order by setting **CANx_TXm_REG** register.

- **CAN Transmission Abort**

For CAN transmission error condition control, user can select message transmission modes of 'Resend' or 'Once' for transmission error conditions by setting **CANx_TXE_MDS** register. When selects 'Resend', the chip will automatically retransmit the message until the message has been successfully transmitted. When selects 'Once', the message will be transmitted only once.

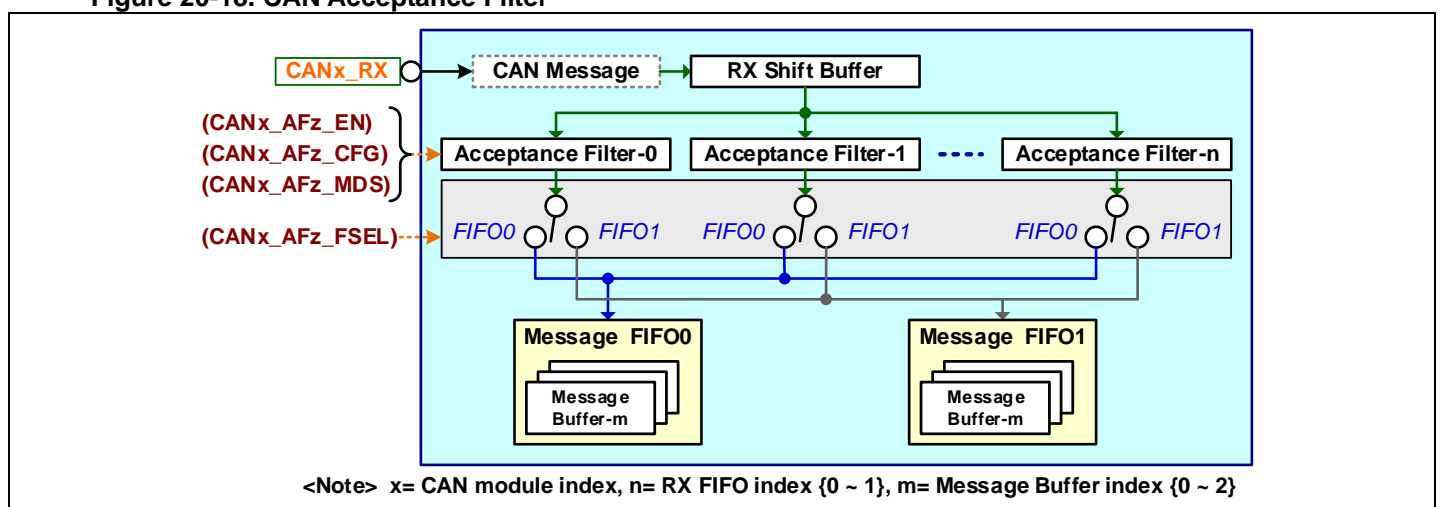
The module is implemented a transmission aborted function that user can stop the next message data transmission request for transmission error conditions by setting **CANx_TXn_STOP** register after use has triggered the CAN module to send message data by setting **CANx_TXm_REG** register. When the **CANx_TXn_STOP** register is enabled and the related TX message buffer has been active and started to transmit, the active TX message buffer will be still transmitted and stop the next message data transmission request for transmission error conditions. When the **CANx_TXn_STOP** register is enabled and the related TX message buffer has not yet been active to transmit, the related TX message buffer will be released and stop to transmit.

Also user can trigger the CAN module to send message data and control once to stop the next message data transmission request for transmission error conditions by setting 'Enable' for both registers of **CANx_TXm_REG** and **CANx_TXn_STOP** simultaneously.

20.12.3. CAN Acceptance Filter

The CAN module provides six configurable CAN receiving acceptance filters for filtering the incoming message data. Each RX acceptance filter has independent registers to define the acceptance filter code and code mask in registers of **CANx_AFzR0** and **CANx_AFzR1**. User can configure to enable or disable for each the RX acceptance filter by setting the independent **CANx_AFz_EN** register. Also each the RX acceptance filter can configure for using RX message FIFO0 or FIFO1 by setting independent **CANx_AFz_FSEL** register. (z= RX Acceptance Filter index {0 ~ 5})

Figure 20-18. CAN Acceptance Filter



Each RX acceptance filter can be configure to one single 32-bit filter or dual 16-bit filters by setting the independent **CANx_AFz_CFG** register. Also each RX acceptance filter can be configure to 'Mask' mode or 'List' mode by setting the independent **CANx_AFz_MDS** register.

The following table is showing the CAN RX Acceptance Filter definitions for different filter modes.

Table 20-5. CAN RX Acceptance Filter Definitions

Filter Mode	Control Register		Acceptance Filter Bit Definitions					
	AFz_CFG	AFz_MDS	Filter Reg	ID #	Bit [31:24]	Bit [23:16]	Bit [15:8]	Bit [7:0]
One 32-bit ID Mask Mode	0	0	AFzR0	n0	SID[10:3]	SID[2:0], EID[17:13]	EID[12:5]	EID[4:0], IDE, RTR, 0
			AFzR1		Mask Bits	Mask Bits	Mask Bits	Mask Bits
Two 32-bit ID List Mode	0	1	AFzR0	n0	SID[10:3]	SID[2:0], EID[17:13]	EID[12:5]	EID[4:0], IDE, RTR, 0
			AFzR1	n1	SID[10:3]	SID[2:0], EID[17:13]	EID[12:5]	EID[4:0], IDE, RTR, 0
Two 16-bit ID Mask Mode	1	0	AFzR0	n0	Bit [15:8] Mask Bits	Bit [7:0] Mask Bits	SID[10:3]	SID[2:0],IDE,RTR,EID[17:15]
			AFzR1	n1	Bit [15:8] Mask Bits	Bit [7:0] Mask Bits	SID[10:3]	SID[2:0],IDE,RTR,EID[17:15]
Four 16-bit ID List Mode	1	1	AFzR0	n1, n0	SID[10:3]	SID[2:0],IDE,RTR,EID[17:15]	SID[10:3]	SID[2:0],IDE,RTR,EID[17:15]
			AFzR1	n3, n2	SID[10:3]	SID[2:0],IDE,RTR,EID[17:15]	SID[10:3]	SID[2:0],IDE,RTR,EID[17:15]

<Sign> SID: Standard identifier, EID: Extended identifier, z: Filter index

When the bits definitions are as 'Identifier' in registers of **CANx_AFzR0** and **CANx_AFzR1**, these bits record the bit patterns used by the acceptance filter in filtering received data in conjunction with the corresponding acceptance mask register. When the bits definitions are as 'Mask' in registers of **CANx_AFzR0** and **CANx_AFzR1**, these bits record the mask patterns used by the acceptance filter in filtering received data. '1's in these bits identify the bits of the incoming data bytes that are required to match the bit values in the corresponding acceptance code registers. '0's mark individual bits as "don't care".

When multiple acceptance filters are matched the same identifier:

- (1) If the FIFO selection of acceptance filters are the same, only one message store into designate RX FIFO.
- (2) If the FIFO selection of acceptance filters are different, the messages store into respective RX FIFO.

20.12.4. CAN Receive

When the module has received CAN message data with requested identifier (ID) through the related Acceptance Filter from external CAN device or node, the module will assert a related RXF flag (**CANx_RXnF**) to indicate one of RX message buffer has received and updated completely in the related RX message FIFO. User can get the message data from RX message FIFO0 or FIFO1 by reading the four 32-bit data registers of **CANx_RDATn0**, **CANx_RDATn1**, **CANx_RDATn2** and **CANx_RDATn3**. (n= RX message FIFO index {0 ~ 1})

The CAN receiver data sampling mode is able to select the oversampling majority vote from three samples or one sample by setting **CANx_OS_MDS** register. When oversampling majority vote is set three samples, the CAN module will decide the received bit value from the three sample value.

The module is implemented all RX message buffer combined function by setting **CANx_RBUF_SEL** register. When the register is selected 'Two', all message buffers are separated to two RX message FIFOs. When selects 'One', all message buffers are combined as one RX message FIFO.

There is **CANx_RXn_NUM** register to indicate the received and updated message buffer number for each RX message FIFO. User can read this register to directly access the RX message FIFO and not control by interrupt process through RXF flag (**CANx_RXnF**). When the **CANx_RBUF_SEL** register is selected 'Two', the **CANx_RXn_NUM** register value range is 0 ~ 3. When the **CANx_RBUF_SEL** register is selected 'One', the **CANx_RX0_NUM** register value range is 0 ~ 6 and **CANx_RX0_NUM** register is unused.

There is one **CANx_RXn_CLR** register to clear and release active message buffer for each RX message FIFO. User must set this register to 'Enable' after user read message data completely for one RX message FIFO, so the module will release this received message buffer and clear **CANx_RXnF** automatically. At the time, the **CANx_RXn_NUM** register value will minus one. If the RX message FIFO is still with remained message data for any message buffer, the **CANx_RXn_NUM** register value is not equal zero. Also the interrupt flag (**CANx_RPENDnF**) will be asserted if the related interrupt enable register **CANx_RPENDn_IE** has enabled. When the receive data FIFO is empty, the chip will clear this **CANx_RPENDnF** flag automatically.

● CAN FIFO Full and Overrun

When RX message FIFO is received full, the interrupt flag (**CANx_RFULnF**) will be asserted if the related interrupt enable register **CANx_RFULn_IE** has enabled. When RX message FIFO has not enough space and received overrun, the interrupt flag (**CANx_ROVRnF**) will be asserted if the related interrupt enable register **CANx_ROVRn_IE** has enabled. For RX message FIFO received overrun control, user can select message buffer overrun mode by setting **CANx_ROVR_MDS** register. When the register is selected 'Overwritten', the new received message data will overwrite the last active message buffer. When selects 'Keep', the new received message data will be skipped.

● CAN FIFO Reset

For RX message FIFO control, user can reset one of the RX message FIFO and clear the flags of **CANx_RXnF**, **CANx_RFULnF** and **CANx_ROVRnF** by setting **CANx_RXn_RST** register.

The following table is showing the CAN RX FIFO Reset Action.

Table 20-6. CAN RX FIFO Reset Action

Action	CAN Register					
Reset FIFO	RPENDnF	ROVRnF	RFULnF	RXnF	ROVRn_STA	RFULn_STA
	Clear 0	Clear 0	Clear 0	Clear 0	Clear 0	Clear 0

<Sign> n: RX FIFO index

● CAN FD Tolerant

When the chip is connected to the CAN bus which is with CAN FD devices, user can select CAN FD tolerant mode by setting **CANx_FDT_MDS** register. For CAN FD the FDF bit is at the same location of r0 is for standard frame or r1 for extended frame. When the register is selected 'Normal' for CAN 2.0, the data frame is normal reception whatever the received r0/r1 bit value. When the register is selected 'Skip' for CAN FD tolerant, the data frame is skipped if r0 is '1' for standard frame or r1 is '1' for extended frame. CAN FD tolerant is not able to receive or transmit FD frame but not disturbing them.

User can select the CAN receive edge detect filter mode after FDF bit is detected by setting **CANx_EDF_MDS** register. When the register is selected 'Normal', the edge detection is hit if detects input high-to-low condition. When the register is selected 'Sample', the edge detection is hit if detects input high-to-low and samples two successive '0' conditions.

20.13. CAN Test Mode

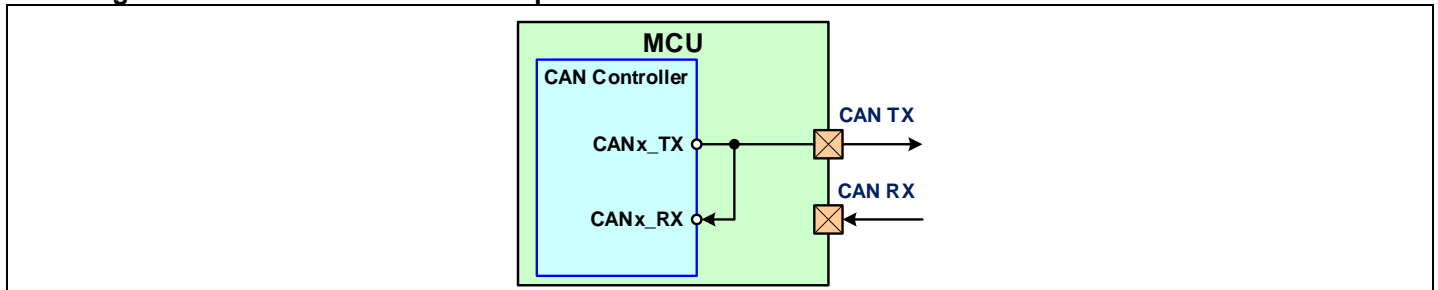
The module implements three test modes of *Silent* mode, *Loop Back* mode and *Loop Back combined with Silent* mode. User can use to test internal CAN module and set the test mode by setting **CANx_TST_MDS** register setting. Others, there is a CAN self-reception request mode and enable by setting **CANx_SRR_EN** register. User can use to test the data transferred path from CAN module to CAN bus through external CAN transceiver.

20.13.1. CAN Test Mode – Loop Back Mode

User can select Loop Back test mode by setting **CANx_TST_MDS** register to 'LBM'. In this test mode, the received input is taken from transmitted output to replace from input pin **CANx_RX**. The **CANx_RX** pin is disconnected to internal CAN module.

The following diagram is showing the CAN Loop Back test mode.

Figure 20-19. CAN Test Mode – Loop Back Mode

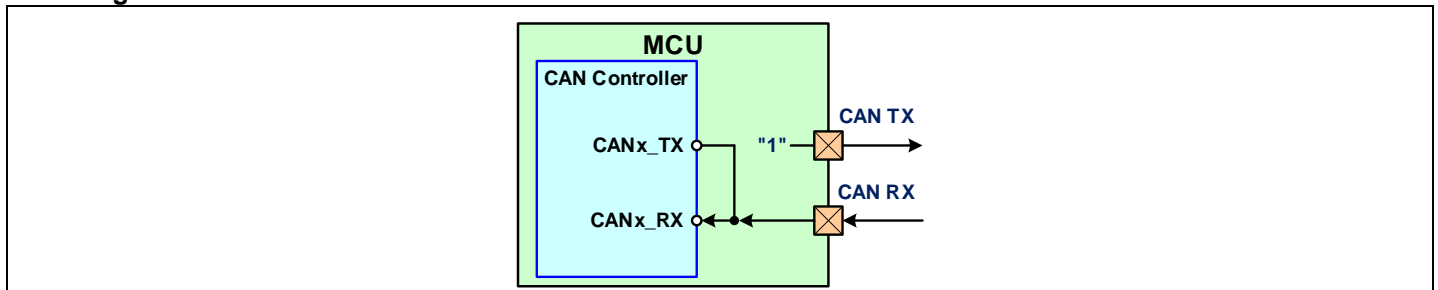


20.13.2. CAN Test Mode – Silent Mode

User can select Silent test mode by setting **CANx_TST_MDS** register to 'SIL'. In this test mode, the module does not send an 'acknowledge' to the CAN bus. That is even if a message is received successfully. When disabled, the error counters are stopped at the current value. The **CANx_TX** pin is always sent "1" and disconnected to internal CAN module.

The following diagram is showing the CAN Silent test mode.

Figure 20-20. CAN Test Mode – Silent Mode

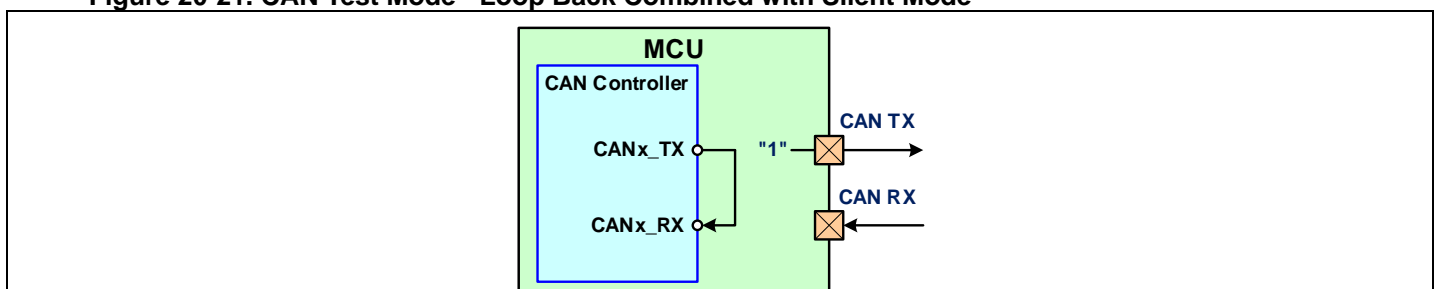


20.13.3. CAN Test Mode – Loop Back Combined with Silent Mode

User can select Loop Back Combined with Silent test mode by setting **CANx_TST_MDS** register to 'LBS'. In this test mode, the received input is taken from transmitted output. The internal module CAN input and output are opened to external CAN pins of **CANx_RX** and **CANx_TX**. The **CANx_TX** pin is always sent "1".

The following diagram is showing the CAN Loop Back Combined with Silent test mode.

Figure 20-21. CAN Test Mode - Loop Back Combined with Silent Mode



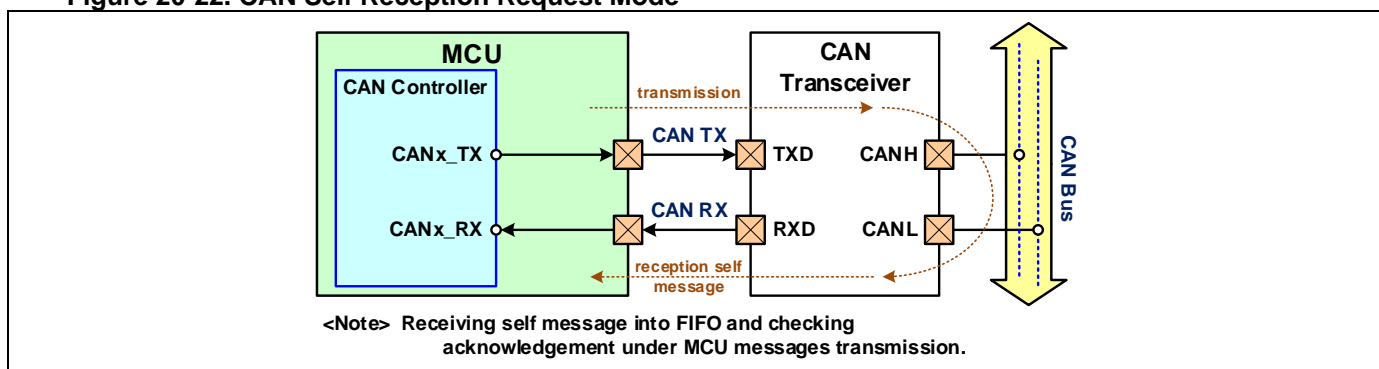
20.13.4. CAN Self Reception Request Mode

User can enable Self Reception Request mode by setting **CANx_SRR_EN** register. User can use to test the data transferred path from CAN module to CAN bus through external CAN transceiver.

When user enables this mode, a message is to be transmitted and received simultaneously. The CAN module sends the CAN message data to CAN bus through external CAN transceiver. At the time, the external CAN transceiver will also send back the data to CAN module on **CAN RX** signal path. The CAN module can receive self-message into FIFO and check acknowledgement to test the data transferred path.

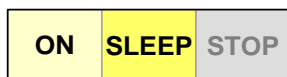
The following diagram is showing the CAN Self Reception Request mode.

Figure 20-22. CAN Self Reception Request Mode



21. Timer (General Timer)

21.1. Introduction



The module can be running in ON and SLEEP modes only.

The chip has maximum seven Timer/Counter modules: TM00, TM01, TM10, TM16, TM20, TM26 and TM36. All of them can be configured as timers or event counters.

TM0x has an 8-bit timer/counter with 8-bit prescaler. TM1x has a 16-bit timer/counter with 16-bit prescaler. TM2x has a 16-bit timer/counter with 16-bit prescaler and embeds two input capture/output compare channels. TM36 has a 16-bit timer/counter with 16-bit prescaler and embeds four input capture/output compare channels.

Notify: The sign of (x = module index; n = input/output channel number; m = I/O line number) is using for Registers, Signals and Pins/Ports in the descriptions of this chapter. [EX]: **TMx_EN**, **TMx_OCn_OEm** ~ x indicates module index number {00,01,10,16,20,26,36}; n = input/output channel number {0~3}; m = I/O line number {0,1,2}.

21.2. Features

- Provide max. seven timers/counters : TM00,TM01,TM10,TM16,TM20,TM26,TM36
- Support multi-level timer modules for different application
- Timer module common functions
 - Selectable Full-counter , Cascade , Separate modes
 - Multiple internal and external signals as timer clock source or trigger source
 - Internal timer events output to pin or other modules as input trigger event
 - Support timer reset , trigger start and clock gating for trigger source function
 - Timer overflow as clock output to external pin output
 - Programmable counter auto-stop mode
 - Main counter support up/down control (TM16/TM26/TM36 only)
 - 2nd counter support up/down control (Cascade/Separate mode)
- Provide TM36 timer modules
 - 32-bit timer/counter : 16-bit timer with 16-bit prescaler
 - 4 CCP (input Capture/output Compare/PWM) channels
 - 3 CCP channels with OCN (complementary output compare)
 - PWM function with center/edge-align and programmable dead time control
 - Support OC comparator split to two separated comparators mode
 - Break input control
 - QEI(Quadrature Encoder Interface) support
 - Max. Two IC and three OC with DMA capability
 - External input timer up/down control
 - Extra repetition counter for auto-stop mode
 - Support duty capture function
- Provide TM2x timer modules (TM20,TM26)
 - 32-bit timer/counter : 16-bit timer with 16-bit prescaler
 - 2 CCP (input Capture/output Compare/PWM) channels
 - 2 CCP channels with OCN (complementary output compare)
 - QEI(Quadrature Encoder Interface) support (TM26 only)
 - PWM function with edge-align
 - Support OC comparator split to two separated comparators mode
 - Extra repetition counter for auto-stop mode
 - Support duty capture function
- Provide TM1x timer modules (TM10,TM16)
 - 32-bit timer/counter : 16-bit timer with 16-bit prescaler
- Provide TM0x timer modules (TM00,TM01)
 - 16-bit timer/counter : 8-bit timer with 8-bit prescaler

21.3. Implementation

21.3.1. Chip Implementation

The following table is showing the implemented Timer modules of chips.

Table 21-1. Timer Implementation

Chip	Timer Module					TM36 DMA channels		TM36 PWM
	TM00/01	TM10/16	TM20	TM26	TM36	IC	OC/PWM	Max. Clock
MG32F02A128/A064	V	V	V	V	V	1	3	48MHz
MG32F02U128/U064	V	V	V	V	V	1	3	48MHz
MG32F02V032	V	V	V	-	V	2	3	96MHz
MG32F02N128/N064	V	V	V	V	V	2	3	96MHz
MG32F02K128/K064	V	V	V	V	V	2	3	96MHz

<Note>

V: Implemented; IC: Input Capture, OC/PWM: Output Compare/PWM
PWM Max. Clock: PWM timer input maximum operation clock frequency

21.3.2. Modules' Functions

The following table is showing the implemented functions of Timer modules.

Table 21-2. Timer Modules' Functions

Module Functions	Timer Modules						Comment
	TM00/01	TM10	TM16	TM20	TM26	TM36	
Timer/Counter total bits	16	32	32	32	32	32	
Timer Cascade Mode	yes	yes	yes	yes	yes	yes	16-bit_counter+16-bit_prescaler or 8-bit_counter+8-bit_prescaler
Timer Separate Mode	yes	yes	yes	yes	yes	yes	two 16-bit_counter or 8-bit_counter
Timer Full-Counter Mode	yes	yes	yes	yes	yes	yes	32-bit_counter or 16-bit counter
Independent channels				2	2	4	
Internal TRGI lines	8	8	8	8	8	8	
External TRGI lines	1	1	1	1	1	1	external pin input as clock source even internal clock disable
Output TRGO lines	1	1	1	1	1	1	support Timer Reset, Enable, Update event, Output state as trigger output
Output CKO lines	1	1	1	1	1	1	timer overflow as clock output to external pin output
Input Capture IC lines				2	2	4	
Output OC/OCN/OCH lines				2/2/2	2/2/2	4/3/4	OCN: complementary outputs, OCH: separated 8-bit compare-high outputs
Input Break lines						1	Break input to put the timer's output signals in hold state or a known (off) state
PWM separated two				yes	yes	yes	support OC comparator split to two separated Low/High 8-bit comparators mode
PWM edge-align				yes	yes	yes	
PWM center-align						yes	
Dead-time generator						yes	programmable dead-time control
Up/Down of 1st Timer	U	U	U/D	U	U/D	U/D	Up/down counting
Up/Down of 2nd Timer	U/D	U/D	U/D	U/D	U/D	U/D	Up/down counting
Timer auto Stop	yes	yes	yes	yes	yes	yes	programmable counter auto-stop one pulse mode
Repetition Counter				yes	yes	yes	extra repetition counter for auto-stop mode
QEI timer U/D control					yes	yes	support QEI mode 1 ~5
3-input XOR to CH-0						yes	
NCO output as clock	yes	yes	yes	yes	yes	yes	NCO_Pn input as Timer clock/trigger signal ITR7 input
Channel output delay				yes	yes	yes	channel-0,1,2,3 output delay 0,1,2,3 step unit delay time
Input duty capture (*1)				yes (*1)	yes (*1)	yes (*1)	
DMA request capability						yes	

<Note> *1: It doesn't support for MG32F02A128/A064/U128/U064.

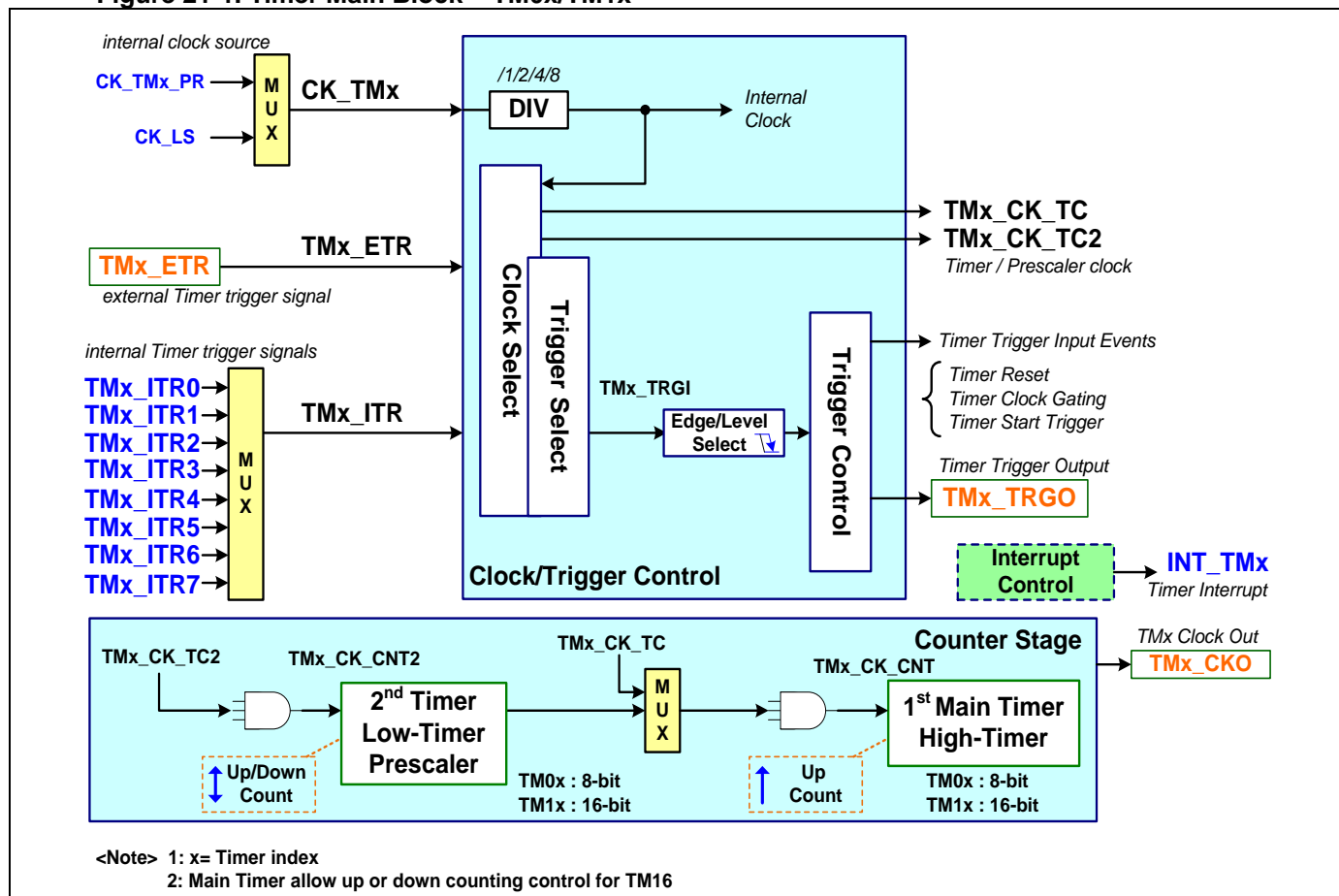
21.4. Control Block

The TMx module is including of a Clock/Trigger control block, a Counter Stage, an Capture/Compare control block and Input/Output Stages of channel I/O control (TM2x, TM3x only) and a Break control block (TM36 only).

The following diagrams are showing the Timer Control blocks of TM0x/TM1x, TM2x, TM3x modules.

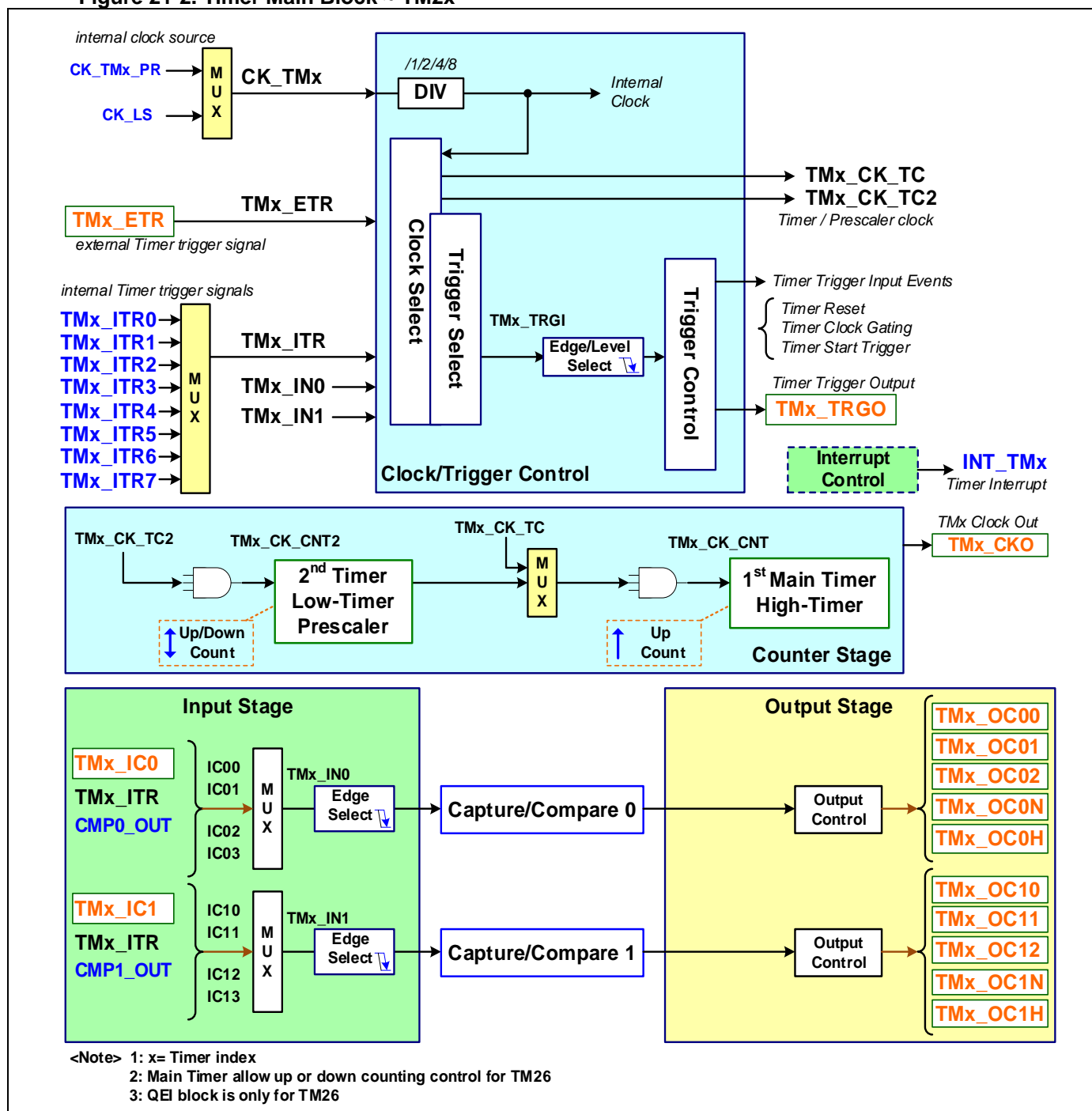
- **TM0x/1x Timer Main Block Diagram**

Figure 21-1. Timer Main Block ~ TM0x/TM1x



- TM2x Timer Main Block Diagram

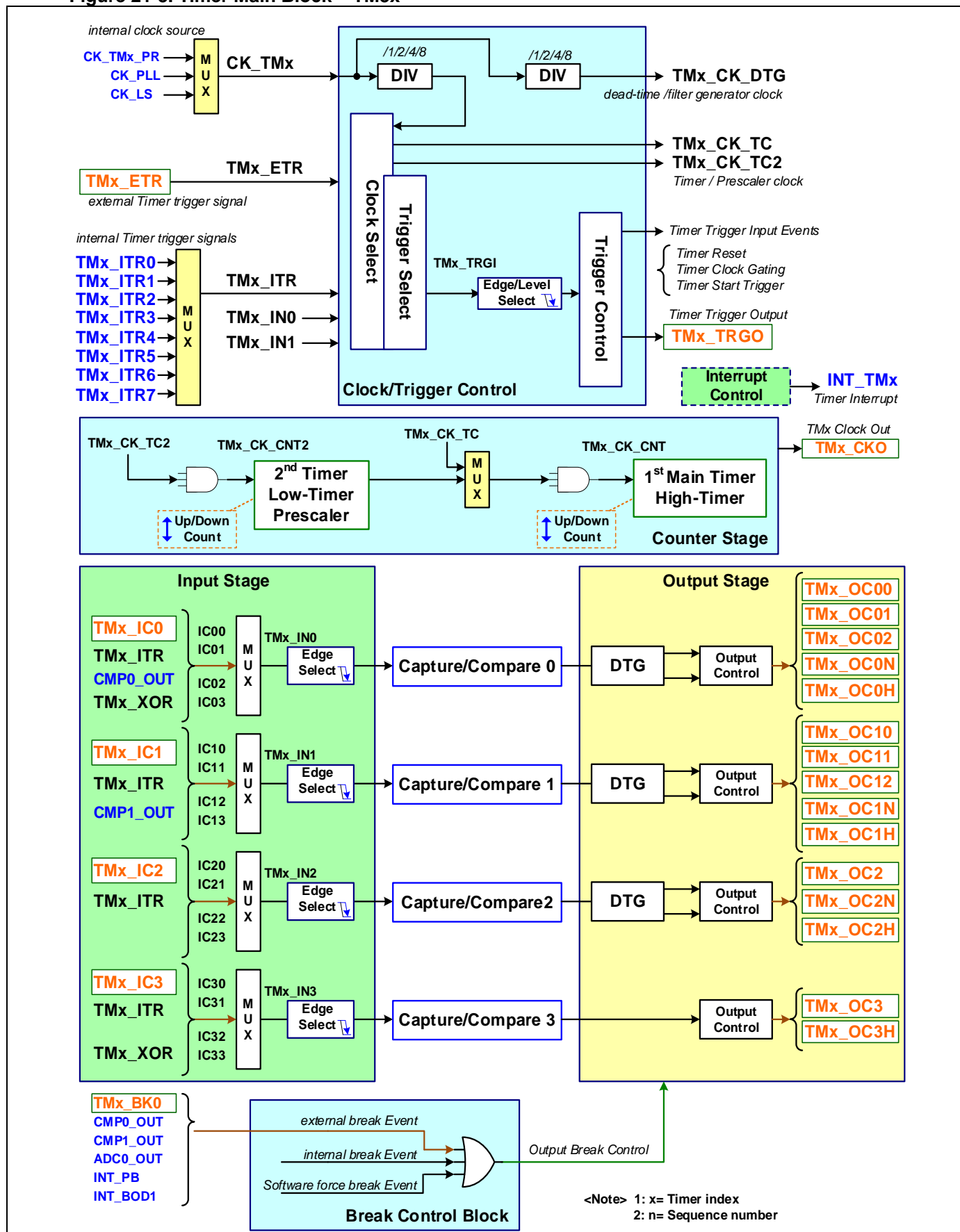
Figure 21-2. Timer Main Block ~ TM2x



● TM3x Timer Main Block Diagram

The **CK_PLL** clock input is not supported in TM36 for MG32F02A128/U128/A064/U064.

Figure 21-3. Timer Main Block ~ TM3x



21.5. IO Lines

21.5.1. IO Signals

- **TMx_CKO**

It is the timer overflow output signal and uses as clock output.

- **TMx_TRGO**

It is the selective output signal which is selected from the timer internal events or signals and output to pin or other internal modules as a trigger signal.

- **TMx_ETR**

It is the external trigger or clock input signal. It can be set and use to do as the timer clock. Also it can be set and input to trigger timer reset, timer counting start or timer clock gating.

- **TMx_ICn**

It is the timer input capture signal for channel-n.

- **TMx_OCn**

It is the timer compare or PWM output signal for channel-n.

- **TMx_OCnN**

It is the complement output signal of **TMx_OCn** for channel-n.

- **TMx_OCnH**

It is the timer compare or PWM output signal for the high channel of channel-n. This signal is using only for "Two 8-bit Compare/PWM" mode.

- **TMx_BK0**

It is the timer break input signal which is used to stop or hold the signals of **TMx_OCn**, **TMx_OCnN** and **TMx_OCnH** independently by register setting.

21.5.2. IO Configure

User must configure the related IO pins in order to use the IO lines of this module. The IO operation modes, output high speed option, pull-high option, output drive strength, IO deglitch filter and input inverse selection are programmable by pin independent. Please refer the section of "[IO Mode](#)" in GPIO chapter for more detail descriptions of IO mode configuration.

Each IO signal may be mapped and selected on several IO pins by configuring the IO AFS matrix. Please refer the section of "[Alternate Function Select](#)" in GPIO chapter for more detail descriptions of IO AFS configuration. About the actual IO pin AFS information, please refer the section of "[Pin Alternate Functions Selected Table](#)" in Pin Description chapter of the chip Data Sheet.

21.6. Enabling and Clock

21.6.1. Timer Enable

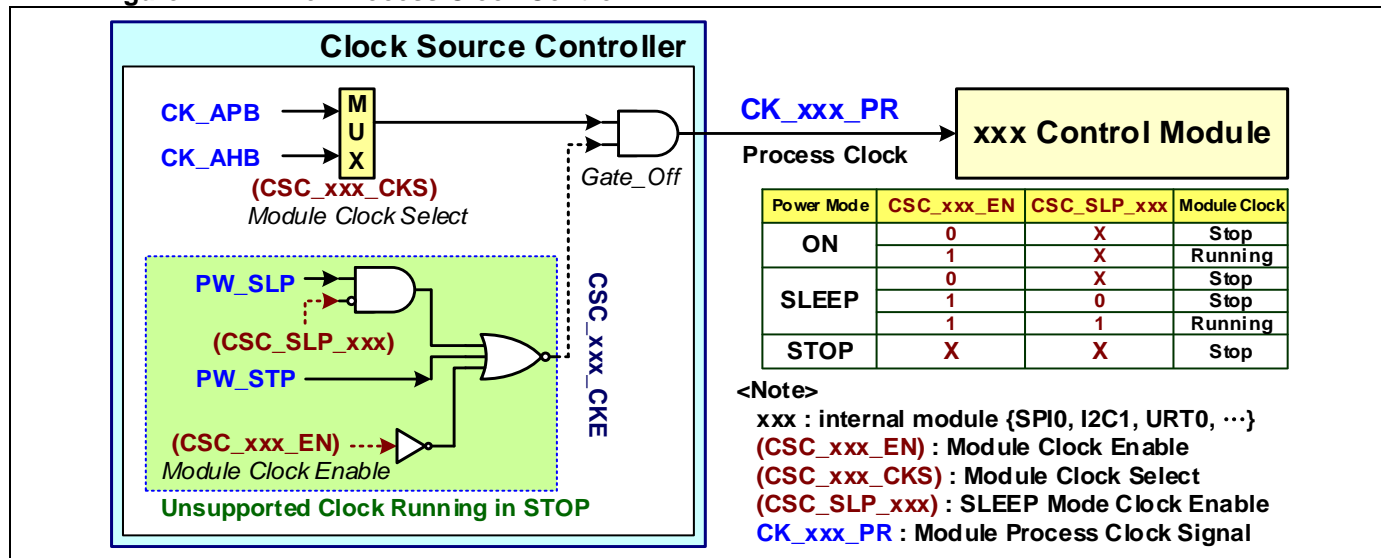
There are two timer enable control bits of **TMx_EN** and **TMx_EN2** for each TMx module. The timer builds in the TMx modules which supports three timer operation modes: (1) Cascade mode (2) Separate mode (3) Full-Counter mode. So the TMx timer can be a full-counter timer or separated to two timers.

The **TMx_EN** bit is used to enable the full-counter timer, High timer or Main timer for Full-Counter mode, Cascade or Separate timer operation modes. The **TMx_EN2** bit is used to enable the Low timer or 2nd timer for Cascade or Separate timer operation mode.

21.6.2. Timer Process Clock Control

The module process clock of **CK_TMx_PR** is using for the interface control logic between APB bus and the module. It is coming from CSC (Clock Source Controller) module. It can be enabled in **CSC_TMx_EN** register and select the clock source from APB clock or AHB clock in **CSC_TMx_CKS** register. User can plan the module clock is running or not beforehand for chip entering **SLEEP** mode by setting **CSC_SLP_TMx** register. Refer the System Clock chapter for more information.

Figure 21-4. Timer Process Clock Control



21.6.3. Timer Module Clock and Trigger Source

● Module Internal Clock

User can select the clock source from the module process clock of **CK_TMx_PR** or internal low speed clock of **CK_LS** by setting **TMx_CKI_SEL** register. Also the module provides one clock divider to generate the internal clock of **CK_TMx_INT** by setting **TMx_CKI_DIV** register. The clock divider can divide the clock frequency by 1/2/4/8.

[Notify]: The **TMx_CKI_DIV** register is only able to set 0 and is divided by 1 when the TMx module operates with QEI function.

The **CK_TMx_INT** signal outputs as the internal clock source for DTG input clock or one of the TMx timer clock sources. Also there is one clock divider by divided 1/2/4/8 to generate the DTG input clock of **TMx_CK_DTG** by setting **TMx_DTG_DIV** register.

[Notify]: Usually the internal clock frequency needs slow down at least 1/2 than module process clock.

● Internal and External Timer Trigger Signals

There are internal timer trigger signals of **TMx_ITR[7:0]** which can be selected by **TMx_ITR_MUX** register to output **TMx_ITR** signal. The **TMx_ITR6** and **TMx_ITR7** trigger signals are the global signals to use for all timer modules. They can be selected the trigger signals from internal timer modules or other peripheral modules. Refer the APB Common Control chapter in Common User Guide for more information.

There is one external timer trigger signal of **TMx_ETR** from **TMx_ETR** input. The **TMx_ETR** input can be configured from which GPIO pin by GPIO AFS setting.

● Timer Input Trigger Signal and External Clock Source

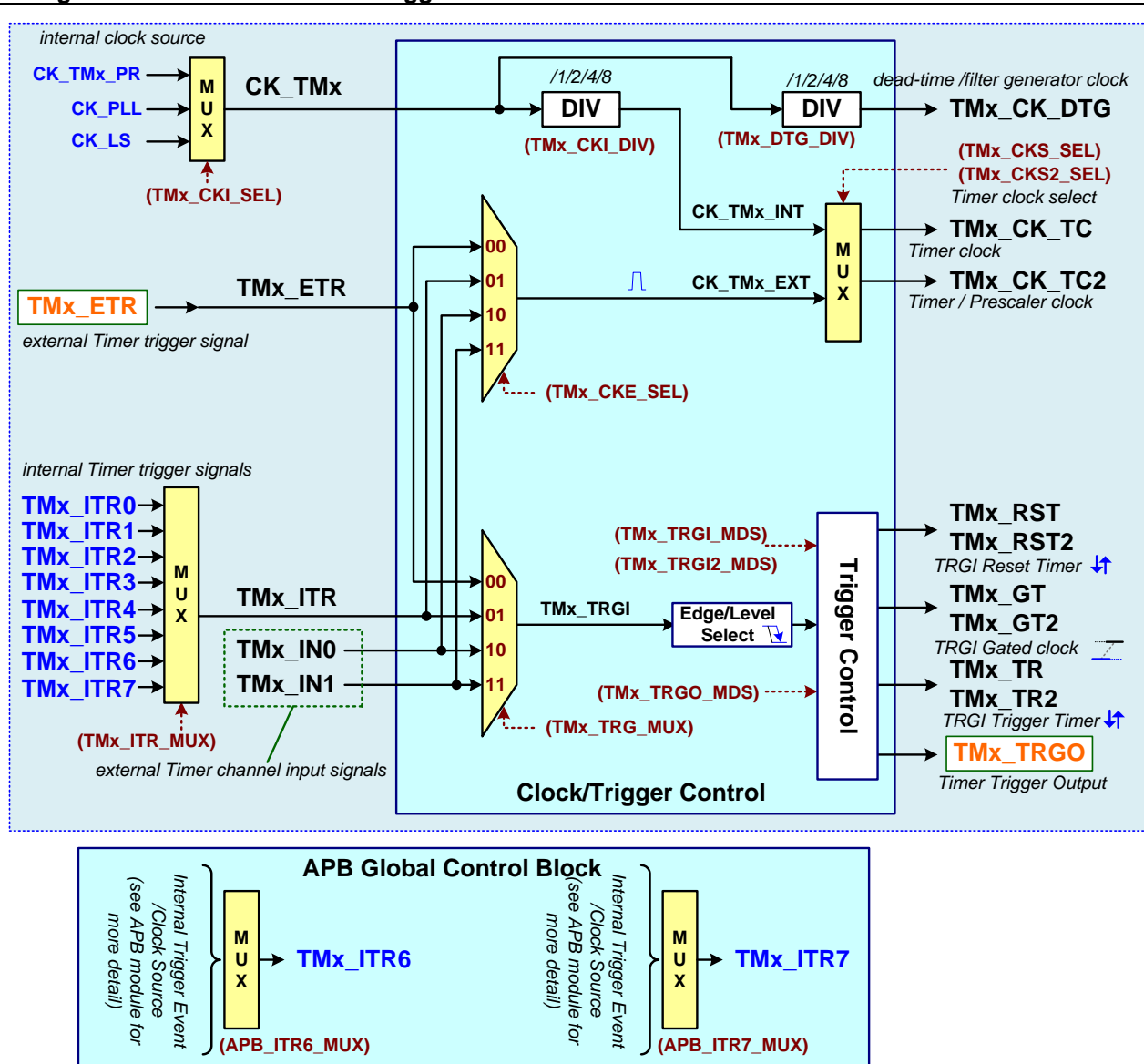
The **TMx_ITR** signal in conjunction with the external timer trigger signal of **TMx_ETR** and the timer external channel input signal of **TMx_IN0/TMx_IN1** to do as the timer input trigger source or the timer external clock source. User can select the timer external clock source to output **TCK_TMx_EXT** by setting **TMx_CKE_SEL** register.

Also user can select the timer input trigger source to output **TMx_TRGI** by setting **TMx_TRG_MUX** register. The **TMx_TRGI** signal can be input as the timer trigger input events of Reset Timer, Gated Clock, Timer-Start Trigger by setting **TMx_TRGI_MDS** and **TMx_TRGI2_MDS** registers for Main Timer and 2nd Timer.

The timer clock source of **TMx_CK_TC** and **TMx_CK_TC2** are able to select from internal clock source of **CK_TMx_INT** or external clock source of **CK_TMx_EXT** by setting **TMx_CKS_SEL** and **TMx_CKS2_SEL** registers.

The following diagram is showing the timer clock and trigger source. The **CL_PLL** clock input is not supported in TM36 for MG32F02A128/U128/A064/U064.

Figure 21-5. Timer Clock and Trigger Source



<Note> 1. x= Timer index
2. CK_LS frequency must $\leq \frac{1}{2}$ (CK_TMx_PR frequency)

21.7. Interrupt and Event

21.7.1. Timer Update Event

The TMx timer is able to separate to Main Timer and 2nd Timer. Refer the descriptions of “[Timer Operation Mode](#)” for the detail of timer operation mode. The Main Timer is able to output overflow/underflow flags of TOF/TUF, up/down counting direction flag of DIRF, counting direction change flag of DIRCF. The 2nd timer is able to output overflow/underflow flags of TOF2/TUF2.

The TMx module can generate two update event signals of **TMx_UEV** and **TMx_UEV2** to **TMx_TRGO** output. The **TMx_UEV** signal can also be used to trigger that loads the compare preload register content to compare shadow register for TM2x/TM3x modules. User can enable or disable the update event function by setting **TMx_UEV_DIS** register. The update event signal sources can come from Main Timer overflow/underflow event, software update event generation bit (**TMx_USW_EN**) or external trigger input of **TMx_TRGI**. User can select **TMx_UEV** output pulse from Main timer overflow TOF event, underflow TUF event or all the events by setting **TMx_UEV_SEL** register.

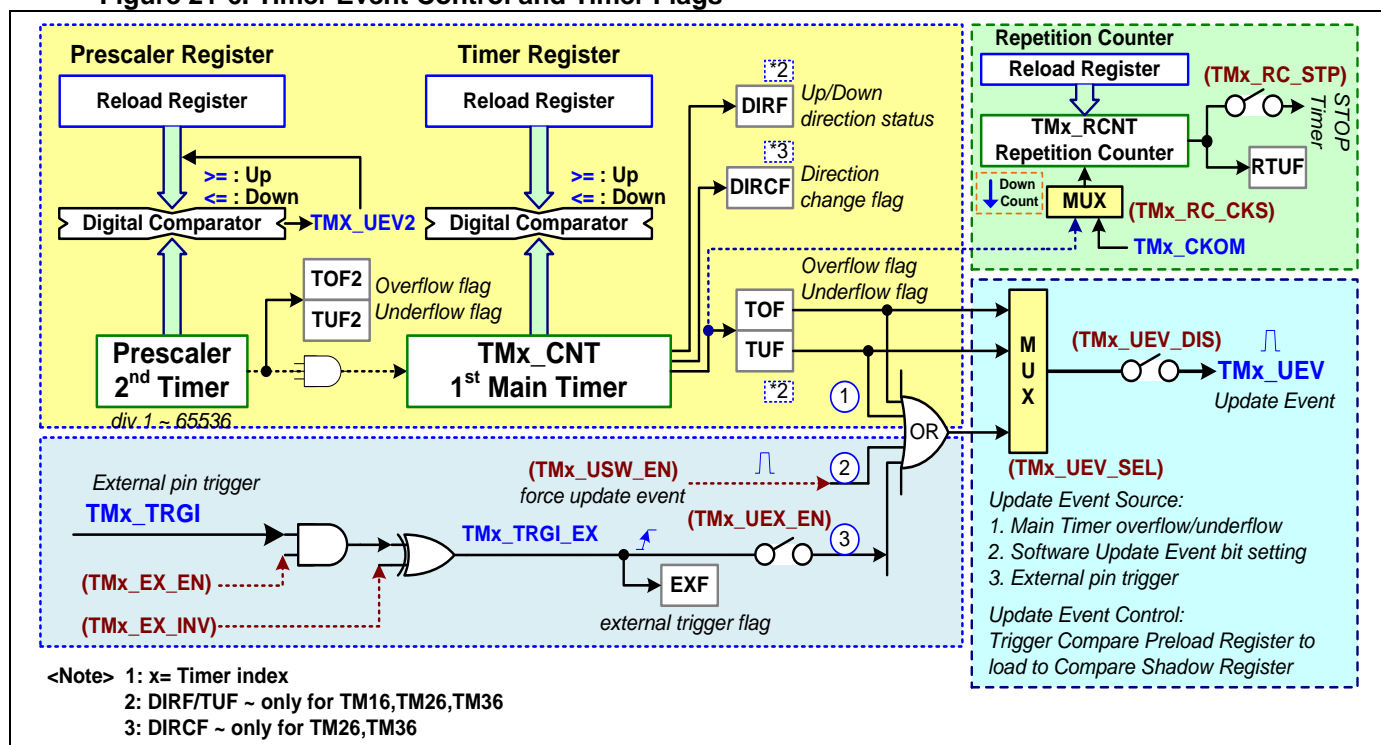
The **TMx_UEV2** signal is generated if the 2nd Timer overflow or underflow event is happened. It is also used to reset the counter for Up-Count or reload Timer Reload Register for Down-Count.

When user sets the **TMx_USW_EN** register bit, this bit will be automatically cleared by hardware after the “update event process” is finished.

For external trigger input of **TMx_TRGI**, there is one external trigger event enable bit of **TMx_EX_EN** to enable **TMx_TRGI** signal input. User can set **TMx_EX_INV** register to invert the **TMx_TRGI** signal. When the chip detects the rising edge of **TMx_TRGI_EX** signal, the EXF flag will be active and generated an interrupt if the related interrupt enable bit is enabled. In the condition, this external pin trigger event will force to do “update event process” if the timer external trigger update event enable bit of **TMx_UEX_EN** is enabled.

The following diagram is showing the timer event control and timer flags.

Figure 21-6. Timer Event Control and Timer Flags



21.7.2. Timer Interrupt Control and Status

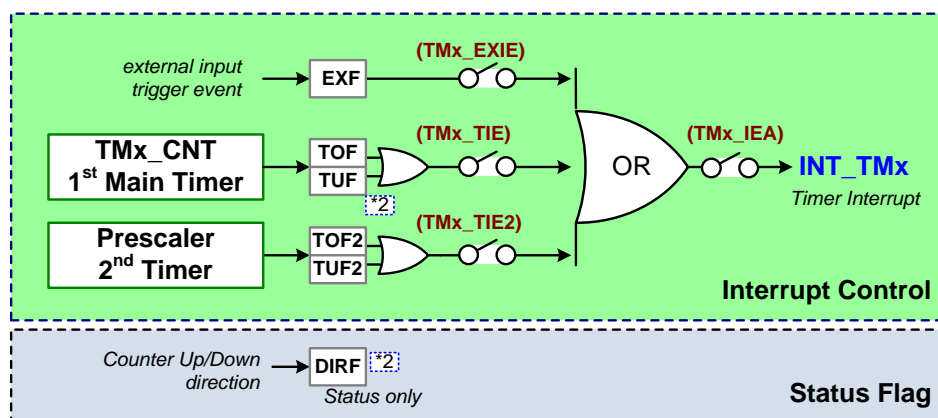
There is one signal of **INT_TMx** to be generated in this timer control module. **INT_TMx** sends to EXIC External Interrupt Controller to do as an interrupt event.

These interrupt flags are using for the interrupt service routine (ISR) flow control. Generally these flags are set by hardware and software clears them when the serviced job is completed in the related ISR. Each interrupt flag has one interrupt enable bit. User can enable or disable it. There is one interrupt all enable bit of **TMx_IEA** to enable or disable all the interrupt sources for this module.

There are some status bits those are reading only to provide internal control status. The DIRF flag (TMx_DIRF) is used to indicate the Main timer up/down counting direction. Refer the register descriptions of the related status bits for more information.

The following diagram is showing the timer status and interrupts control for TM0x and TM1x modules.

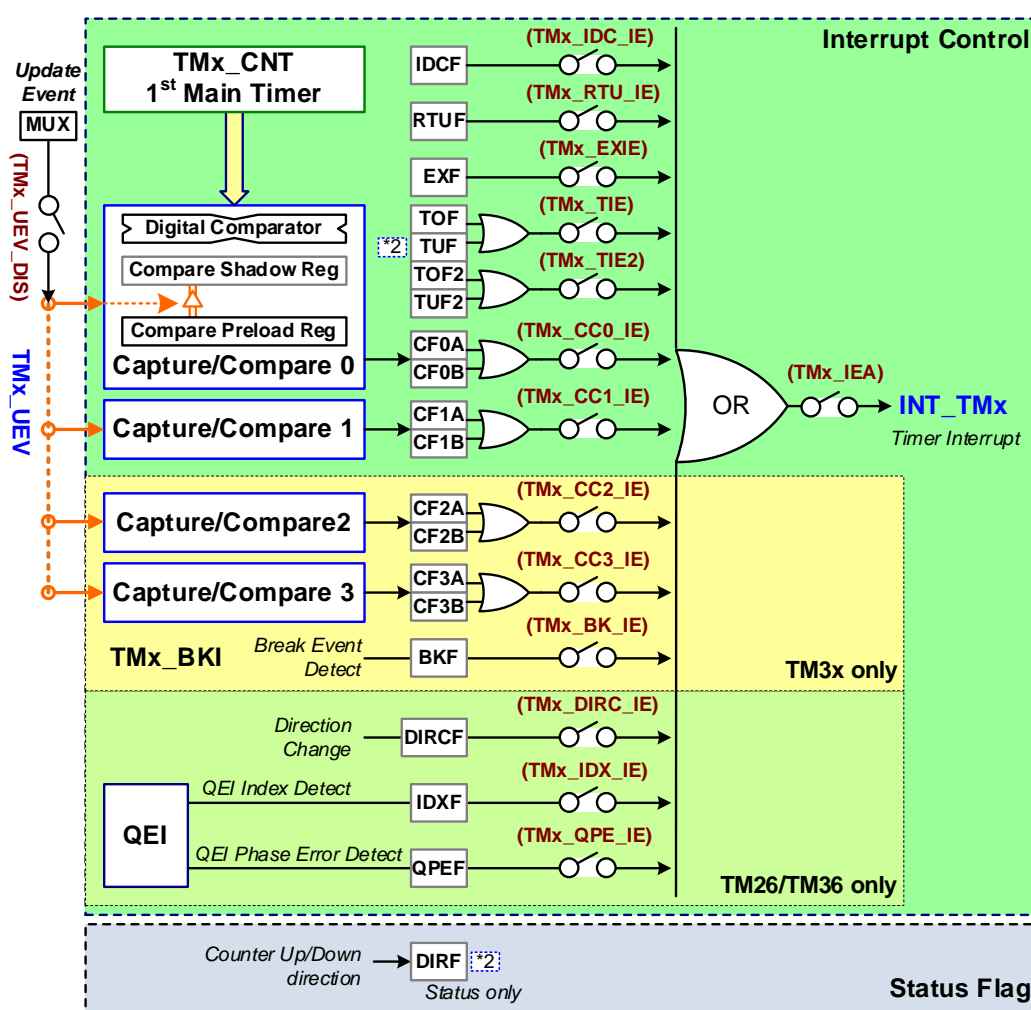
Figure 21-7. Timer Status and Interrupt Control – TM0x/1x



<Note> 1: x= Timer index
2: DIRF/TUF ~ only for TM16

The following diagram is showing the timer interrupt control and Compare Register update control for TM2x and TM3x modules.

Figure 21-8. Timer Status and Interrupt Control – TM2x/3x



<Note> 1: x= Timer index
2: DIRF/TUF ~ only for TM26,TM36

21.7.3. Timer Interrupt Flags

Generally these interrupt flags are set by hardware and clear by software writing 1. Refer the register descriptions for more information about the related interrupt flags and interrupt enable bit.

- **BKF**

Timer break input flag (**TMx_BKF**). There is a related interrupt enable register bit of **TMx_BKIE**.

- **EXF**

Timer external trigger flag (**TMx_EXF**). There is a related interrupt enable register bit of **TMx_EXIE**.

- **TOF / TUF**

Main Timer overflow/underflow flag (**TMx_TOF** / **TMx_TUF**). There is a related interrupt enable register bit of **TMx_TIE** for these two flags.

- **TOF2 / TUF2**

2nd Timer overflow/underflow flag (**TMx_TOF2** / **TMx_TUF2**). There is a related interrupt enable register bit of **TMx_TIE2** for these two flags.

- **CFnA / CFnB**

Timer input capture and output compare event flag (**TMx_CFnA** / **TMx_CFnB**). There is a related interrupt enable register bit of **TMx_CCnE** for these two flags. (n = timer IC/OC/PWM channel index)

- **DIRCF**

Main Timer up/down counting direction change flag (**TMx_DIRCF**). There is a related interrupt enable register bit of **TMx_DIRC_IE** for this flag. When the Main Timer up/down counting direction is changed by external QEI signal(s), this flag will be active and the chip will the interrupt if the interrupt enable bit is enabled.

- **IDXF**

Main Timer QEI external index signal input active detect and internal timer reset flag (**TMx_IDXF**). There is a related interrupt enable register bit of **TMx_IDX_IE** for this flag. When Main Timer QEI external index signal input is detected active, this flag will be active and the Main Timer will be reset. Also the chip will assert the interrupt if the interrupt enable bit is enabled.

- **QPEF**

Main Timer QEI phase state transition errors detect flag (**TMx_QPEF**). There is a related interrupt enable register bit of **TMx_QPE_IE** for this flag. When detects the Main Timer QEI input phase state transition is error, this flag will be active and the chip will assert the interrupt if the interrupt enable bit is enabled.

- **RTUF**

Repetition timer underflow flag (**TMx_RTUF**). There is a related interrupt enable register bit of **TMx_RTU_IE** for this flag. When detects the repetition timer is underflow, this flag will be active and the chip will assert the interrupt if the interrupt enable bit is enabled.

- **IDCF**

Input duty capture complete flag (**TMx_IDCF**). There is a related interrupt enable register bit of **TMx_IDC_IE** for this flag. When detects the duty capture is complete, this flag will be active and the chip will assert the interrupt if the interrupt enable bit is enabled.

[Notify]: The IDCF flag is not supported for MG32F02A128/U128/A064/U064.

21.8. Timer Operation

21.8.1. Timer Operation Mode

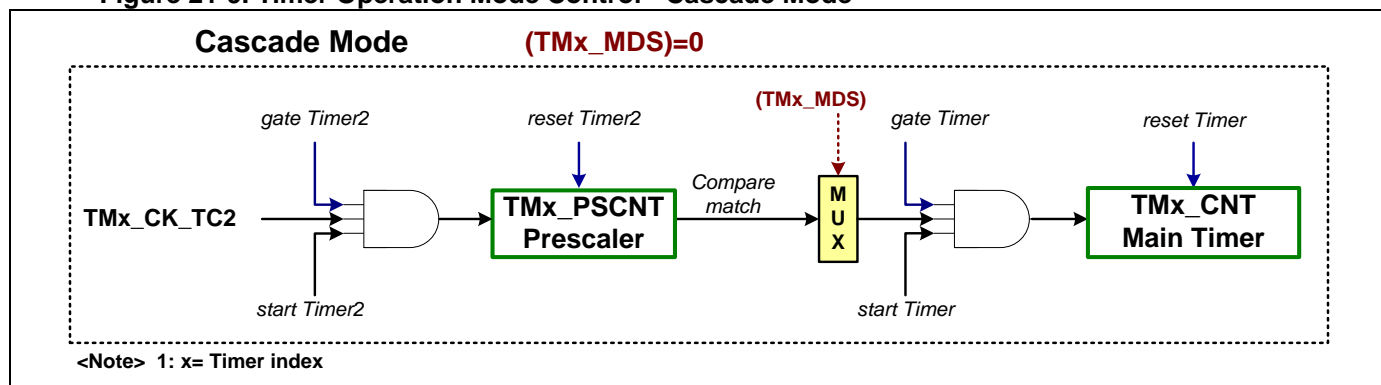
TMx supports three timer operation modes: (1) Cascade mode (2) Separate mode (3) Full-Counter mode. User can set the timer operation mode by setting **TMx_MDS** register. The following diagram is showing the control block of the three timer modes.

- **Timer Operation Mode – Cascade Mode**

The module has a Main Timer with a Prescaler. The clock source of **TMx_CK_TC2** can divide by the Prescaler and output as the input clock of the Main Timer/Counter.

The following diagram is showing the Timer operation mode control block of Cascade mode.

Figure 21-9. Timer Operation Mode Control - Cascade Mode

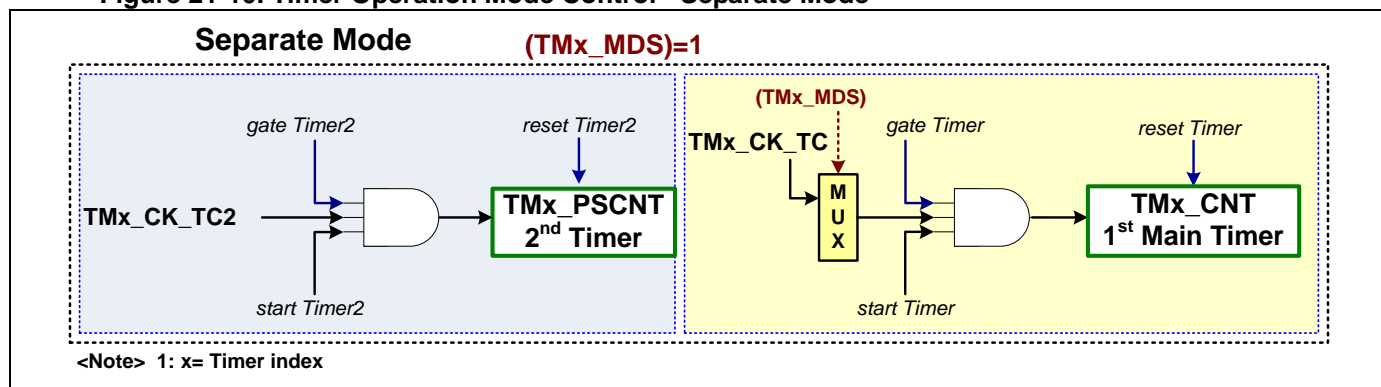


- **Timer Operation Mode – Separate Mode**

The Main Timer and the Prescaler are used as independent timers of 1st Timer and 2nd Timer. The clock source of **TMx_CK_TC** is used for the 1st Timer and the clock source of **TMx_CK_TC2** is used for the 2nd Timer.

The following diagram is showing the Timer operation mode control block of Separate mode.

Figure 21-10. Timer Operation Mode Control - Separate Mode

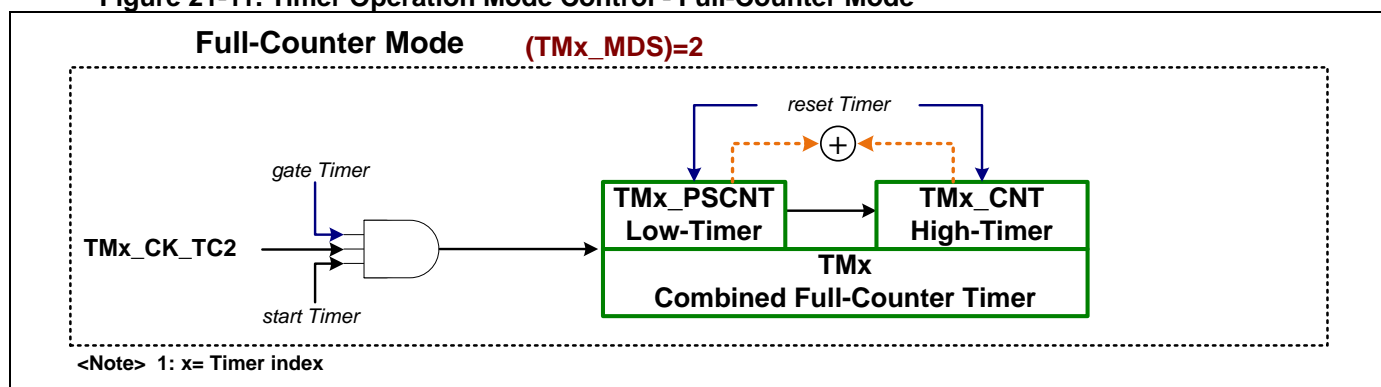


- **Timer Operation Mode – Full-Counter Mode**

The Main Timer and the Prescaler/2nd Timer are able to merge as an integral timer or counter. The clock source of **TMx_CK_TC2** is used as the input clock for this merged full-counter timer.

The following diagram is showing the Timer operation mode control block of Full-Counter mode.

Figure 21-11. Timer Operation Mode Control - Full-Counter Mode



21.8.2. Timer and Counter Control

The timer module is built in a Main timer and a clock Prescaler/2nd timer for three types of operation mode. Refer the previous section for the operation modes.

The Main timer is implemented with 16-bit counter for TM1x/2x/3x modules and 8-bit counter for TM0x modules. Also the clock Prescaler/2nd timer is implemented with 16-bit counter for TM1x/2x/3x modules and 8-bit counter for TM0x modules. There is one counter value register of **TMx_CNT** and one timer reload register of **TMx_ARR** for the Main timer. Also there is one counter value register of **TMx_PSCNT** and one timer reload register of **TMx_PSARR** for the clock Prescaler/2nd timer. By design, there is one **TMx_CNTA** register which is the alias of **TMx_CNT** and combines with **TMx_PSCNT** in a 32-bit register for Full-Counter mode using.

User can set the Main timer counting direction in **TMx_DIR** register and set the 2nd timer counting direction in **TMx_DIR2** register.

[Notify]: The Main timer counting direction control is only supported for TM16, TM26 and TM36 modules.

- **Auto-Reload Register Value Limit**

The setting range of Main timer/counter auto-reload register of **TMx_ARR** is normally able to set 0~0xFF for 8-bit timer and 0~0xFFFF for 16-bit timer. But the range is limited for some specially conditions as following.

When all channels set Two 8bit OC/PWM mode, the auto-reload register value is limited to 0x00zz (zz={0x00~0xFF}). When some channel(s) set Two 8bit OC/PWM mode and some channel(s) set 16bit OC/PWM mode, the reload register value is limited to 0zzxFF (zz={0x00~0xFF}).

Refer the section of "[Timer Output Compare and PWM](#)" for more information.

21.8.3. Timer Operation Mode and Control Validation

The following table is showing the validation of control and the control action of Main and 2nd timers under QEI control function for the three operation modes.

Table 21-3. Timer Operation Mode and Control Validation

Mode	Setting	Timer Control	Main Timer Control				2nd Timer Control				QEI Control
	MDS		EN	DIR	RST	GT	EN2	DIR2	RST2	GT2	
Cascade	0	Independent	V	V	V	V	V	V	V	V	Both Timers
Separate	1	Independent	V	V	V	V	V	V	V	V	Main Timer
Full-Counter	2	Main Only	V	V	V	V	X	X	X	X	Both Timers

Main only : control both Main and 2nd timers simultaneously by Main Timer register

V: control is valid , X: control is no effect

QEI: Timer external Input Up/Down Control

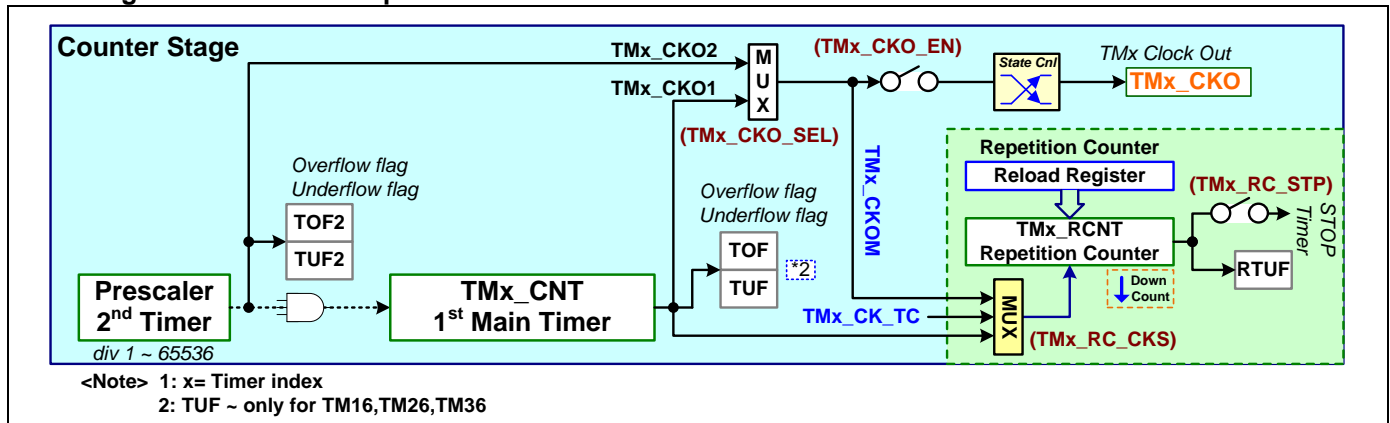
The EN and EN2 are meaning timer enable/start control (**TMx_EN** and **TMx_EN2** registers). The DIR and DIR2 are meaning timer counting direction control (**TMx_DIR** and **TMx_DIR2** registers). The RST and RST2 are meaning trigger input timer reset control. The GT and GT2 are meaning trigger input timer clock gating control.

21.8.4. Repetition Counter

The timer modules of TM2x, TM3x are built in an extra Repetition Counter for stopping Main timer and 2nd timer. User can use the Repetition Counter and set the counting value to stop the timer clock output signal (**TMx_CKO**) or the timer output compare / PWM signals (**TMx_OCn**).

The following diagram is showing the Timer Repetition Counter control block of TM2x, TM3x modules.

Figure 21-12. Timer Repetition Counter Control Block



The Repetition Counter is implemented with 8-bit count-down counter for TM2x/3x modules. User can enable this Repetition Counter in **TMx_RC_EN** register. User can select the counter input clock source from the main timer overflow / underflow output signal, the **TMx_CKOM** pulse signal or the **TMx_CK_TC** clock signal by setting **TMx_RC_CKS** register. The **TMx_CKOM** pulse signal can be selected from the main timer or 2nd timer of overflow / underflow output signal in **TMx_CKO_SEL** register.

*[Notify]: The **TMx_CK_TC** clock signal input for the repetition counter is not supported for MG32F02A128/U128/A064/U064.*

There is one counter value register of **TMx_RCNT** and one timer auto-reload register of **TMx_RARR** for the Repetition Counter. This **TMx_RARR** register is used to set the main timer overflow / underflow number or **TMx_CKOM** pulse number which is as the next updated auto-reload value after the Repetition counter is underflow. When the Repetition counter has been started and counting underflow, the chip will be asserting a RTUF flag (**TMx_RTUF**).

In the condition, user also can enable to stop the main timer and 2nd timer by setting **TMx_RC_STP** register. So user can use this function to stop the timer clock output signal (**TMx_CKO**) or the timer output compare / PWM signals (**TMx_OCn**). Refer the sections of "Timer Clock Output" and "Timer Output Compare and PWM" for more information about the timer clock output and the timer output compare / PWM.

21.9. Timer Trigger Control Block

The Trigger Control block has two functions, one is to control the timer trigger input events and another is to control the timer trigger output events.

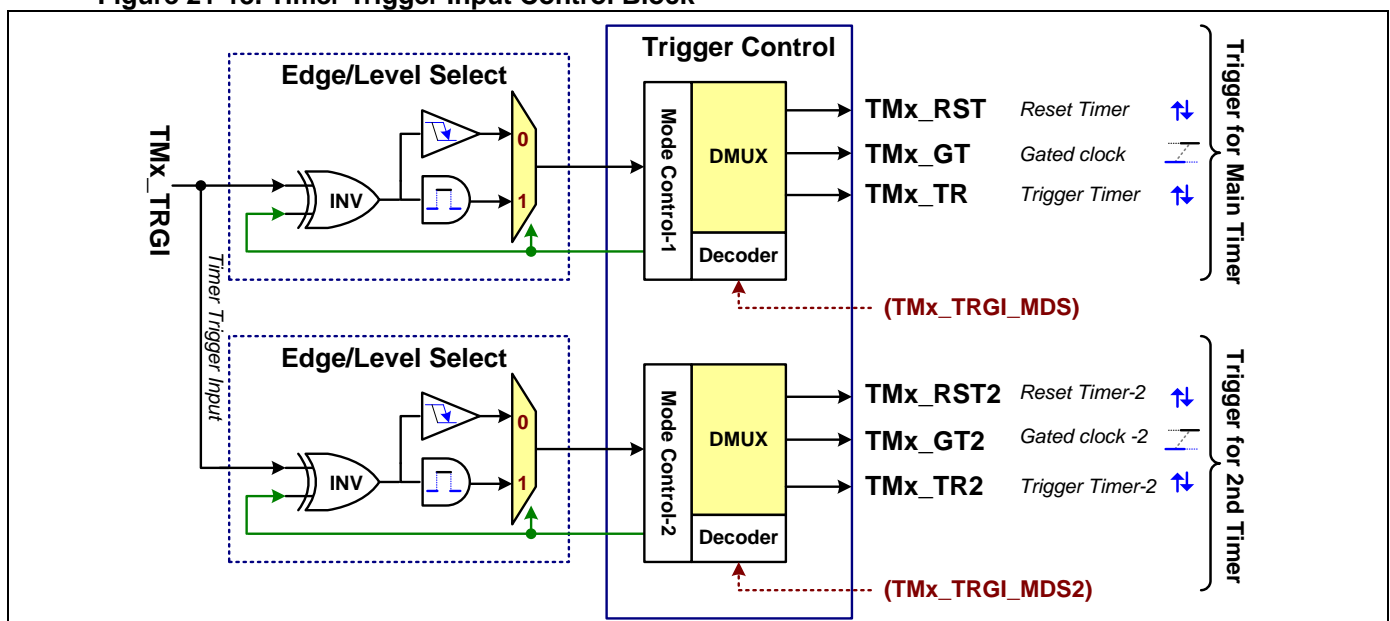
21.9.1. Timer Trigger Input Events

The timer trigger input event can use to do “Reset Timer”, “Gated Clock” and “Timer-Start Trigger”. Use can select by separated **TMx_TRGI_MDS** and **TMx_TRGI2_MDS** registers for Main Timer and 2nd Timer. The input source of the timer trigger input events is selected from external trigger signal of **TMx_ETR**, internal trigger signals of **TMx_ITR[7:0]** or external channel input signal of **TMx_IN0/TMx_IN1** by setting **TMx_TRG_MUX** register.

The “Reset Timer” is to reset the counter for Up-Count or reload Timer Reload Register for Down-Count when detects the selected edge of input signal. The “Gated Clock” is to stop the counter counting by gating timer clock off when detects the selected level of input signal. The “Timer-Start Trigger” is to start the counter counting when the timer is enabled and detects the selected edge of input signal.

The following diagram is showing the timer trigger input control block.

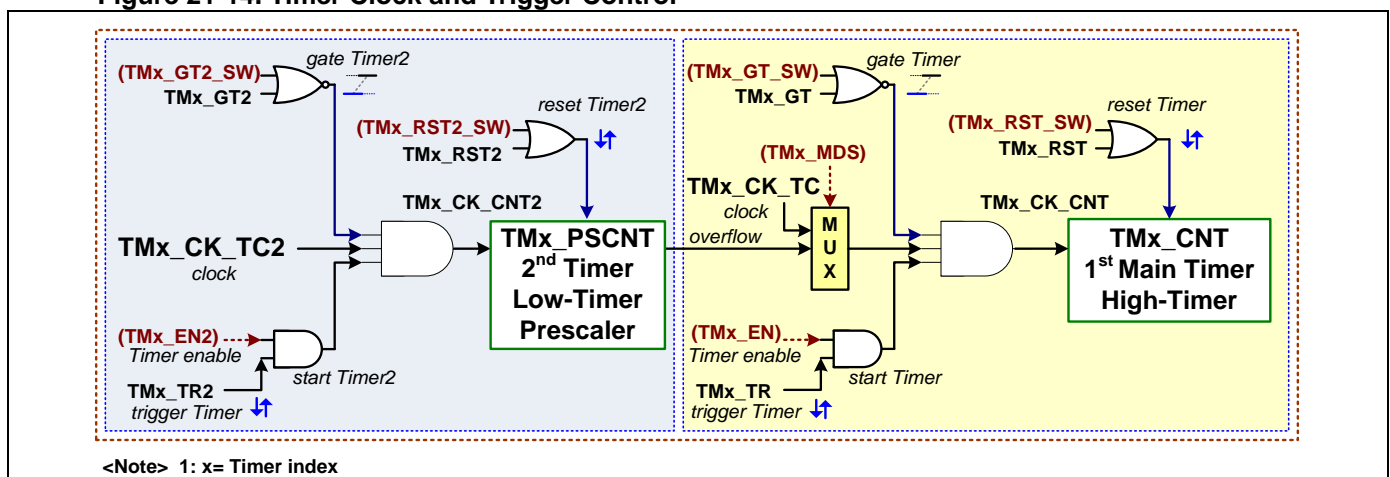
Figure 21-13. Timer Trigger Input Control Block



- **Timer Internal Trigger Software Control**

The module supports the software control function of reset timer and gated clock by setting **TMx_RST_SW**, **TMx_GT_SW** registers for Main Timer and **TMx_RST2_SW**, **TMx_GT2_SW** registers for 2nd Timer.

Figure 21-14. Timer Clock and Trigger Control



● Timer Internal Trigger Input Signals

The following table is showing the internal trigger signals as timer trigger input for each timer module. Specially the **ITR6** and the **ITR7** are the common signals for all timer modules and select the trigger source signal by setting **APB_ITR6_MUX** and **APB_ITR7_MUX** registers. Refer the section of “[Timer Common Trigger/Clock Source Select](#)” in APB Common Control chapter for more detail information.

Table 21-4. Timer Internal Trigger Signals Table

Module	TM00	TM01	TM10	TM16	TM20	TM26	TM36
ITR0 Signal	TM10_TRGO	TM16_TRGO	TM00_TRGO	TM01_TRGO	TM00_TRGO	TM01_TRGO	TM10_TRGO
ITR1 Signal	TM20_TRGO	TM26_TRGO	TM20_TRGO	TM26_TRGO	TM10_TRGO	TM16_TRGO	-
ITR2 Signal	CMP0_OUT	CMP1_OUT	CMP0_OUT	CMP1_OUT	CMP0_OUT	CMP1_OUT	CMP0_OUT
ITR3 Signal	-	-	-	-	-	-	CMP1_OUT
ITR4 Signal	INT_PA	INT_PC	INT_PA	INT_PC	INT_PA	INT_PC	INT_PA
ITR5 Signal	INT_PB	INT_PD	INT_PB	INT_PD	INT_PB	INT_PD	INT_PC
ITR6 Signal	ITR6	ITR6	ITR6	ITR6	ITR6	ITR6	ITR6
ITR7 Signal	ITR7	ITR7	ITR7	ITR7	ITR7	ITR7	ITR7

<Note> 1. ITR6 = {TM00_TRGO, TM10_TRGO, TM20_TRGO, TM36_TRGO, INT_PB, URT1_TMO, URT2_BRO, URT2_TMO}
 2. ITR7 = {TM01_TRGO, TM16_TRGO, TM26_TRGO, ADC0_OUT, INT_PD, URT1_BRO, ICKO_INT, RTC_OUT, TM36_XOR, NCO_P0}
 3. TM26/URT2 and related signals are not supported for MG32F02V032
 4. CMP0/CMP1 and related signals are not supported for MG32F02V032

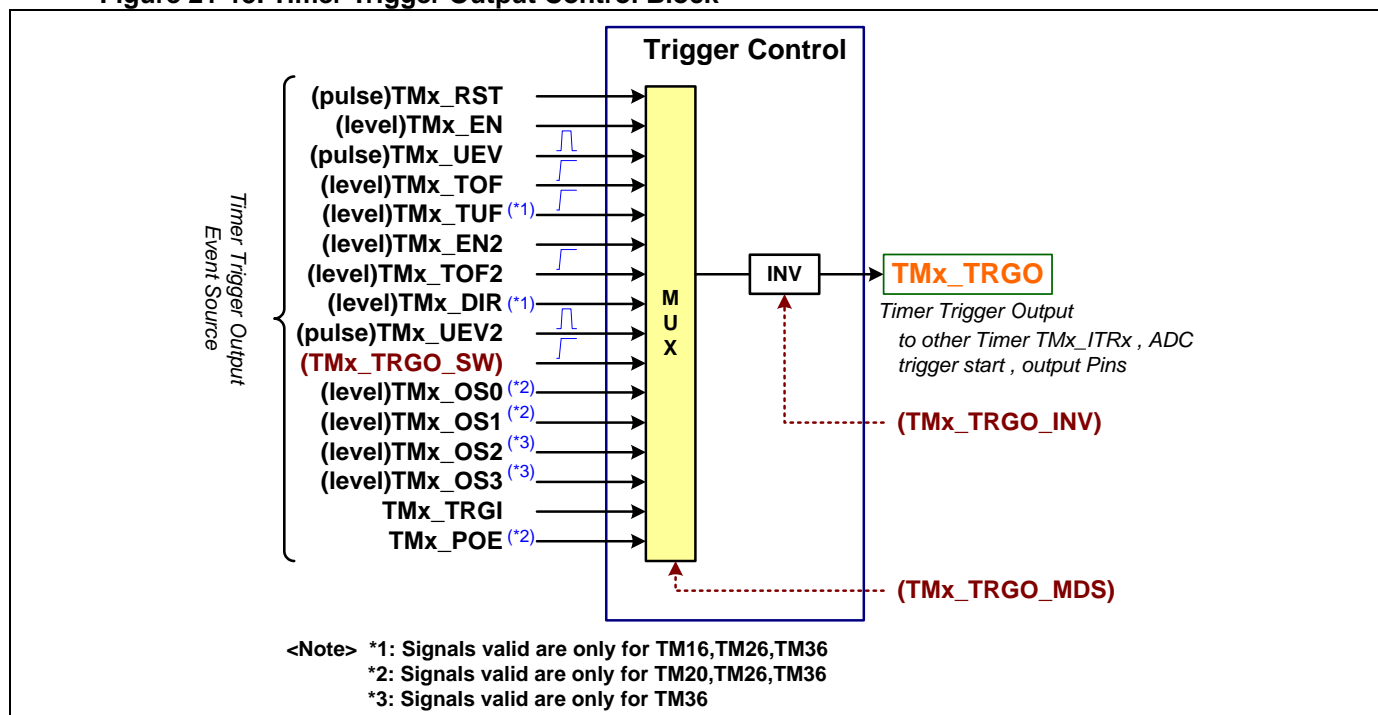
21.9.2. Timer Trigger Output Events

The source of the timer trigger output events are able to come from many internal events or signals of this timer module and output to **TMx_TRGO** output. Also user can use the software register of **TMx_TRGO_SW** to set the trigger output directly. These sources of output event can select by setting **TMx_TRGO_MDS** register and invert the output signal by setting **TMx_TRGO_INV** register.

When **TMx_TRGO_MDS** selects **TMx_UEV** signal as output signal, user can select output pulse function from Main timer overflow TOF event and/or underflow TUF event by setting **TMx_UEV_SEL** register.

The following diagram is showing the timer trigger output control block.

Figure 21-15. Timer Trigger Output Control Block



21.10. Timer Input/Output Channels

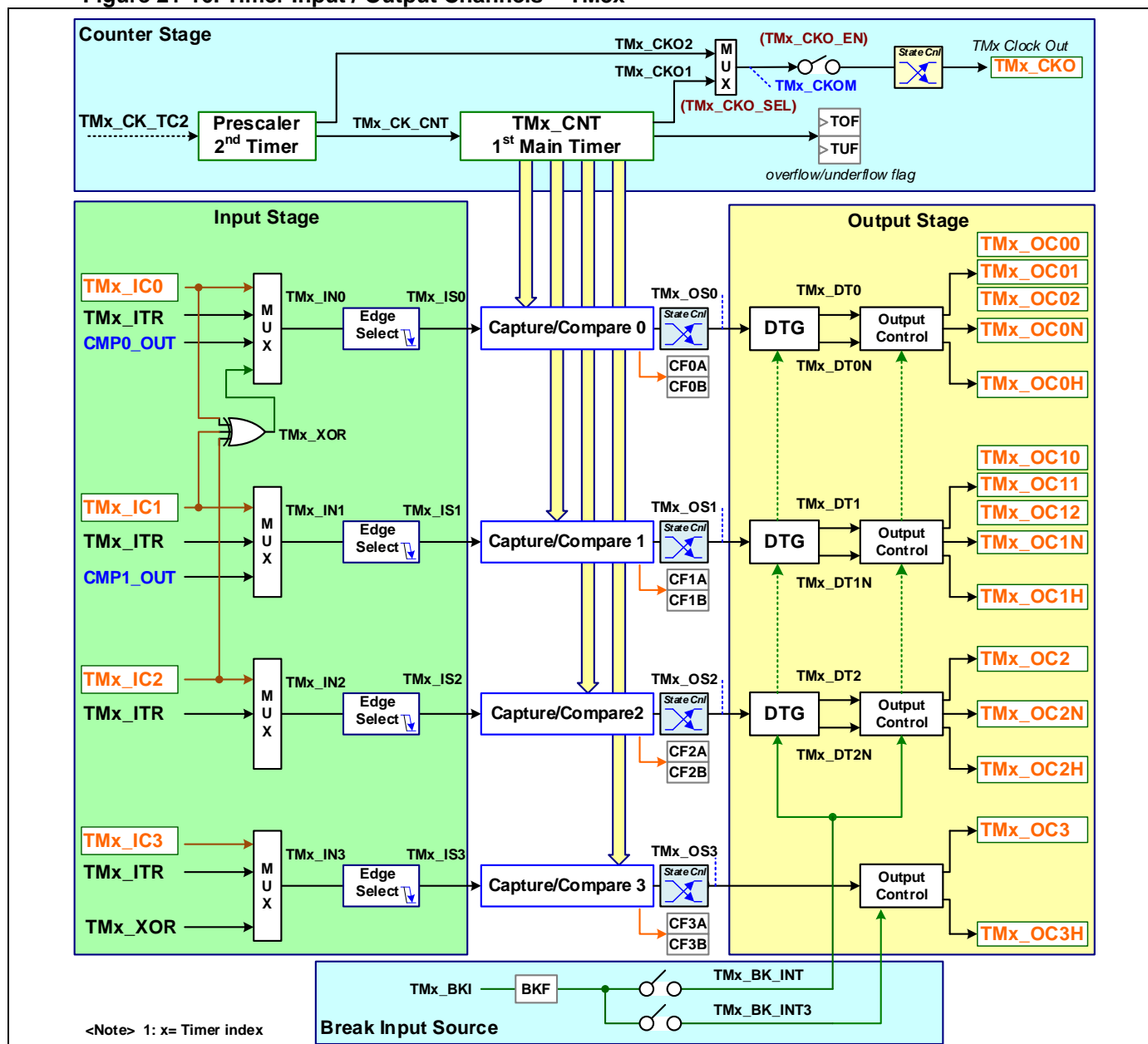
The TM0x and TM1x modules are no channel input selection function as the input capture (IC) and output compare (OC) functions are not support. The TM2x modules have two IC/OC/PWM channels and the TM36 module has four IC/OC/PWM channels.

21.10.1. TM3x Timer Input/Output Channels Block

There are four channels of IC/OC/PWM for TM3x module(s). The following diagram is showing the Timer Input/Output Channels block of TM3x.

[Notify]: MG32F02 V032 is not supported CMP module and all CMPn_OUT (n=0~2).

Figure 21-16. Timer Input / Output Channels ~ TM3x

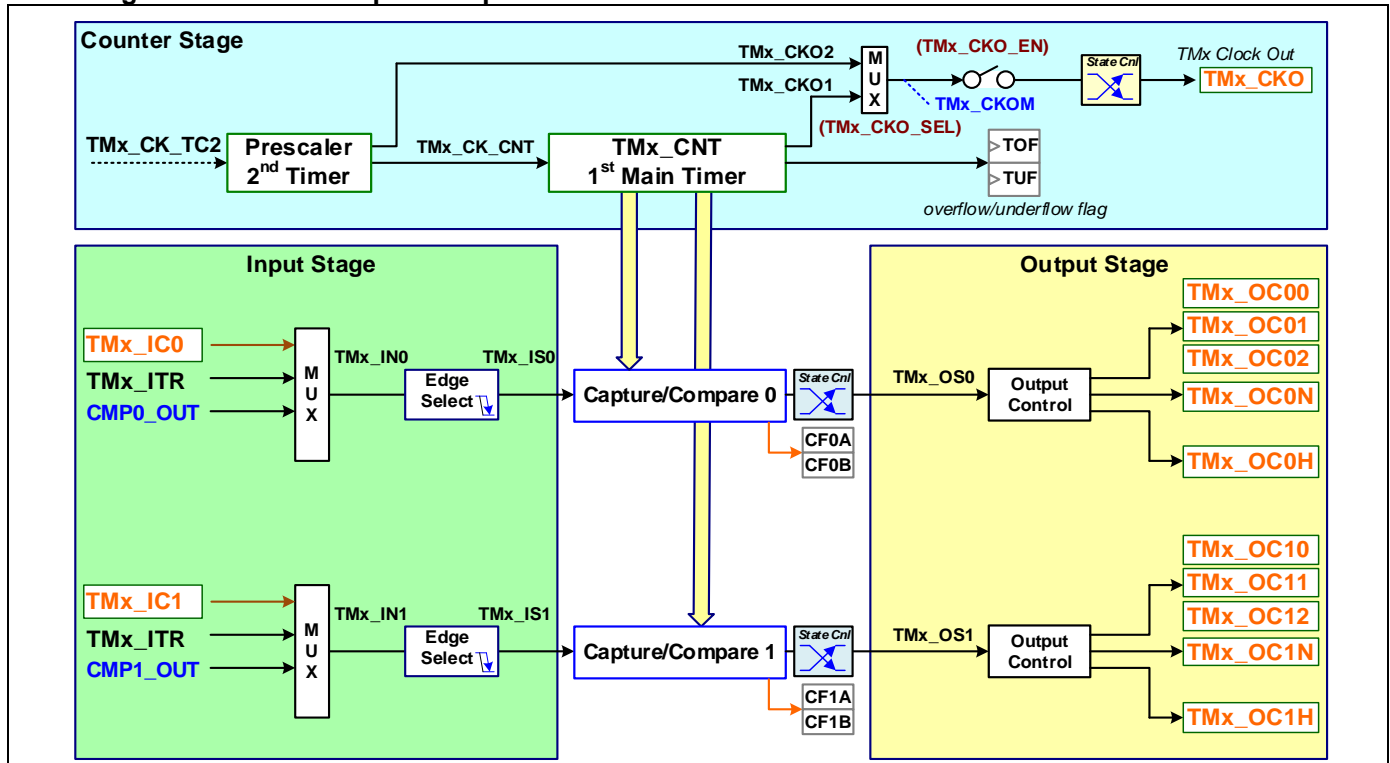


21.10.2. TM2x Timer Input/Output Channels Block

There are two channels of IC/OC/PWM for TM2x module(s). The following diagram is showing the Timer Input/Output Channels block of TM2x.

[Notify]: MG32F02V032 is not supported CMP module and all CMPn_OUT (n=0~1).

Figure 21-17. Timer Input / Output Channels ~ TM2x



21.10.3. Timer Input Channel Control

The signal source of input channels can select by setting **TMx_ICn_MUX** registers (x = module index, n= channel number).

Each channel has four input lines. The **TMx_ICn** signal is direct input from pins and **TMx_ITR** signal is input from internal "Trigger/Clock Control Block". User can select the trigger edge of input signal for input capture function by setting **TMx_ICn_TRGS** register.

Channel-0 and Channel-1 have five output lines of **TMx_OCn0**, **TMx_OCn1**, **TMx_OCn2**, **TMx_OCnN** and **TMx_OCnH**. Channel-2 has three output lines of **TMx_OC2**, **TMx_OC2N** and **TMx_OC2H**. Channel-3 has two output lines of **TMx_OC3**, **TMx_OC3H**.

The **TMx_OCnN** signal is the complement signal of **TMx_OCn**. The **TMx_OCnH** signal is the output signal of compare-H if the channel is configured as two 8-bit of compare or PWM mode by setting **TMx_CCn_MDS** register.

The following table is showing the channel input signals for each timer module. Please refer the table of “Timer Implementation” in the section of “Chip Implementation” about the chip support of timer modules.

Table 21-5. Timer Channel Input Signals Table

Module		TM00	TM01	TM10	TM16	TM20	TM26	TM36
IN0 Signal	IC00	X	X	X	X	TM20_IC0	TM26_IC0	TM36_IC0
	IC01					TM20_ITR	TM26_ITR	TM36_ITR
	IC02					CMP0_OUT	CMP0_OUT	CMP0_OUT
	IC03					-	-	TM36_XOR
IN1 Signal	IC10	X	X	X	X	TM20_IC1	TM26_IC1	TM36_IC1
	IC11					TM20_ITR	TM26_ITR	TM36_ITR
	IC12					CMP1_OUT	CMP1_OUT	CMP1_OUT
	IC13					-	-	Reserved
IN2 Signal	IC20	X	X	X	X	X	X	TM36_IC2
	IC21							TM36_ITR
	IC22							-
	IC23							Reserved
IN3 Signal	IC30	X	X	X	X	X	X	TM36_IC3
	IC31							TM36_ITR
	IC32							-
	IC33							TM36_XOR

<Note> 1. TM26 and related signals are not supported for MG32F02V032
2. CMP0/CMP1 and related signals are not supported for MG32F02V032

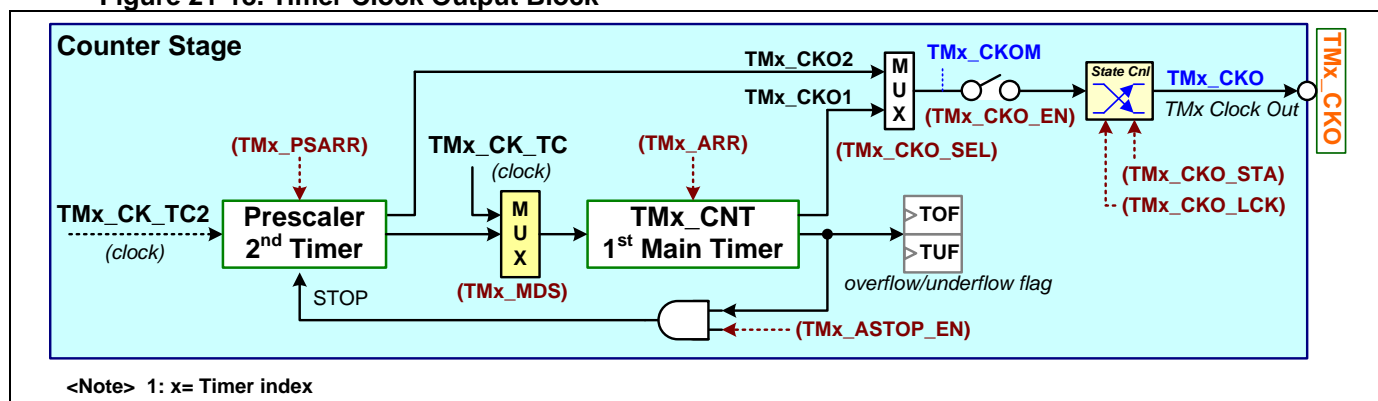
21.10.4. Timer Clock Output

All timer modules can provide the **TMx_CKO** signal as a clock output to external port of **TMx_CKO** and enable to output by setting **TMx_CKO_EN** register. The clock output signal can generate from the timer overflow/underflow event of Main Timer or 2nd Timer and select by setting **TMx_CKO_SEL** register.

User can set the initial state of the clock output signal by setting **TMx_CKO_STA** register. Specially, the initial state register has a write protection register bit of **TMx_CKO_LCK**. It can be written only when the write protection bit is written “1” simultaneously.

The following diagram is showing the Timer CKO Clock Output block.

Figure 21-18. Timer Clock Output Block



- **CKO output from Main Timer**

User can select the timer clock output coming from Main Timer by setting **TMx_CKO_SEL** register. When the timer operation mode is set Separate Mode, use **TMx_CK_TC** as timer input clock. The time period is set by Main Timer auto-reload register of **TMx_ARR**. When the timer operation mode is set Full-Counter Mode or Cascade Mode, use **TMx_CK_TC2** as timer input clock. The time period is set by both **TMx_ARR** and **TMx_PSARR** registers.

- **CKO output from 2nd Timer**

User can select the timer clock output coming from Main Timer by setting **TMx_CKO_SEL** register. In the condition, use **TMx_CK_TC2** as timer input clock and the timer operation mode must set Separate Mode. The time period set by 2nd Timer auto-reload register of **TMx_PSARR**.

● Clock Output Auto Stop Mode

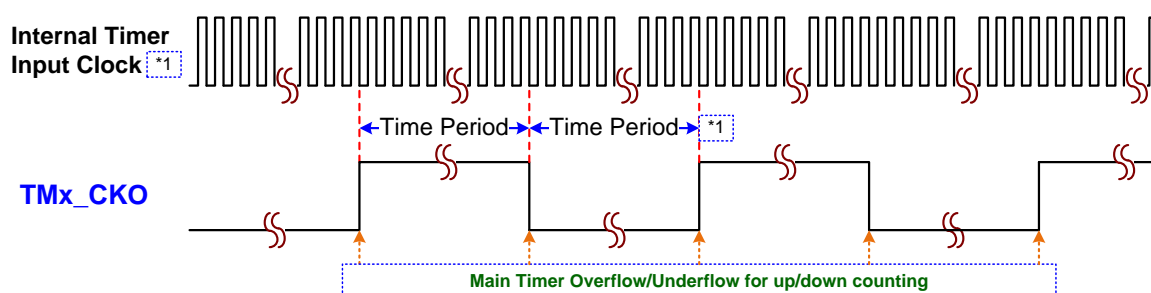
The timer module supports the timer clock output **auto stop mode** when selects clock output from 2nd Timer. The **TMx_ASTOP_EN** register is used to enable the timer clock output auto stop mode. When it enables, the Main Timer will act like as a repetition timer. The 2nd Timer will auto stop after Main timer counting overflow/underflow and stops the **TMx_CKO** signal.

User can enable to force the timer overflow or underflow flag auto-clear for clock output **auto stop mode** by setting **TMx_ACLEAR_EN** register. This bit is no effect if **TMx_ASTOP_EN** is disabled. When enables, the timer will auto clear the flag of TMx_TOF or TMx_TUF after timer counting is overflow or underflow. For timer trigger start by external signal application, the clock output will enter **auto stop mode** for next timer trigger start.

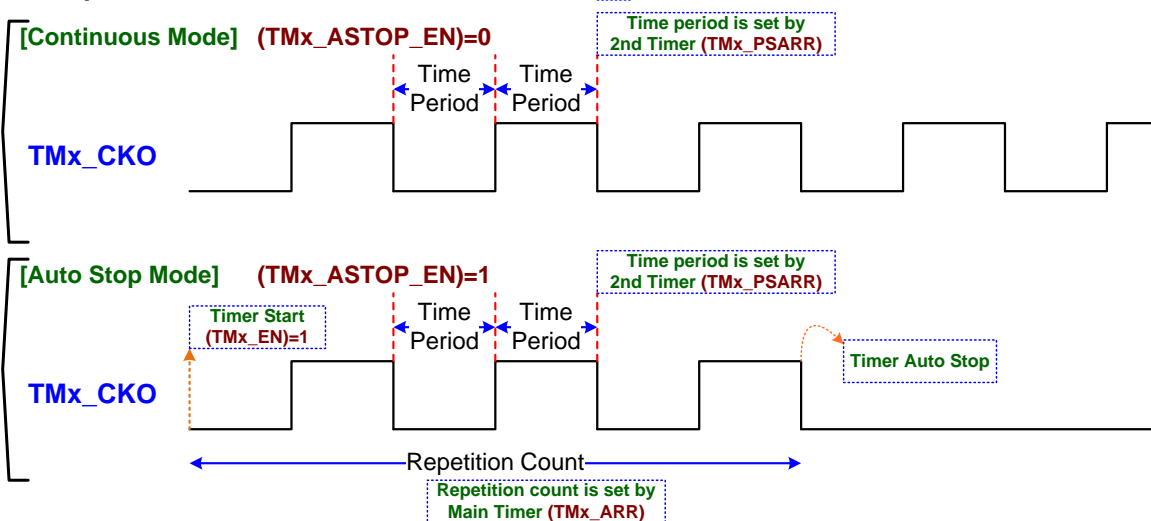
The following diagram is showing the Timer CKO Clock Output Timing.

Figure 21-19. Timer CKO Clock Output Timing

A. Output from TMx_CKO1 (TMx_CKO_SEL)=1



B. Output from TMx_CKO2 (TMx_CKO_SEL)=0



- <Note-1> (1) When select [Separate Mode] , use **TMx_CK_TC** as timer input clock. The time period is set by Main Timer (**TMx_ARR**).
 (2) When select [Full-Counter Mode] or [Cascade Mode] , use **TMx_CK_TC2** as timer input clock. The time period is set by both (**TMx_ARR**) and (**TMx_PSARR**).
 <Note-2> Use **TMx_CK_TC2** as timer input clock. The time period set by 2nd Timer (**TMx_PSARR**).

● Clock Output with Repetition Counter

The timer modules of TM2x, TM3x are built in an extra Repetition Counter for stopping Main timer and 2nd timer. User can use the Repetition Counter and set the counting value to stop the timer clock output **TMx_CKO** signal. Refer the section of "[Repetition Counter](#)" for more information about Repetition Counter.

21.11. Timer Capture and Compare Block

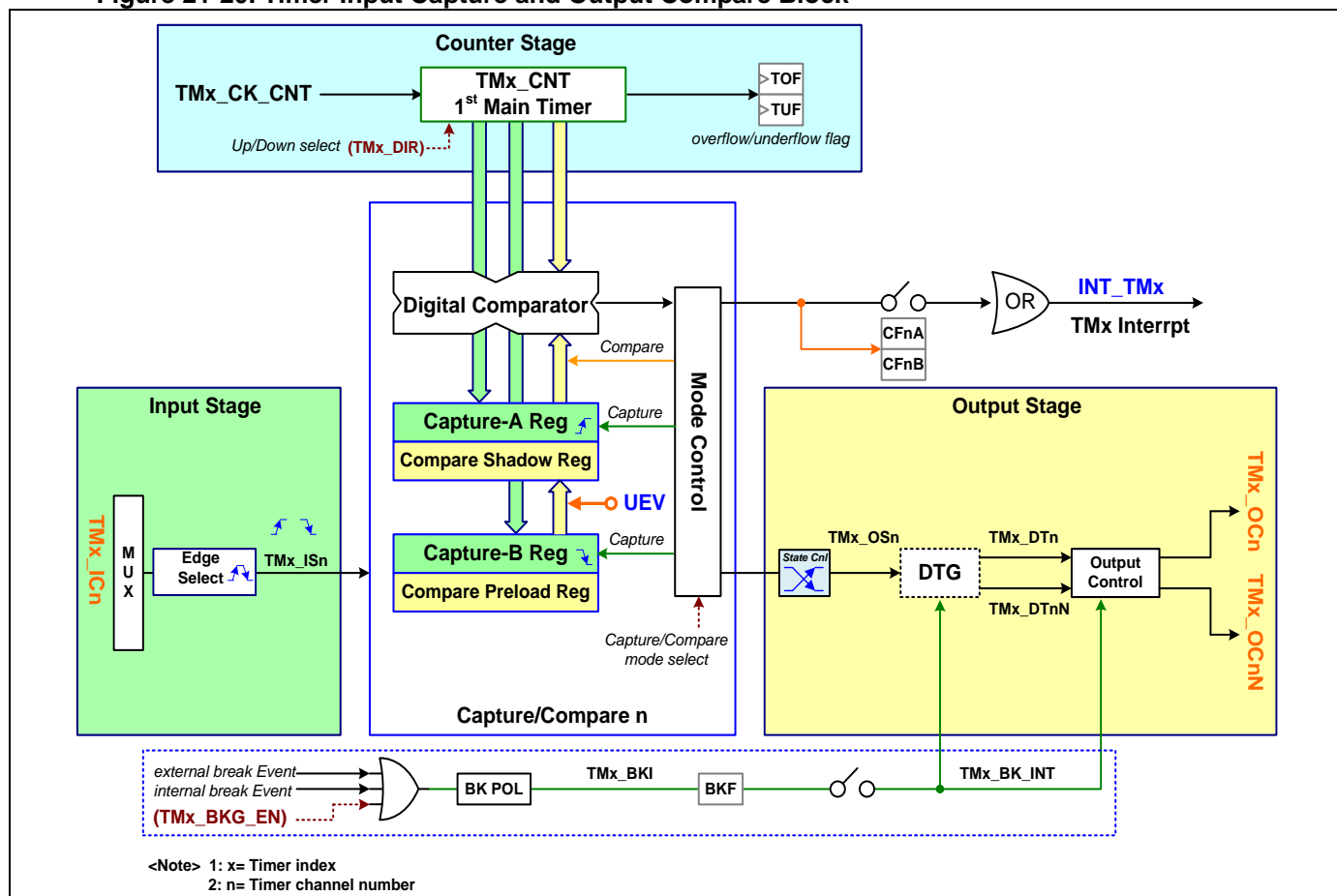
The input capture (IC) and output compare (OC) functions are only supported for TM2x and TM3x modules. TM0x and TM1x modules are no the functions of the input capture/output compare.

21.11.1. Timer Channel Mode

User can configure each of the timer IC/OC/PWM channel independently as input capture, output compare or PWM mode by setting **TMx_CCn_MDS** register.

The following diagram is showing the Input Capture and Output Compare blocks. The DTG (Dead Time Generator) and Break control are only support for TM36 module.

Figure 21-20. Timer Input Capture and Output Compare Block



21.11.2. Software Input Capture and Output Compare Generation

The module supports the IC/OC/PWM event generation by software control. There are two software enable bits to generate IC/OC/PWM event by setting **TMx_CCnA_SEN** and **TMx_CCnB_SEN** registers for each timer channel.

For input capture mode, the **TMx_CCnA_SEN** register is used to trigger to generate rising edge capture event. The **TMx_CCnB_SEN** register is used to trigger to generate falling edge capture event. For output compare or PWM mode, the **TMx_CCnA_SEN** register is used to set **TMx_CFnA** flag only. The **TMx_CCnB_SEN** register is used to set **TMx_CFnB** flag only. When user set the software control bit of **TMx_CCnA_SEN** or **TMx_CCnB_SEN**, it will be automatically cleared by hardware after the IC or OC event is generated.

21.11.3. Timer Capture and Compare Register Control

There are two timer capture and compare registers of **TMx_CC0A** and **TMx_CC0B** for each input/output channel. These registers are used to store the timer capture value for input capture mode or the timer compare threshold value for output compare mode or output PWM mode.

The following table is showing the registers' control function of timer capture and compare. Please refer the sections of ["Timer Input Capture"](#) and ["Timer Output Compare and PWM"](#) for more information.

Table 21-6. Timer Capture and Compare Register Control

Capture/Compare		Input Capture				Output Compare/PWM	
Operation Mode		Cascade/Separate Mode		Full-Counter Mode		Cascade/Separate/Full-Counter Mode	
Function		Dual Edge Capture	Single Edge Capture	Dual Edge Capture	Single Edge Capture	One 16-bit Compare Out	Two 8-bit Compare Out
TMx Register							
CCnA	H-byte	rising edge capture 16bit data	first capture 16bit data	rising and falling edge capture 32bit data	rising or falling edge capture 32bit data	compared shadow register	compared-H path shadow register
	L-byte						compared-L path shadow register
CCnB	H-byte	falling edge capture 16bit data	2nd capture 16bit data			compared preload register	compared-H path preload register
	L-byte						compared-L path preload register

<Note> CCnA, CCnB : Timer Capture and Compare Register, n= {0,1,2,3}

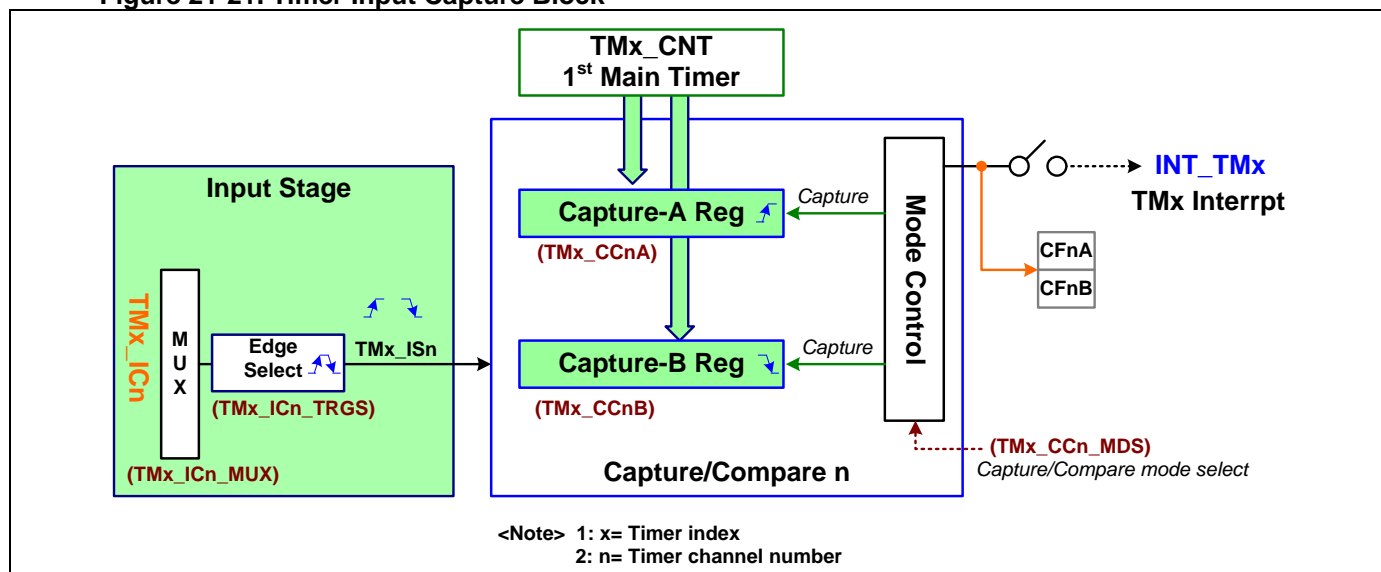
21.12. Timer Input Capture

21.12.1. Capture Data and Edge Selection

If the channel is configured as input capture mode, the registers of **TMx_CCnA** and **TMx_CCnB** are used to capture the counter value when the trigger event is happened for the input capture signal.

The following diagram is showing the Input Capture Block.

Figure 21-21. Timer Input Capture Block



The register of **TMx_ICn_TRGS** is used to select the trigger edge of the input capture signal.

- **Single Edge**

When selects single edge in **TMx_ICn_TRGS** register, the register of **TMx_CCnA** is used to store the first capture data and the register of **TMx_CCnB** is used to store the 2nd capture data for timer Cascade and Separate Modes. For Full-Counter Mode, the register of **TMx_CCnA** is used to store the Msb 16-bit capture data and the register of **TMx_CCnB** is used to store the Lsb 16-bit capture data.

The **TMx_CFnA** and **TMx_CFnB** bits are use as the Input capture event main flag and sub flag.

- **Dual Edge**

When selects dual edge in **TMx_ICn_TRGS** register, the register of **TMx_CCnA** is used to store the rising edge capture data and the register of **TMx_CCnB** is used to store the falling edge capture data.

The **TMx_CFnA** and **TMx_CFnB** bits are use as the Input capture rising edge event flag and falling edge event flag.

21.12.2. Capture Data Overrun

For the Input Capture mode, the capture data buffer will overrun if both the **TMx_CCnA** and **TMx_CCnB** registers are filled capture data and not yet been gotten out by firmware at next capture event happened for single edge capture mode. Also the **TMx_CCnA** or **TMx_CCnB** register is filled capture data and not yet been gotten out by firmware at next rising or falling capture event happened for dual edge capture mode. User can select the capture data buffer overrun mode by setting independent register of **TMx_OVRn_MDS** for each timer channel. When selects “Overwritten” and the capture data buffer is overrun, the **TMx_CCnA** or **TMx_CCnB** register will be updated by new capture data. When selects “Keep” and the capture data buffer is overrun, the **TMx_CCnA** or **TMx_CCnB** register will not be updated and keep the old capture data.

21.12.3. Capture Control and Status

The following table is showing the related control registers and status registers for timer Input Capture mode.

Table 21-7. Timer Input Capture Mode

Mode	Setting	Edge Select	Setting	Capture Register		Status Register		Capture Buffering
	MDS		ICn_TRGS	CCnA	CCnB	CFnA	CFnB	
Cascade	0	Rising edge	1	1st CNT	2nd CNT	1st capture	2nd capture	Yes
		Falling edge	2	1st CNT	2nd CNT	1st capture	2nd capture	Yes
		Dual edge	3	rising edge CNT	falling edge CNT	rising edge	falling edge	No
Separate	1	Rising edge	1	1st CNT	2nd CNT	1st capture	2nd capture	Yes
		Falling edge	2	1st CNT	2nd CNT	1st capture	2nd capture	Yes
		Dual edge	3	rising edge CNT	falling edge CNT	rising edge	falling edge	No
Full-Counter	2	Rising edge	1	CNT	PSCNT	1st capture	2nd capture	No
		Falling edge	2	CNT	PSCNT	1st capture	2nd capture	No
		Dual edge	3	CNT	PSCNT	rising edge	falling edge	No

CNT : Main Timer counter value , PSCNT : Prescaler/2nd Timer counter value

Capture Buffering: CCnA use as capture buffer of first edge. CCnB use as capture buffer of next edge.

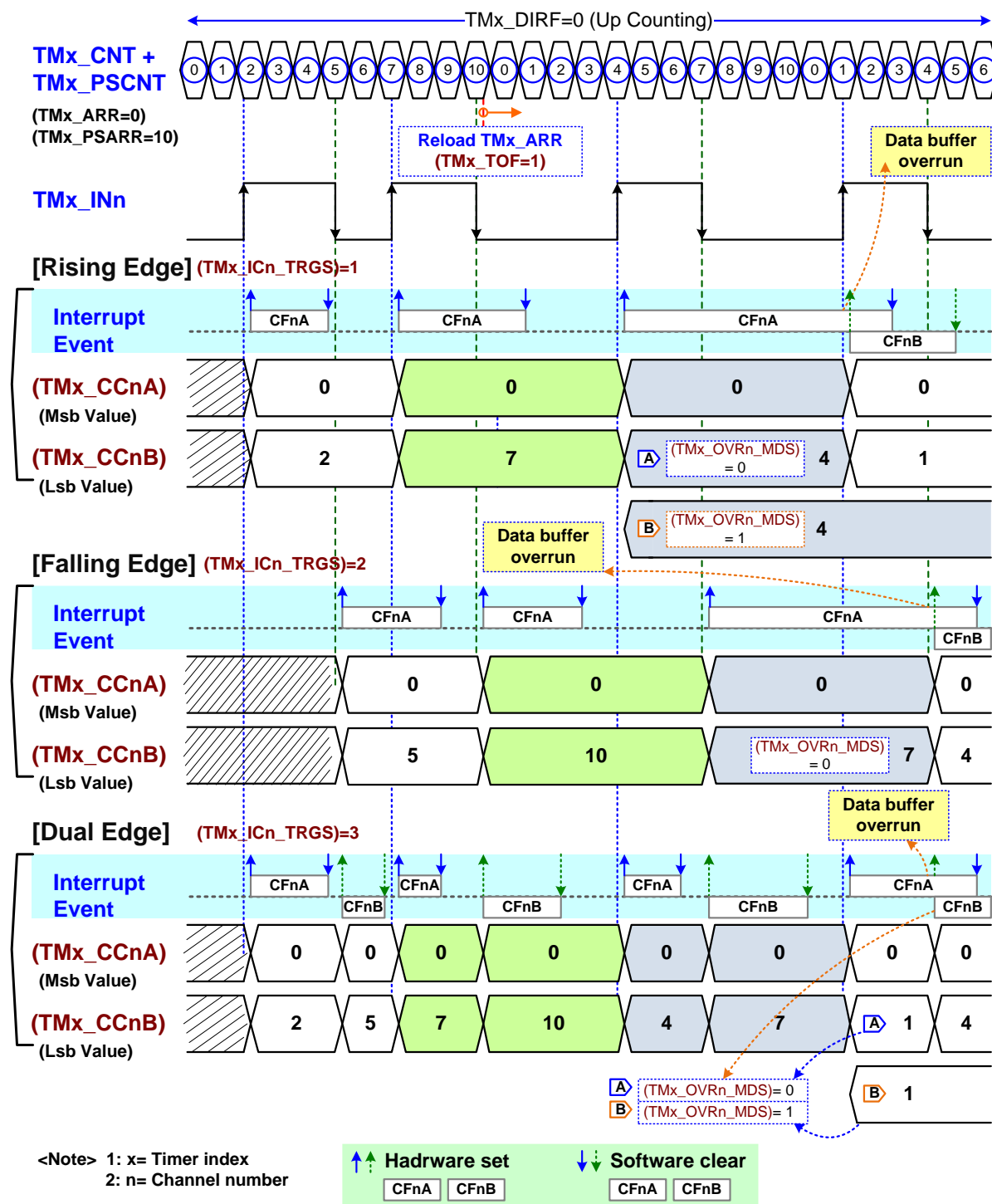
Hardware will copy CCnB to CCnA after reading CCnA. (CCnA, CCnB : n = channel number)

● Full-Counter Mode

The Main Timer and the Prescaler/2nd Timer are able to merge as an integral timer or counter for timer Full-Counter Mode.

The following diagram is showing the timer input capture timing of Full-Counter mode with up counting by **TMx_ARR = 10**.

Figure 21-22. Timer Input Capture Timing for Full-Counter Mode



The module has a Main Timer with a Prescaler for timer Cascade mode. The Main Timer and the Prescaler are used as independent timers of 1st Timer and 2nd Timer for timer Separate mode.

Figure 21-23. Timer Input Capture Timing for Cascade and Separate Modes



21.12.4. Duty Capture Control

The timer input capture is supported the duty capture function for TM2x or TM3x modules. User can enable this function to capture the duty or cycle time of input waveform after pass through three edges of input capture signal by setting **TMx_IDC_EN** register.

[Notify]: The duty capture function is not supported for MG32F02A128/U128/A064/U064.

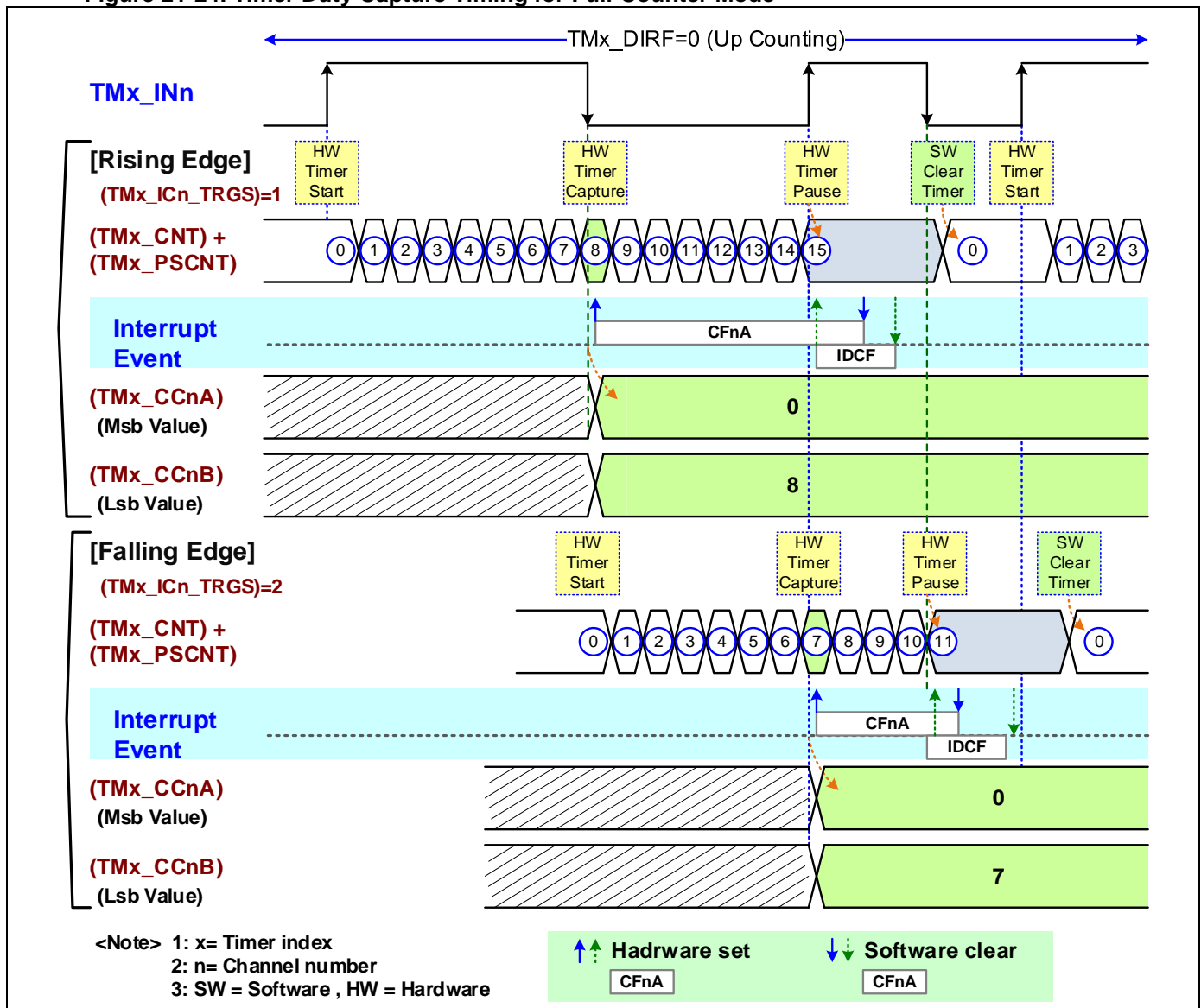
User can input the capture signal from external channel input signal of **TMx_IN0/TMx_IN1** and select "Timer-Start Trigger" by setting **TMx_TRGI_MDS** or **TMx_TRGI2_MDS** register for different operation modes of Cascade mode, Separate mode and Full-Counter mode. Also user can set the same registers to select the first trigger edge whether is rising edge or falling edge.

It also implements an input duty capture complete flag (**TMx_IDCF**) and a related interrupt enable register bit of **TMx_IDC_IE**. When detects the duty capture is complete, this flag will be active and the chip will assert the interrupt if the interrupt enable bit is enabled. User can get the pulse width and cycle time in this **IDCF** interrupt service routine to calculate the duty of input waveform.

● Full-Counter Mode

The Main Timer and the Prescaler/2nd Timer are able to merge as an integral timer or counter for timer Full-Counter Mode. The following diagram is showing the timer duty capture timing of Full-Counter mode with up counting.

Figure 21-24. Timer Duty Capture Timing for Full-Counter Mode



For the Full-Counter Mode, the timer will start at leading edge of input capture signal and capture counter to the registers of **TMx_CCnA** and **TMx_CCnB** at trailing edge of input capture signal after the duty capture function was enabled. User can get the capture value from these registers to calculate the pulse width. Following, then timer is stopped at next leading edge. User can get the timer counter from the registers of **TMx_CNT** and **TMx_PSCNT** for the Main timer and Prescaler/2nd timer. So user can calculate the capture cycle time and duty of the input waveform. The duty cycle is $\{TMx_CCnA, TMx_CCnB\} / \{TMx_CNT, TMx_PSCNT\}$. The TMx_CCnA and TMx_CNT are MSB.

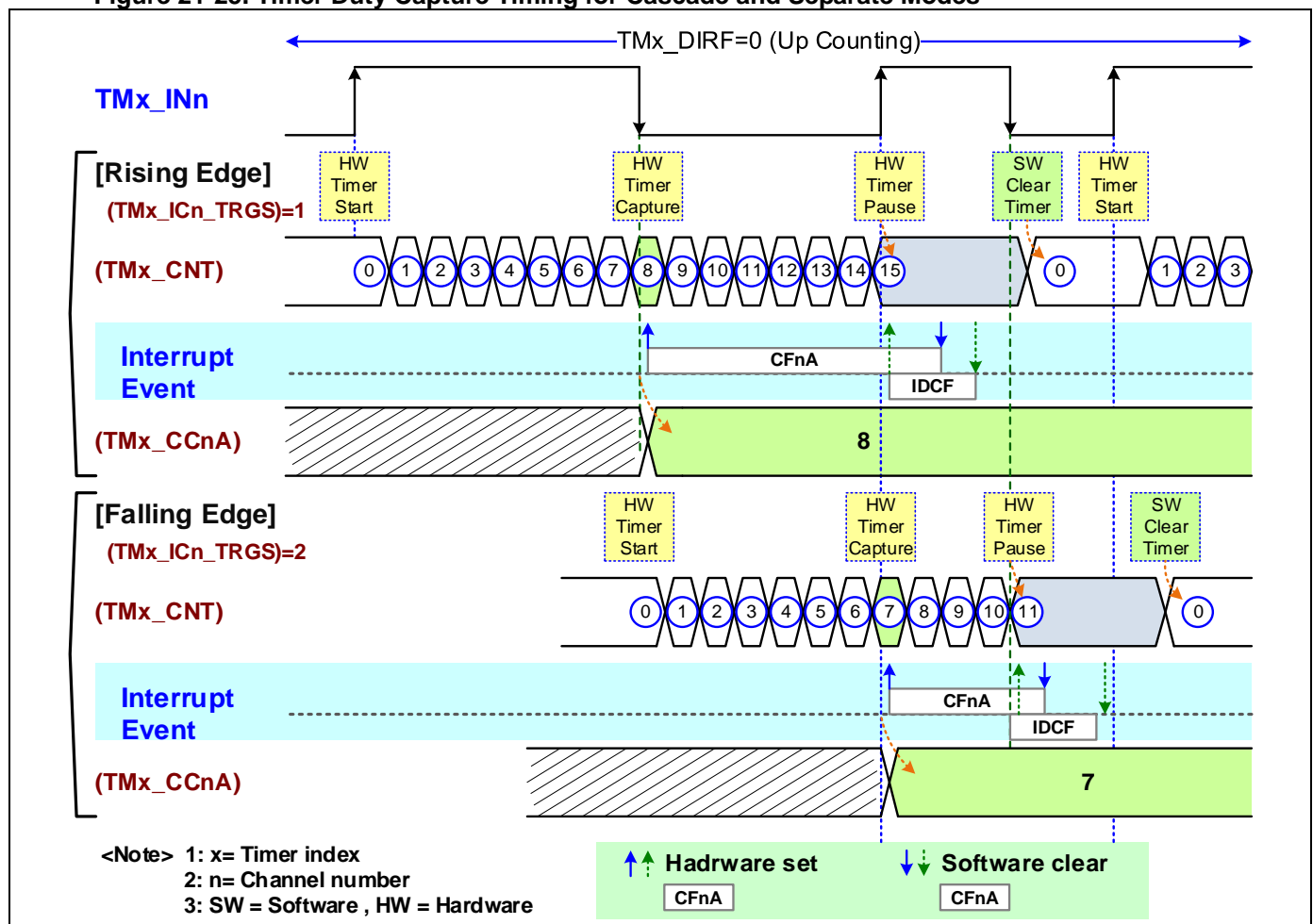
● Cascade and Separate Modes

The module has a Main Timer with a Prescaler for timer Cascade mode. The Main Timer and the Prescaler are used as independent timers of 1st Timer and 2nd Timer for timer Separate mode.

For the Cascade and Separate Modes, the timer will start at leading edge of input capture signal and capture counter to the register of **TMx_CCnA** at trailing edge of input capture signal after the duty capture function was enabled. User can get the capture value from this registers to calculate the pulse width. Following, then timer is stopped at next leading edge. User can get the timer counter from the register of **TMx_CNT** or **TMx_PSCNT** for the Main timer or Prescaler/2nd timer. So user can calculate the capture cycle time and duty of the input waveform. The duty cycle is $\{TMx_CCnA\} / \{TMx_CNT\}$.

The following diagram is showing the timer duty capture timing for Cascade and Separate mode with up counting.

Figure 21-25. Timer Duty Capture Timing for Cascade and Separate Modes



21.13. Timer Output Compare and PWM

21.13.1. Compare Reload Register

If the channel is configured as output compare/PWM mode, the register of **TMx_CCnA** is used as the compared preload register for software setting and the register of **TMx_CCnB** is used as the compared shadow register for timer output compare. The value of **TMx_CCnB** register will be copied to **TMx_CCnA** register when **TMx_CCnB** was written.

By application, there is a timer output compare reload function lock enable bit of **TMx_OC_LCK** for all channels. When this bit is enabled and timer update event is happened, it is locked that the compare preload registers of **TMx_CCnB** reload to compare shadow buffer registers of **TMx_CCnA**. Until this bit disables, these compare preload registers will update the compare shadow buffer at next timer update event happened.

When the channel is configured as output Two 8-bit compare/PWM mode, the register of **TMx_CCnA** is separated to low 8-bit compared shadow register for compare-L path and high 8-bit compared shadow register for compare-H path. The register of **TMx_CCnB** is separated to low 8-bit compared preload register for compare-L path and high 8-bit compared preload register for compare-H path.

When the PWM mode is selected for a channel, the register of **TMx_PWM_MDS** is used to select the PWM alignment mode of Edge Left-aligned or Center-aligned.

When both **TMx_CCnA** and **TMx_CCnB** value is equal **TMx_ARR** or 0x0000 in central-align mode, the output high and low width are 0x10000 clocks' width.

21.13.2. Compare Output State

The TMx timer module is able to set the initial state of the timer compare output **OSn** by setting **TMx_OSn_STA** registers for 16-bit Compare/PWM mode. Also user can set the initial state of the timer compare-L output **OSn** by setting **TMx_OSn_STA** registers and the timer compare-H output **OSnH** by setting **TMx_OSnH_STA** registers for Two 8-bit Compare/PWM mode.

These initial state registers have the independent write protection register bit of **TMx_OSn_LCK** or **TMx_OSnH_LCK**. They can be written only when the related write protection bit is written "1" simultaneously.

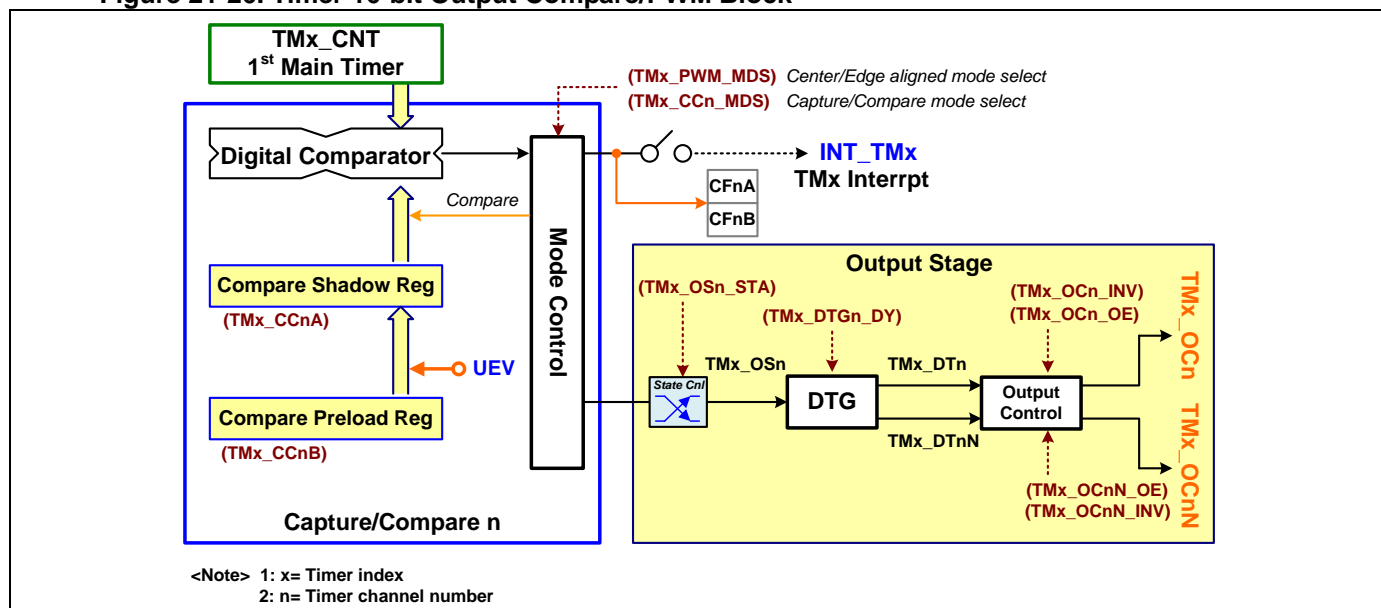
21.13.3. 16-bit Compare/PWM Mode

The **TMx_CFnA** bit is use as output compare event flag for 16-bit comparator mode and is used as up counting PWM compare flag for center-alignment PWM mode.

The **TMx_CFnB** bit is no using for 16-bit comparator mode. But it is used as down counting PWM compare flag for center-alignment PWM mode.

The following diagram is showing the 16-bit Compare/PWM Output Block.

Figure 21-26. Timer 16-bit Output Compare/PWM Block

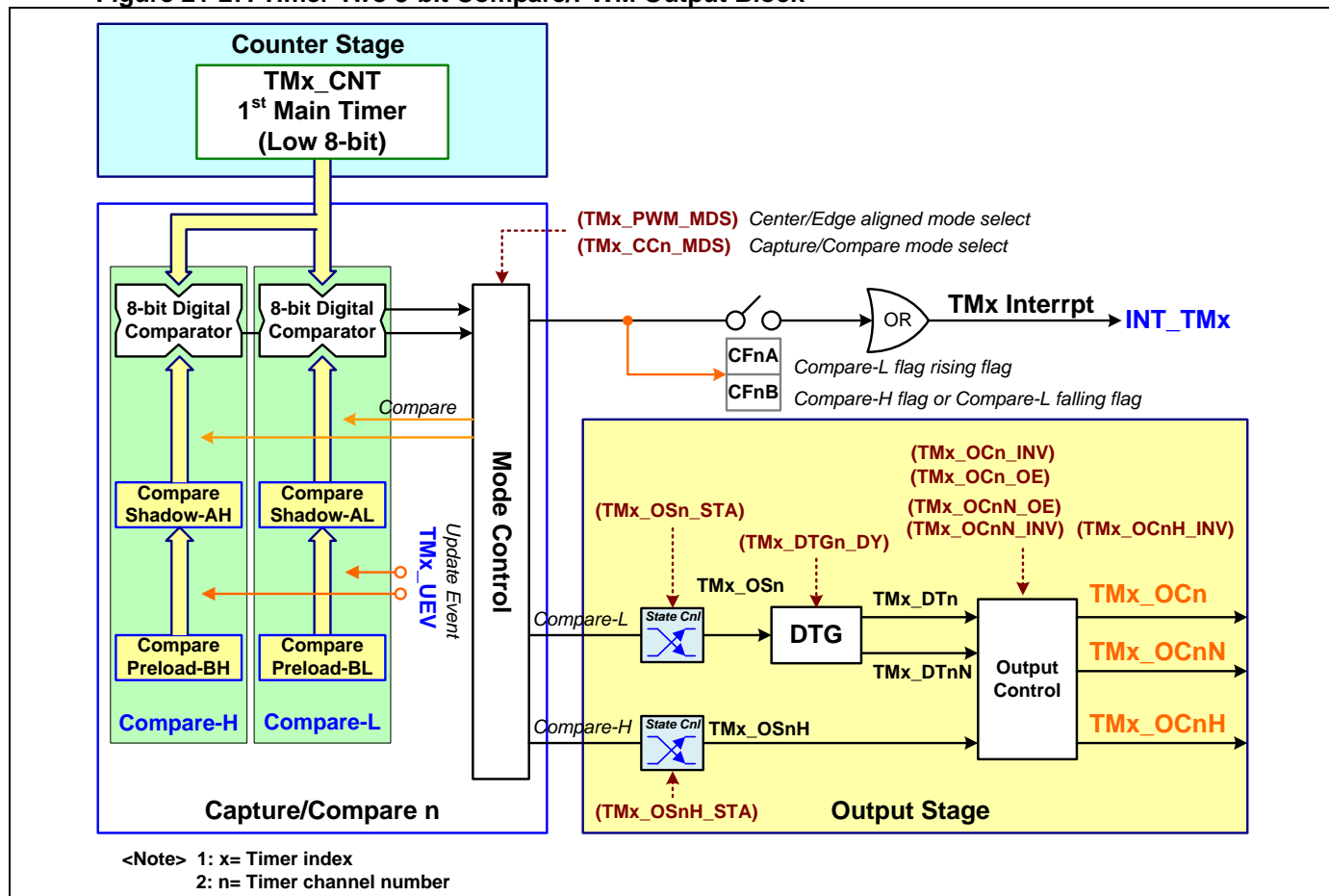


21.13.4. Two 8-bit Compare/PWM Mode

When compare-L is PWM mode and center-alignment mode, the **TMx_CFnA** bit is used as up counting PWM compare-L flag and the **TMx_CFnB** bit is used as down counting PWM compare-L flag. Other conditions, the **TMx_CFnA** bit is use as output compare-L event flag and the **TMx_CFnB** bit is used as compare-H event flag for the 8-bit comparator mode.

The following diagram is showing the Two 8-bit Compare/PWM Output Block.

Figure 21-27. Timer Two 8-bit Compare/PWM Output Block



21.13.5. Output Compare/PWM Control and Status

The following table is showing the related control registers and status registers for Timer Output Compare/PWM Mode.

Table 21-8. Timer Output Compare/PWM Mode

Mode	Setting	PWM mode	Setting	Compare Register				Status Register	
	CCn_MDS		PWM_MDS	CCnA High	CCnA Low	CCnB High	CCnB Low	CFnA	CFnB
16-bit Compare	2	x	x	compare 16-bit value	preload compare 16-bit value	preload compare 16-bit value	preload compare 16-bit value	compare	x
16-bit PWM	4	Edge Alignment	0					compare	x
		Center Alignment	1					up count	down count
two 8-bit Compare	3	x	x	compare-H 8-bit value	compare-L 8-bit value	preload compare-H 8-bit value	preload compare-L 8-bit value	compare-L	compare-H
two 8-bit PWM	5	Edge Alignment	0					compare-L	compare-H
		Center Alignment	1					compare-L up count	compare-L down count

CCnA, CCnB : n = channel number

21.13.6. Timer Output Compare and PWM Timing

❖ Timer Output Compare Timing with Up Counting

The following diagram is showing the Timer Output Compare timing with up counting by **TMx_ARR = 8**. The TOF flag is active at the time which **TMx_CNT** counter value changes from 8 to 0 and timer is overflow.

- **Compare Threshold TMx_CCnB = 2**

The **TMx_OSn** signal will toggle level at the time which **TMx_CNT** counter value changes from 1 to 2. Also the CFnA flag will be active at the time.

- **Compare Threshold TMx_CCnB = 8**

The **TMx_OSn** signal will toggle level at the time which **TMx_CNT** counter value changes from 7 to 8. Also the CFnA flag will be active at the time.

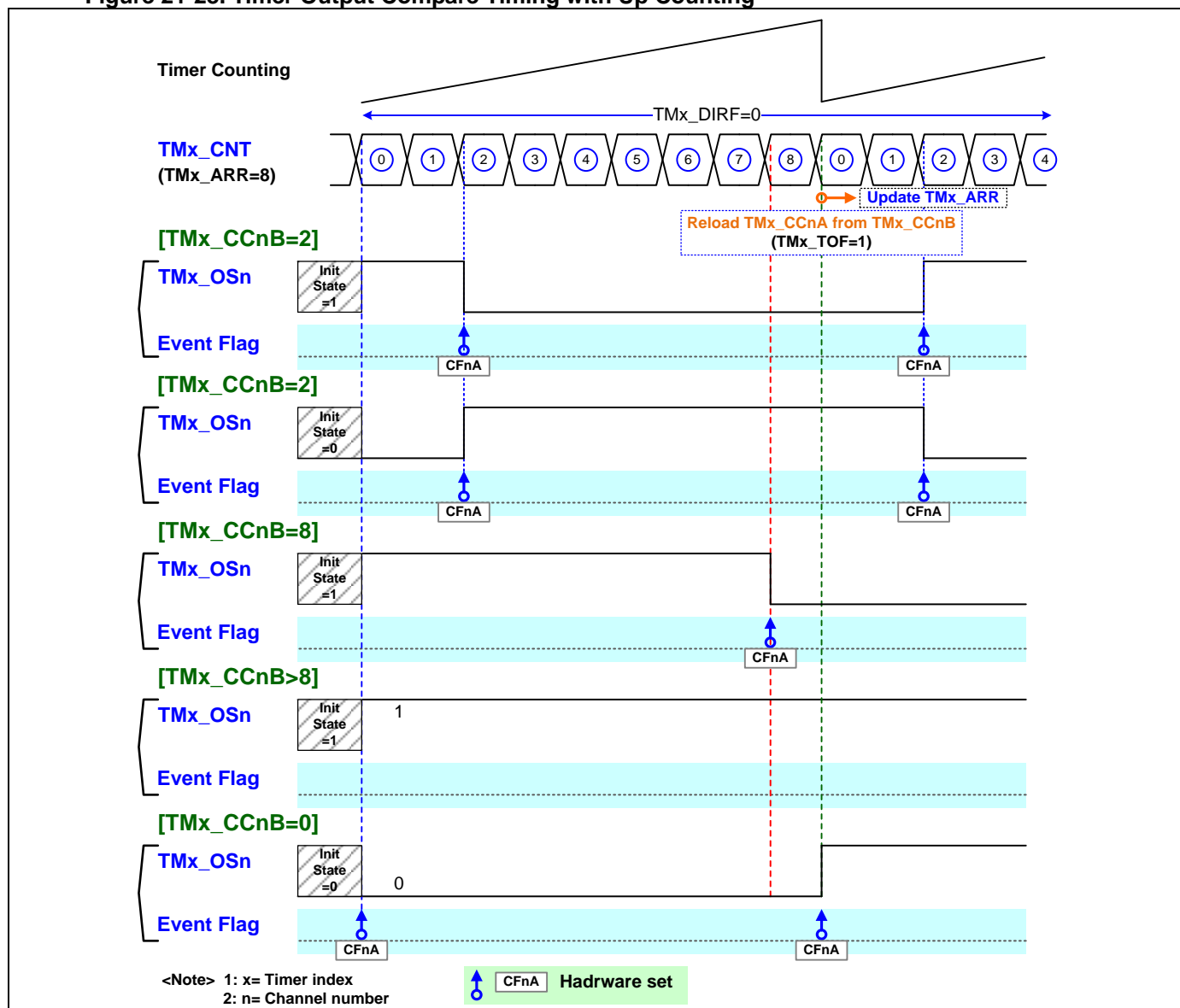
- **Compare Threshold TMx_CCnB > 8**

The **TMx_OSn** signal will always keep initial level and the CFnA flag will never be active.

- **Compare Threshold TMx_CCnB = 0**

The **TMx_OSn** signal will toggle level at the time which **TMx_CNT** counter value changes from 8 to 0. Also the CFnA flag will be active at the time.

Figure 21-28. Timer Output Compare Timing with Up Counting



❖ **Timer Edge Aligned PWM Timing with Up Counting**

The following diagrams are showing the Timer Edge Aligned PWM timing with up counting by **TMx_ARR = 8**.

- **Compare Threshold TMx_CCnB = 2**

The **TMx_OS_n** signal will toggle level at the time which **TMx_CNT** counter value changes from 1 to 2. Also the CFnA flag will be active at the time. Then the **TMx_OS_n** signal will toggle back to initial level after timer overflow and the TOF flag is active.

- **Compare Threshold TMx_CCnB = 8**

The **TMx_OS_n** signal will toggle level at the time which **TMx_CNT** counter value changes from 7 to 8. Also the CFnA flag will be active at the time. Then the **TMx_OS_n** signal will toggle back to initial level after timer overflow and the TOF flag is active.

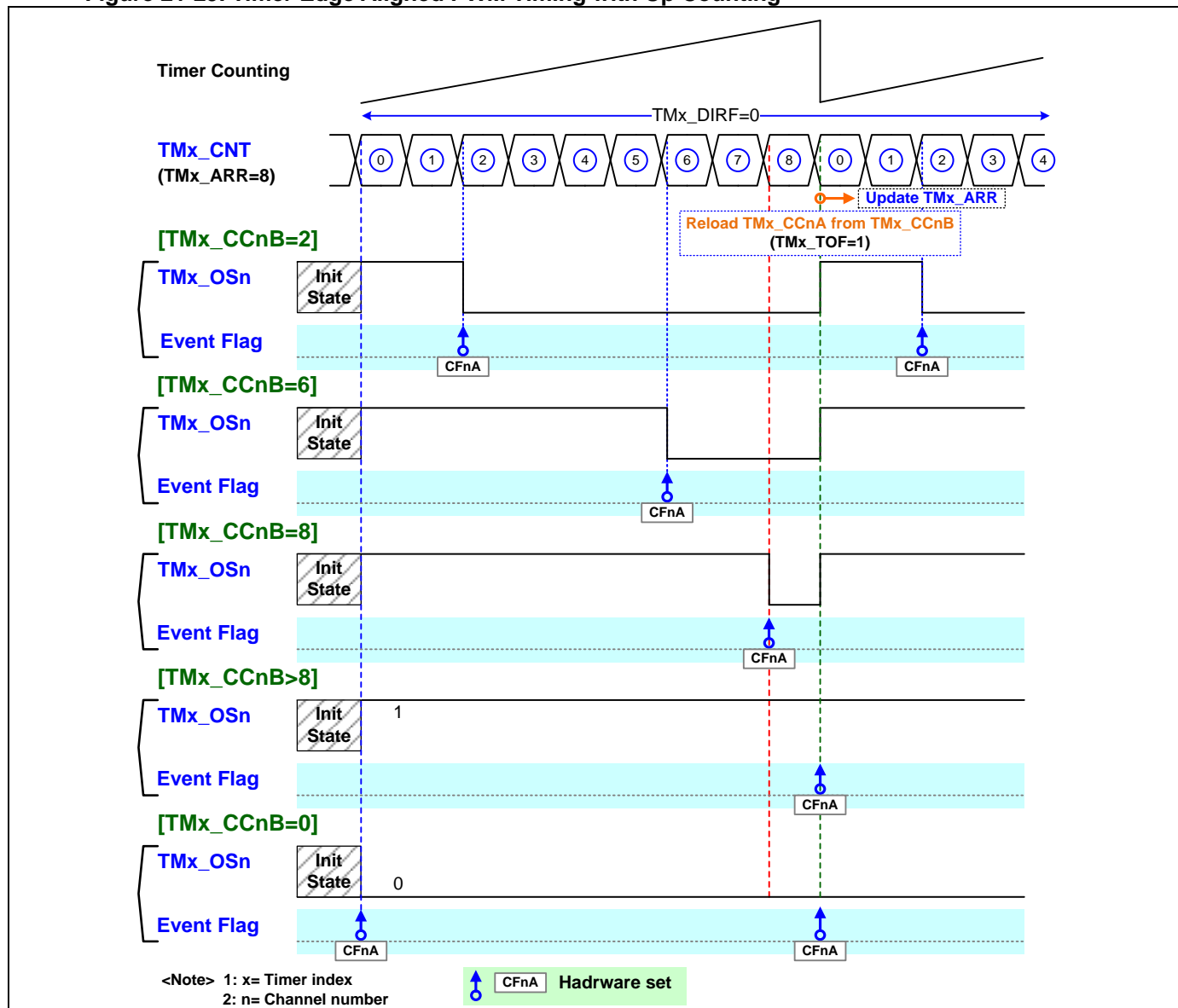
- **Compare Threshold TMx_CCnB > 8**

The **TMx_OS_n** signal will always keep high level whatever the setting of initial level. The CFnA and TOF flags will be active at the time which **TMx_CNT** counter value change from 8 to 0.

- **Compare Threshold TMx_CCnB = 0**

The **TMx_OS_n** signal will always keep low level whatever the setting of initial level. The CFnA and TOF flags will be active at the time which **TMx_CNT** counter value changes from 8 to 0. The CFnA flag is also active in the first time which **TMx_CNT** counter value equals 0 after timer counting is started.

Figure 21-29. Timer Edge Aligned PWM Timing with Up Counting



❖ **Timer Edge Aligned PWM Timing with Down Counting**

The following diagrams are showing the Timer Edge Aligned PWM timing with down counting by **TMx_ARR** = 8.

- **Compare Threshold TMx_CCnB = 2**

The **TMx_OSn** signal will toggle level at the time which **TMx_CNT** counter value changes from 2 to 1. Also the CFnA flag will be active at the time. Then the **TMx_OSn** signal will toggle back to initial level after timer underflow and the TUF flag is active.

- **Compare Threshold TMx_CCnB = 8**

The **TMx_OSn** signal will toggle level at the time which **TMx_CNT** counter value changes from 8 to 7. Also the CFnA flag will be active at the time. Then the **TMx_OSn** signal will toggle back to initial level after timer overflow and the TUF flag is active.

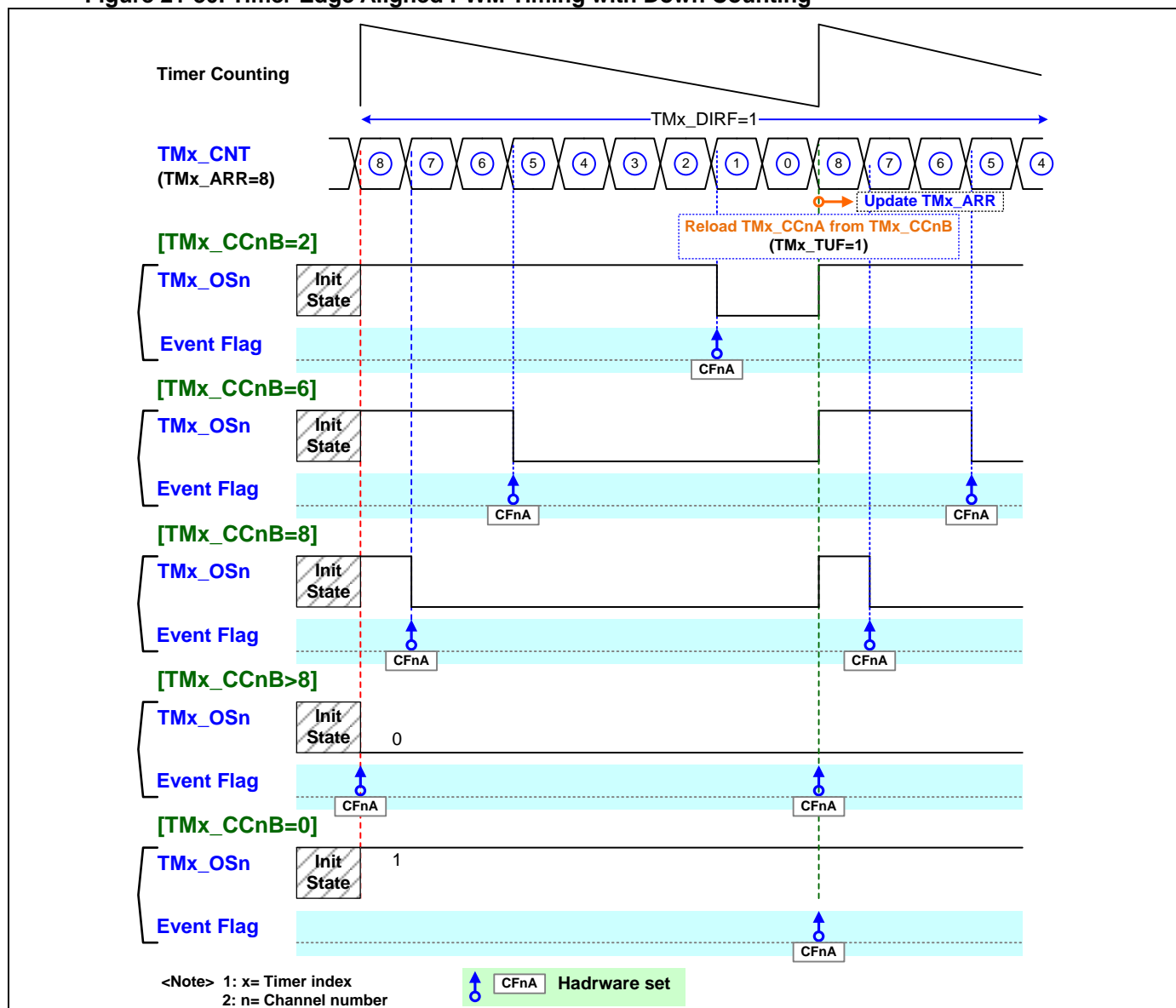
- **Compare Threshold TMx_CCnB > 8**

The **TMx_OSn** signal will always keep low level whatever the setting of initial level. The CFnA and TUF flags will be active at the time which **TMx_CNT** counter value changes from 0 to 8. The CFnA flag is also active in the first time which **TMx_CNT** counter value equals 8 after timer counting is started.

- **Compare Threshold TMx_CCnB = 0**

The **TMx_OSn** signal will always keep high level whatever the setting of initial level. The CFnA and TUF flags will be active at the time which **TMx_CNT** counter value changes from 0 to 8.

Figure 21-30. Timer Edge Aligned PWM Timing with Down Counting



❖ Timer Center Aligned PWM Timing

The following diagrams are showing the Timer Center Aligned PWM timing by **TMx_ARR = 6**. The TOF flag is active at the time which **TMx_CNT** counter value changes from 5 to 6 and timer is overflow. The TUF flag is active at the time which **TMx_CNT** counter value changes from 1 to 0 and timer is underflow.

- Compare Threshold **TMx_CCnB = 2**

The **TMx_OSn** signal will toggle level at the time which **TMx_CNT** counter value changes from 1 to 2. Also the CFnA flag will be active at the time. Then the **TMx_OSn** signal will toggle back to initial level at the time which **TMx_CNT** counter value changes from 3 to 2. Also the CFnB flag will be active at the time.

- Compare Threshold **TMx_CCnB = 6**

The **TMx_OSn** signal will always keep high level whatever the setting of initial level. The CFnB flag will be active at the time which **TMx_CNT** counter value changes from 5 to 6. The CFnA flag will never be active.

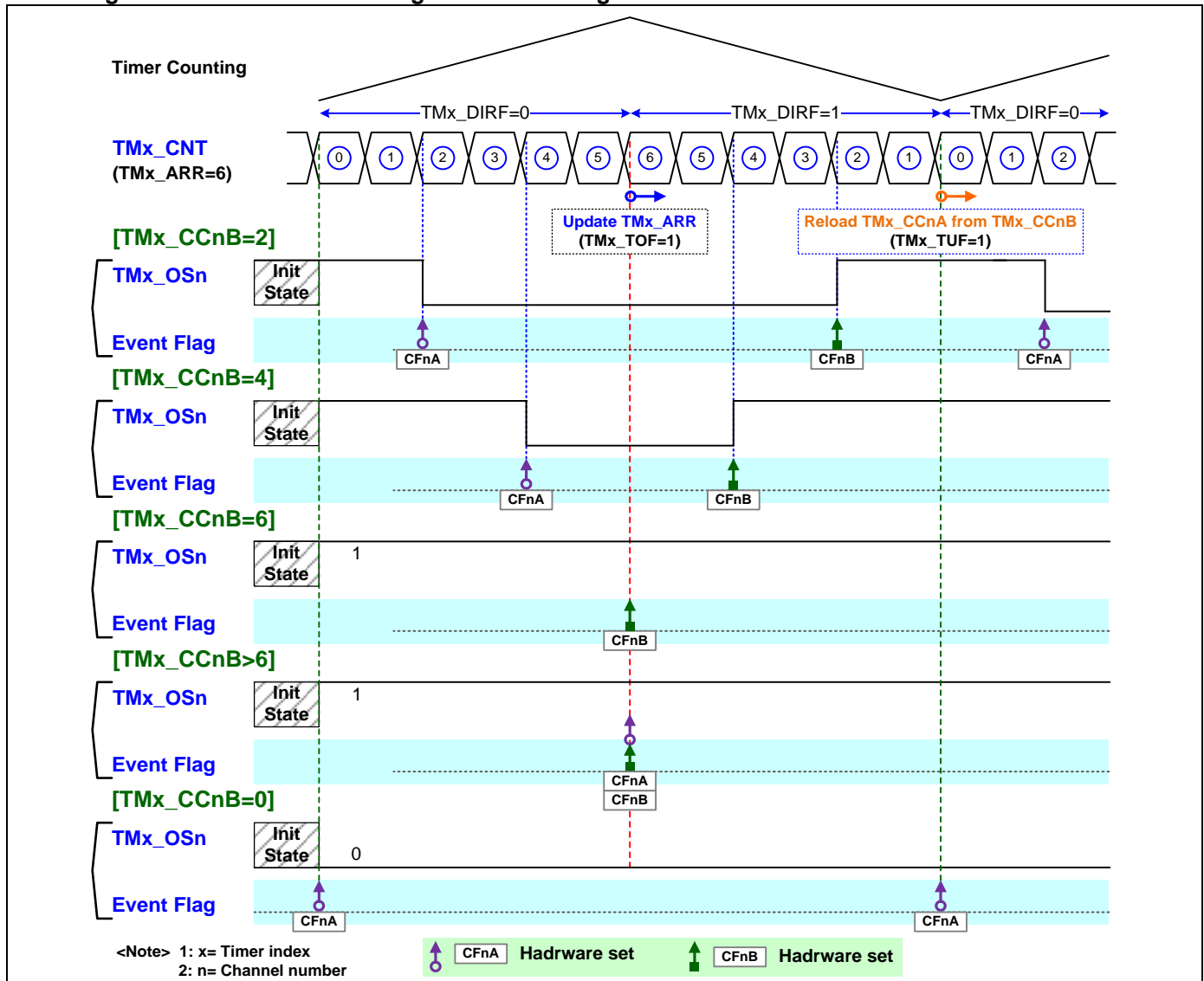
- Compare Threshold **TMx_CCnB > 6**

The **TMx_OSn** signal will always keep high level whatever the setting of initial level. The CFnA and CFnB flags will be active at the time which **TMx_CNT** counter value change from 5 to 6.

- Compare Threshold **TMx_CCnB = 0**

The **TMx_OSn** signal will always keep low level whatever the setting of initial level. The CFnA flag will be active at the time which **TMx_CNT** counter value changes from 1 to 0. The CFnA flag is also active in the first time which **TMx_CNT** counter value equals 0 after timer counting is started.

Figure 21-31. Timer Center Aligned PWM Timing



❖ **Timer Output Auto Stop Mode**

The timer module supports the timer output **auto stop mode** for compare and PWM output by setting **TMx_ASTOP_EN** register. When it enables, the timer output will stop after timer counting is overflow or underflow.

When the timer output **auto stop mode** is enabled, user can enable to auto clear the timer overflow **TMx_TOF** flag or underflow **TMx_TUF** flag for timer output **auto stop mode** by setting **TMx_ACLEAR_EN** register. For timer trigger start by external signal application, the timer compare and PWM output will enter **auto stop mode** for next timer trigger start.

❖ **Timer Output with Repetition Counter**

The timer modules of TM2x, TM3x are built in an extra Repetition Counter for stopping Main timer and 2nd timer. User can use the Repetition Counter and set the counting value to stop the timer output compare / PWM **TMx_OCn** signals. Refer the section of "[Repetition Counter](#)" for more information about Repetition Counter.

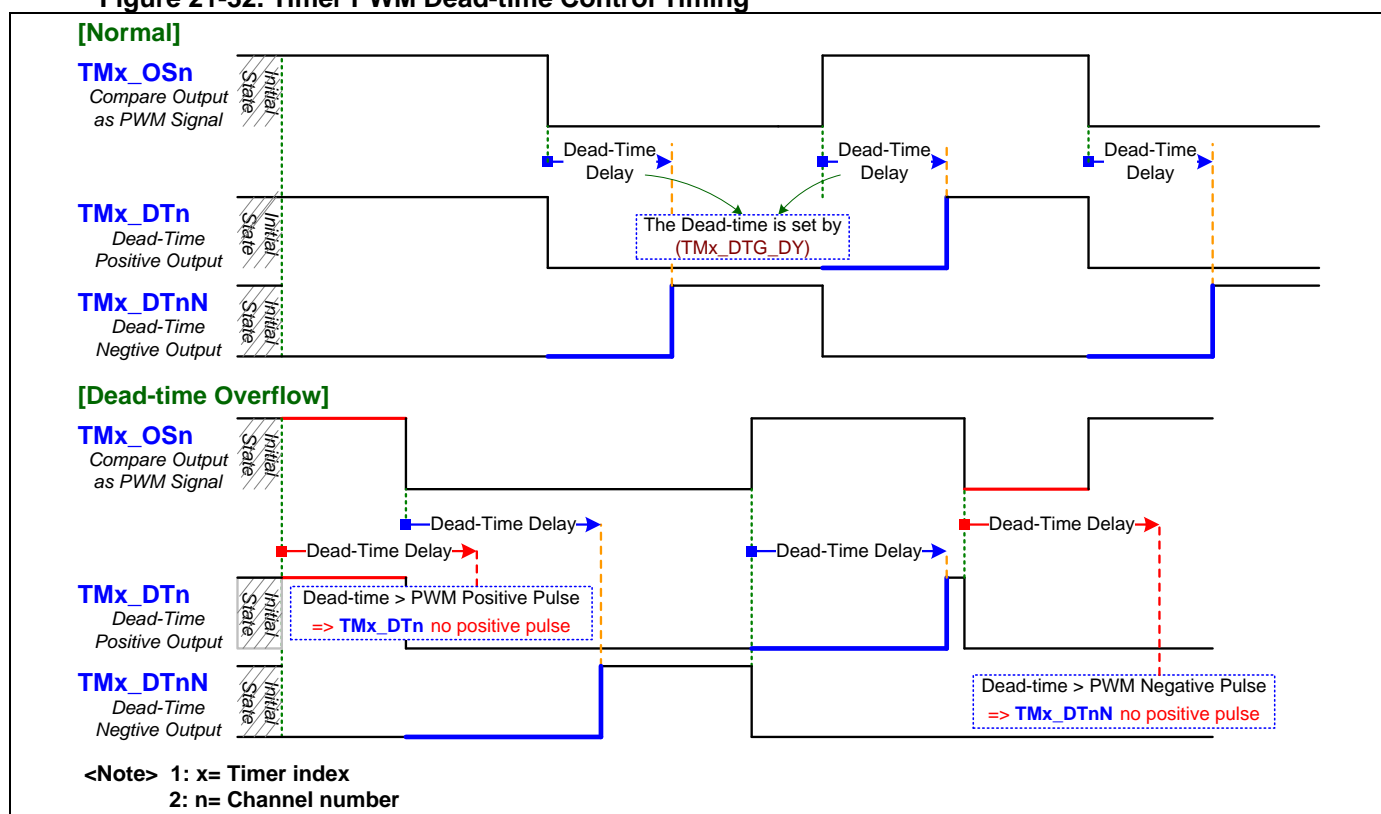
21.13.7. PWM Dead-Time Control

The Dead Time Generator (DTG) is only supported for TM36 module. User can use with the DTG function and configure the timer channel as 16bit PWM mode or Two 8bit PWMs mode by setting **TMx_CCn_MDS** register.

The register of **TMx_DTG_DY** is used to set the dead-time delay between **TMx_DTn** and **TMx_DTnN** signals in the unit of CK_DTG clock time base for all channels. **TMx_DTn** and **TMx_DTnN** are the internal signals those come from DTG output.

The following diagram is showing the Timer PWM Dead-time Control Timing.

Figure 21-32. Timer PWM Dead-time Control Timing



21.14. Timer Output Control Block

There are maximum four output channels which has three types output signal of **OCn**, **OCnH** and **OCnN** for each channel. The channel-0 and 1 have three independent output lines of **OCnm**. (n = output channel index, m = output line index {0, 1, 2})

The output control block can invert the output signal **OCn**, **OCnH** and **OCnN** by setting **TMx_OCn_INV**, **TMx_OCnH_INV** and **TMx_OCnN_INV** registers. User can set these **OCn** outputs to a fixed state by setting independent registers of **TMx_OCn_OEm** for output channel 0, 1 and independent registers of **TMx_OCn_OE** for output channel 2, 3. Also user can set these **OCnN** outputs to a fixed state by setting independent registers of **TMx_OCnN_OE** for output channel 0, 1, 2. (n = output channel index, m = output line index {0, 1, 2})

There are the independent state initial register bits of **TMx_STPn_STA** for **OCn** output channel-0/1/2/3 and **TMx_STPnN_STA** for **OCnN** output channel-0/1/2.

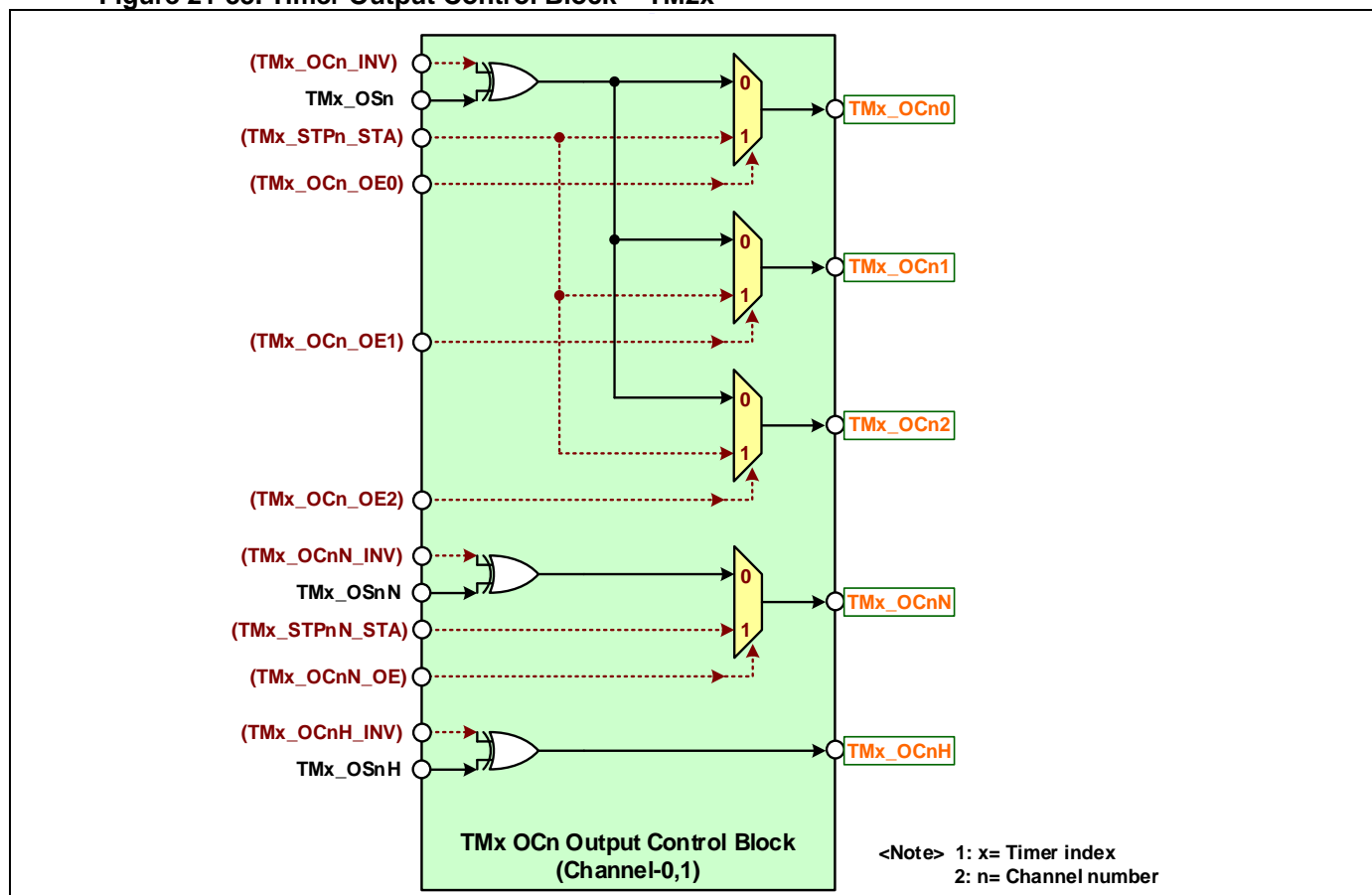
21.14.1. TM2x Timer Output Control Block

The Output Control blocks of Channel-0 and Channel-1 are the same. There are five output lines of **TMx_OCn0**, **TMx_OCn1**, **TMx_OCn2**, **TMx_OCnN** and **TMx_OCnH**.

The **TMx_OSn**, **TMx_OSnN** and **TMx_OSnH** signals are the timers compare output. The **TMx_OCn0**, **TMx_OCn1** and **TMx_OCn2** output are with the same common signal source **TMx_OSn**. The **TMx_OCnN** and **TMx_OCnH** output are with the independent signal source **TMx_OSnN** and **TMx_OSnH**.

The following diagram is showing the Timer Output Control block of TM2x.

Figure 21-33. Timer Output Control Block ~ TM2x



21.14.2. TM3x Timer Output Control Block

Also the Output Control blocks of Channel-0 and Channel-1 are the same. Channel-2 has three output lines of **TMx_OC2**, **TMx_OC2N** and **TMx_OC2H**. Channel-3 has two output lines of **TMx_OC3** and **TMx_OC3H**.

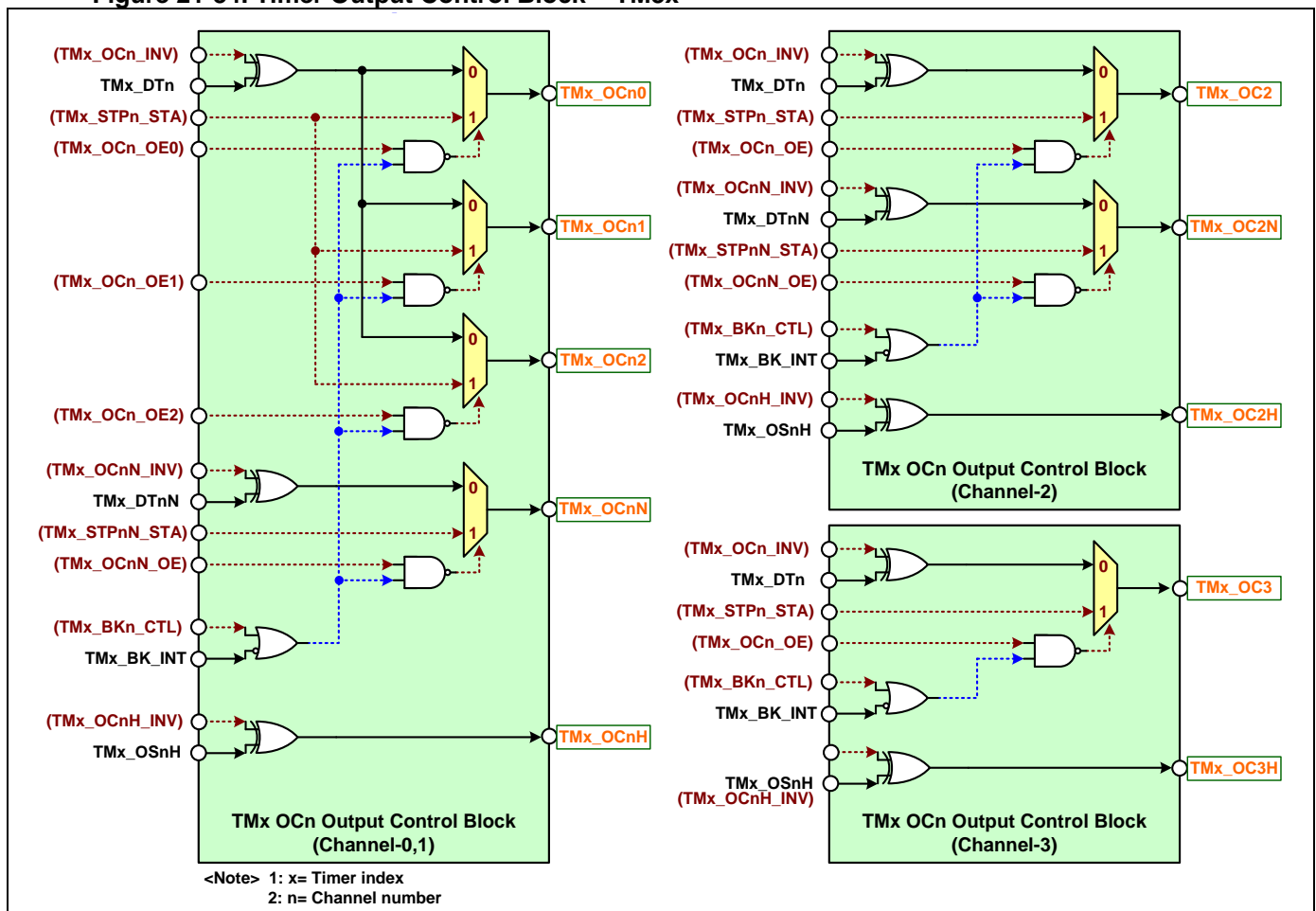
The **TMx_DTn** and **TMx_DTnN** signals are the DTG (Dead Time Generator) output. The **TMx_OSnH** signal is the timer compare output.

For Channel-0 and Channel-1, the **TMx_OCn0**, **TMx_OCn1** and **TMx_OCn2** output are with the same common signal source **TMx_DTn**. The **TMx_OCnN** and **TMx_OCnH** output are with the independent signal source **TMx_DTnN** and **TMx_OSnH**. For Channel-2, the **TMx_OC2**, **TMx_OC2N** and **TMx_OC2H** output are with the independent signal source **TMx_DTn**, **TMx_DTnN** and **TMx_OSnH**. For Channel-3, the **TMx_OC3** and **TMx_OC3H** output are with the independent signal source **TMx_DTn** and **TMx_OSnH**.

TMx_BK_INT is the timer internal break signal. User can set the **TMx_BKn_CTL** register to configure the action of switching to stop state or hold output state when break event is happened. Refer the section of "[Break Control Block](#)" for more information about timer break control.

The following diagram is showing the Timer Output Control block of TM3x.

Figure 21-34. Timer Output Control Block ~ TM3x



21.14.3. Timer Output Enable Preload Control

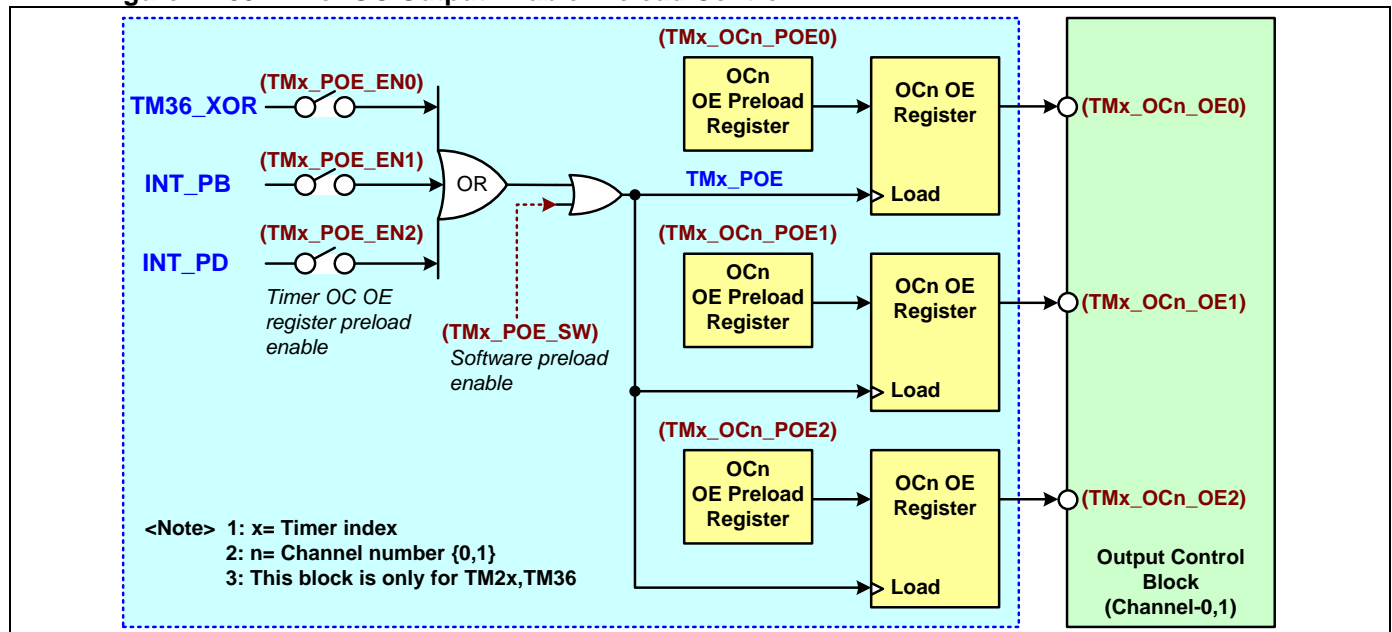
The module provides the output preload registers of **TMx_OCn_POEm** which can use to store the planned output control state and preload to output enable register of **TMx_OCn_OEm** when one of preload-event is happened. The **TMx_OCn_OEm** registers are used to enable the output line of **TMx_OCn0**, **TMx_OCn1** and **TMx_OCn2** output for output channel 0, 1.

There are the independent state initial register bits of **TMx_STPn_STA** for **OCn** output channel-0/1/2/3 and **TMx_STPnN_STA** for **OCnN** output channel-0/1/2.

The preload-events are able to coming from the source of TM36 XOR output, internal EXIC global interrupt events of GPIO B/D port or software control register setting of **TMx_POE_SW**. User can enable the preload-event source by setting the independent **TMx_POE_ENz** register. (z = preload-event source input line index {0, 1, 2})

The following diagram is showing the Timer OC Output Enable Preload Control.

Figure 21-35. Timer OC Output Enable Preload Control



21.15. Timer Break Control Block

The break control block is only supported for TM36 module. The module can input the break events from internal events, external events or software forced event by register setting to break the timer output signals.

21.15.1. Break Enable and Event Sources

The **TMx_BK_EN** register is used to enable the break control for all of the channel-0, 1, 2 and the independent **TMx_BK_EN3** register is used for the channel-3 only.

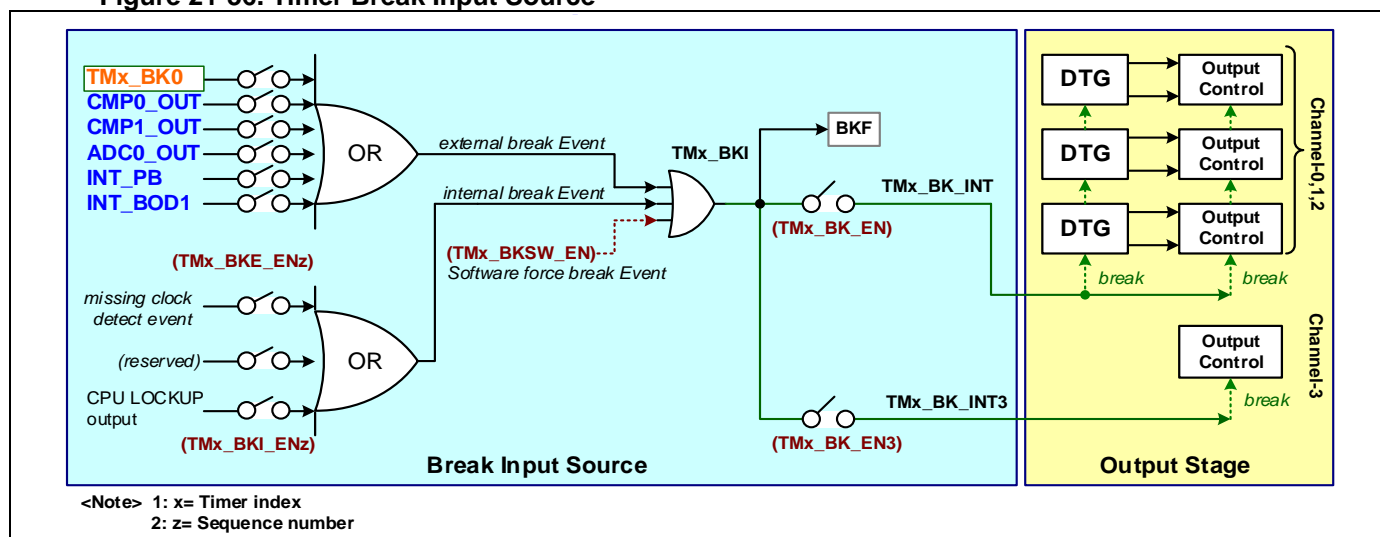
The internal break event sources are including of missing clock detect event and CPU LOCKUP output event. The external break event sources are including of external pin input TMx_BK0, analog comparator outputs, ADC voltage window detect output, internal EXIC global interrupt event of GPIO B port and BOD1 interrupt event. Also the module provides a software forced break event by setting **TMx_BKSW_EN** register.

User can enable the internal break event sources by setting **TMx_BKI_ENz** register and the external break event sources by setting **TMx_BKE_ENz** register as the showing in the following diagram. (z = source index number)

When break function is enabled and one of break event is happened, the break control signal **TMx_BKI** will send to DTG and output control block to stop the output signal.

The following diagram is showing the Timer Break Input Source Block.

Figure 21-36. Timer Break Input Source



21.15.2. Break Control Mode

User can select the break function mode of Cycle-to-Cycle mode or Latch mode for all channels by setting **TMx_BK_MDS** register.

Also user can set the **TMx_BKn_CTL** register to configure the action of switching to stop state or hold output state when break event is happened. Refer the descriptions of “[Timer Output Control Block](#)” for more detail information. There are the independent state initial register bits of **TMx_STPn_STA** for **OCn** output channel-0/1/2/3 and **TMx_STPnN_STA** for **OCnN** output channel-0/1/2.

The following table is showing the related control registers and status registers for Timer Output Break or Stop Control. The **TMx_OCn_OEm** registers are used to enable the output lines.

Table 21-9. Timer Output Break or Stop Control

TMx Register		Break Signal TMx_BK_INT	TMx_OCn/TMx_OCnN Output
OCnN_OEm	BKn_CTL		
Disable	X	X	STPnN_STA setting
Enable	X	inactive	Timer Compare or PWM output state
Enable	Hold	active	Timer Compare or PWM output last state
Enable	Stop	active	STPnN_STA setting

x = Timer module index , n= Channel number , m= Line number

OCnN : N = indicate complement channel

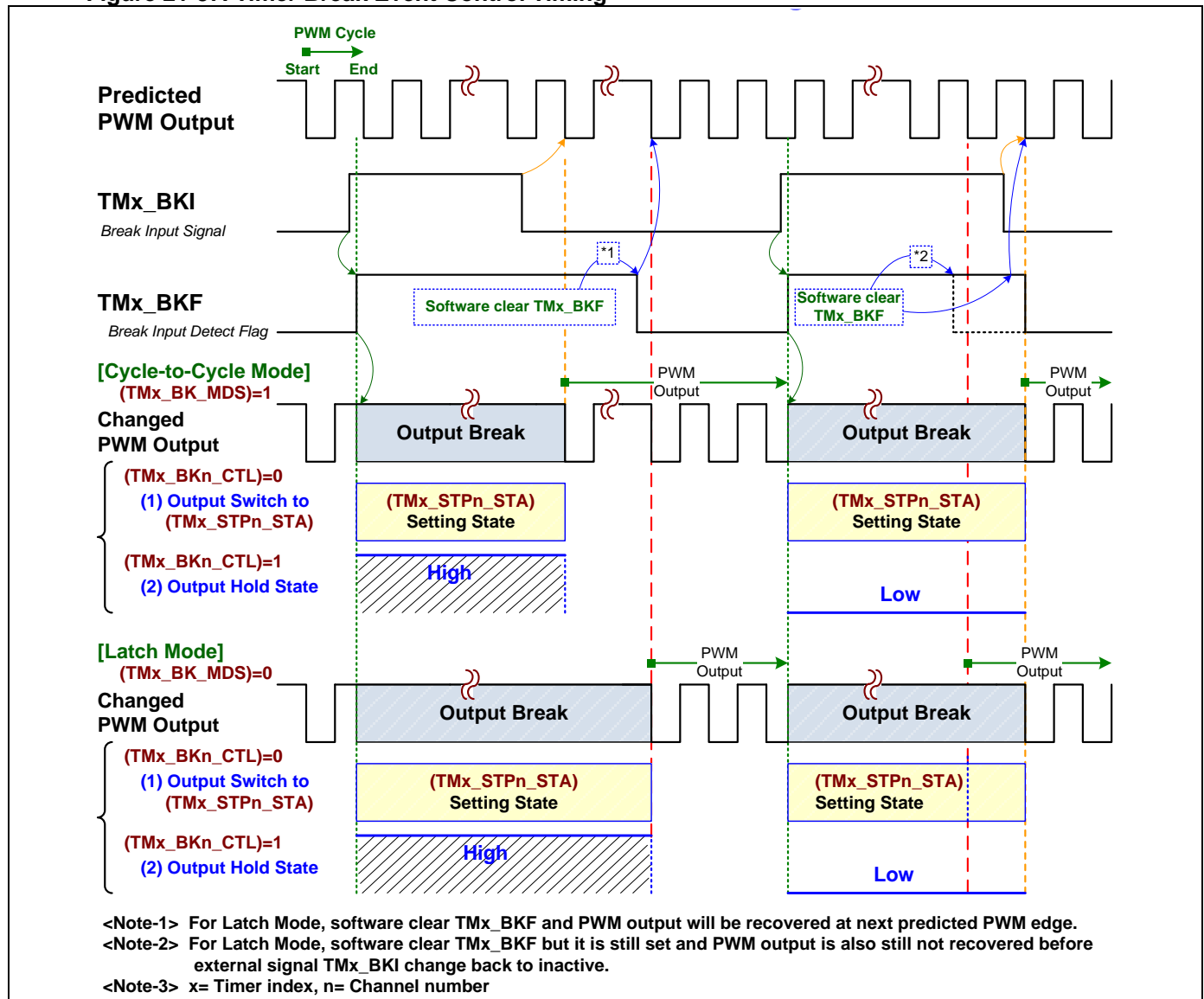
● Break Event Control Timing

When selects Cycle-to-Cycle mode, the compare or PWM output is stopped during the break control signal **TMx_BKI** active period. The output will be recovered and started at the beginning of next integral cycle of compare or PWM output after the **TMx_BKI** signal is inactive. Also the BKF flag is active when detects the active break control signal and is cleared by serviced firmware.

When selects Latch mode, the compare or PWM output is stopped and the BKF flag is active after the chip detects the active break control signal. Until the BKF flag is cleared by serviced firmware, it will end the output break. Also the output will be recovered and started at the beginning of next integral cycle of compare or PWM output.

The following diagram is showing the Timer Break Event Control Timing.

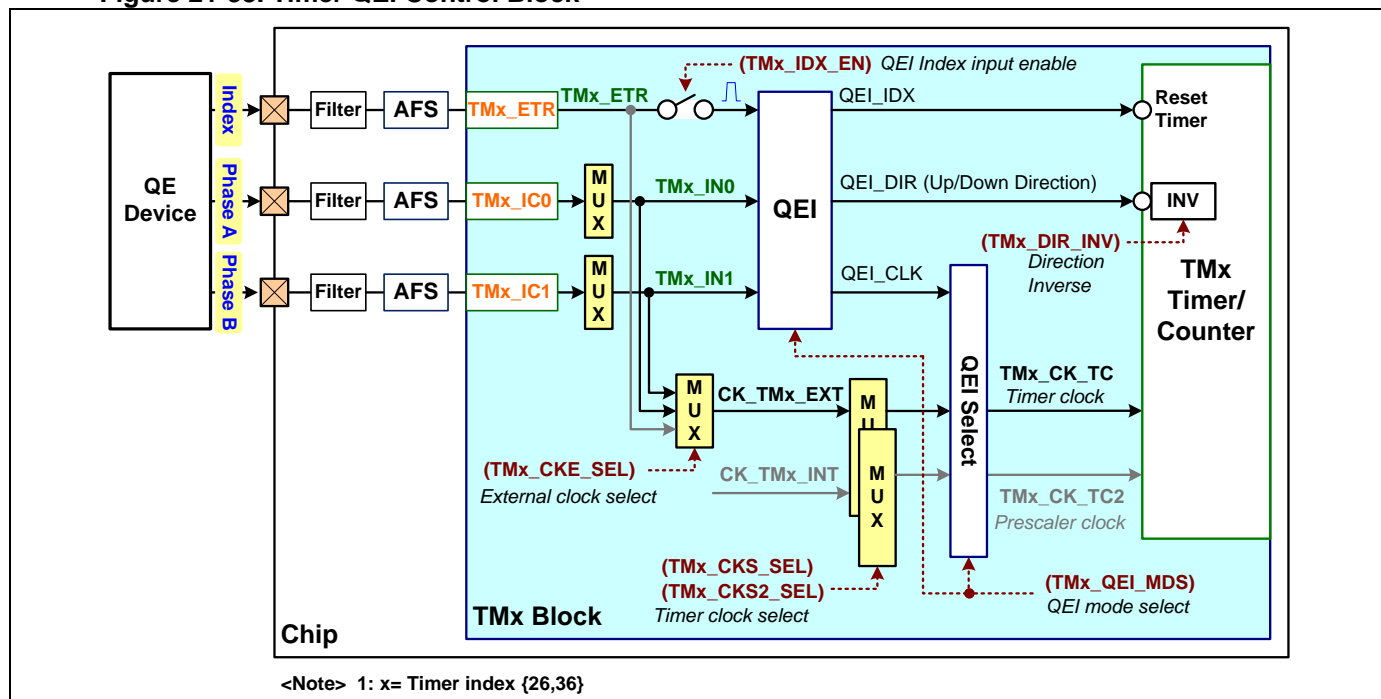
Figure 21-37. Timer Break Event Control Timing



21.16. QEI Control Block

The QEI (Quadrature Encoder Interface) control block is only supported for TM26 and TM36 modules. The following diagram is showing the Timer QEI Control Block.

Figure 21-38. Timer QEI Control Block



21.16.1. QEI Control Mode

The QEI block provides five control modes and user can enable QEI control and configure the QEI control mode by setting the **TMx_QEI_MDS** register. The QEI function is disabled if the **TMx_QEI_MDS** register is set "0". The following table is showing the external input signals' type for Timer QEI External Input Up/Down Control Mode.

Table 21-10. Timer QEI External Input Up/Down Control Mode

QEI Mode	TMx_IN0	TMx_IN1	Timer Counting
TMx_IN0 positive (TMx_QEI_MDS)=1	High	X	Up count continued
	Low	X	Down count continued
TMx_IN0 negative (TMx_QEI_MDS)=2	High	X	Down count continued
	Low	X	Up count continued
TMx_IN0 trigger (TMx_QEI_MDS)=3	Rising-edge	High	Down count once
	Rising-edge	Low	Up count once
	Falling-edge	High	Up count once
	Falling-edge	Low	Down count once
TMx_IN1 trigger (TMx_QEI_MDS)=4	High	Rising-edge	Up count once
	Low	Rising-edge	Down count once
	High	Falling-edge	Down count once
	Low	Falling-edge	Up count once
Both edge (TMx_QEI_MDS)=5	Rising-edge	High	Down count once
	Rising-edge	Low	Up count once
	Falling-edge	High	Up count once
	Falling-edge	Low	Down count once
	High	Rising-edge	Up count once
	Low	Rising-edge	Down count once
	High	Falling-edge	Down count once
	Low	Falling-edge	Up count once

TMx : x = Timer module index

21.16.2. QEI Input Signals

The QEI block can input two external signals of **TMx_IC0** and **TMx_IC1** to control the Main Timer up/down counting and one optional external signals of **TMx_ETR** to reset the timer. User needs to select the timer channel-0, 1 input signal source from **TMx_IC0** and **TMx_IC1** by setting **TMx_IC0_MUX** and **TMx_IC1_MUX** registers.

The **TMx_IDX_EN** register bit is used to enable to input the QEI **Index** signal. This signal will be detected by QEI block and output the **QEI_IDX** signal to reset the Main Timer. When the QEI control block is enabled and detects the external **Index** signal active high pulse, the timer will reset during up counting or reload the auto-reload value during down counting until the external **Index** signal is inactive.

The QEI block supports two types of input signal. The one is able to input one clock signal and one up/down direction signal. User can set QEI control mode 0, 1 for this input signal type.

The two is able to input two phase A, B signals. User can set QEI control mode 3, 4, 5 for this input signal type. The QEI block needs to convert to one clock signal (**QEI_CLK**) and one up/down direction signal (**QEI_DIR**). Others, there is one direction control bit and uses to invert the timer counting direction for actual input signals by setting **TMx_DIR_INV** register.

21.16.3. QEI Control Mode 1, 2

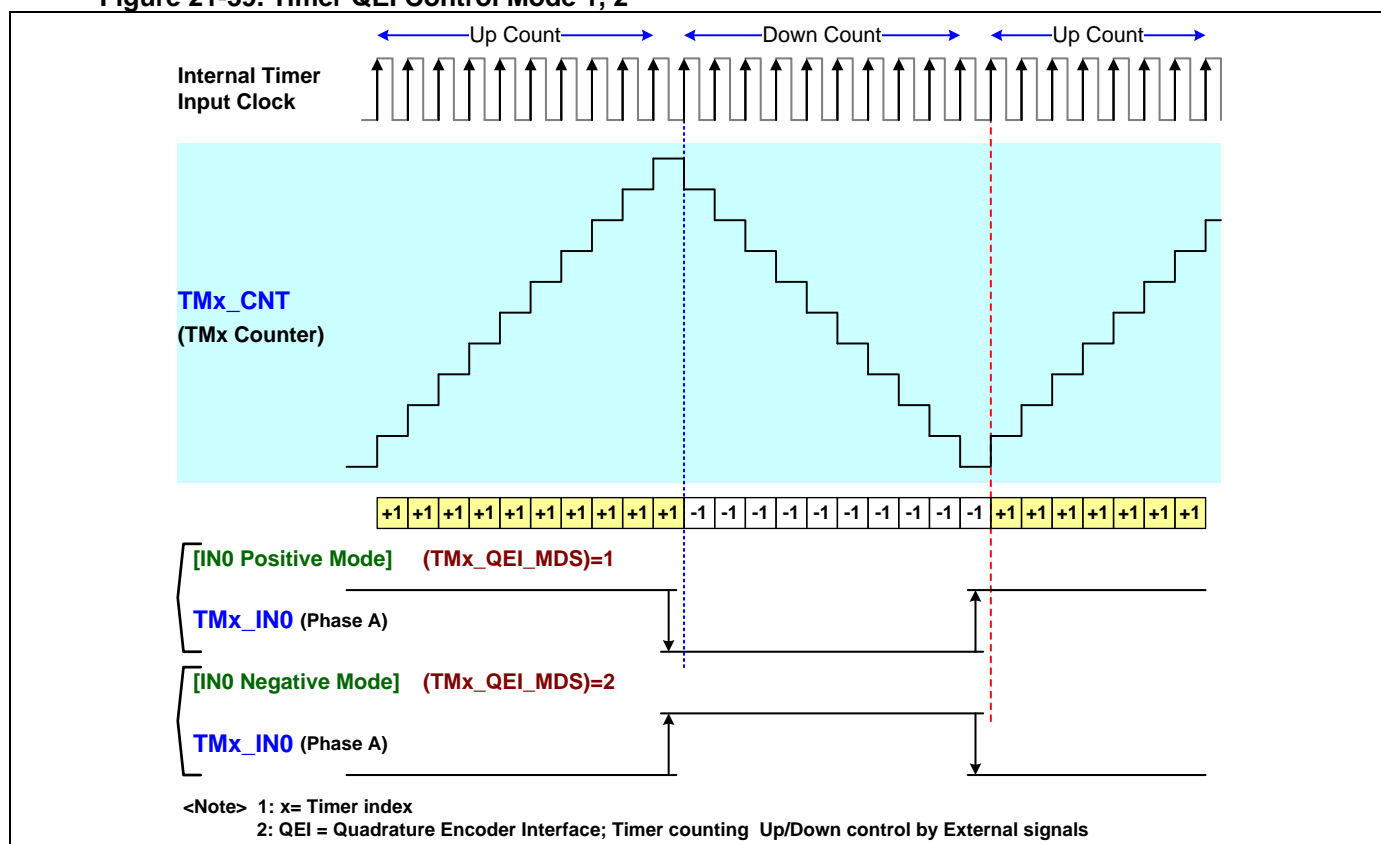
The QEI control block is able to input one clock signal from **TMx_IC1** and one up/down direction signal from **TMx_IC0**. User needs set QEI control mode 0, 1 by setting the **TMx_QEI_MDS** register.

If the timer clock is coming from the external **TMx_IC1**, user needs to select the timer clock source from **TMx_IN1** signal by setting **TMx_CKE_SEL** register and **CK_TMx_EXT** signal by setting **TMx_CKS_SEL** register. Of course the timer clock is able to come from internal clock source by user application. The timer up or down counting is directly control by the level of input signal **TMx_IN0**.

The difference between the QEI control mode-0 and mode-1 is only the timer counting direction polarity by input **TMx_IN0** signal.

The following diagram is showing the timing of QEI Control Mode 1, 2.

Figure 21-39. Timer QEI Control Mode 1, 2



21.16.4. QEI Control Mode 3, 4, 5

The QEI control block is able to input two phase A, B signals from **TMx_IC0** and **TMx_IC1**. User needs set QEI control mode 3, 4, 5 by setting the **TMx_QEI_MDS** register.

The difference between the QEI control mode-3 and mode-4 is only the swapping the **Phase A** and **Phase B** function definitions for input signals of **TMx_IN0** and **TMx_IN1**. The difference between the QEI control mode-3,4 and mode-5 is that the timer counting is only at the edge change of **Phase A** or **Phase B** for QEI control mode-3,4 but the timer counting is at the edge change of both **Phase A** and **Phase B** for QEI control mode-5.

The QEI block receives these two **Phase A, B** signals from **TMx_IN0** and **TMx_IN1** signal. It converts these two signals to one clock signal (**QEI_CLK**) as timer clock and one up/down direction signal (**QEI_DIR**) to control timer up or down counting. In other words, the timer up or down counting is control by the level and the edge of both input signals of **TMx_IN0** and **TMx_IN1**.

- **QEI Encoder States and Transition**

The QEI state is encoding by these two input signals of **Phase A** and **Phase B** for QEI block control. The following table is showing the QEI Encoder States.

Table 21-11. Timer QEI Encoder States

Phase A	Phase B	State
1	0	1
1	1	2
0	1	3
0	0	4

The following table is showing the QEI Encoder State Transaction for timer up/down counting direction control.

Table 21-12. Timer QEI Encoder State Transition

from State	to State	Direction
1	2	Positive
2	3	
3	4	
4	1	
4	3	Negative
3	2	
2	1	
1	4	

- **QEI Index Control**

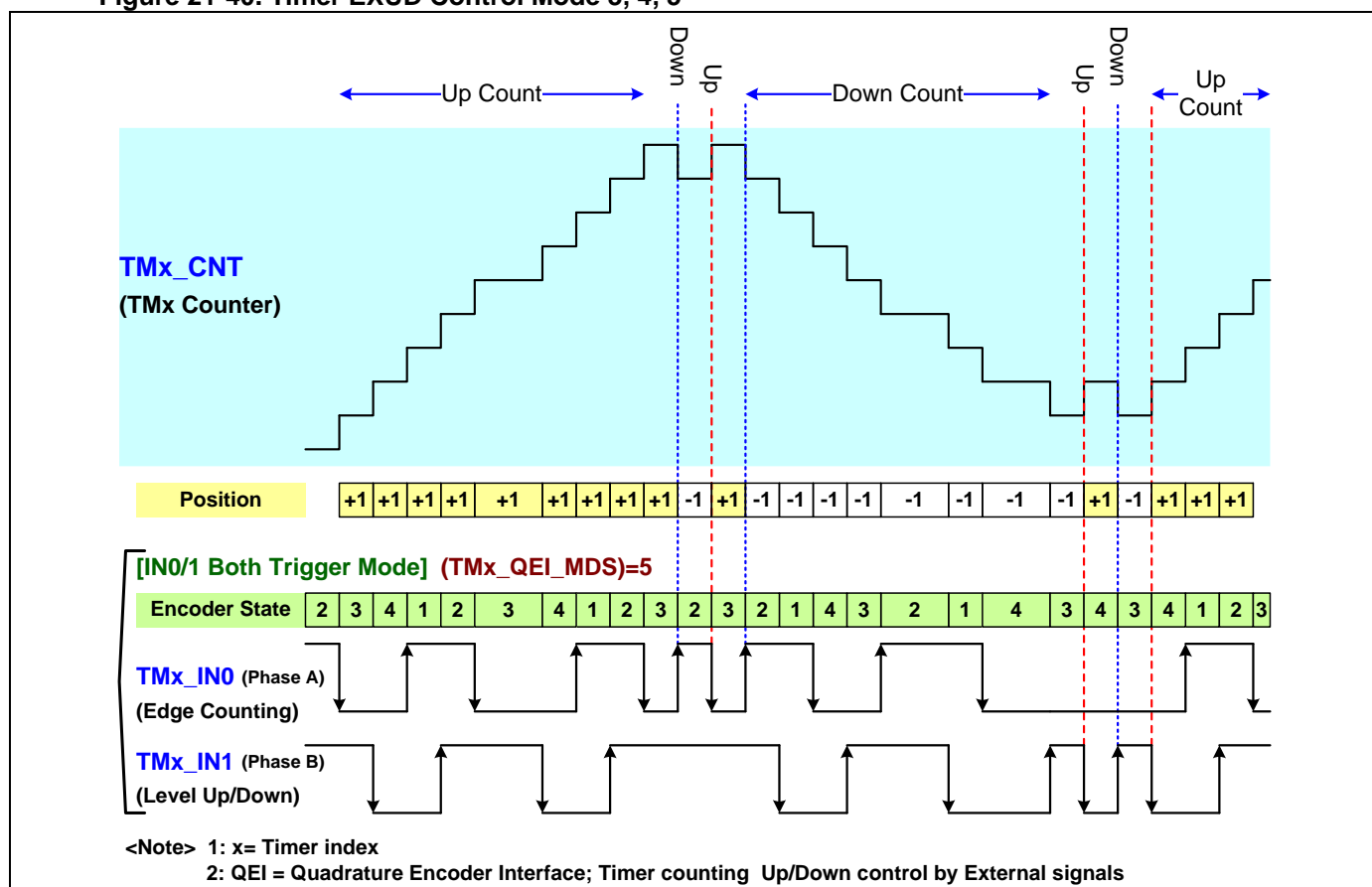
The **TMx_IDX_EN** register is used to enable QEI external **Index** signal input from **TMx_ETR**. When the external QE device output signals are only one clock signal and one up/down direction signal, user needs to disable the **Index** input function by setting **TMx_IDX_EN** register. So the QEI block can directly use the two signals to control timer up/down counting.

When the external QE device output are two of the Phase signals with one **Index** signal, user can enable the **TMx_IDX_EN** register and the QEI block will be able to detect the **Index** signal to reset the timer.

The **TMx_IDX_MDS** register is used to select the transition state of QEI encoder state to reset timer by QEI external **Phase** signals input.

The following diagram is showing the timing of QEI Control Mode 3, 4, 5.

Figure 21-40. Timer EXUD Control Mode 3, 4, 5



21.17. Timer DMA Operation

There is one channel of input capture and three channels of output compare with DMA capability for TM36 module only. The timer input capture data can store to memory or transfer to other peripherals by through DMA controller. Also the timer output compare value can reload from memory or other peripherals by through DMA controller.

21.17.1. DMA Module Configure

When the chip supports a DMA (direct memory access) controller, user can configure the DMA setting of transferred source/destination devices, channel request arbitration and others in the DMA module before a DMA data transaction. The DMA source and the destination can be memory or peripheral.

Refer the DMA chapter for more detail information about the DMA module configuration.

21.17.2. Timer DMA Control

After DMA configuration is finished, user needs to set the Timer module DMA enable bits of **TMx_DMA_CCnE** and **TMx_DMA_ICnE**. (n = Timer channel index)

[Notify]: The **TMx_DMA_ICnE** are not supported for MG32F02A128/U128/A064/U064.

Finally, the related channel request start bit of **DMA_CHn_REQ** is necessary to be set to start the DMA transaction (n = DMA channel index). Then the transferred source/destination devices will assert the RX/TX request signal to DMA controller and the DMA controller will assert the acknowledge signal to the request source/destination devices. At the time, the data transferred connection is built for DMA transaction.

● Timer IC to DMA

There are two registers of **TMx_DMA_IC2E** and **TMx_DMA_IC3E** those are used to enable DMA data transfer from timer input capture to DMA destination for timer input channel-2 and channel-3. For MG32F02A128/A064/U128/U064, these chips are only support timer input capture from channel-3 and enables by setting **TMx_DMA_CC3E** register bit.

[Notify]: The DMA timer input capture function does not support for Full-Counter mode.

[Notify]: The **TMx_DMA_CC3E** is only used for MG32F02A128/U128/A064/U064.

During the DMA transaction cycle, the IC/OC/PWM flags of **TMx_CF3A** and **TMx_CF3B** are masked by hardware.

- **Timer OC from DMA**

The **TMx_DMA_CC0E**, **TMx_DMA_CC1E** and **TMx_DMA_CC2E** register bits are used to enable DMA data transfer from DMA source to timer output compare reload registers for timer output channel-0/1/2 independently.

During the DMA transaction cycle, the IC/OC/PWM flags of **TMx_CFnA** and **TMx_CFnB** are masked by hardware. (n = Timer channel index 0, 1, 2)

For the timer output compare DMA function, user can select timer output DMA request mode by setting **TMx_DMA_OMDS** register. When selects ITR, the DMA request is asserted at the time of TOF active or ITR input signal active. When selects TOF, the DMA request is asserted at the time of TOF active only. The asserted DMA request makes to update **TMx_CCnB** registers (output compare reload registers) for the channels those DMA function (**TMx_DMA_CCnE**) are enabled. (n = Timer channel index 0, 1, 2)

[Notify]: The TMx will copy the **TMx_CCnB** register content to the compared register of **TMx_CCnA** when every time the DMA update **TMx_CCnB** register. So the **TMx_CCnB** register must be filled the first compared data before do DMA data transaction.

21.17.3. Timer Interrupt Flag Control during DMA

During DMA operation cycle, the module's interrupt flags will control and act three types as following table. One is masked during the DMA process. Another is to disable the DMA function after the flag has asserted. At the time, hardware will disable the **TMx_DMA_CCnE** bits in this condition. Others are normally as same as the action of not processing in DMA operation.

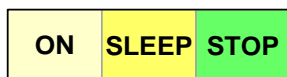
Table 21-13. Timer Interrupt Flag Control for DMA Function

Action	Mask the Flag during DMA Process	DMA Disable after the Flag asserted (*1)	Normal Control	
Peripheral	(Data flow flags)	(Error/Detect flags)	(Other flags)	
TM36	TM36_CF0A/0B TM36_CF1A/1B TM36_CF2A/2B TM36_CF3A/3B	TM36_TOF TM36_TUF	TM36_EXF TM36_TOF2 TM36_TUF2 TM36_DIRCF	TM36_IDXF TM36_QPEF TM36_BKF

Note-1 : When the flag is asserted, it will not force peripheral DMA disabled if the related interrupt enable bit is not enabled.

22. IWDT (Independent WatchDog Timer)

22.1. Introduction



The module can be running in all power operation modes.

The chip has one independent watch-dog timer (IWDT) to use as a recovery method in situations where the CPU may be subjected to software upset. It will trigger system reset when the counter reaches a given timeout value.

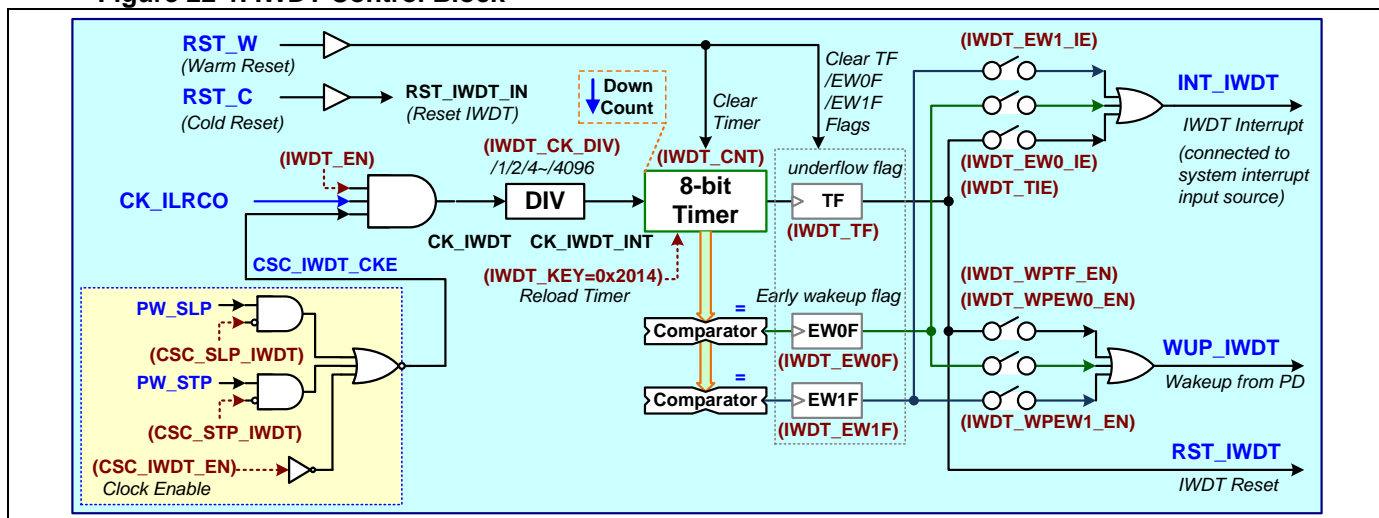
22.2. Features

- 8-bit down counter with 12-bit prescaler and clocked by its own CK_ILRCO
- Operating capability in SLEEP and STOP modes
- Selectable reset or interrupt when the counter underflow
- Support two early wakeup comparators with interrupt
- Support register key-protected and reset-locked functions

22.3. Control Block

The IWDT consists of one 12-bit clock divider, one 8-bit timer and two early wakeup digital comparators. The following diagram is showing the IWDT Control block.

Figure 22-1. IWDT Control Block



22.4. Enabling and Clock

22.4.1. IWDT Global Enable

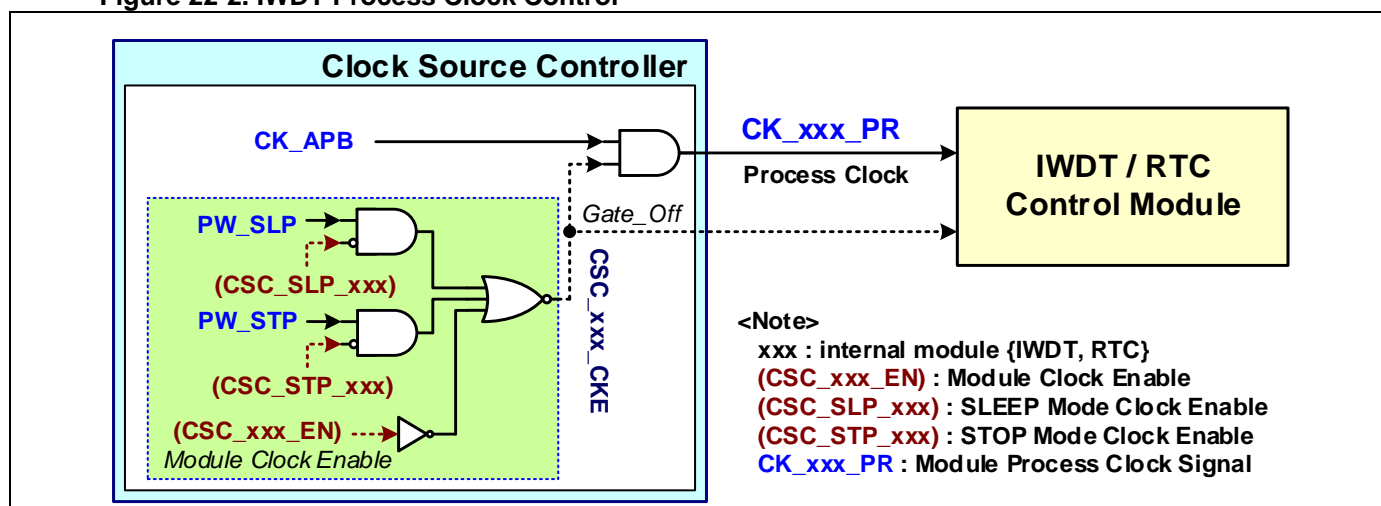
There is a global enable bit of **IWDT_EN** for all functions of this module. When this bit is disabled, all the IWDT functions are not working.

22.4.2. IWDT Clock Control

● Module Process Clock

The module process clock of **CK_IWDT_PR** is using for the interface control logic between APB bus and the module. It is coming from CSC (Clock Source Controller) module. It can be enabled in **CSC_IWDT_EN** register. User can plan the module clock is running or not beforehand for chip entering **SLEEP** or **STOP** modes by setting **CSC_SLP_IWDT** or **CSC_STP_IWDT** registers. Refer the System Clock chapter for more information.

Figure 22-2. IWDT Process Clock Control



● Module Internal Clock

The IWDT timer clock source is coming from the internal ILRCO clock of **CK_ILRCO**. The internal timer clock is gated off when the module global enable bit of **IWDT_EN** is disabled.

The internal timer clock is also gating control by the **CSC_IWDT_EN**, **CSC_SLP_IWDT** and **CSC_STP_IWDT** registers in CSC as like as the module process clock.

One 12-bit clock divider is using for the IWDT timer input clock. User can set the divider by divided 1/2/4/8/~4096 in **IWDT_CK_DIV** register.

22.5. Interrupt and Event

There is one signal of **INT_IWDT** to be generated in this IWDT control module. **INT_IWDT** sends to EXIC External Interrupt Controller to do as an interrupt event.

22.5.1. IWDT Interrupt Control and Status

These interrupt flags are using for the interrupt service routine (ISR) flow control. Generally these flags are set by hardware and software clears them when the serviced job is completed in the related ISR. Each interrupt flag has one interrupt enable bit. User can enable or disable it.

22.5.2. IWDT Interrupt Flags

Generally these interrupt flags are set by hardware and clear by software writing 1. Refer the register descriptions for more information about the related interrupt flags and interrupt enable bit.

● TF

IWDT timer timeout interrupt flag (**IWDT_TF**). There is a related interrupt enable register bit of **IWDT_TIE**.

● EW0F

IWDT early wakeup-0 interrupt flag (**IWDT_EW0F**). There is a related interrupt enable register bit of **IWDT_EW0_IE**. This flag is set when the counter value reaches to **0x20**.

● EW1F

IWDT early wakeup-1 interrupt flag (**IWDT_EW1F**). There is a related interrupt enable register bit of

IWDT_EW1_IE. This flag is set when the counter value reaches to **0x40**.

22.6. Register Protect and Lock

After IWDT domain reset, all IWDT registers are write-access protected except **IWDT_STA**, **IWDT_KEY** registers. Write **0xA217** value to the **IWDT_KEY** register to unprotect the registers and process the control continuously. Oppositely write other value except **0xA217** value to protect the registers. Read the **IWDT_KEY** register to get the register value is Protected (=1) or Unprotected (=0).

It also provides the register lock function after warm reset condition. Write value **0x712A** to the **IWDT_LOCK** register to lock the register write access except **IWDT_STA**, **IWDT_KEY** registers. When registers lock, the registers cannot change event through the system warm reset until the hardware cold reset. Write other value except **0x712A** is no effect. Read the **IWDT_LOCK** register to get the register value is Locked (=1) or Unlocked (=0).

Special for hardware function control, the following registers of **CSC_IWDT_EN**, **CSC_SLP_IWDT**, **CSC_STP_IWDT** and **PW_WKSTP_IWDT** are locked control by the **IWDT_LOCK** register. When the **IWDT_LOCK** register is set locked, these registers are not to be reset by warm reset.

Refer the table and descriptions of “[Register Protect and Lock](#)” in System Reset chapter for more information.

The hardware option of **CFG_IWDT_WP** defines the IWDT registers write protected default value those are not volatile after power off. The register value is loaded from the option bytes (**OB**) flash memory after system reset. Refer the Hardware Option chapter for more information.

The following table is showing the IWDT register write protection for different setting.

Table 22-1. IWDT Register Write Protection Table

Register Setting			IWDT Register Protection		
CFG_IWDT_WP	IWDT_LOCK	IWDT_KEY	IWDT_KEY IWDT_STA	IWDT_EW1_IE IWDT_EW0_IE IWDT_TIE IWDT_EW1_WPEN IWDT_EW0_WPEN IWDT_TF_WPEN	Others
Enable	Locked	Protected	V	-	-
		Unprotected		-	
	Unlocked	Protected		-	
		Unprotected		V	
Disable	Locked	Protected	V	-	-
		Unprotected		-	-
	Unlocked	Protected		-	-
		Unprotected		V	V

<Sign> V: write access allowed, - : write access protected

22.7. IWDT Function Control

22.7.1. IWDT Timer Control

The IWDT watch-dog timer consists of a 12-bit prescaler and an 8-bit timer. The timer is enabled by setting **IWDT_EN** register. When the watch-dog timer is enabled, software should always reset the timer by writing value **0x2014** to the **IWDT_KEY** register before the timer is timeout. The watch-dog timer is a down-counting timer and user can read the register of **IWDT_CNT** to get the actual counter value. When the watch-dog timer is reset, the timer will be reloaded 0xFF value to restart counting.

If the chip is out of control by any disturbance, that means the CPU may not run the software normally. Then the firmware may miss to reset the timer (writing value **0x2014** to the **IWDT_KEY** register) and the timer timeout will be coming. The timeout of watch-dog timer makes the IWDT generating a reset event **RST_IWDT** and sends it to Reset Source Controller (RST) to do as the warm reset events or cold reset events.

This reset event can be enabled to reset the chip by setting the registers in RST. Finally, this reset event will make CPU to restart if the related reset control bits have enabled. Refer the System Reset chapter for more information about the reset events and control.

The IWDT is able to record default initialized value in hardware option byte (**OB**) about IWDT on/off, input clock divider value, IWDT registers write protection and associates with the **IWDT_EN**, **IWDT_CK_DIV** and **IWDT_LOCK** registers. These registers can be automatically initialized during power-on or cold reset cycle by the configurable hardware option byte (**OB**).

22.7.2. Operation in STOP

The IWDT is able to operate in **STOP** mode if the user set the **CSC_IWDT_EN** and **CSC_STP_IWDT** registers beforehand and the chip entering **SLEEP** or **STOP** mode. When the IWDT is operation in **STOP** mode, the APB clock is stopped and the module is asynchronous control for all logic.

22.7.3. Wakeup from STOP

The IWDT supports to wakeup chip in **STOP** mode by the events of watch-dog timer underflow, early wakeup-0 detection (**IWDT_EW0F** flag is asserted) and early wakeup-1 detection (**IWDT_EW1F** flag is asserted). There are independent wakeup enable bits of **IWDT_TF_WPEN**, **IWDT_EW0_WPEN** and **IWDT_EW1_WPEN** for these three wakeup events.

When the chip is entering **STOP** mode and any of these IWDT wakeup events is happened, the IWDT will send the wakeup event to Power Controller (PW) to do as the system wakeup events. If the related control registers are enabled, the wakeup event will make to wakeup the chip.

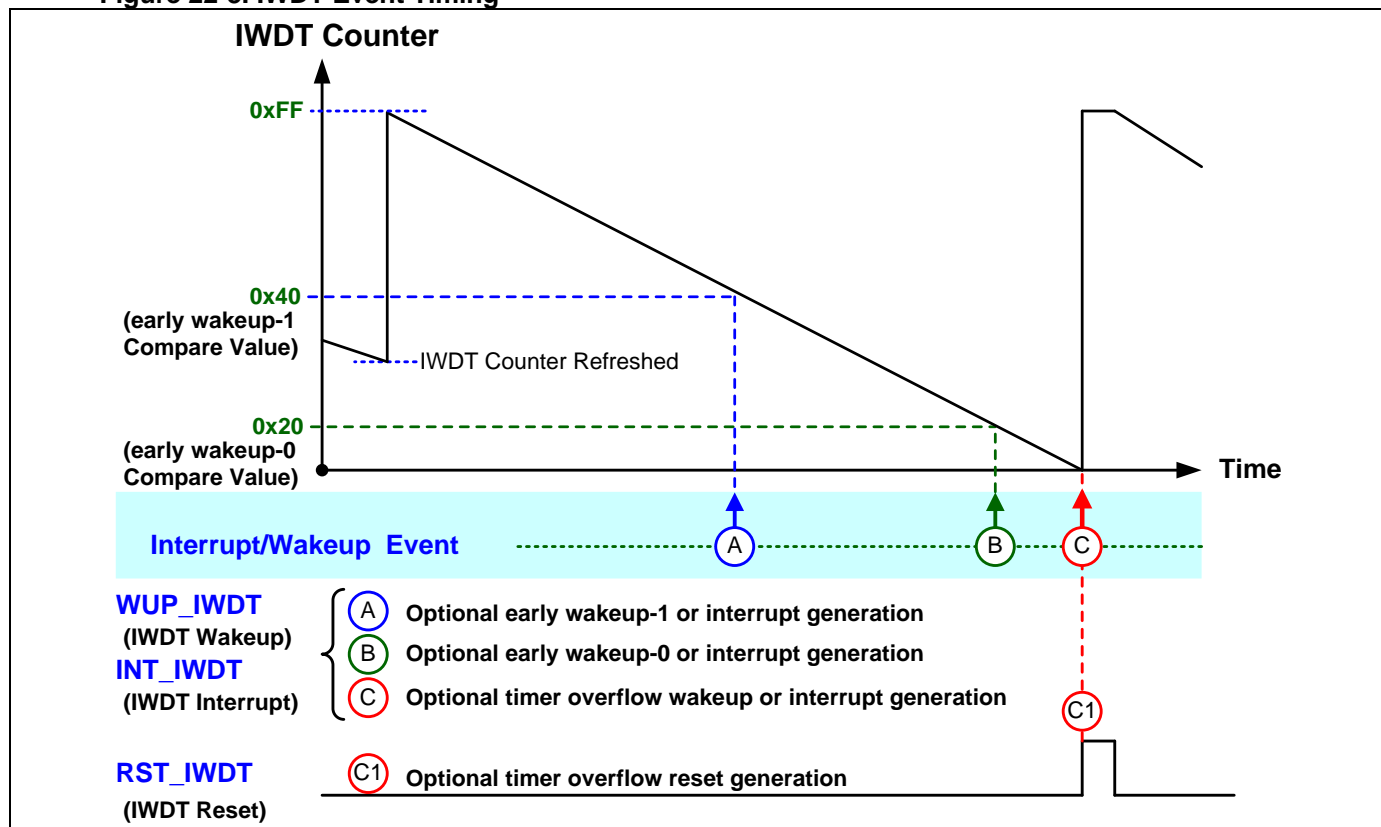
Refer the descriptions of System Power and Interrupt chapters for more information about the wakeup events and control.

22.7.4. IWDT Event Timing

The IWDT watch-dog timer is a down-counting timer and the timer is always counting start from 0xFF value. The IWDT is able to output **IWDT_EW0F** and **IWDT_EW1F** flags for IWDT timer counting hit events at fixed 0x20, 0x40 counter values. User can implement the management code in these two ISR to fix the probable problem beforehand before the IWDT timer underflow. Also IWDT is able to output **IWDT_TF** as IWDT timer timeout flag.

The following diagram is showing the IWDT event timing.

Figure 22-3. IWDT Event Timing



23.4. Enabling and Clock

23.4.1. WWDT Global Enable

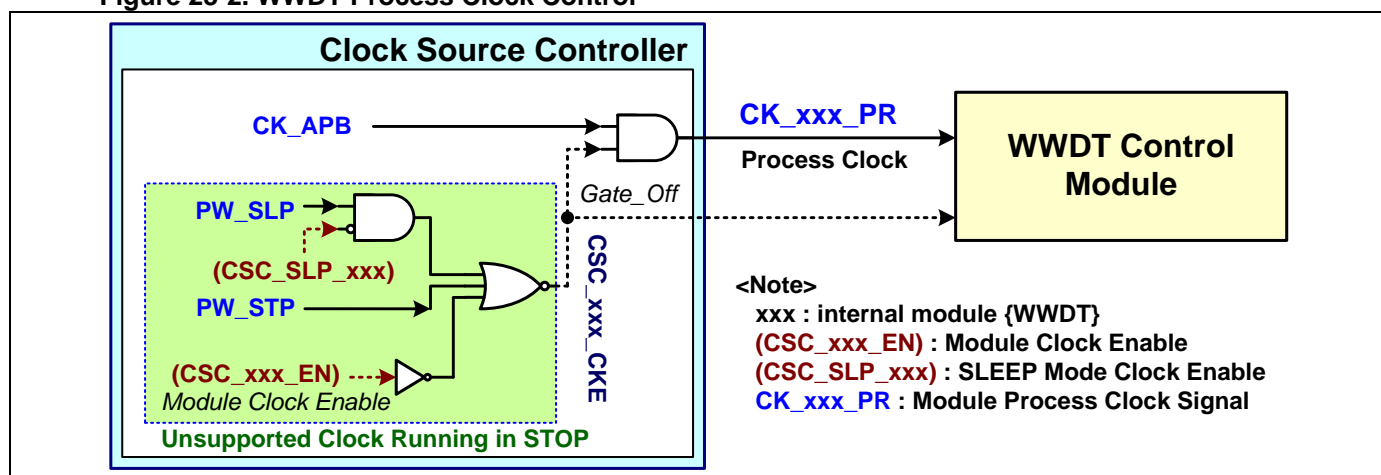
There is a global enable bit of **WWDT_EN** for all functions of this module. When this bit is disabled, all the WWDT functions are not working.

23.4.2. WWDT Clock Control

- **Module Process Clock**

The module process clock of **CK_WWDT_PR** is coming from CSC (Clock Source Controller) module. It can be enabled in **CSC_WWDT_EN** register. User can plan the module clock is running or not beforehand for chip entering **SLEEP** modes by setting **CSC_SLP_WWDT** register. Refer the System Clock chapter for more information.

Figure 23-2. WWDT Process Clock Control



- **Module Internal Clock**

User can select the internal timer clock source from the internal APB clock of **CK_APB** or the internal unit clock of **CK_UT** by setting **WWDT_CK_SEL** register. The internal timer clock is gated off when the module global enable bit of **WWDT_EN** is disabled.

The internal timer clock is also gating control by the **CSC_WWDT_EN** and **CSC_SLP_WWDT** registers in CSC as like as the module process clock.

One clock prescaler and one clock divider are using for the WWDT timer input clock. User can set the prescaler by divided 1 or 256 in **WWDT_CK_PDIV** register and the divider by divided 1/2/4/8~128 in **WWDT_CK_DIV** register.

[Notify]: Usually the internal clock frequency needs slow down at least 1/2 than module process clock.

23.5. Interrupt and Event

There is one signal of **INT_WWDT** to be generated in this WWDT control module. **INT_WWDT** sends to EXIC External Interrupt Controller to do as an interrupt event.

23.5.1. WWDT Interrupt Control and Status

These interrupt flags are using for the interrupt service routine (ISR) flow control. Generally these flags are set by hardware and software clears them when the serviced job is completed in the related ISR. Each interrupt flag has one interrupt enable bit. User can enable or disable it.

23.5.2. WWDT Interrupt Flags

Generally these interrupt flags are set by hardware and clear by software writing 1. Refer the register descriptions for more information about the related interrupt flags and interrupt enable bit.

- **TF**

WWDT timer timeout interrupt flag (**WWDT_TF**). There is a related interrupt enable register bit of **WWDT_TIE**.

- **WINF**

WWDT counter refreshing and value over the window compare threshold condition flag (**WWDT_WINF**). There is a related interrupt enable register bit of **WWDT_WIN_IE**. It is set when the **WWDT_KEY** is written **0x2014** to reload timer by firmware and the counter value is over the threshold value of **WWDT_WIN** in the same time.

- **WRNF**

WWDT counter warning flag (**WWDT_WRNF**). There is a related interrupt enable register bit of **WWDT_WRN_IE**. It is set when the WWDT counter reaches the value of **WWDT_WRN**.

23.6. WWDT Register Protect

After WWDT domain reset, all WWDT registers are write-access protected except **WWDT_STA**, **WWDT_KEY** registers. Write **0xA217** value to the **WWDT_KEY** register to unprotect the registers and process the control continuously. Oppositely write other value except **0xA217** value to protect the registers. Read the **WWDT_KEY** register to get the register value is Protected (=1) or Unprotected (=0).

Refer the table and descriptions of “[Register Protect and Lock](#)” in System Reset chapter for more information.

23.7. WWDT Function Control

23.7.1. WWDT Timer Control

The WWDT watch-dog timer consists of one /1 or /256 clock prescaler, one 7-bit clock divider and one 10-bit timer. The timer is enabled by setting **WWDT_EN** register. When the watch-dog timer is enabled, software should always reset the timer by writing value **0x2014** to the **WWDT_KEY** register before the timer is timeout. The watch-dog timer is a down-counting timer and user can read the register of **WWDT_CNT** to get the actual counter value. When the watch-dog timer is reset, the timer will reload the value from **WWDT_RLR** register to restart counting.

When the firmware is out of control, which may miss to reset the timer (writing value **0x2014** to the **WWDT_KEY** register) and the timer timeout will be coming. The timeout of watch-dog timer makes the WWDT generating a reset event **RST_WWDT** and sends it to Reset Source Controller (RST) to do as the warm reset events or cold reset events. If the firmware is written **0x2014** to **WWDT_KEY** register (reset the timer) to reload timer and the counter value is over the window compare threshold value of **WWDT_WIN** in the same time, it also makes the WWDT generating a reset event. There are independent reset event enable bits of **WWDT_RSTF_EN** and **WWDT_RSTW_EN** for these reset events.

This reset events can be enabled to reset the chip by setting the registers in RST. Finally, this reset event will make CPU to restart if the related reset control bits have enabled. Refer the System Reset chapter for more information about the reset events and control.

23.7.2. WWDT Module Reset Control

The WWDT module can be reset by Cold reset. Also it can be reset by Warm reset if the WWDT warm reset disable register of **RST_WWDT_DIS** is set 0. When the **RST_WWDT_DIS** register is set 1, the WWDT module cannot be reset by Warm reset. In the conditions, the WWDT can still count for watch-dog function once during the following Warm reset period by other reset event. Subsequently, the WWDT will be reset also if the WWDT is counting underflow.

There is one reset enable register of **RST_WWDT_EN** to directly force WWDT module reset for software control.

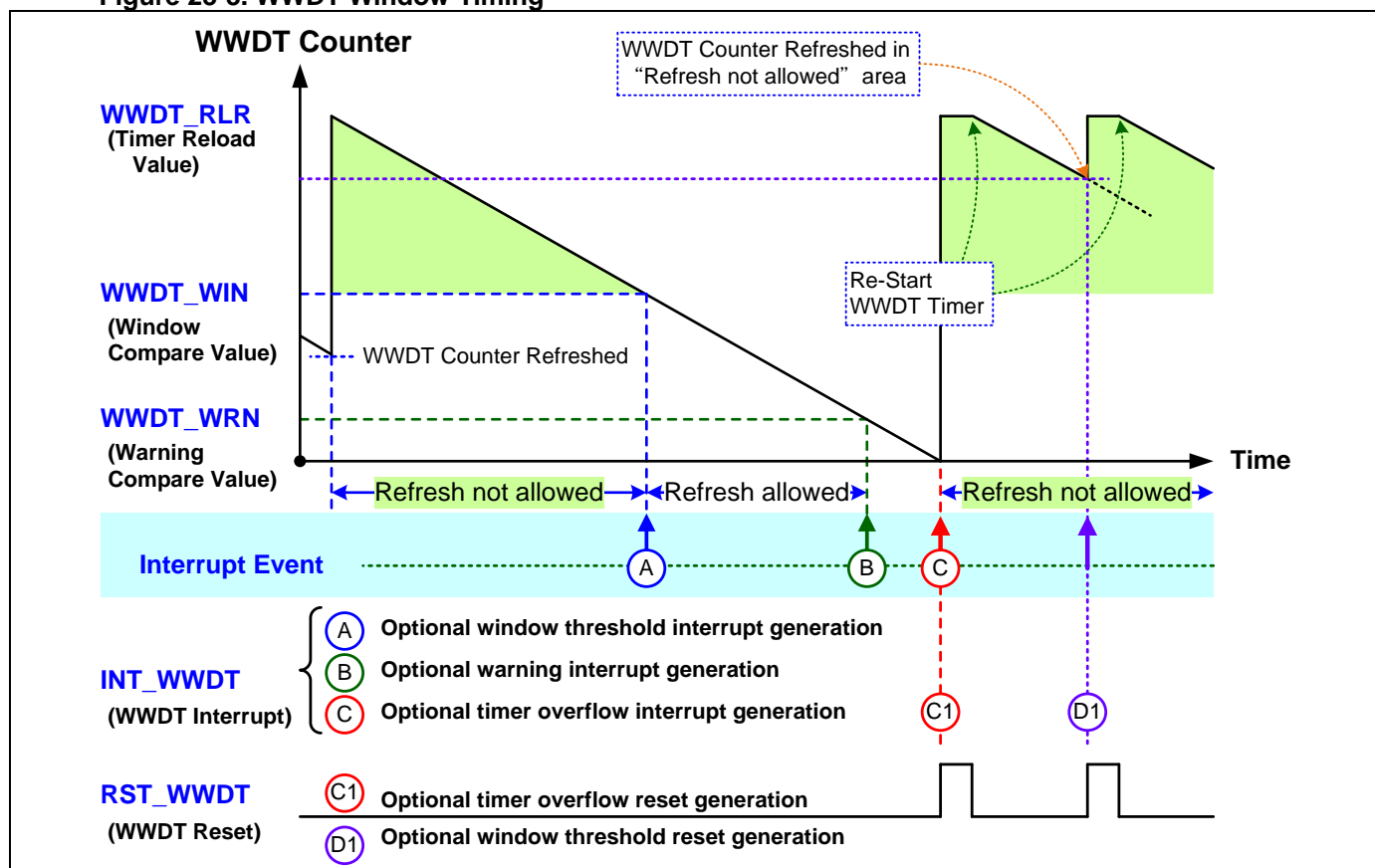
23.7.3. WWDT Window Timing

The WWDT watch-dog timer is a down-counting timer and the timer is counting start from the reload value from **WWDT_RLR** register.

WWDT is able to output **WWDT_WINF** and **WWDT_WRNF** flags for WWDT timer counting events of window compare and warning with programmable compared counter value. There are **WWDT_WIN** and **WWDT_WRN** registers to set the compared counter value for asserting **WWDT_WINF** and **WWDT_WRN** flags. Also WWDT is able to output **WWDT_TF** as WWDT timer timeout flag.

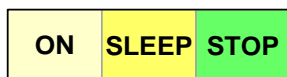
The following diagram is showing the WWDT window timing.

Figure 23-3. WWDT Window Timing



24. RTC (Real Time Clock)

24.1. Introduction



The module can be running in all power operation modes.

The real-time clock is an independent 32-bit timer. The RTC provides a time clock with programmable alarm interrupt. User can use as a calendar with software programmable alarm seconds, minutes, hours, day, and date.

The RTC provides a wakeup flag to perform auto wakeup from power down mode with interrupt.

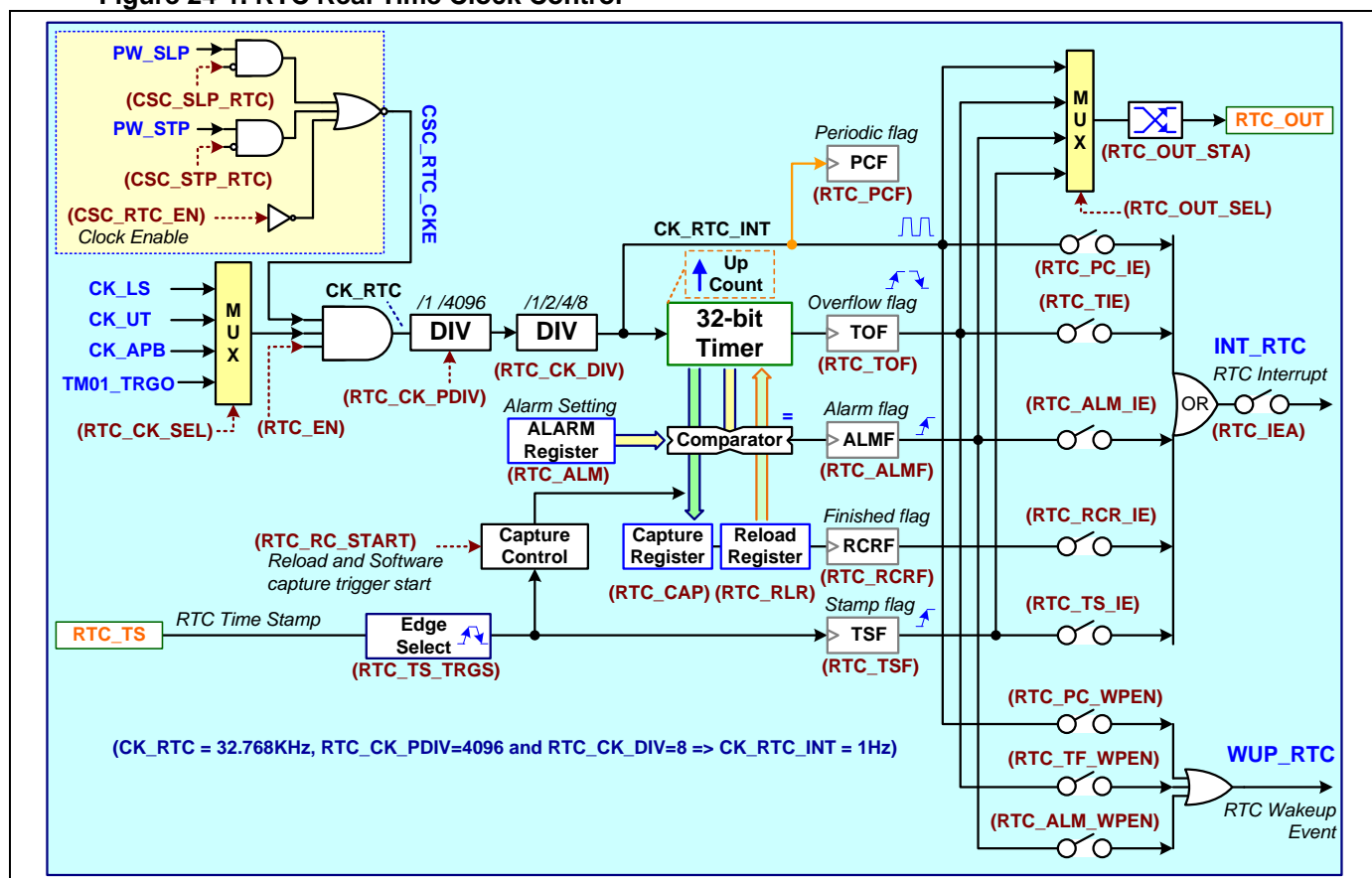
24.2. Features

- Built-in 32-bit counter with selectable clock source
- Support alarm function and time-stamp function
 - Support alarm function with 32-bit programmable compare register
- Support wakeup from Stop mode
- Support periodic timer tick interrupt or wakeup
- Support register key-protected and reset-locked functions

24.3. Control Block

The RTC consists of one clock prescaler and one clock divider, a 32-bit timer, alarm digital comparator and time stamp control logic. The following diagram is showing the RTC Control block.

Figure 24-1. RTC Real Time Clock Control



24.4. IO Lines

24.4.1. IO Signals

- **RTC_TS**

It is the RTC time stamp input signal. When RTC receives a time stamp signal, the TSF flag will be asserted and the RTC timer value will be captured to the capture register.

- **RTC_OUT**

It is the selective output signal which is selected from the RTC internal events or signals and output to pin or other internal modules.

24.4.2. IO Configure

User must configure the related IO pins in order to use the IO lines of this module. The IO operation modes, output high speed option, pull-high option, output drive strength, IO deglitch filter and input inverse selection are programmable by pin independent. Please refer the section of “[IO Mode](#)” in GPIO chapter for more detail descriptions of IO mode configuration.

Each IO signal may be mapped and selected on several IO pins by configuring the IO AFS matrix. Please refer the section of “[Alternate Function Select](#)” in GPIO chapter for more detail descriptions of IO AFS configuration. About the actual IO pin AFS information, please refer the section of “[Pin Alternate Functions Selected Table](#)” in Pin Description chapter of the chip Data Sheet.

24.5. Enabling and Clock

24.5.1. RTC Global Enable

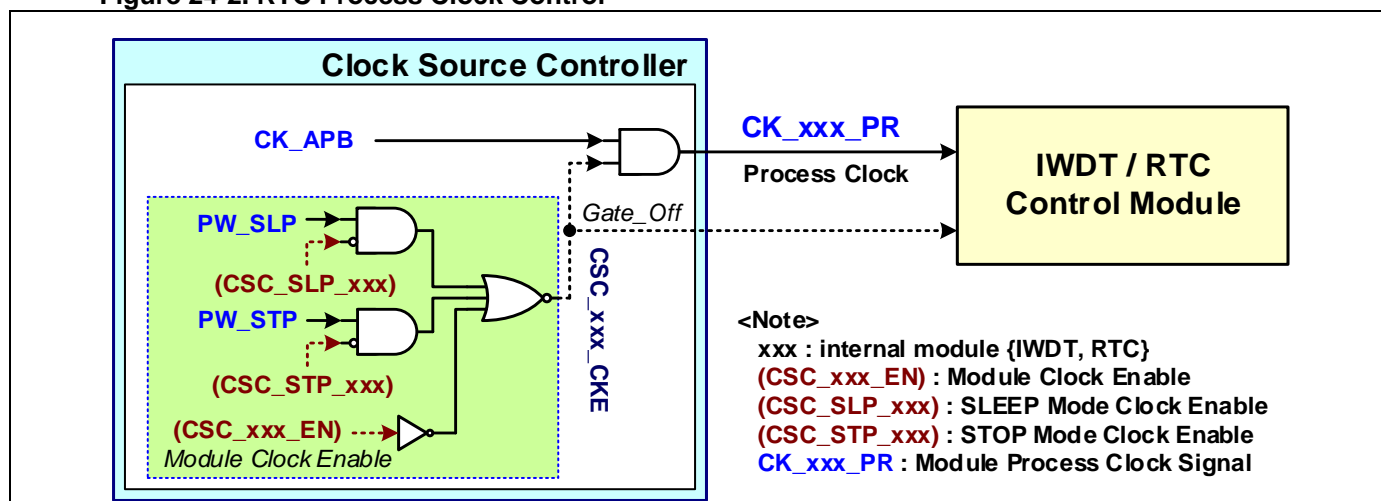
There is a global enable bit of **RTC_EN** for all functions of this module. When this bit is disabled, all the RTC functions are not working.

24.5.2. RTC Clock Control

- **Module Process Clock**

The module process clock of **CK_RTC_PR** is using for the interface control logic between APB bus and the module. It is coming from CSC (Clock Source Controller) module. It can be enabled in **CSC_RTC_EN** register. User can plan the module clock is running or not beforehand the chip entering **SLEEP** or **STOP** modes by setting **CSC_SLP_RTC** or **CSC_STP_RTC** registers. Refer the System Clock chapter for more information.

Figure 24-2. RTC Process Clock Control



- **Module Internal Clock**

User can select the internal timer clock source from the internal ILRCO clock of **CK_ILRCO**, the internal unit clock of **CK_UT**, the APB clock of **CK_APB** or the timer trigger output signal of **TM01_TRGO** by setting **RTC_CK_SEL** register. The internal timer clock is gated off when the module global enable bit of **RTC_EN** is disabled.

The internal timer clock is also gating control by the **CSC_RTC_EN**, **CSC_SLP_RTC** and **CSC_STP_RTC** registers in CSC as like as the module process clock.

One clock prescaler and one clock divider are using for the RTC timer input clock. User can set the prescaler

by divided 1 or 4096 in **RTC_CK_PDIV** register and the divider by divided 1/2/4/8 in **RTC_CK_DIV** register.
[Notify]: Usually the internal clock frequency needs slow down at least 1/2 than module process clock.

24.6. Interrupt and Event

There is one signal of **INT_RTC** to be generated in this RTC control module. **INT_RTC** sends to EXIC External Interrupt Controller to do as an interrupt event.

24.6.1. RTC Interrupt Control and Status

These interrupt flags are using for the interrupt service routine (ISR) flow control. Generally these flags are set by hardware and software clears them when the serviced job is completed in the related ISR. Each interrupt flag has one interrupt enable bit. User can enable or disable it. There is one interrupt all enable bit of **RTC_IEA** to enable or disable all the interrupt sources for this module.

24.6.2. RTC Interrupt Flags

Generally these interrupt flags are set by hardware and clear by software writing 1. Refer the register descriptions for more information about the related interrupt flags and interrupt enable bit.

- **ALMF**

RTC alarm matched interrupt flag (**RTC_ALMF**). There is a related interrupt enable register bit of **RTC_ALM_IE**.

- **PCF**

RTC periodic interrupt flag (**RTC_PCF**). There is a related interrupt enable register bit of **RTC_PC_IE**. Usually user can adjust the timer input divider to set the timer input clock as 1Hz. Then the user can service the periodic work every 1sec by the periodic interrupt trigger.

- **TSF**

RTC time stamp interrupt flag (**RTC_TSF**). There is a related interrupt enable register bit of **RTC_TS_IE**. It indicates one time stamp event is detected.

- **TOF**

RTC timer counting overflow interrupt flag (**RTC_TOF**). There is a related interrupt enable register bit of **RTC_TIE**.

- **RCRF**

RTC reload or capture flag (**RTC_RCRF**). This flag is active when RTC_RLR register reload finished, RTC_CAP register software capture finished or RTC_ALM register value update allowed flag... There is a related interrupt enable register bit of **RTC_RCR_IE**.

24.6.3. RTC Clock Status

- **CK_STA**

RTC input clock source select MUX switching status (**RTC_CK_STA**). If the read back value is 0, it indicates the clock source select MUX is switching and clock is not yet stable. This bit is read only that is set and cleared by hardware control.

24.7. Register Protect and Lock

After RTC domain reset, all RTC registers are write-access protected except **RTC_STA**, **RTC_KEY** registers. Write **0xA217** value to the **RTC_KEY** register to unprotect the registers and process the control continuously. Oppositely write other value except **0xA217** value to protect the registers. Read the **RTC_KEY** register to get the register value is Protected (=1) or Unprotected (=0).

It also provides the register lock function after warm reset condition. Write value **0x712A** to the **RTC_LOCK** register to lock the register write access except **RTC_STA**, **RTC_KEY** registers. When registers lock, the registers cannot change event through the system warm reset until the hardware cold reset. Write other value except **0x712A** is no effect. Read the **RTC_LOCK** register to get the register value is Locked (=1) or Unlocked (=0).

Special for hardware function control, the following registers of **CSC_RTC_EN**, **CSC_SLP_RTC**, **CSC_STP_RTC** and **PW_WKSTP_RTC** are locked control by the **RTC_LOCK** register. When the **RTC_LOCK** register is set locked, these registers are not to be reset by warm reset.

Refer the table and descriptions of "[Register Protect and Lock](#)" in System Reset chapter for more information.

24.8. RTC Function Control

24.8.1. RTC Alarm

The RTC supports an alarm function and enables by **RTC_ALM_EN** bit. The **RTC_ALM** register sets the RTC alarm compare value. This register is able to update under **RTC_ALM_EN** disabled. When **RTC_ALM_EN** disables, hardware will assert the **RTC_RCRF** flag to notify software. Then software can update the **RTC_ALM** register value.

When the RTC timer value is matched with **RTC_ALM** value, the RTC alarm flag (**RTC_ALMF**) is asserted and generates an interrupt if **RTC_ALM_IE**=1.

24.8.2. RTC Time Stamp

The RTC supports a time stamp function by external input. User can select input trigger edge of rising edge, falling edge or dual-edge by **RTC_TS_TRGS** register.

When an external input signal is matched, the RTC time stamp flag (**RTC_TSF**) is asserted and generates an interrupt if **RTC_TS_IE**=1.

24.8.3. RTC Timer Capture and Reload

The RTC can capture from the 32-bit timer value to **RTC_CAP** register or reload value to the 32-bit timer from **RTC_RLR** register. There are four control modes of "Directly capture", "Delayed capture", "Forced reload", "Auto reload" for the using of **RTC_RLR** and **RTC_CAP** registers.

*[Notify]: The value 0xFFFFFFFF is invalid setting for **RTC_RLR** register.*

The RTC reload or capture control mode is selected by **RTC_RCR_MDS** bits. If selects "Directly capture" or "Delayed capture" mode, the RTC timer counter value will capture into the **RTC_CAP** register when software capture event (**RTC_RC_START**=1) or hardware time stamp event happened. If selects "Force reload", the RTC timer counter will be updated by **RTC_RLR** register value when **RTC_RLR** has been written. If selects "Auto reload" mode, the RTC timer counter will be update by **RTC_RLR** register value when RTC timer is overflow.

When **RTC_RLR** register reload finished, **RTC_CAP** register software capture finished or **RTC_ALM** register value update allowed, the **RTC_RCRF** flag is asserted and generates a interrupt if **RTC_RCR_IE**=1.

24.8.4. RTC Output

One **RTC_OUT** output is able to output the RTC internal signals to internal modules or external pin. There are four signals of timer overflow signal toggle output, time stamp trigger event, timer input periodic clock signal **CK_RTC_INT** and alarm compare output event which can be sent from **RTC_OUT** output and selected the output signal by setting **RTC_OUT_SEL** register.

User can set the initial state of the output signal by setting **RTCx_OUT_STA** register. Specially, the initial state register can be written only when the register of **RTCx_OUT_LCK** is written "1" simultaneously.

24.8.5. Operation in STOP

The RTC is able to operate in **STOP** mode if the user set the **CSC_RTC_EN** and **CSC_STP_RTC** registers beforehand the chip entering **STOP** mode. When the RTC is operation in **STOP** mode, the APB clock is stopped and the module is asynchronous control for all logic.

24.8.6. Wakeup from STOP

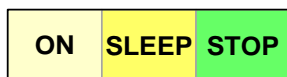
The RTC supports to wakeup chip in **STOP** mode by the events of timer overflow, timer input periodic clock and alarm compare output. There are independent wakeup enable bits of **RTC_TF_WPEN**, **RTC_PC_WPEN** and **RTC_ALM_WPEN** for these three wakeup events.

When the chip is entering **STOP** mode and any of these RTC wakeup events is happened, the RTC will send the wakeup event to Power Controller (PW) to do as the system wakeup events. If the related control registers are enabled, the wakeup event will make to wakeup the chip.

Refer the descriptions of System Power and Interrupt chapters for more information about the wakeup events and control.

25. LCD (Liquid Crystal Display Controller)

25.1. Introduction



The module can be running in all power operation modes.

The chip is built-in a LCD module which is including of the LCD controller, charge pump, bias generator and output voltage drivers. The output voltage drivers can directly drive the external monochrome passive LCD glass. It supports total 44 LCD driver lines with configurable LCD common or segment for maximum 8 common (COM) lines and 40 segment (SEG) lines. It supports static, 1/2, 1/3, 1/4 bias voltage and static, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8 duty.

The LCD controller can directly drive LCD displays and automatically generate the requested voltage waveform to avoid an electrophoresis effect for each configured common pin and segment pin. It is embedded the LCD display memory with blinking display function. It also supports both type-A and type-B frame timing mode with dead time control.

Notify: The sign of (n = LCD pin number) is using for Registers, Signals and Pins in the descriptions of this chapter. [EX]: **LCD_Pn**, **LCD_Mn** ~ n indicates LCD pin number.

25.2. Features

- Support external power input, internal VDD or internal charge pump voltage source
 - VDD voltage range 2.0V to 5.5V for internal charge pump
 - External LCD_VT voltage range 1.8V to 5.5V for external power input
 - Internal charge pump with adjustable contrast adjustment
- Embedded LCD bias reference ladder
 - LCD power rails (LCD_V1, LCD_V2, LCD_V3, LCD_VT) with external decoupling capability
 - Support external resistor ladder optional
- Provide maximum 8 common/40 segment and total 44 pins with GPIO alternate function
 - Up to drive 160 (40x4), 228 (38x6) or 288 (36x8) LCD dots
 - Support configurable common or segment for each LCD pins
 - Support hardware phase inversion
- Support Static, 1/2, 1/3 and 1/4 bias voltage
- Support Static, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7 and 1/8 duty
- Supports Type-A or Type-B frame control mode
 - Configurable LCD frame frequency
- Embedded LCD data RAM
- Programmable dead time between frames or duty phases.
- Support LCD blinking display
- LCD start frame interrupt for LCD data RAM update
- Support LCD display on SLEEP and STOP modes

25.3. Implementation

25.3.1. Chip Implementation

The following table is showing the implementation of LCD module.

Table 25-1. LCD Implementation

Chip	Bias and Duty			COM/SEG LCD Pin		LCD Function		
	Bias Source	Bias Voltage	Duty	Total Pins	Configurable	Frame Timing	Blinking	Dead Time
MG32F02N128/N064 MG32F02K128/K064	AVDD/EXT/CP	1/2, 1/3, 1/4	1/2 ~ 1/8	44	V	Type-A/B	V	V
MG32F02A128/A064 MG32F02U128/U064 MG32F02V032	Not Implemented							

<Note> V: Implemented; AVDD: internal power; EXT: external power; CP: internal charge pump

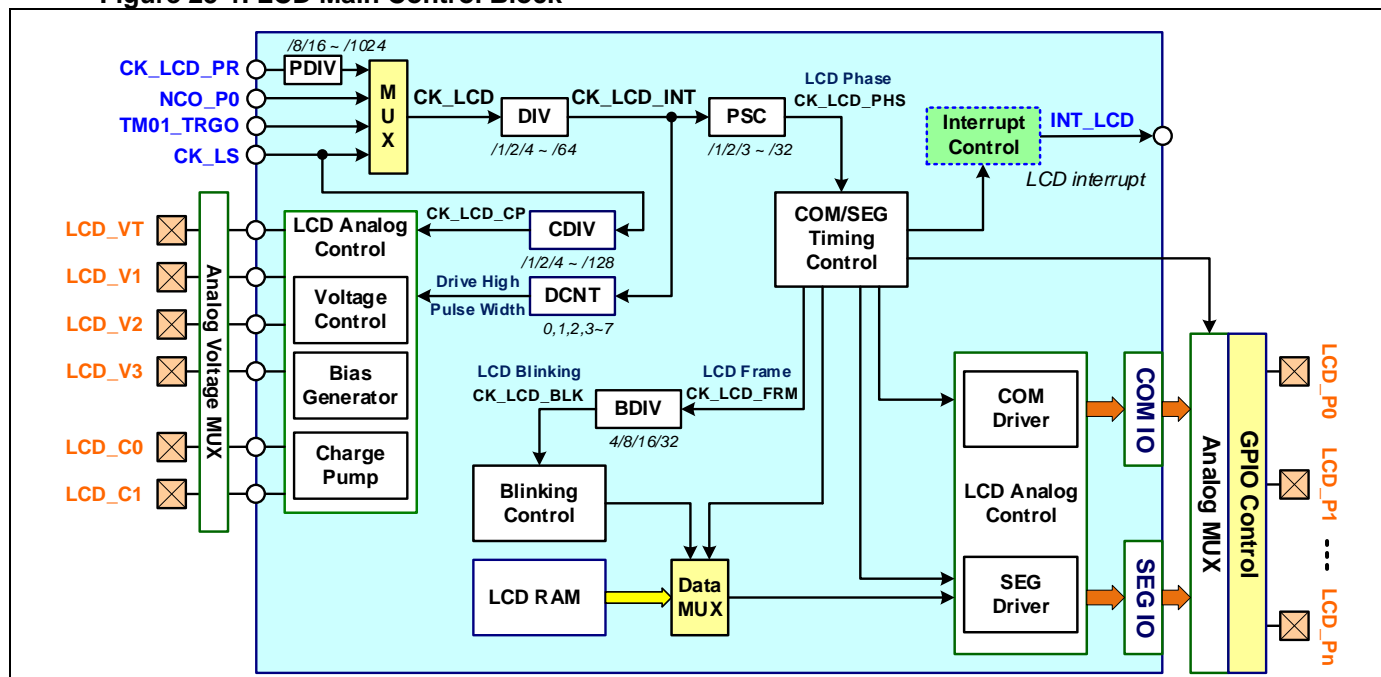
25.4. Control Block

The LCD module is built-in charge pump, bias generator and output voltage drivers. The bias generator supports static, 1/2, 1/3, 1/4 bias voltage. The output voltage drivers can directly drive the external monochrome passive LCD glass.

The module is embedded the LCD RAM and implements an independent 8-bit data register for each LCD pins. User can update the LCD RAM through these registers to update the LCD display.

The following diagram is showing the LCD Control block.

Figure 25-1. LCD Main Control Block



25.5. IO Lines

25.5.1. IO Signals

- **LCD_P[0..43]**

These signals are the LCD common or segment signals. These output voltage drivers of **LCD_P[0..43]** can directly drive the external monochrome passive LCD glass. The chip supports total 44 LCD driver lines with configurable LCD common or segment for maximum 8 common (COM) lines and 40 segment (SEG) lines.

25.5.2. IO Configure

User must configure the related IO pins in order to use the IO lines of this module. The IO operation modes, output high speed option, pull-high option, output drive strength, IO deglitch filter and input inverse selection are programmable by pin independent. Please refer the section of “[IO Mode](#)” in GPIO chapter for more detail descriptions of IO mode configuration.

Each IO signal may be mapped and selected on several IO pins by configuring the IO AFS matrix. Please refer the section of “[Alternate Function Select](#)” in GPIO chapter for more detail descriptions of IO AFS configuration. About the actual IO pin AFS information, please refer the section of “[Pin Alternate Functions Selected Table](#)” in Pin Description chapter of the chip Data Sheet.

25.6. Enabling and Clock

25.6.1. LCD Global Enable

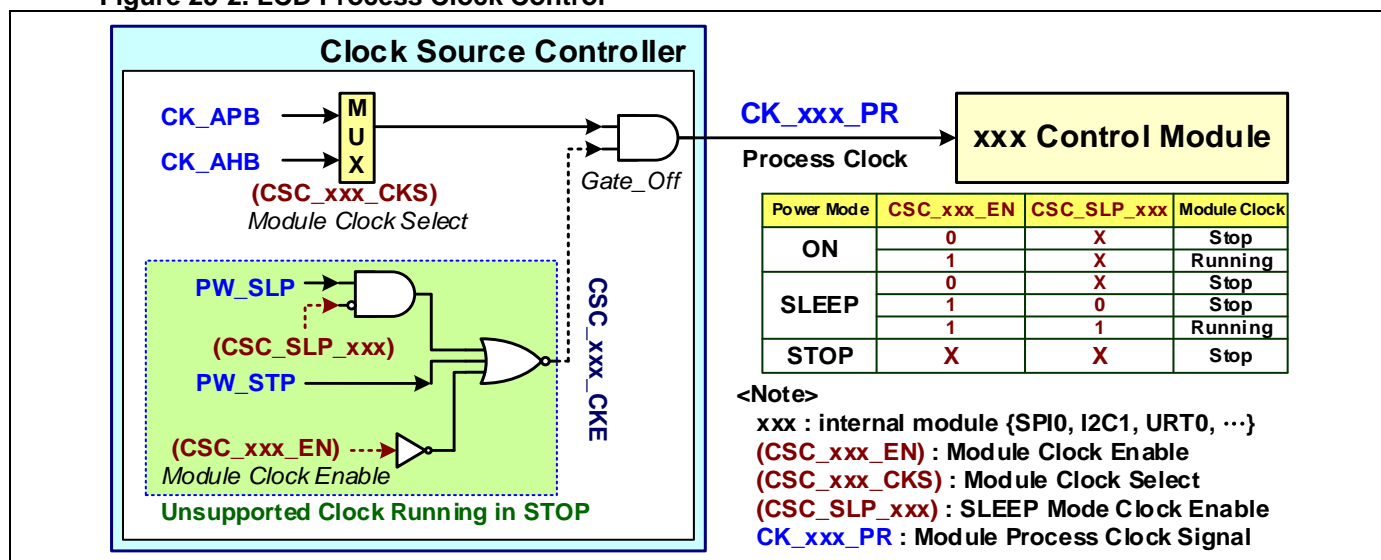
There is a global enable bit of **LCD_EN** for all functions of this module. When this bit is disabled, all the LCD functions are not working. The module can select all the LCD drive output state of **LCD_Pn** to tristate or output low by setting **LCD_OFF_CTL** register before the **LCD_EN** register is enabled. When the **LCD_OFF_CTL** bit is selected 'HiZ', the **LCD_Pn** outputs are set to tristate. When the **LCD_OFF_CTL** bit selects 'Low', the **LCD_Pn** outputs are set to low level. (n= LCD_Pn pin index number)

25.6.2. LCD Clock Control

- **Module Process Clock**

The module process clock of **CK_LCD_PR** is using for the interface control logic between APB bus and the module. It is coming from CSC (Clock Source Controller) module. It can be enabled in **CSC_LCD_EN** register and select the clock source from APB clock or AHB clock in **CSC_LCD_CKS** register. User can plan the module clock is running or not beforehand for chip entering **SLEEP** mode by setting **CSC_SLP_LCD** register. Refer the System Clock chapter for more information.

Figure 25-2. LCD Process Clock Control



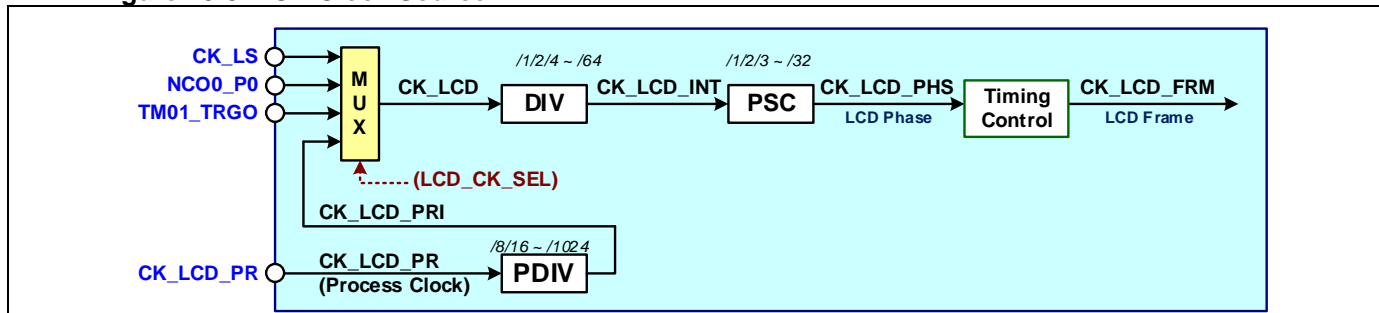
● Module Internal Clock

User can select the module clock source of **CK_LCD** from the clock of **CK_LCD_PRI** and **NCO_P0**, the timer trigger output signal of **TM01_TRGO** or internal ILRCO clock of **CK_ILRCO** by setting **LCD_CK_SEL** register. The clock of **CK_LCD_PRI** is outputted by a clock divider from the module process clock of **CK_LCD_PR**.

The LCD module is able to generate the internal clock of **CK_LCD_INT** by a clock divider from the module clock of **CK_LCD** as the LCD clock source for LCD display timing.

The following diagram is showing the LCD clock source block.

Figure 25-3. LCD Clock Source



25.7. Interrupt and Event

25.7.1. LCD Interrupt Control and Status

There is one signal of **INT_LCD** to be generated in this LCD control module. **INT_LCD** sends to EXIC External Interrupt Controller to do as an interrupt event.

These interrupt flags are using for the interrupt service routine (ISR) flow control. Generally these flags are set by hardware and software clears them when the serviced job is completed in the related ISR. Each interrupt flag has one interrupt enable bit. User can enable or disable it. There is one interrupt all enable bit of **LCD_IEA** to enable or disable all the interrupt sources for this module.

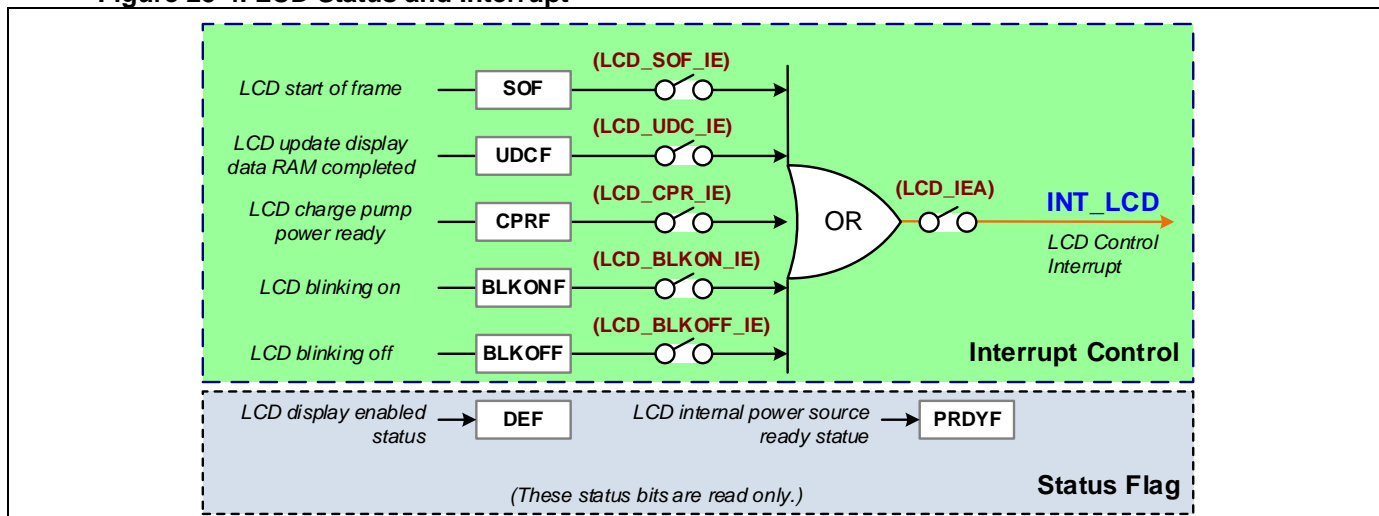
There are some status bits those are reading only to provide internal control status. Refer the register descriptions of the related status bits for more information.

25.7.2. LCD Interrupt Flags

Refer the register descriptions for more information about the related interrupt flags and interrupt enable bit.

The following diagram is showing the LCD status and interrupt control block.

Figure 25-4. LCD Status and Interrupt



❖ Interrupt Flags

The following interrupt flag bits are able to write or read. They are set by hardware and can clear by software writing 1.

- **SOF**

LCD start of frame interrupt flag (**LCD_SOF**). There is a related interrupt enable register bit of **LCD_SOF_IE**. This bit is set when the LCD display starts a new frame and the display data has updated.

- **UDCF**

LCD update display data completed interrupt flag (**LCD_UDCF**). There is a related interrupt enable register bit of **LCD_UDC_IE**. This bit is set by hardware when the current display data has updated to display memory and user can start to write next display frame data.

- **CPRF**

LCD charge pump power ready interrupt flag (**LCD_CPRF**). There is a related interrupt enable register bit of **LCD_CPR_IE**.

- **BLKONF**

LCD blinking on interrupt flag (**LCD_BLKONF**). There is a related interrupt enable register bit of **LCD_BLKON_IE**. This bit is set at the start of LCD frame which is the first frame to switch display on during blinking period.

- **BLKOFF**

LCD blinking off interrupt flag (**LCD_BLKOFF**). There is a related interrupt enable register bit of **LCD_BLKOFF_IE**. This bit is set at the start of LCD frame which is the first frame to switch display off during blinking period.

❖ Hardware Statue

The following status bits are read only. These bits are set by hardware and cleared by hardware.

- **DEF**

LCD display enabled status (**LCD_DEF**). It indicates the real LCD display status. It will be active at start of first frame and be inactive at the end of the last displayed frame.

- **PRDYF**

LCD internal power source ready status (**LCD_PRDYF**). When this bit is set, it indicated the power source is ready to output voltage.

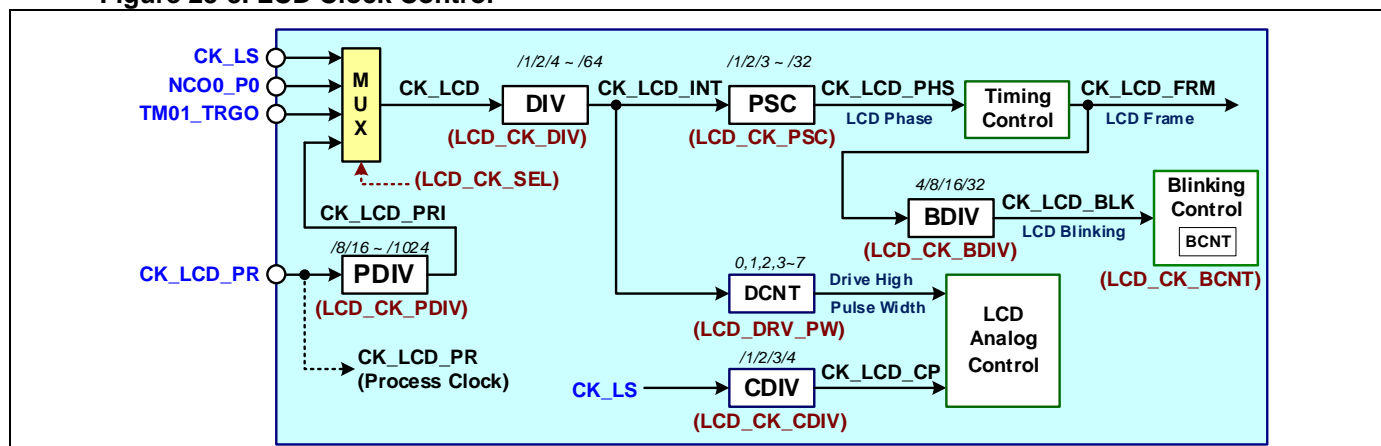
25.8. LCD Fundamental Control

25.8.1. LCD Clock

User can select the module clock source of **CK_LCD** from the clock of **CK_LCD_PRI** and **NCO_P0**, the timer trigger output signal of **TM01_TRGO** or internal ILRCO clock of **CK_ILRCO** by setting **LCD_CK_SEL** register. The clock of **CK_LCD_PRI** is outputted by a clock divider from the module process clock of **CK_LCD_PR**. The clock divider can divide the module process clock by 8/16/32/64/ ~1024 in **LCD_CK_PDIV** register.

The following diagram is showing the LCD clock control block.

Figure 25-5. LCD Clock Control



The LCD module is able to generate the internal clock of **CK_LCD_INT** by a clock divider from the module clock of **CK_LCD** as the LCD clock source for LCD display timing. The clock divider can divide the **CK_LCD** clock by 1/2/4/8/ ~64 in **LCD_CK_DIV** register.

The module provides one clock prescaler to generate the LCD phase clock of **CK_LCD_PHS**. The clock prescaler can divide the input clock by 1/2/3/4/ ~32 in **LCD_CK_PSC** register and the register value must be set > 0. The module is built in a LCD timing controller to output the LCD frame clock **CK_LCD_FRM** for LCD display. Also this LCD frame clock can input as LCD Blinking controller input clock through a clock divider. The clock divider can divide the LCD frame clock by 4/8/16/32 in **LCD_CK_BDIV** register.

The module is built the internal charge pump circuit and used the **CK_LS** clock as timing control clock for the charge pump circuit through a clock divider. The clock divider can divide the **CK_LS** clock by 1/2/3/4 in **LCD_CK_CDIV** register.

User can calculate related LCD clock frequency by following formulas.

$CK_LCD_PHS = \frac{CK_LCD}{DIV * PSC} = \frac{CK_LCD_INT}{PSC}$	DIV= {1,2,4,8 ~ 64} by setting LCD_CK_DIV PSC= {1,2,3,4 ~ 32} by setting LCD_CK_PSC
$CK_LCD_FRM = \frac{CK_LCD_PHS}{COMs * 2} = \frac{CK_LCD_PHS * DUTY}{2}$	COMs: LCD COM number = 1/DUTY DUTY= {1,1/2 ~ 1/8} by setting LCD_DUTY
$CK_LCD_BLK = \frac{CK_LCD_FRM}{BDIV}$	BDIV= {4, 8, 16, 32} by setting LCD_CK_BDIV
$LCD \text{ Blinking Frequency} = \frac{CK_LCD_BLK}{(LCD_CK_BCNT+1)}$	
$CK_LCD_CP = \frac{CK_LCD_INT}{CDIV}$	CDIV= {1,2,4,8~ 128} by setting LCD_CK_CDIV
$Drive \text{ High Pulse Width} = CK_LCD_INT * DPW$	DPW= {0,1,2,3 ~ 7} by setting LCD_DRV_PW

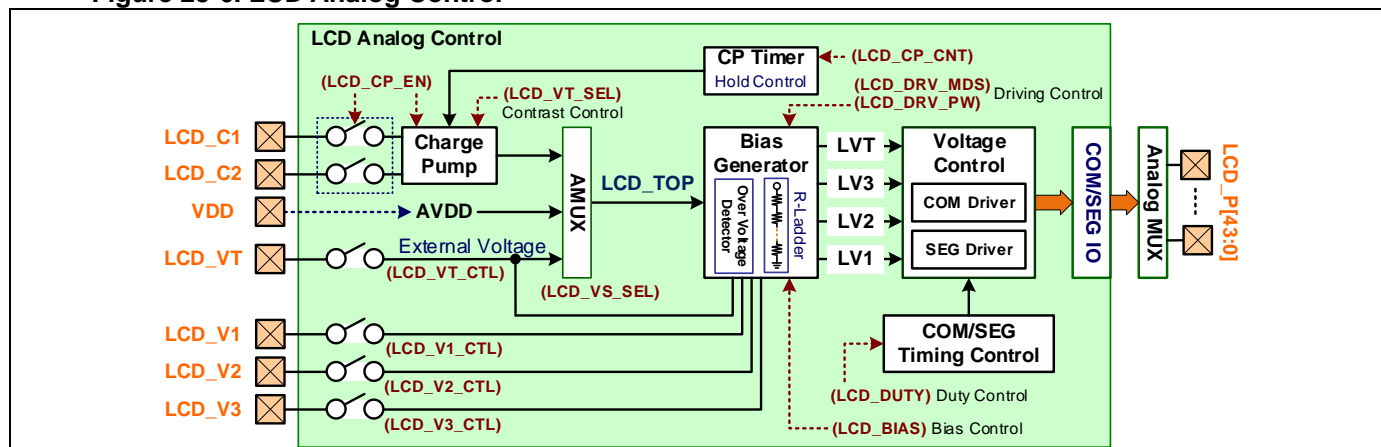
25.8.2. LCD Analog Control

The LCD module is built-in charge pump, bias generator and output voltage drivers. The output voltage drivers can directly drive the external monochrome passive LCD glass.

The IO mode of each using LCD bias power rail or charge pump capacitor pins of **LCD_VT**, **LCD_V1**, **LCD_V2**, **LCD_V3**, **LCD_C1** and **LCD_C2** must set to analog IO mode. Please refer the section of “[IO Mode](#)” in GPIO chapter for more detail descriptions of IO mode configuration.

The following diagram is showing LCD analog control block.

Figure 25-6. LCD Analog Control



● LCD Power Source

User can select the LCD power source as the operation voltage of bias generator from internal AVDD (through **VDD**), external voltage from **LCD_VT** pin or internal charge pump by setting **LCD_VS_SEL** register.

When user selects external voltage from **LCD_VT** pin or internal charge pump, user must enable LCD over voltage detection function in **LCD_OVD_DIS** register if **LCD_TOP** voltage is larger than **VDD** voltage.

● LCD BIAS

The bias generator supports static, 1/2, 1/3, 1/4 bias voltage. User can configure the LCD bias by setting **LCD_BIAS** register. The **LCD_BIAS** register must be initiated before the **LCD_CP_EN** bit is enabled. There are four pins of **LCD_VT**, **LCD_V1**, **LCD_V2** and **LCD_V3** can be independent configuration to connect internal bias power rail to external capacitor for LCD bias voltage stability. User can configure these pins by setting independently **LCD_VT_CTL**, **LCD_V1_CTL**, **LCD_V2_CTL** and **LCD_V3_CTL** registers.

The following table is showing register settings of LCD power rail pin control.

Table 25-2. LCD Power Rail Pin Control

Conditions	Bias	LCD Power Rail Pins	LCD Register			
			VT_CTL	V1_CTL	V2_CTL	V3_CTL
Internal R-Ladder without external Capacitor	x		B	0	0	0
1. Internal Charge Pump	Static	LCD_VT	1	0	0	0
2. Internal R-Ladder with External Capacitor	1/2 Bias	LCD_VT, LCD_V1	1	1	0	0
	1/3 Bias	LCD_VT, LCD_V2, LCD_V1	1	1	1	0
3. External R-Ladder	1/4 Bias	LCD_VT, LCD_V3, LCD_V2, LCD_V1	1	1	1	1

<Sign> x: don't care, B: =1 if select External Power

The module can support to use internal resistor ladder or external resistor ladder to generate bias voltage. User can select internal resistor ladder or external resistor ladder by setting **LCD_RL_SEL** register.

Also the module provide high drive strength function and select the driving mode by setting **LCD_DRV_MDS** register. When this register is set 'High', the drive strength is always forced to drive high. When this register is set 'Normal', user can configure the drive high pulse duration in terms of **CK_LCD_INT** pulses by setting **LCD_DRV_PW** register.

User can calculate drive high strength pulse width by following formula.

$$\text{Drive High Pulse Width} = \text{CK_LCD_INT} * \text{DPW} \quad \text{DPW} = \{0, 1, 2, 3 \sim 7\} \text{ by setting } \text{LCD_DRV_PW}$$

● Internal Charge Pump

The LCD module is built-in charge pump circuit. When the internal charge pump is used, user needs to enable the charge pump circuit in **LCD_CP_EN** register. When this register is enabled, the GPIO **PD0** and **PD1** will be forced to switch to do as **LCD_C1** and **LCD_C2** connections. User must set the IO mode to analog IO for these two pins. On application, a suggested 1nF capacitor must be connected between the **LCD_C1** and **LCD_C2** pins.

When LCD power is selected internal charge pump, user can set the bias generator operation top voltage level for LCD contrast control by register setting in **LCD_VT_SEL** register. Refer the related device data sheet for more information about the voltage level value.

The following table is showing register settings of LCD power source and R-Ladder control mode.

Table 25-3. LCD Power Source and R-Ladder Control Mode

Bias Control Mode	R-Ladder	LCD Register			
		VS_SEL	RL_SEL	VT_SEL	OVD_DIS
LCD Power with Internal Charge Pump	Internal	3	0	Valid	0 (Enable)
LCD Power with Internal AVDD	Internal	1	0	x	1 (Disable)
	External	1	1	x	1 (Disable)
LCD Power with External Power	Internal	2	0	x	A
	External	2	1	x	A
Off	x	0	x	x	1 (Disable)

<Sign> x: don't care, Valid: register setting is valid, A: =0 if LCD_TOP > VDD

25.8.3. LCD Power Source and Pin Connections

The LCD operation power voltage source can be from follows by setting **LCD_VS_SEL** register.

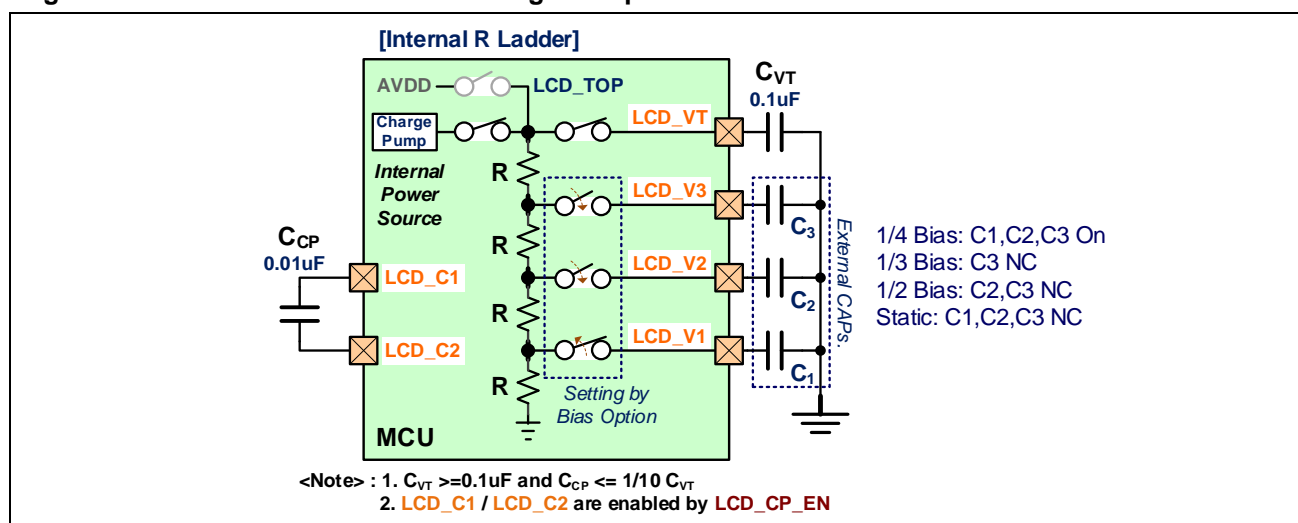
- (1) Internal charge pump power
- (2) Internal analog power from **VDD** pin
- (3) External power from **LCD_VT** pin

❖ LCD Bias with Internal Charge Pump

When **LCD_VS_SEL** is selected internal charge pump, user needs connect internal bias power rail to external capacitor through **LCD_VT**, **LCD_V1**, **LCD_V2** and **LCD_V3** pins for selected bias voltage by setting independently **LCD_VT_CTL**, **LCD_V1_CTL**, **LCD_V2_CTL** and **LCD_V3_CTL** registers. These external capacitors are suggested to use 0.1uF capacitor. For internal charge pump using, a suggested 0.01uF capacitor must be connected between the **LCD_C1** and **LCD_C2** pins.

The following diagram is showing the application connection of LCD Bias with internal Charge Pump.

Figure 25-7. LCD Bias with Internal Charge Pump

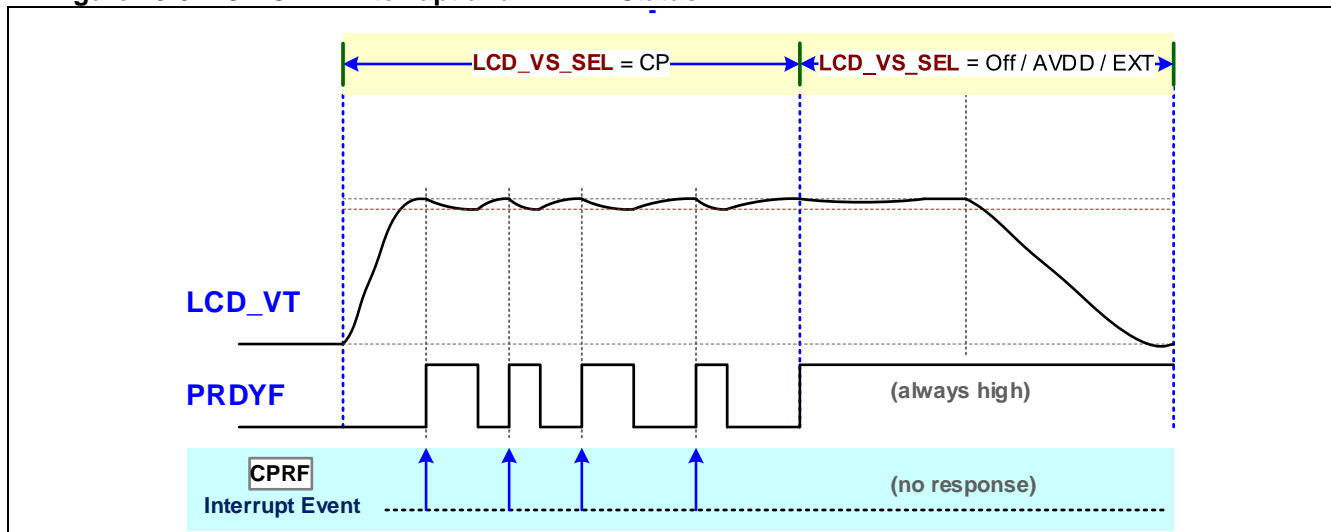


For internal charge pump using, there are one hardware status (**LCD_PRDYF**) and one interrupt flag (**LCD_CPRF**) to indicate whether the power source is ready. User can directly read **LCD_PRDYF** status bit to

check the voltage stable of power source after the first time switch internal charge pump by setting **LCD_VS_SEL** register or wakeup from power down modes.

The following diagram is showing the internal charge pump power voltage vs **LCD_PRDYF** and **LCD_CPRF**.

Figure 25-8. LCD CPRF Interrupt and PRDYF Status

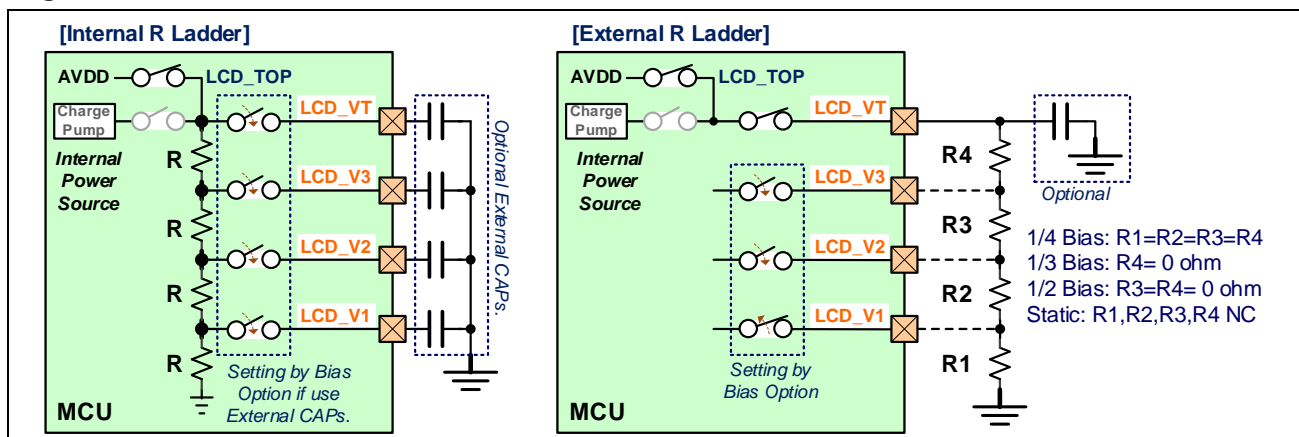


❖ LCD Bias with Internal AVDD

When **LCD_VS_SEL** is selected internal AVDD, user can optionally connect internal bias power rail to external capacitor through **LCD_VT**, **LCD_V1**, **LCD_V2** and **LCD_V3** pins for selected bias voltage by setting independently **LCD_VT_CTL**, **LCD_V1_CTL**, **LCD_V2_CTL** and **LCD_V3_CTL** registers. User can select internal resistor ladder or external resistor ladder to generate bias voltage by setting **LCD_RL_SEL** register.

The following diagram is showing the application connection of LCD Bias with internal AVDD.

Figure 25-9. LCD Bias with Internal AVDD

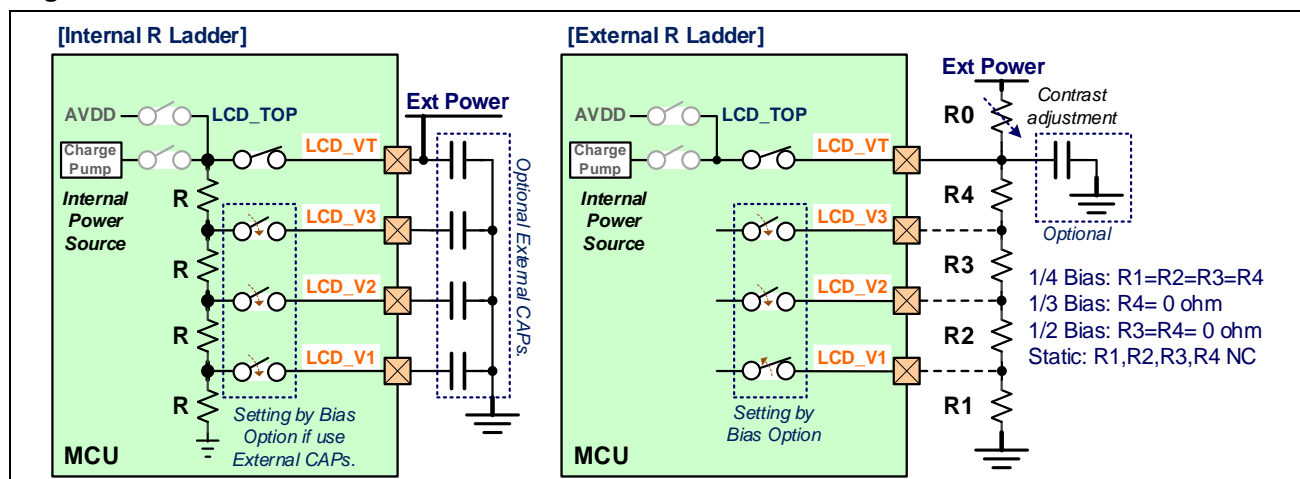


❖ LCD Bias with External Power

When **LCD_VS_SEL** is selected external power voltage, user can optionally connect internal bias power rail to external capacitor through **LCD_VT**, **LCD_V1**, **LCD_V2** and **LCD_V3** pins for selected bias voltage by setting independently **LCD_VT_CTL**, **LCD_V1_CTL**, **LCD_V2_CTL** and **LCD_V3_CTL** registers. User can select internal resistor ladder or external resistor ladder to generate bias voltage by setting **LCD_RL_SEL** register.

The following diagram is showing the application connection of LCD Bias with external power.

Figure 25-10. LCD Bias with External Power

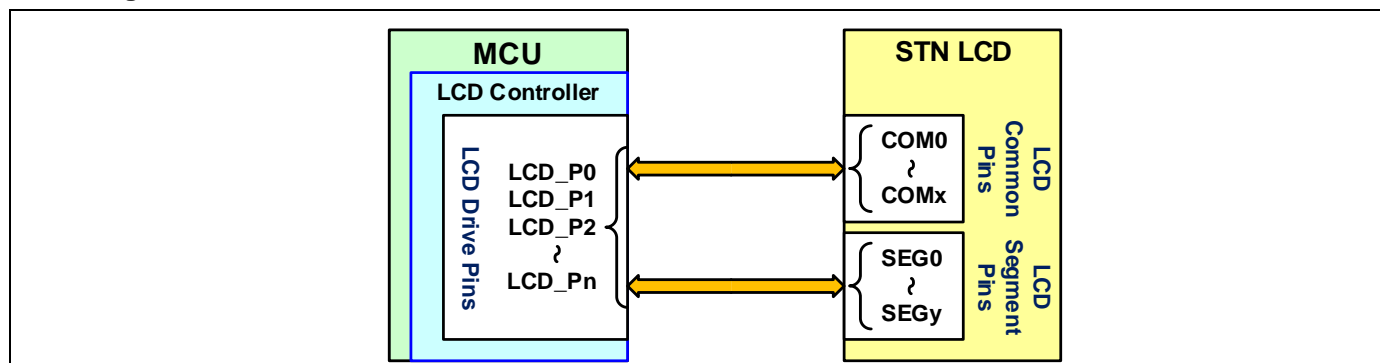


25.8.4. LCD COM and SEG Control

This module provides maximum 44 LCD drive output of **LCD_P[0..43]** to external monochrome passive LCD glass. Each one can configure as LCD common or segment line. The module can configure these LCD drive output for maximum 8 common (COM) lines and 40 segment (SEG) lines.

The following diagram is showing the LCD COM/SEG connection.

Figure 25-11. LCD COM/SEG Connection



❖ LCD Off Output State

The module can select all the LCD drive output state of **LCD_Pn** to tristate or output low by setting **LCD_OFF_CTL** register before the module was enabled in **LCD_EN** register.

Before user sets some of the IO pins to do as LCD common and segment, user can plan the output state of all these planned LCD common and segment lines for application request. When the **LCD_OFF_CTL** bit is selected 'HiZ', the **LCD_Pn** outputs are set to tristate. When the **LCD_OFF_CTL** bit selects 'Low', the **LCD_Pn** outputs are set to low level. (n= LCD_Pn pin index number)

❖ LCD Drive IO Mode

For LCD drive application, the IO mode of each using LCD drive pins must set to analog IO mode. Please refer the section of “[IO Mode](#)” in GPIO chapter for more detail descriptions of IO mode configuration.

When some of the LCD drive pins are not using for LCD drive application, they can set the IO mode to push-pull or open-drain for other IO output signals or digital input or open-drain for other IO input signals.

❖ LCD COM and SEG Configure

At first, user needs to select which LCD drive pins for using on LCD application or request of external monochrome passive LCD glass. These using LCD drive pins must set the pin alternate function to **LCD_Pn** in **Px_AFSz** register for LCD drive application. Please refer the section of “[Alternate Function Select](#)” in GPIO chapter and the section of “Pin Alternate Functions Selected Table” in Pin Description chapter of the chip Data Sheet for more detail descriptions of IO AFS configuration. (Px = GPIO port {PA, PB, PC, PD, PE}, z= GPIO pin index number, n= LCD_Pn pin index number)

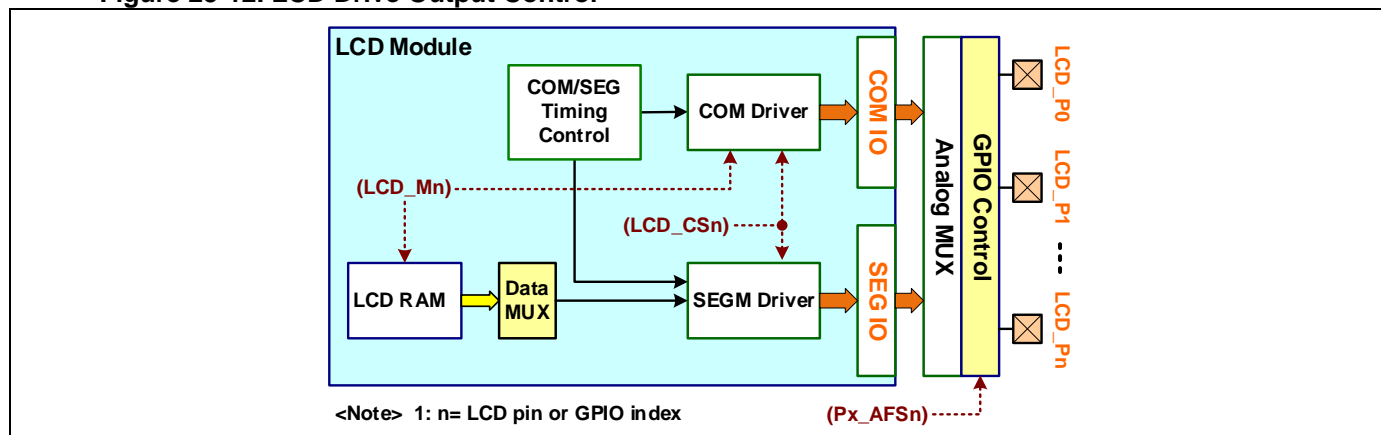
The second, user needs to select which using LCD drive pin do as common or segment line by setting

LCD_CS_n register for request of external monochrome passive LCD glass.

The third, user needs to configure the common line index for each of using LCD drive common pins those are selected as common line by setting **LCD_CS_n** register. By the common line definitions of external monochrome passive LCD glass, user can set **LCD_M_n** register to map the LCD common line index definitions for each of using LCD drive common pins. The least significant bit of each **LCD_M_n** register is mapping to COM0 and most significant bit is mapping to COM7. When the corresponding bit is set 1, it indicates the pin is enabled as the corresponding COM line.

The following diagram is showing the LCD Drive Output control block.

Figure 25-12. LCD Drive Output Control



25.9. LCD Drive Timing

25.9.1. LCD Clock and Frame

The module is implemented a COM/SEG timing controller. It can support Static, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7 and 1/8 duty by using 1, 2, 3, 4, 5, 6, 7 and 8 common lines. User can configure the LCD duty timing by setting **LCD_DUTY** register. There are two type LCD frame timing of Type-A or Type-B for request of external monochrome passive LCD glass and user set in **LCD_FRM_MDS** register.

The COM/SEG timing controller can provide programmable LCD dead time control between frames or duty phases. Also it can support LCD blinking display and LCD COM/SEG waveform phase inverse functions.

Before COM/SEG timing controller is operation, user must configure the module clock settings, LCD common/segment line assignment, LCD power source and analog control configuration. Please refer the section of "[LCD Fundamental Control](#)" for more information. For example, user needs to configure the LCD bias by setting **LCD_BIAS** register. The following table is showing a suggestion of LCD Bias and Duty Configure.

Table 25-4. LCD Bias and Duty Configure Suggestion

	1/2 Duty	1/3 Duty	1/4 Duty	1/5 Duty	1/6 Duty	1/7 Duty	1/8 Duty	LCD Power Rail Pins
Static	-	-	-	-	-	-	-	LCD_VT
1/2 Bias	V	V	-	-	-	-	-	LCD_VT, LCD_V1
1/3 Bias	-	V	V	V	V	V	V	LCD_VT, LCD_V2, LCD_V1
1/4 Bias	-	-	-	-	-	V	V	LCD_VT, LCD_V3, LCD_V2, LCD_V1

<Sign> "-": Not suggested

User needs to configure the LCD duty timing for actual common lines and plan the LCD display frame rate for application request. User can calculate LCD phase and frame frequency by following formulas.

$$CK_LCD_PHS = \frac{CK_LCD}{DIV * PSC} = \frac{CK_LCD_INT}{PSC}$$

DIV= {1,2,4,8 ~ 64} by setting **LCD_CK_DIV**
PSC= {1,2,3,4 ~ 32} by setting **LCD_CK_PSC**

$$CK_LCD_FRM = \frac{CK_LCD_PHS}{COMs * 2} = \frac{CK_LCD_PHS * DUTY}{2}$$

COMs: LCD COM number = 1/DUTY
DUTY= {1, 1/2 ~ 1/8} by setting **LCD_DUTY**

The following table is showing an example of LCD frame rate calculation and register settings with fixed 32 KHz clock frequency of **CK_LCD** for different LCD common lines.

Table 25-5. LCD Frame Rate and Clock Configure Examples

CK_LCD	COM	CK_LCD_INT	CK_LCD_PHS	Frame Rate	LCD Register	
KHz	Lines	Hz	Hz	Hz	LCD_CK_DIV	LCD_CK_PSC
32	2	2000.0	117.6	29.4	4	16
32	2	4000.0	235.3	58.8	3	16
32	2	8000.0	470.6	117.6	2	16
32	3	2000.0	181.8	30.3	4	10
32	3	4000.0	363.6	60.6	3	10
32	3	8000.0	727.3	121.2	2	10
32	4	2000.0	250.0	31.3	4	7
32	4	4000.0	500.0	62.5	3	7
32	4	8000.0	1000.0	125.0	2	7
32	5	2000.0	285.7	28.6	4	6
32	5	4000.0	571.4	57.1	3	6
32	5	8000.0	1142.9	114.3	2	6
32	6	2000.0	333.3	27.8	4	5
32	6	4000.0	666.7	55.6	3	5
32	6	8000.0	1333.3	111.1	2	5
32	7	2000.0	400.0	28.6	4	4
32	7	4000.0	800.0	57.1	3	4
32	7	8000.0	1600.0	114.3	2	4
32	8	2000.0	500.0	31.3	4	3
32	8	4000.0	1000.0	62.5	3	3
32	8	8000.0	2000.0	125.0	2	3

<Note> Type B LCD Frame Rate = Even Frame + Odd Frame

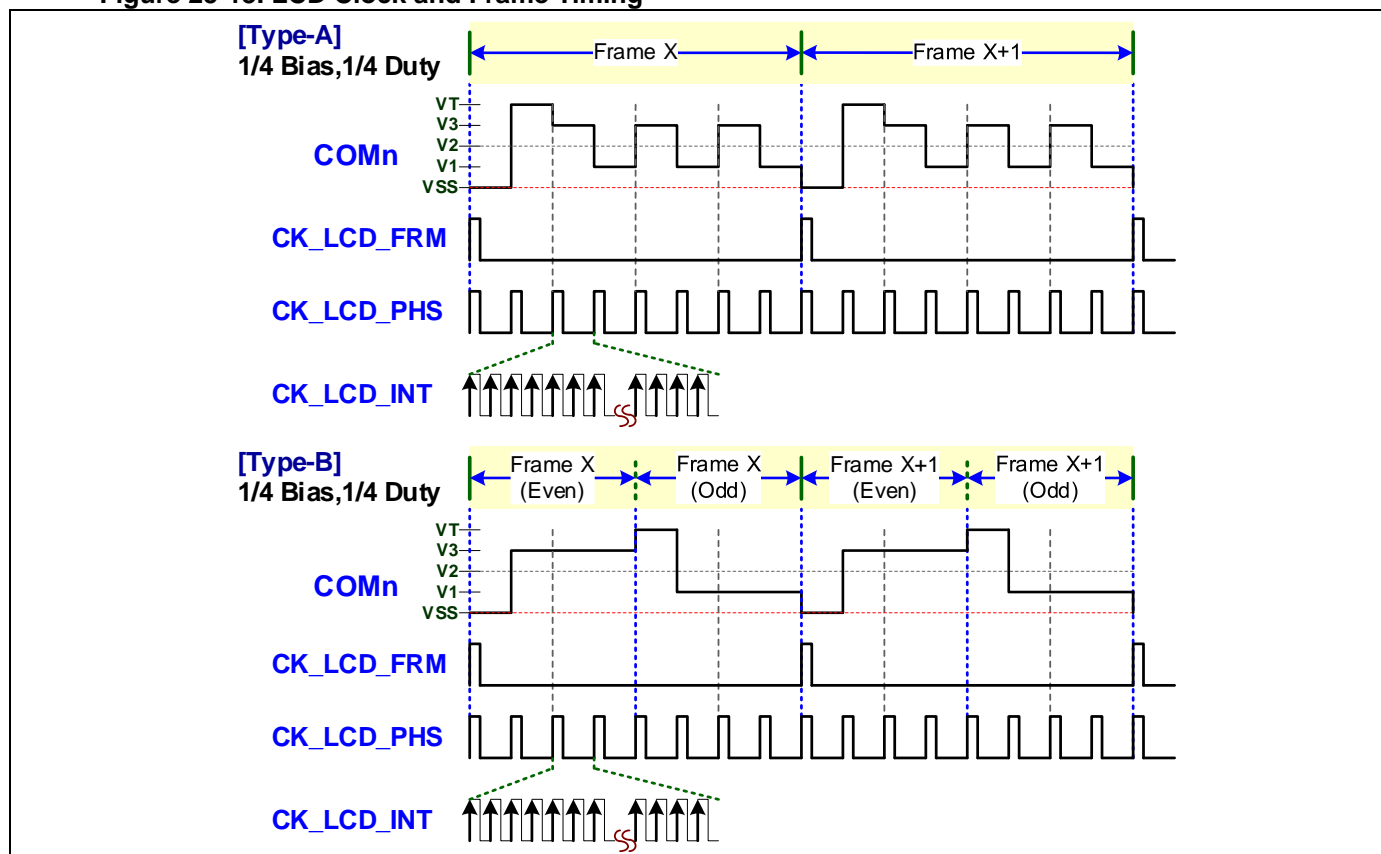
The following table is showing an example of LCD maximum and minimum frame rate for different LCD common lines and different clock frequency of **CK_LCD**.

Table 25-6. LCD Clock Frequency Range Examples

CK_LCD	CK_LCD_INT (Hz)		CK_LCD_PHS (Hz)		Frame Rate (Hz) = CK_LCD_PHS/2/COMs					
KHz	CK_LCD/(1,2,4~64)		CK_LCD_INT/(1,2,3~32)		COMs=2		COMs=4		COMs=8	
	Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.
32	500.0	32000.0	15.6	32000.0	3.9	8000.0	2.0	4000.0	1.0	2000.0
32.768	512.0	32768.0	16.0	32768.0	4.0	8192.0	2.0	4096.0	1.0	2048.0
46.875	732.4	46875.0	22.9	46875.0	5.7	11718.8	2.9	5859.4	1.4	2929.7
93.75	1464.8	93750.0	45.8	93750.0	11.4	23437.5	5.7	11718.8	2.9	5859.4
375	5859.4	375000.0	183.1	375000.0	45.8	93750.0	22.9	46875.0	11.4	23437.5
562.5	8789.1	562500.0	274.7	562500.0	68.7	140625.0	34.3	70312.5	17.2	35156.3
750	11718.8	750000.0	366.2	750000.0	91.6	187500.0	45.8	93750.0	22.9	46875.0
1125	17578.1	1125000.0	549.3	1125000.0	137.3	281250.0	68.7	140625.0	34.3	70312.5

The following diagram is showing an example of the LCD clock and frame timing of “1/4 Bias and 1/4 Duty” for LCD frame timing of Type-A or Type-B.

Figure 25-13. LCD Clock and Frame Timing



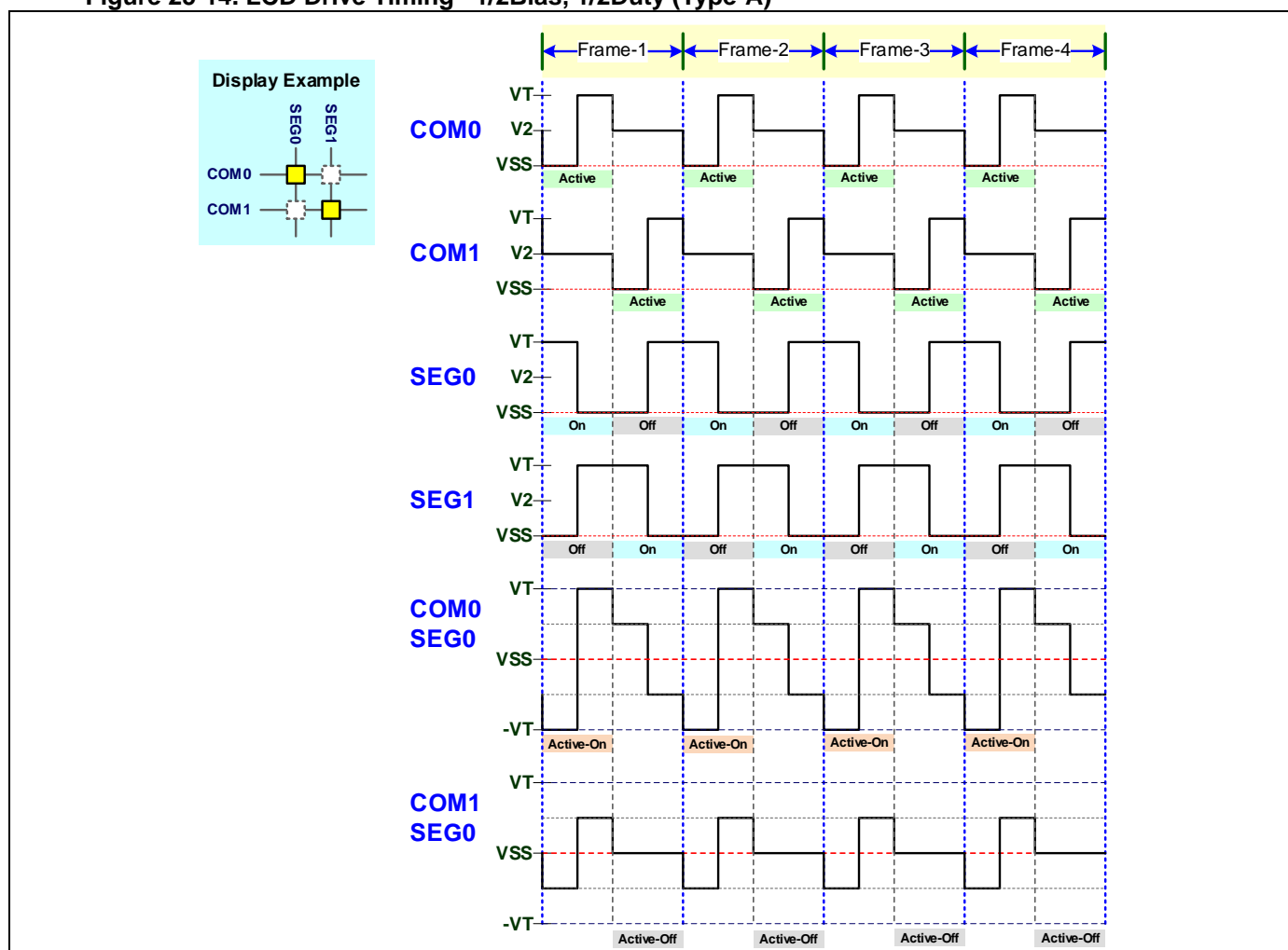
25.9.2. LCD Drive 1/2 Bias

❖ LCD Drive Timing – 1/2Bias, 1/2Duty (Type-A)

The following diagram is showing an example of LCD Type-A drive timing for 1/2Bias and 1/2Duty. It uses 2 common lines and 2 segment lines. The dots of COM0-SEG0 and COM1-SEG1 are displayed on. Other dots are displayed off.

- (1) **LCD_BIAS** = 0x01 (1/2 bias)
- (2) **LCD_DUTY** = 0x01 (1/2 duty)
- (3) **LCD_FRM_MDS** = 0x0 (Type-A)

Figure 25-14. LCD Drive Timing –1/2Bias, 1/2Duty (Type-A)

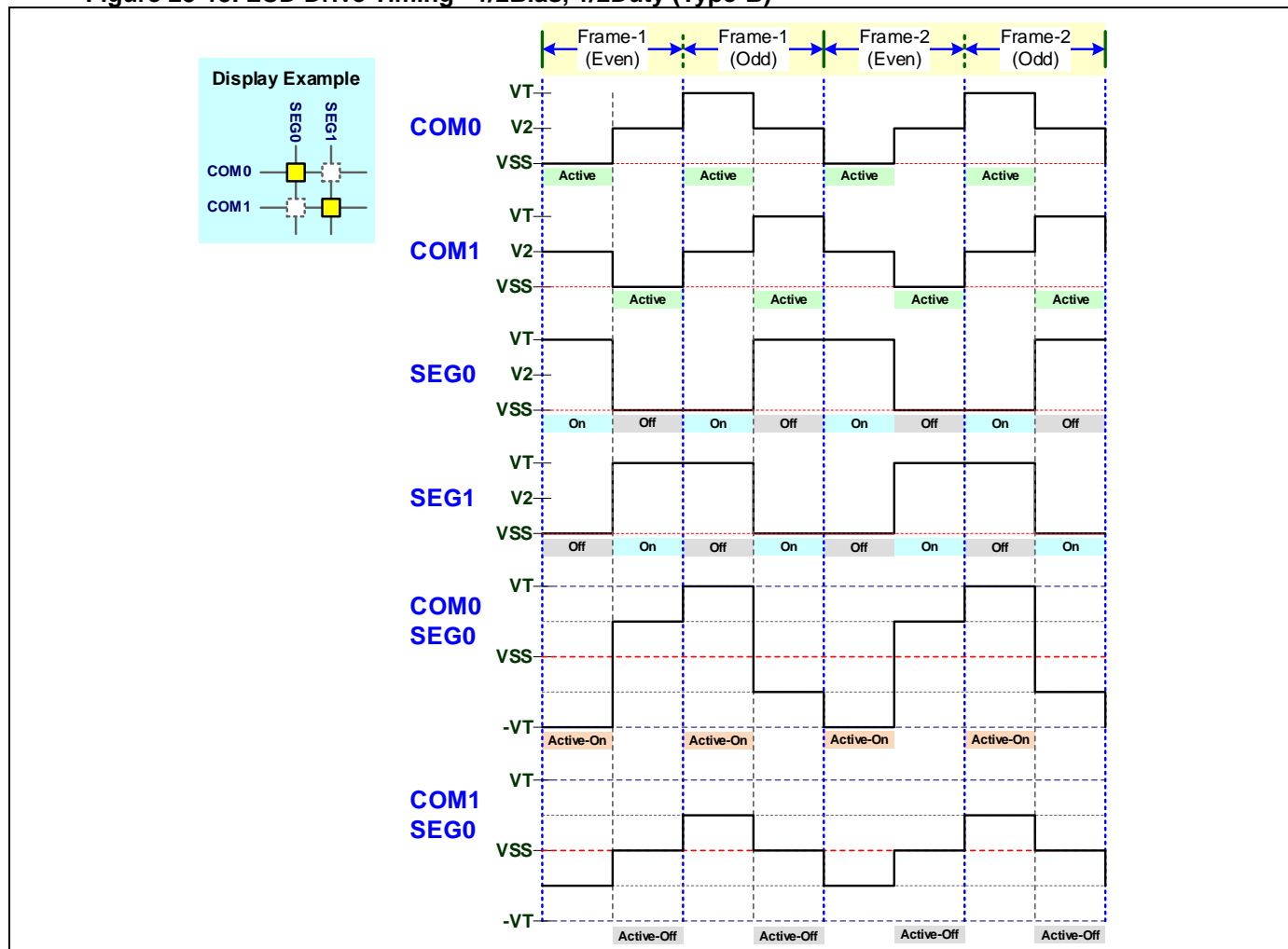


❖ LCD Drive Timing – 1/2Bias, 1/2Duty (Type-B)

The following diagram is showing an example of LCD Type-B drive timing for 1/2Bias and 1/2Duty. It uses 2 common lines and 2 segment lines. The dots of COM0-SEG0 and COM1-SEG1 are displayed on. Other dots are displayed off.

- (1) **LCD_BIAS** = 0x01 (1/2 bias)
- (2) **LCD_DUTY** = 0x01 (1/2 duty)
- (3) **LCD_FRM_MDS** = 0x1 (Type-B)

Figure 25-15. LCD Drive Timing –1/2Bias, 1/2Duty (Type-B)



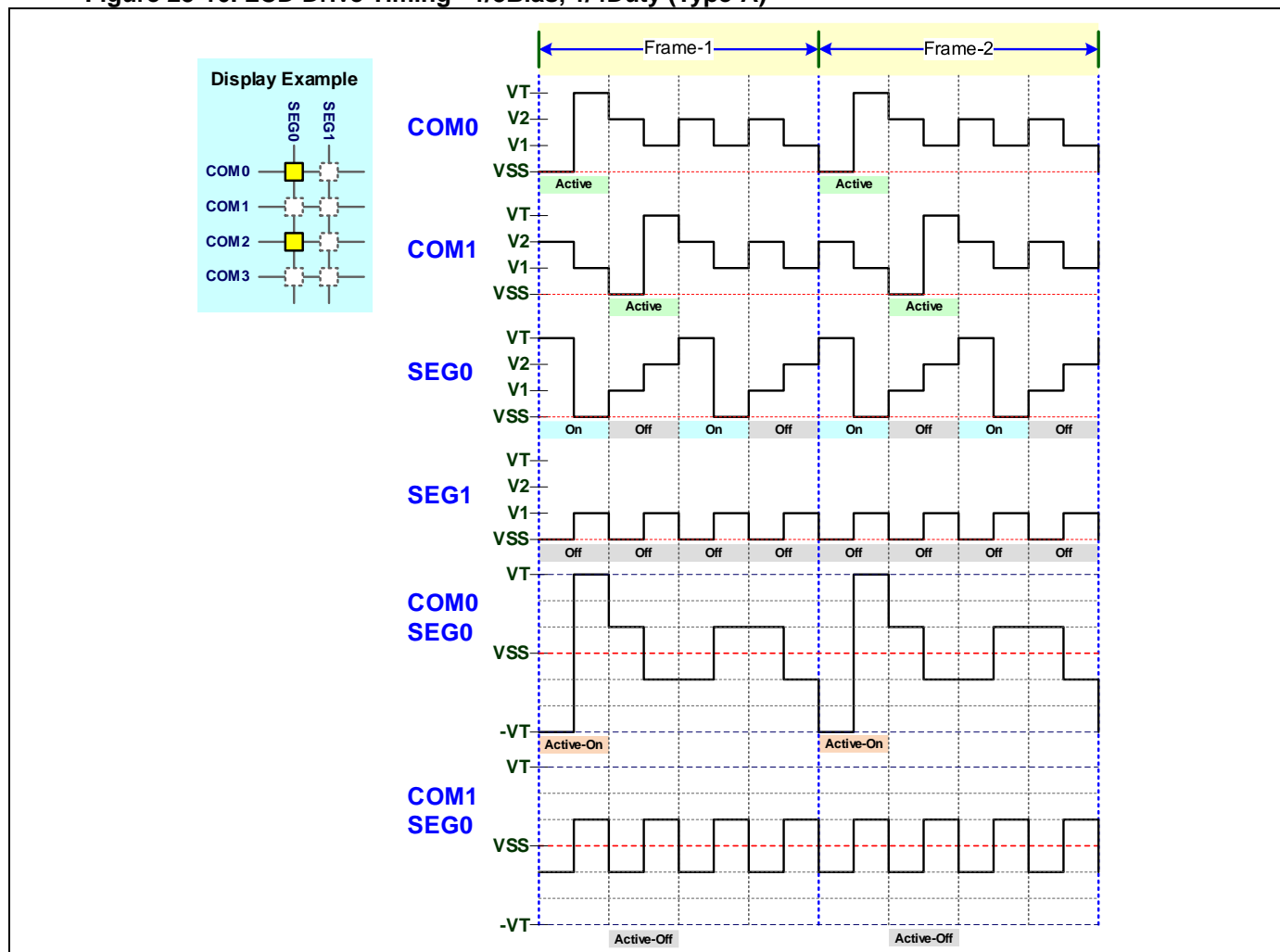
25.9.3. LCD Drive 1/3 Bias

❖ LCD Drive Timing – 1/3Bias, 1/4Duty (Type-A)

The following diagram is showing an example of LCD Type-A drive timing for 1/3Bias and 1/4Duty. It uses 4 common lines and 2 segment lines. The dots of COM0-SEG0 and COM2-SEG0 are displayed on. Other dots are displayed off.

- (1) **LCD_BIAS** = 0x02 (1/3 bias)
- (2) **LCD_DUTY** = 0x03 (1/4 duty)
- (3) **LCD_FRM_MDS** = 0x0 (Type-A)

Figure 25-16. LCD Drive Timing –1/3Bias, 1/4Duty (Type-A)

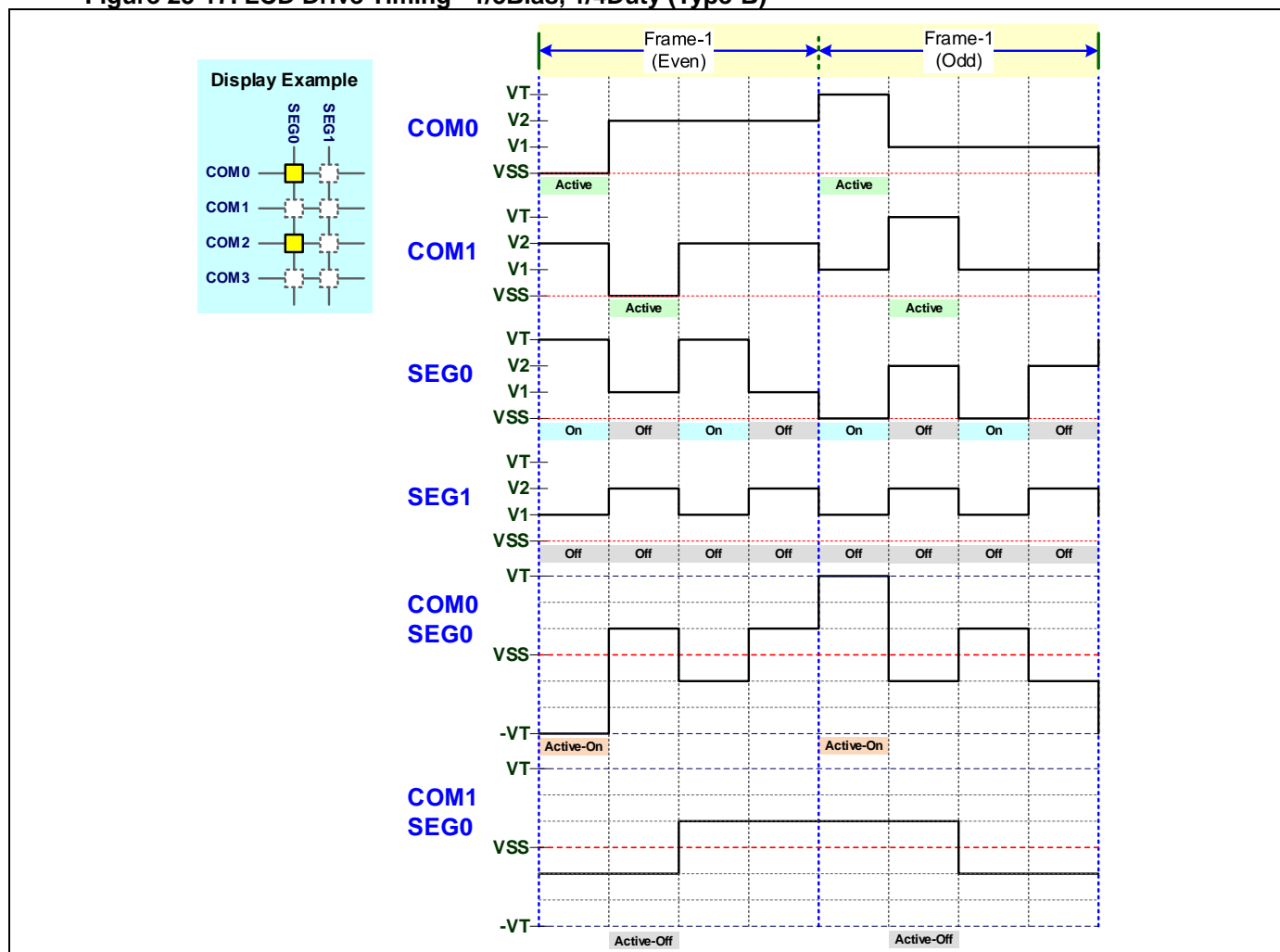


❖ LCD Drive Timing – 1/3Bias, 1/4Duty (Type-B)

The following diagram is showing an example of LCD Type-B drive timing for 1/3Bias and 1/4Duty. It uses 4 common lines and 2 segment lines. The dots of COM0-SEG0 and COM2-SEG0 are displayed on. Other dots are displayed off.

- (1) **LCD_BIAS** = 0x02 (1/3 bias)
- (2) **LCD_DUTY** = 0x03 (1/4 duty)
- (3) **LCD_FRM_MDS** = 0x1 (Type-B)

Figure 25-17. LCD Drive Timing –1/3Bias, 1/4Duty (Type-B)



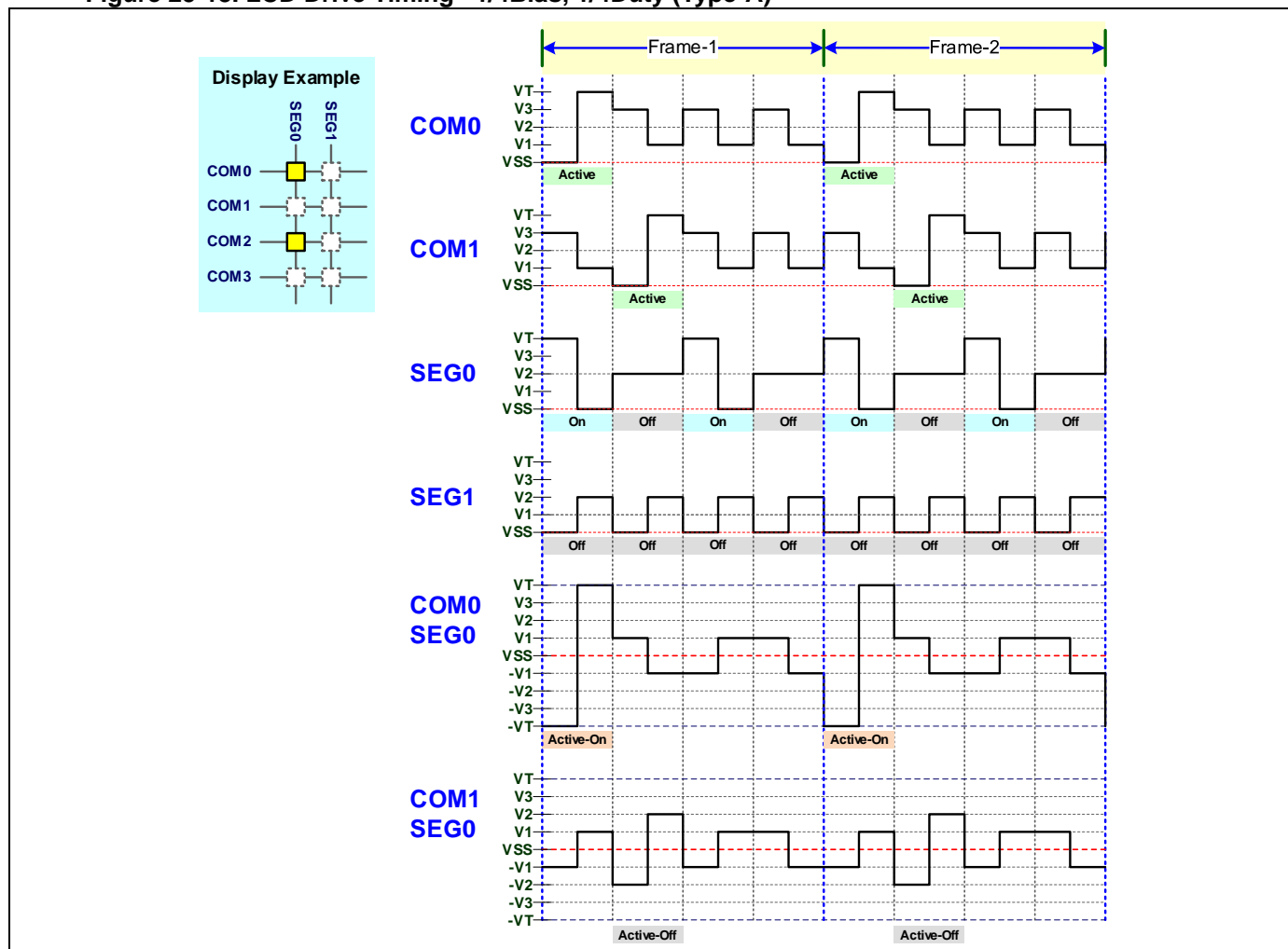
25.9.4. LCD Drive 1/4 Bias

❖ LCD Drive Timing – 1/4Bias, 1/4Duty (Type-A)

The following diagram is showing an example of LCD Type-A drive timing for 1/4Bias and 1/4Duty. It uses 4 common lines and 2 segment lines. The dots of COM0-SEG0 and COM2-SEG0 are displayed on. Other dots are displayed off.

- (1) **LCD_BIAS** = 0x03 (1/4 bias)
- (2) **LCD_DUTY** = 0x03 (1/4 duty)
- (3) **LCD_FRM_MDS** = 0x0 (Type-A)

Figure 25-18. LCD Drive Timing –1/4Bias, 1/4Duty (Type-A)

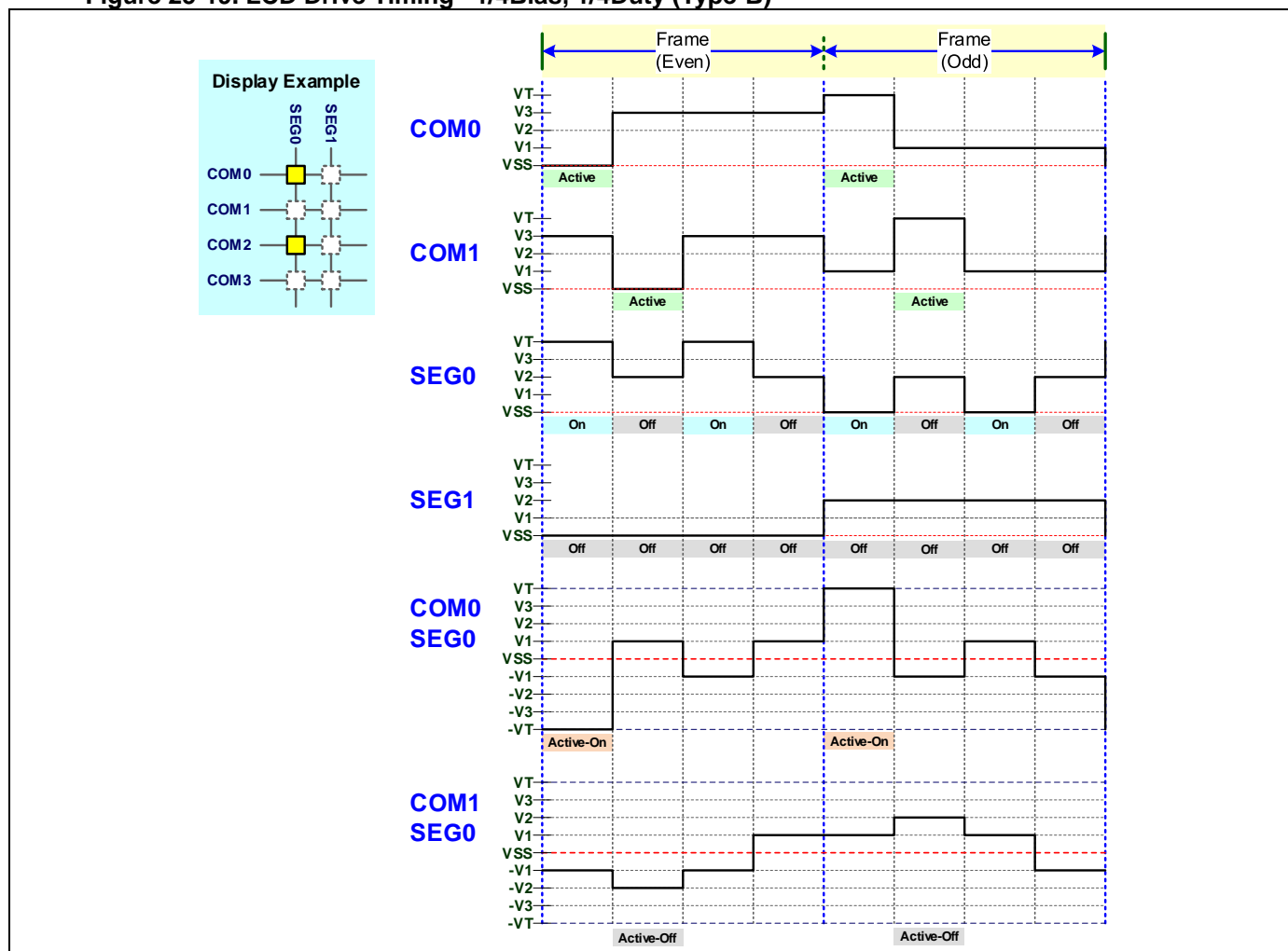


❖ LCD Drive Timing – 1/4Bias, 1/4Duty (Type-B)

The following diagram is showing an example of LCD Type-B drive timing for 1/4Bias and 1/4Duty. It uses 4 common lines and 2 segment lines. The dots of COM0-SEG0 and COM2-SEG0 are displayed on. Other dots are displayed off.

- (1) **LCD_BIAS** = 0x03 (1/4 bias)
- (2) **LCD_DUTY** = 0x03 (1/4 duty)
- (3) **LCD_FRM_MDS** = 0x1 (Type-B)

Figure 25-19. LCD Drive Timing –1/4Bias, 1/4Duty (Type-B)

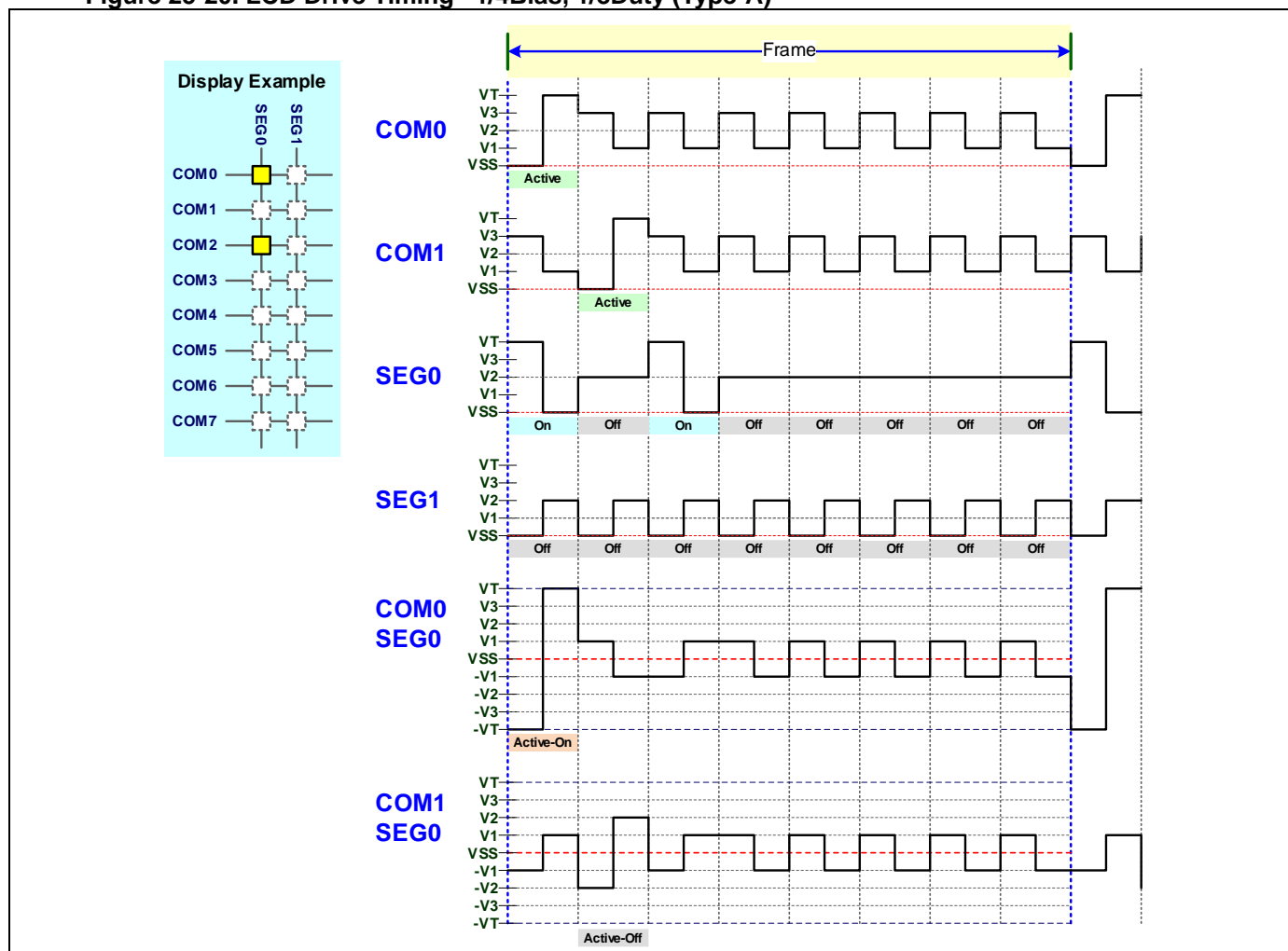


❖ LCD Drive Timing – 1/4Bias, 1/8Duty (Type-A)

The following diagram is showing an example of LCD Type-A drive timing for 1/4Bias and 1/8Duty. It uses 8 common lines and 2 segment lines. The dots of COM0-SEG0 and COM2-SEG0 are displayed on. Other dots are displayed off.

- (1) **LCD_BIAS** = 0x03 (1/4 bias)
- (2) **LCD_DUTY** = 0x07 (1/8 duty)
- (3) **LCD_FRM_MDS** = 0x0 (Type-A)

Figure 25-20. LCD Drive Timing –1/4Bias, 1/8Duty (Type-A)

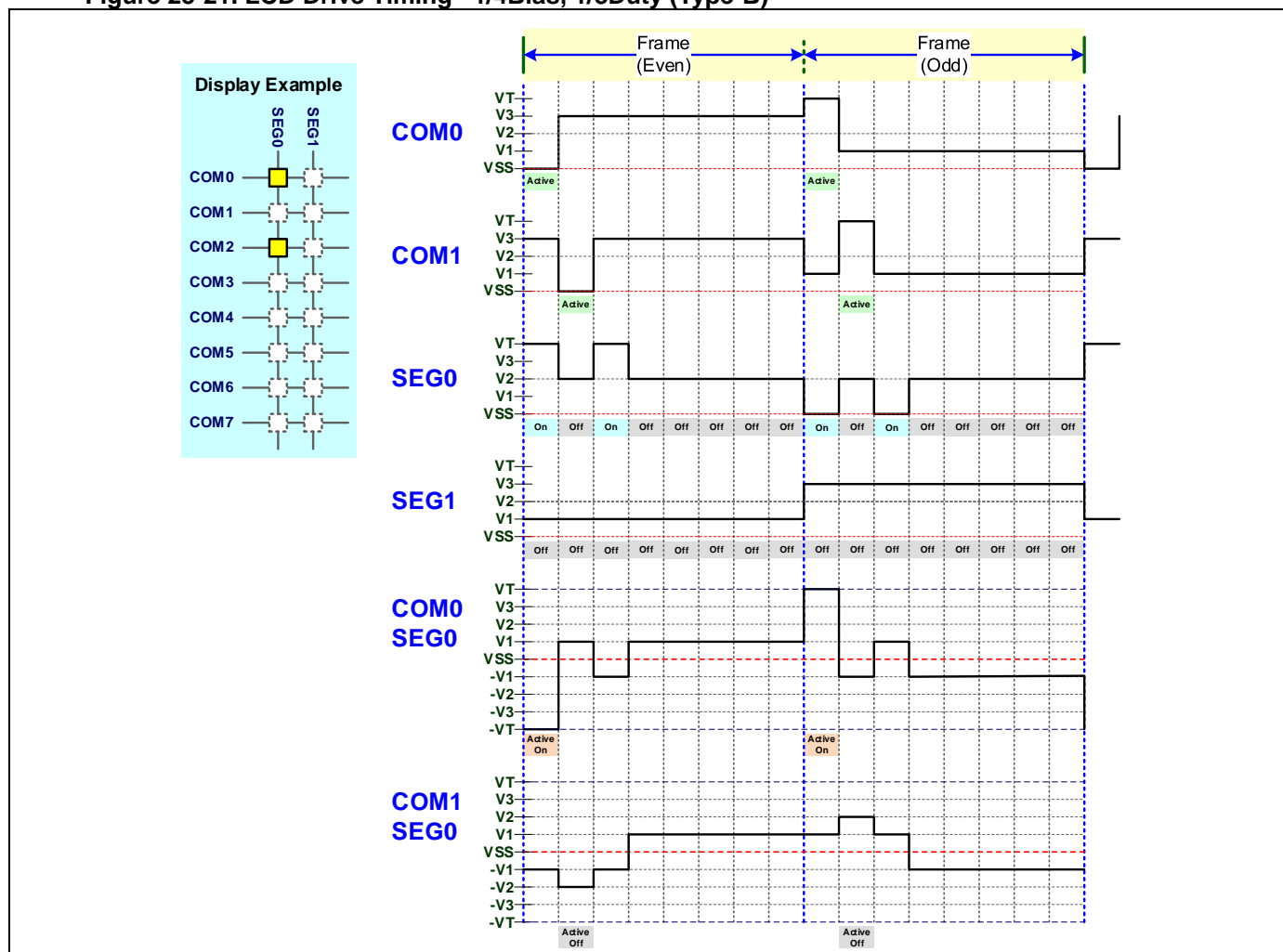


❖ LCD Drive Timing – 1/4Bias, 1/8Duty (Type-B)

The following diagram is showing an example of LCD Type-B drive timing for 1/4Bias and 1/8Duty. It uses 8 common lines and 2 segment lines. The dots of COM0-SEG0 and COM2-SEG0 are displayed on. Other dots are displayed off.

- (1) **LCD_BIAS** = 0x03 (1/4 bias)
- (2) **LCD_DUTY** = 0x07 (1/8 duty)
- (3) **LCD_FRM_MDS** = 0x1 (Type-B)

Figure 25-21. LCD Drive Timing –1/4Bias, 1/8Duty (Type-B)



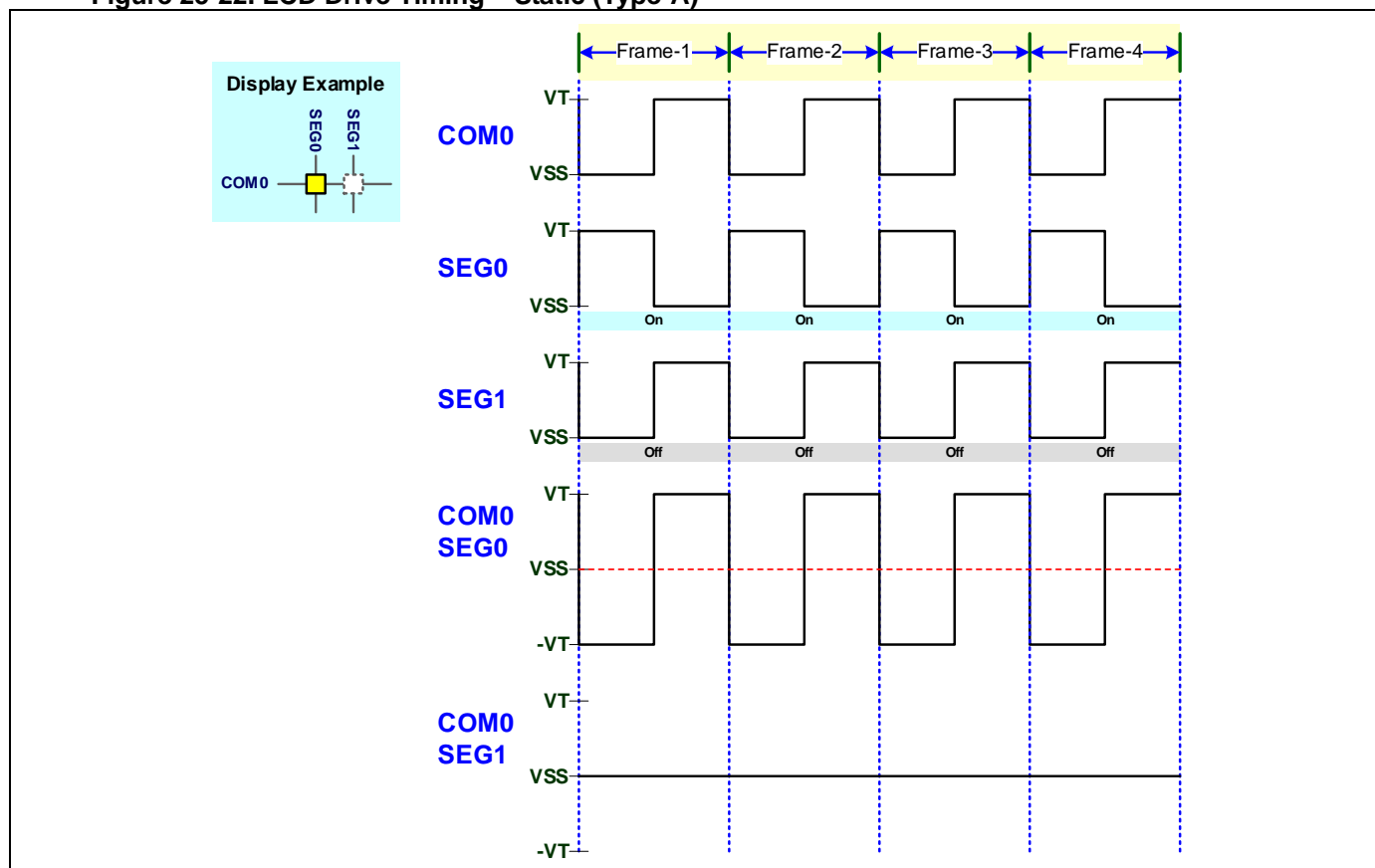
25.9.5. LCD Drive Static

❖ LCD Drive Timing – Static (Type-A)

The following diagram is showing an example of LCD Type-A drive timing for Static Bias. It uses 1 common lines and 2 segment lines. The dot of COM0-SEG0 is displayed on. The dot of COM0-SEG1 is displayed off.

- (1) **LCD_BIAS** = 0x00 (Static bias)
- (2) **LCD_DUTY** = 0x00 (Static duty)
- (3) **LCD_FRM_MDS** = 0x0 (Type-A)

Figure 25-22. LCD Drive Timing – Static (Type-A)

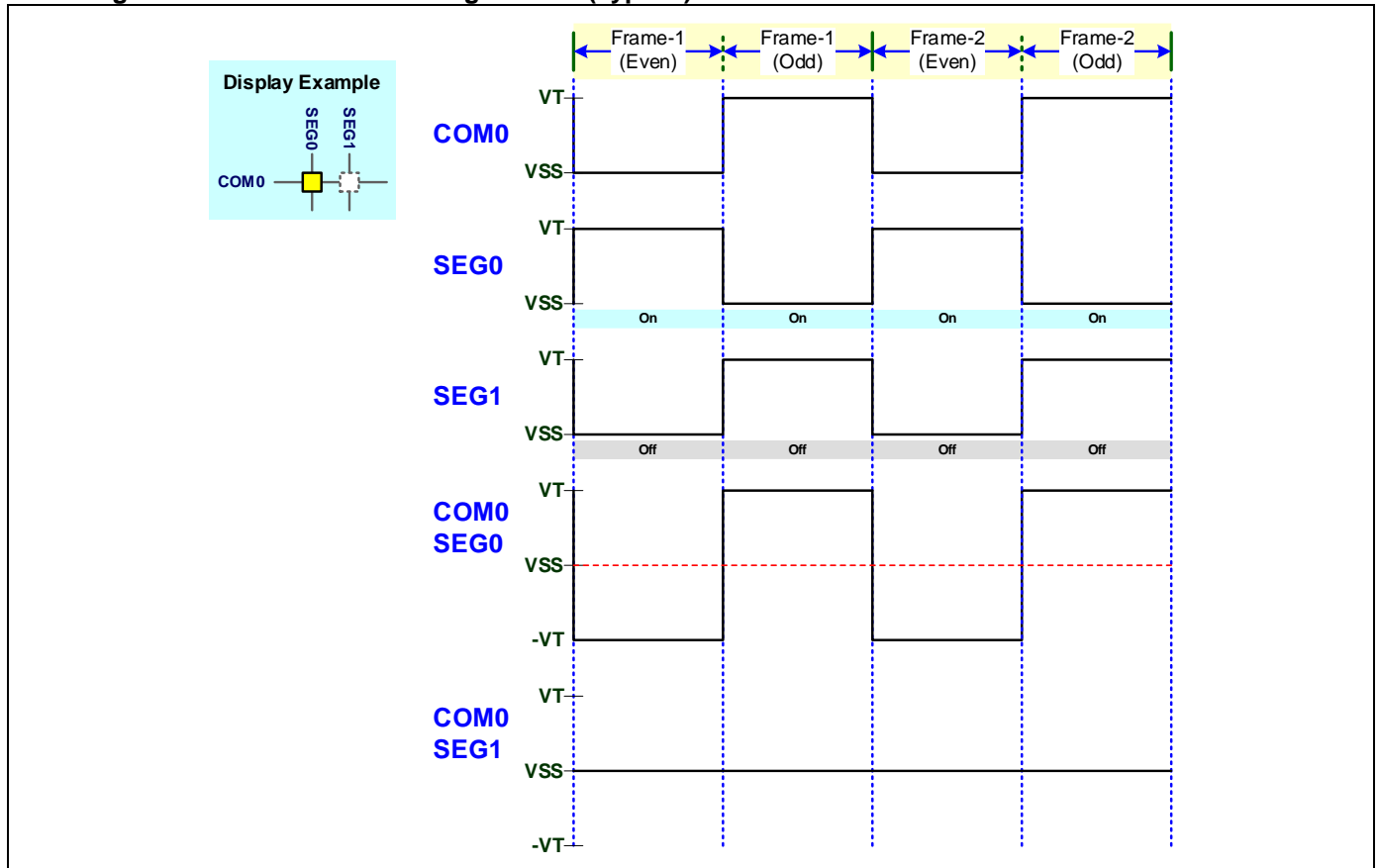


❖ LCD Drive Timing – Static (Type-B)

The following diagram is showing an example of LCD Type-B drive timing for Static Bias. It uses 1 common lines and 2 segment lines. The dot of COM0-SEG0 is displayed on. The dot of COM0-SEG1 is displayed off.

- (1) **LCD_BIAS** = 0x00 (Static bias)
- (2) **LCD_DUTY** = 0x00 (Static duty)
- (3) **LCD_FRM_MDS** = 0x1 (Type-B)

Figure 25-23. LCD Drive Timing – Static (Type-B)



25.9.6. LCD Dead Timing

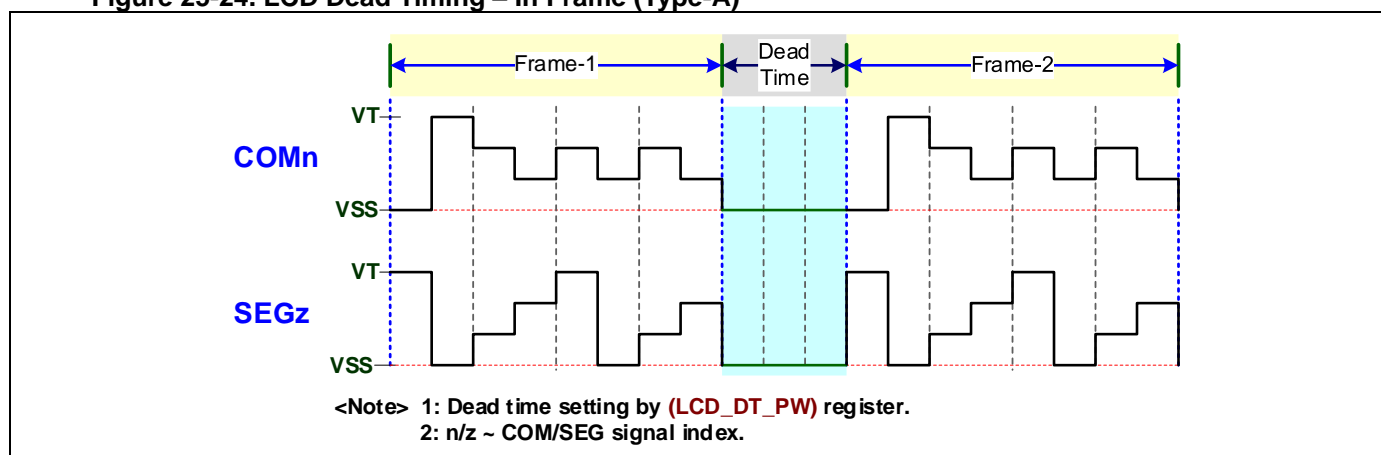
The LCD COM/SEG timing controller supports LCD timing with dead time control between LCD frame periods or phase duty periods by setting **LCD_DT_MDS** register. When user selects Type-B LCD timing, user is only able to set dead time control in LCD frame period. During the dead time, all LCD COM and SEG pins are driven to ground.

Also user can configure LCD output dead time period width in **LCD_DT_PW** register. The dead time period setting unit is one **CK_LCD_PHS** (LCD phase period) for using 'frame period' or one **CK_LCD_INT** period for using 'phase duty period'. During the dead time, all COM and SEG pins are driven to ground.

❖ LCD Dead Timing – In Frame

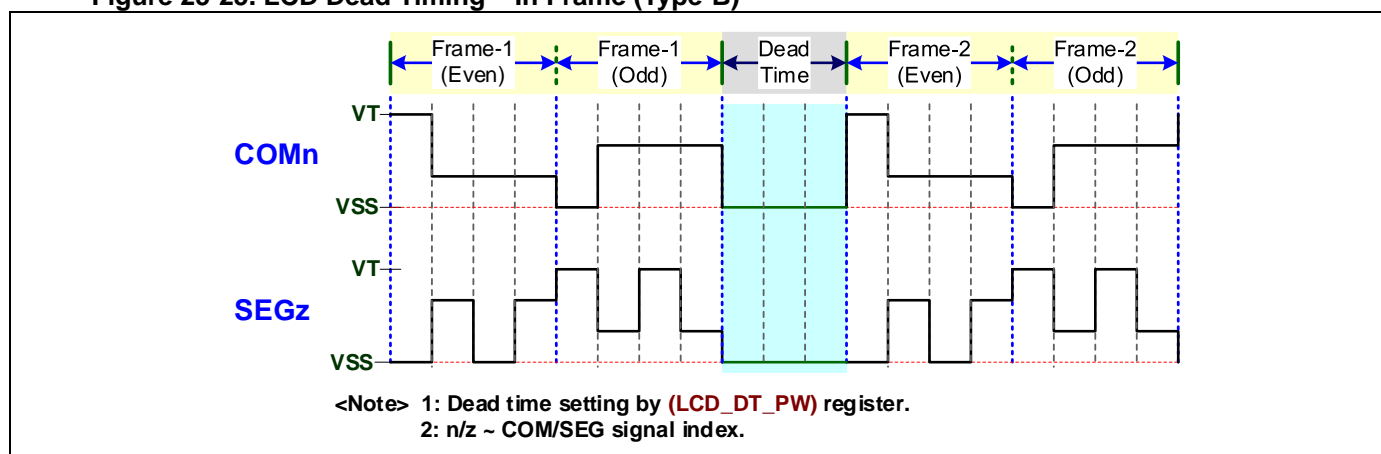
The following diagram is showing an example of LCD Type-A drive timing with dead time control between frame periods by setting three clock times of **CK_LCD_PHS** (LCD phase clock).

Figure 25-24. LCD Dead Timing – In Frame (Type-A)



The following diagram is showing an example of LCD Type-B drive timing with dead time control between frame periods by setting three clock times of **CK_LCD_PHS** (LCD phase clock).

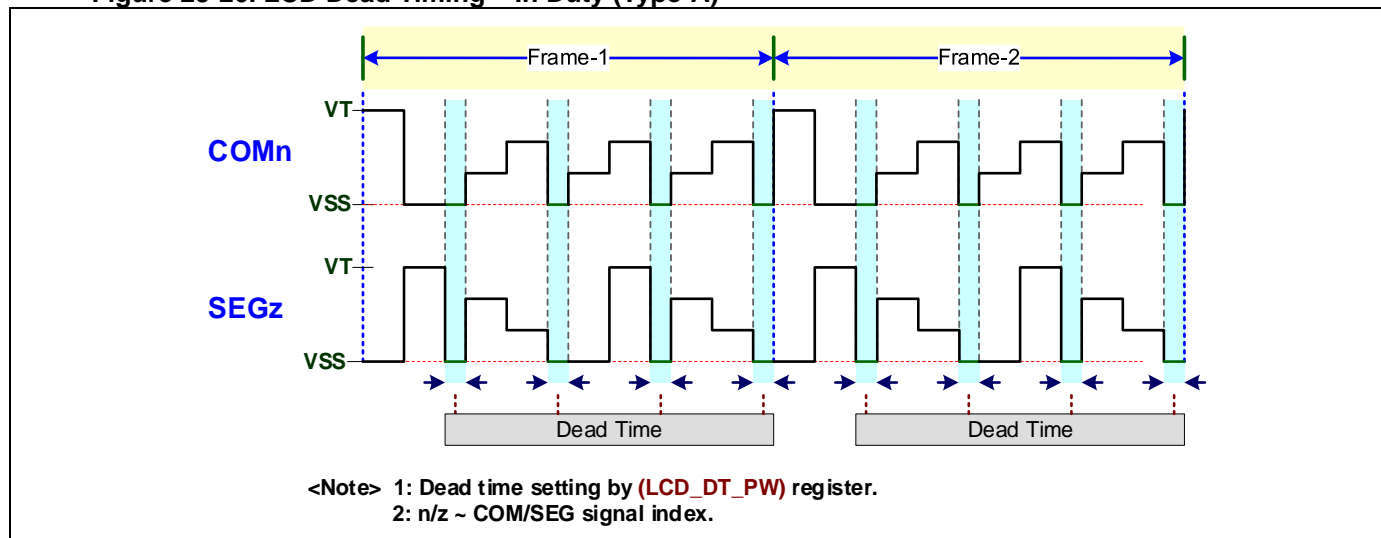
Figure 25-25. LCD Dead Timing – In Frame (Type-B)



❖ LCD Dead Timing – In Duty

The following diagram is showing an example of LCD Type-A drive timing with dead time control between phase duty periods by setting four clock times of **CK_LCD_INT** (LCD internal clock).

Figure 25-26. LCD Dead Timing – In Duty (Type-A)



25.9.7. LCD Blinking and Segment Off

The LCD COM/SEG timing controller supports LCD timing with display blinking and segment lines off functions. User can enable the LCD blinking function by setting **LCD_BLK_MDS** register. Also user can directly switch all of the segment lines to dot off timing by setting **LCD_SEG_OFF** register.

A blinking clock divider is used to divide the LCD frame clock **CK_LCD_FRM** to generate the LCD blinking clock **CK_LCD_BLK**. The clock divider can divide the LCD frame clock by 4/8/16/32 in **LCD_CK_BDIV** register. Please refer the section of “[LCD Clock](#)” for more information about the clock configuration.

There is one LCD frame counter and user can set the maximum frame counter threshold value for blinking function in **LCD_BCNT** register. The frame counter will start at 0 and be automatically increased by 1 at each frame end. When the frame counter reaches blinking maximum frame counter value, it will resets and restarts from 0. And the LCD display will toggle the blinking display between normal display and blank display at the calculated blinking frequency. When user enable the LCD blinking function, the LCD COM/SEG timing controller will switch the LCD display to blank display at the end of the first frame. The first frame is the period which user writes **LCD_BLK_MDS** register to enable LCD blinking. The second frame will be switched the LCD display to normal display. The next frame will be switched to blank display again.

User can calculate LCD blinking frequency by following formulas.

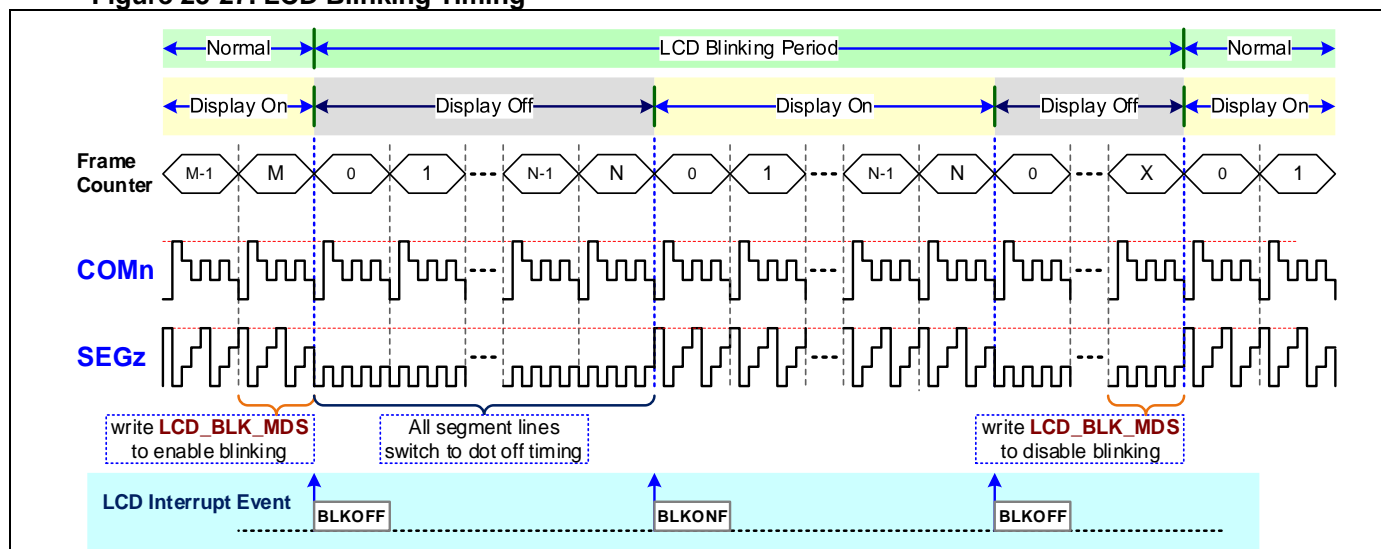
$$CK_LCD_BLK = \frac{CK_LCD_FRM}{BDIV} \quad BDIV = \{4, 8, 16, 32\} \text{ by setting } \mathbf{LCD_CK_BDIV}$$

$$LCD \text{ Blinking Frequency} = \frac{CK_LCD_BLK}{(LCD_CK_BCNT+1)}$$

There are two interrupt flags of LCD blinking on (**LCD_BLKONF**) and LCD blinking off (**LCD_BLKOFF**). There is a related interrupt enable register bit of **LCD_BLKON_IE**. The **LCD_BLKONF** bit is set at the start of LCD frame which is the first frame to switch display on during blinking period if the interrupt enable register bit of **LCD_BLKON_IE** is enabled. Also the **LCD_BLKOFF** bit is set at the start of LCD frame which is the first frame to switch display off during blinking period if the interrupt enable register bit of **LCD_BLKOFF_IE** is enabled.

The following diagram is showing an example of LCD Type-A drive timing with blinking display for 1/3Bias and 1/4Duty. For the example, the LCD display will toggle the blinking display between N+1 frames of normal display and N+1 frames of blank display. During blank display period, all of the segment lines are switched to dot off timing and all of the common lines are as normal display.

Figure 25-27. LCD Blinking Timing



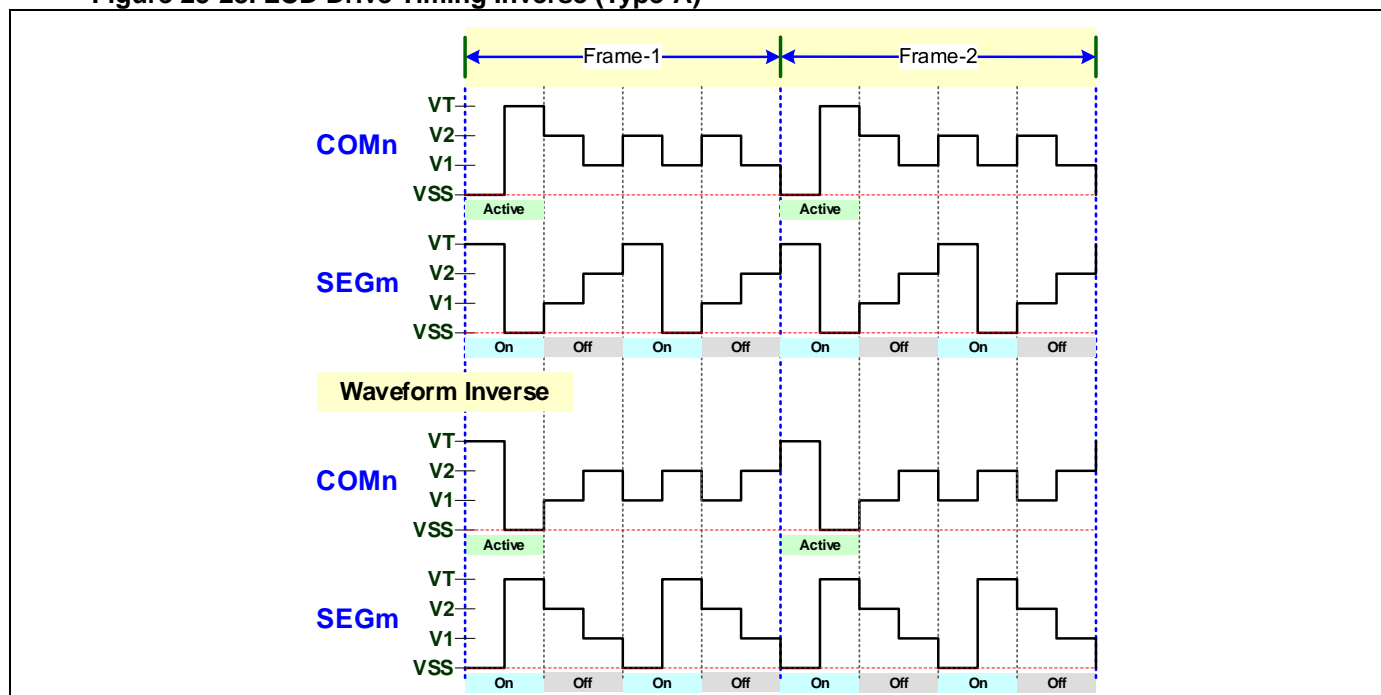
25.9.8. LCD Drive Timing Phase Inverse

The LCD COM/SEG timing controller supports LCD timing with output waveform phase inverse control and user can enable by setting **LCD_CS_INV** register. When user enables the LCD output waveform inverse function, all COM and SEG output waveform phase are inverted.

❖ LCD Drive Timing Inverse (Type-B)

The following diagram is showing an example of LCD Type-A drive timing with phase inverse.

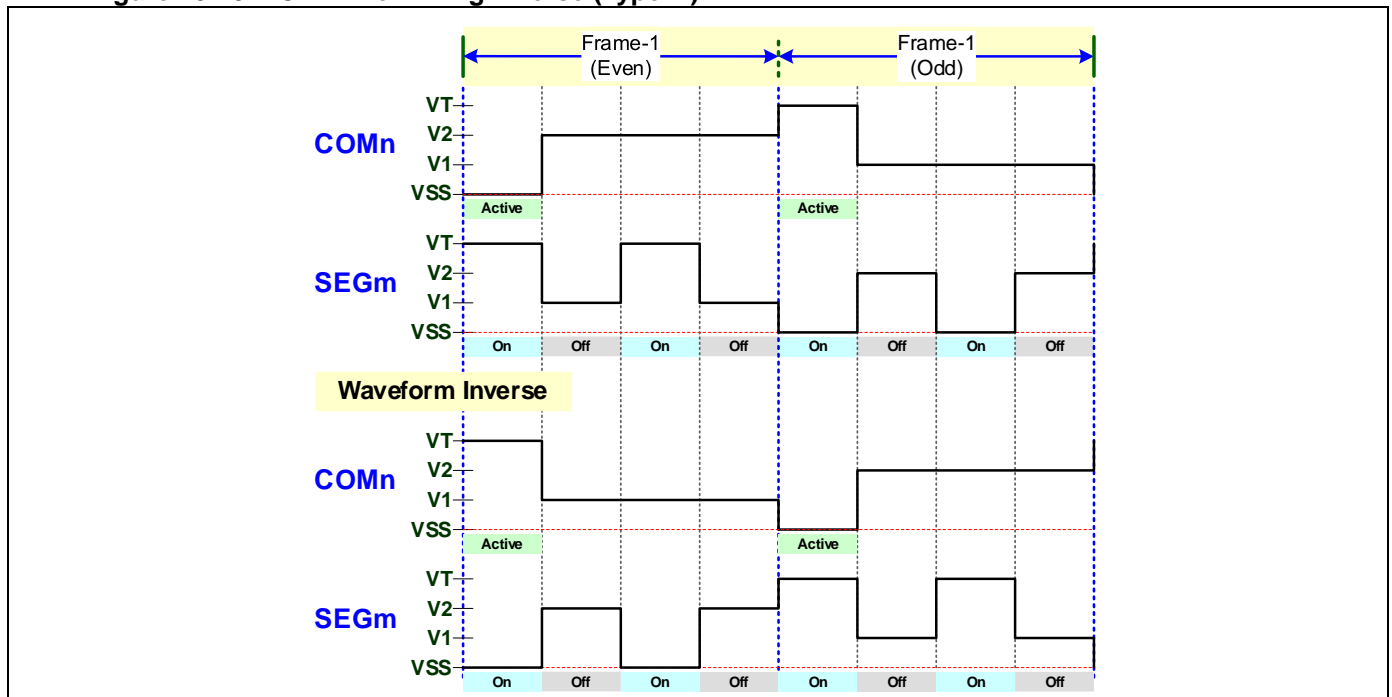
Figure 25-28. LCD Drive Timing Inverse (Type-A)



❖ LCD Drive Timing Inverse (Type-B)

The following diagram is showing an example of LCD Type-B drive timing with phase inverse.

Figure 25-29. LCD Drive Timing Inverse (Type-B)



25.10. LCD Data Control

The module is embedded the LCD data RAM to do as LCD display memory and implements an independent 8-bit data register of **LCD_Mn** for each LCD pins of **LCD_Pn**. This module provides maximum 44 LCD drive output of **LCD_P[0..43]** to external monochrome passive LCD glass. Each of the 8-bit data register bits can use to do display data bits. (n= LCD_Pn pin index number)

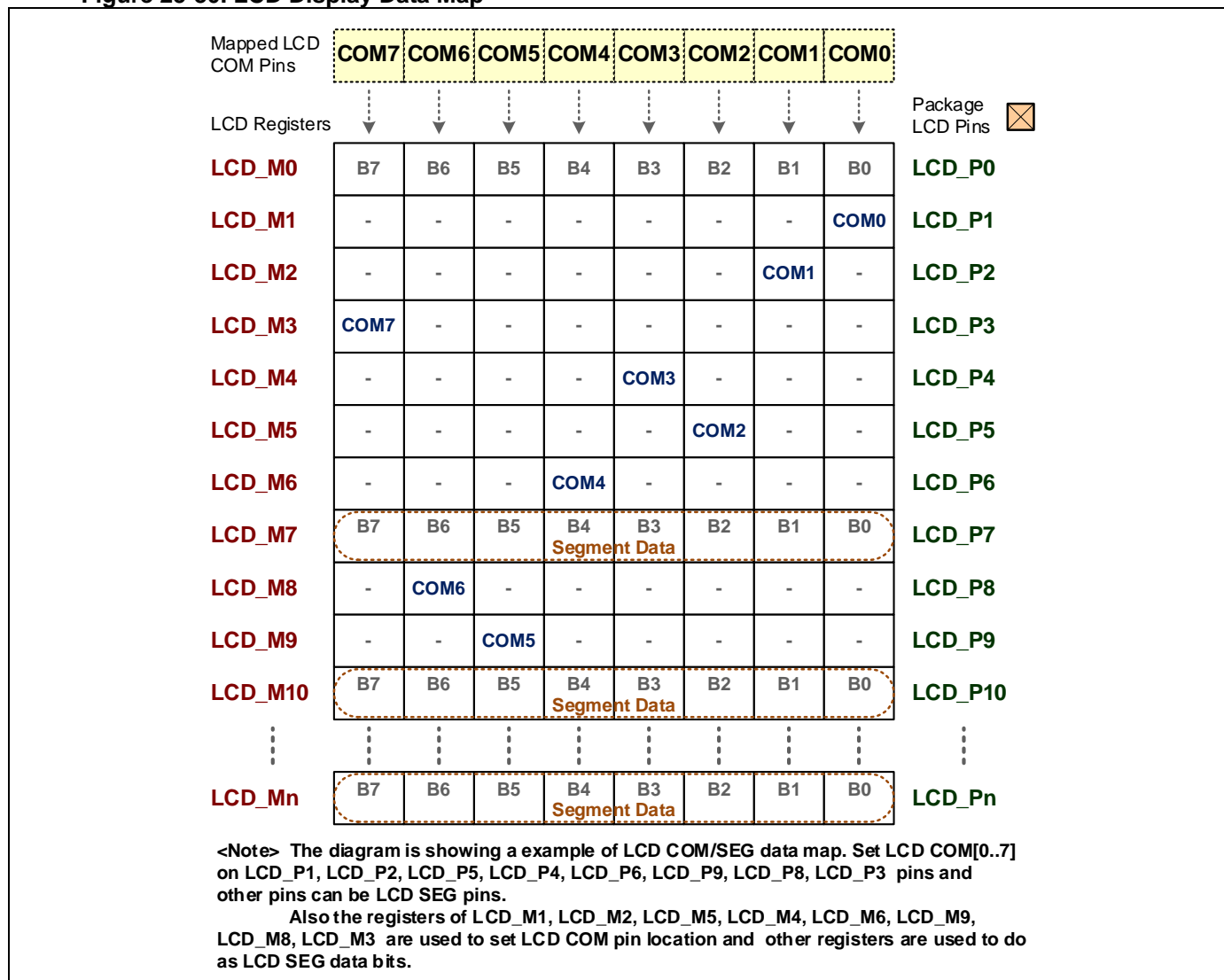
Before LCD data RAM access, user must configure LCD common/segment line assignment. Please refer the section of "[LCD COM and SEG Control](#)" for more information about the pin configuration of LCD common and segment lines.

25.10.1. LCD Display Data

When some drive lines of **LCD_Pn** are configured to common lines by setting **LCD_CSn** register, user can set the related **LCD_Mn** registers to map the LCD common line index definitions. For other drive lines of **LCD_Pn** are configured to segment lines by setting **LCD_CSn** register, the related **LCD_Mn** register is used as the **LCD_Pn** segment line display data bits which are as LCD display data of one dot for each of COM0 to COM7 active cycle. User can set the LCD dot on or off data on a segment line in related **LCD_Mn** register. If one bit is 0, it is meaning the dot is inactive for corresponding COMn active cycle. If one bit is 1, it is meaning the dot is active for corresponding COMn active cycle.

The following diagram is showing an example of LCD display data map. It is 8 common lines by using **LCD_P[1..6]** and **LCD_P[8..9]** pins. The LCD pins of **LCD_P7** and **LCD_P10** are assigned as segment lines.

Figure 25-30. LCD Display Data Map



For a common line example, the bit-3 of **LCD_M4** register is set 1 and **LCD_P4** pin is assigned to COM4 line. Except bit-3, the other bits of **LCD_M4** register must set 0. The second example, the bit-5 of **LCD_M9** register is set 1 and **LCD_P9** pin is assigned to COM5 line. Also except bit-5, the other bits of **LCD_M9** register must set 0.

For a segment line example, the drive lines of **LCD_P7** pin is assigned to a segment line. User can set 1 for bits 0, 1, 4 and 7 of **LCD_M7** register. These dots of this segment line those are connected to the lines of COM0, COM1, COM4 and COM7 will be display on for corresponding active cycle of COM0, COM1, COM4 and COM7.

● LCD Data RAM Clear

The module supports to clear all LCD display memory **LCD_Mn** registers by setting **LCD_MEM_CLR** register except those registers are used to assign LCD COM line index. The **LCD_MEM_CLR** bit is automatically reset when the LCD memory is cleared.

25.10.2. LCD SOF and UDCF Interrupt

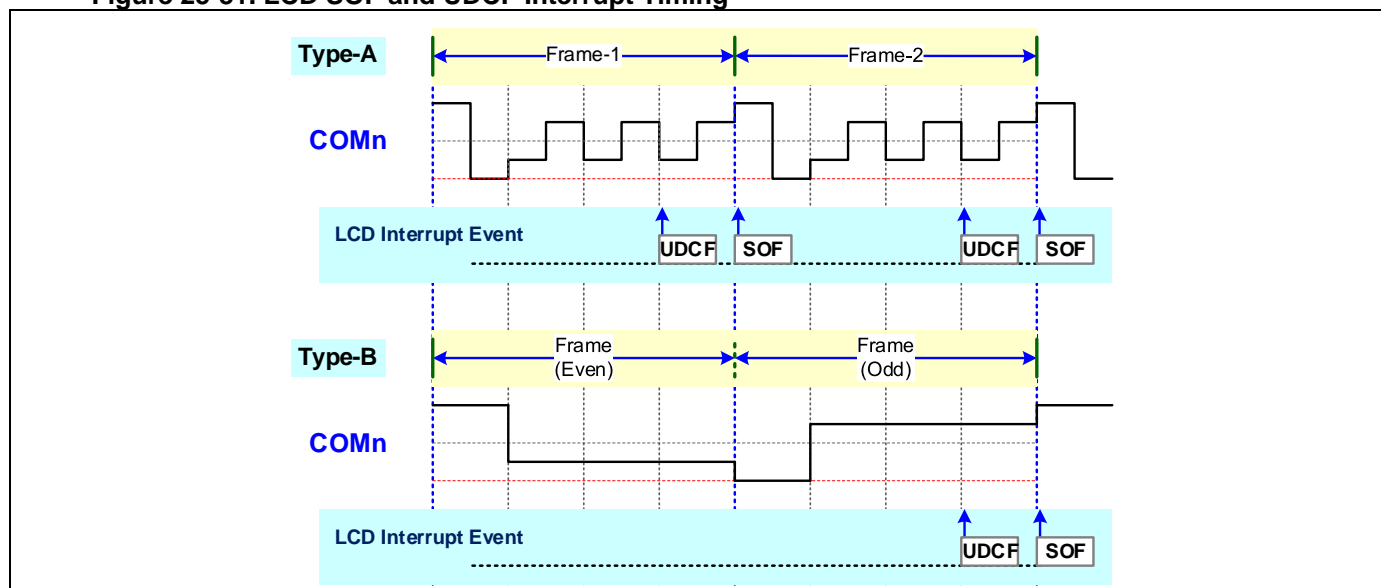
The module can generate a LCD start of frame interrupt flag (SOF) and a LCD update display data completed (UDCF) interrupt flag for user updating the LCD data RAM. User can read SOF flag in **LCD_SOF** register. This SOF flag is set when the LCD display starts a new frame and the display data has updated if related interrupt enable register bit of **LCD_SOF_IE** is enabled. Also user can read UDCF flag in **LCD_UDCF** register. This UDCF flag is set by hardware when the current display data has updated to display memory and user can start to write next display frame data if related interrupt enable register bit of **LCD_UDC_IE** is enabled.

When the LCD display is to be changed, the LCD data RAM *content needs to be updated with new data*. The LCD data RAM *is designed to allow that data to be updated at any time by user firmware*. But that will affect the integrity of LCD display and make the LCD display unexpected behavior if the data updating does not synchronize to LCD fame timing.

So the user firmware *must make sure that LCD data RAM is updated synchronously with the start of frame and the LCD data RAM refresh rate must be faster than the LCD frame frequency*. For suggestion, user can update new LCD data RAM in the interrupt service routine of UDCF. This UDCF flag is set at the start of last phase period in a frame.

The following diagram is showing SOF and UDCF interrupt timing.

Figure 25-31. LCD SOF and UDCF Interrupt Timing



26. OPA (Operational Amplifier)

26.1. Introduction

ON	SLEEP	STOP
----	-------	------

The module can be running in ON and SLEEP modes only.

The chip builds in one OPA module which embeds one general purpose operational amplifier of OP0 with two inputs and one output. It also can be configured as one general purpose analog comparator. The OPA output can be connected to internal input channel of ADC and CMP modules.

The module can be running in ON and SLEEP mode only but can be wakeup MCU from STOP mode by the input comparison change detection.

Notify: The sign of (n= operational amplifier macro index number) is using for Registers, Signals and Pins in the descriptions of this chapter.

26.2. Features

- Embedded one operational amplifier
- Provide two external inputs and one output for one OPA
 - Provide internal ADC PGA output and VBUF voltage connection
 - Support OPA output to ADC and CMP internal channel input
- Support low power mode for optimal current consumption

26.3. Implementation

26.3.1. Chip Implementation

The following table is showing the implementation of OPA module.

Table 26-1. OPA Implementation

Chip	OPA Analog Comparator		Analog Macro
	OP0	Total Channels	Internal Channel
MG32F02N128/N064 MG32F02K128/K064	V	2	VBUF, PGA_OUT
MG32F02A128/A064 MG32F02U128/U064 MG32F02V032	Not Implemented		

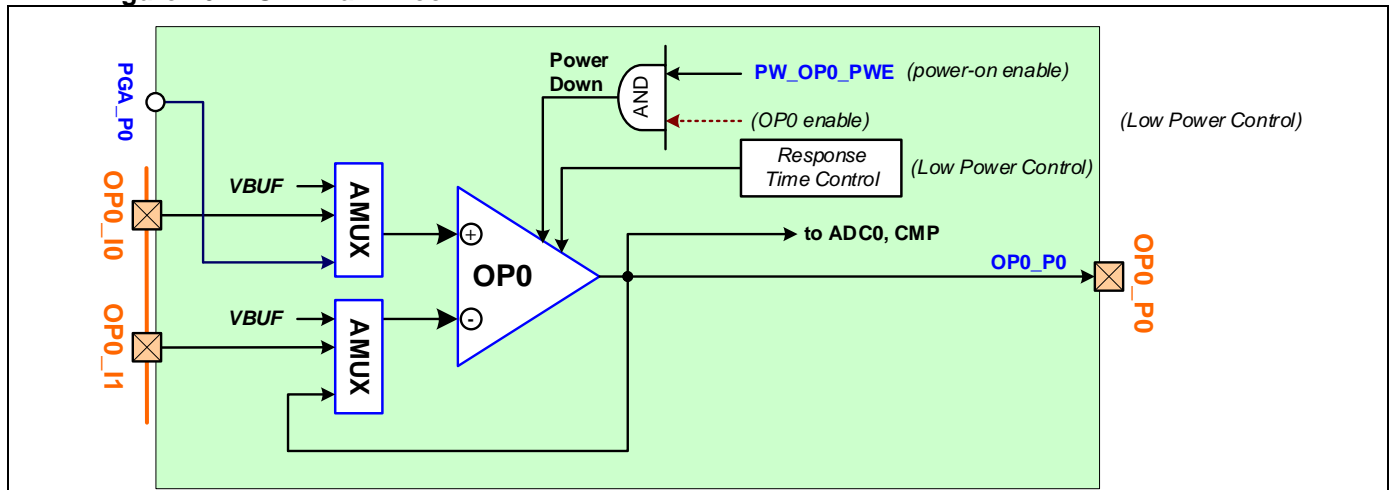
<Note> V: Implemented

26.4. Control Block

The OPA module includes one analog operational amplifier OP0. It is with the independent input multiplexer for plus input port and minus input port. The input multiplexer can connect the internal voltage source of VBUF and PGA_OUT. The OPA can support low power mode for optimal current consumption.

The following diagram is showing the analog operational amplifier block.

Figure 26-1. OPA Main Block



26.5. IO Lines

26.5.1. IO Signals

- **OPn_I[0..1]**

They are the operational amplifier analog input signal for OPA **OPn** which is able to use for plus input or minus input.

- **OPn_P0**

It is the operational amplifier analog output for OPA **OPn**.

26.5.2. IO Configure

User must configure the related IO pins in order to use the IO lines of this module. The IO operation modes, output high speed option, pull-high option, output drive strength, IO deglitch filter and input inverse selection are programmable by pin independent. The using of analog input **OPn_I[0..1]** and analog output **OPn_P0** must set the IO mode to AIO mode for related IO pins. Please refer the section of “[IO Mode](#)” in GPIO chapter for more detail descriptions of IO mode configuration.

Each IO signal may be mapped and selected on several IO pins by configuring the IO AFS matrix. Please refer the section of “[Alternate Function Select](#)” in GPIO chapter for more detail descriptions of IO AFS configuration. About the actual IO pin AFS information, please refer the section of “[Pin Alternate Functions Selected Table](#)” in Pin Description chapter of the chip Data Sheet.

26.6. Power and Clock

26.6.1. OPA Power Control

The operation power VDDA of OPA analog macros is from the IO power **VDD** pin. The operational amplifier is with one enable bits of **OPA_OPn_EN**. The OPA is enabled only when the **OPA_OPn_EN** bit is set to logic 1. User can plan the power control of OPA is on or off beforehand for chip entering **SLEEP** or **STOP** mode by setting **PW_SLP_OPn** or **PW_STP_OPn** registers. Refer the System Power chapter for more information. (n= {0})

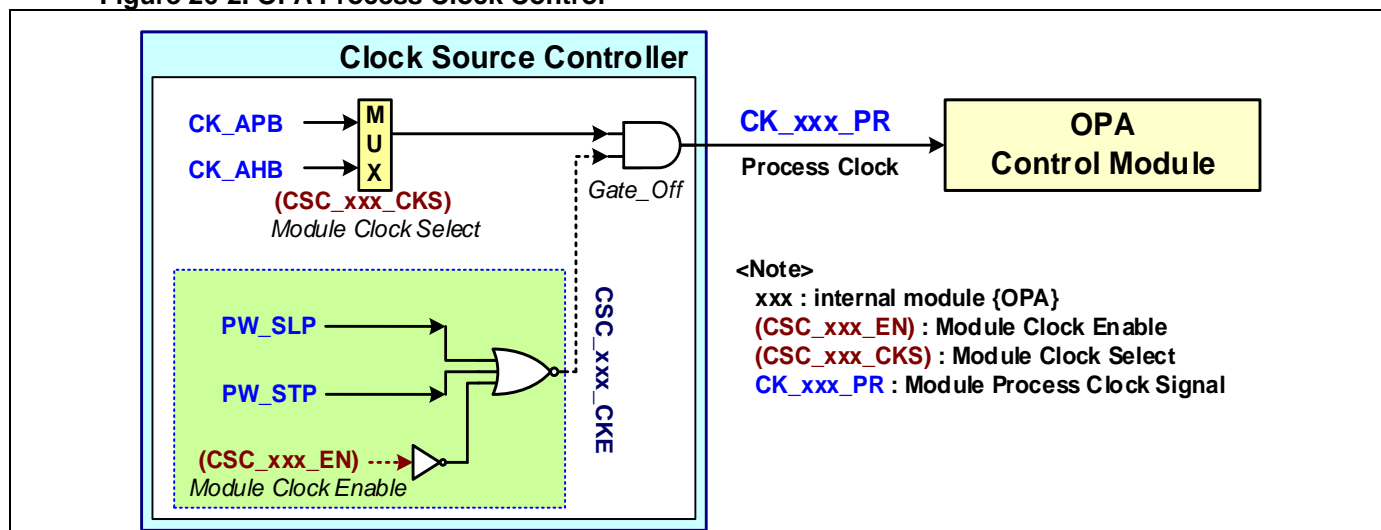
- **Low Power Mode**

The OPA can support the low power mode for optimal current consumption by setting **OPA_OPn_LPEN** bit. For analog comparator mode function, it is also control the response time of comparator for different applications. (n= {0})

26.6.2. OPA Clock Control

The module process clock of **CK_OPA_PR** is using for the interface control logic between APB bus and the module. It is coming from CSC (Clock Source Controller) module. It can be enabled in **CSC_OPA_EN** register and select the clock source from APB clock or AHB clock in **CSC_OPA_CKS** register. Refer the System Clock chapter for more information.

Figure 26-2. OPA Process Clock Control



26.7. OPA Operational Amplifier

26.7.1. Input Channels

- **Analog Input Multiplexer**

Each the analog multiplexer (AMUX) of the positive or negative input of the operational amplifier has flexible 3 channels input. These channels are including of one comparator independently external channels and two internal channels. The AMUX can be configured by **OPA_OPn_PMUX**, **OPA_OPn_NMUX** registers. (n= {0})

The independent external channels are from **OPn_I0** and **OPn_I1** pins those can input to the operational amplifier OP-n. The two internal channels are from internal voltage reference of **VBUF** and internal voltage source of **PGA_P0** or **OPn_P0**. The voltage source **PGA_P0** is coming from ADC PGA output and **OP0_P0** is coming from the operational amplifier self-output.

- **I/O Pins Used with OPA Function**

The analog input pins used for the operational amplifier also have its I/O port digital input and output function. In order to give the proper analog performance, a pin that is being used should have its digital output as disabled. It is done by putting the port pin into the input-only mode. And when an analog signal is applied to the analog input pin and the digital input from this pin is not needed, software could set the corresponding pin to analog-input-only in **PX_IOMn** (X={A},n={0~15}) to reduce power consumption in the digital input buffer.

26.7.2. Input Offset Trimming

For user application and environment, the operation voltage and temperature will affect the OPA characters. The chip is built-in the offset calibration function for OPA macro. The calibrated offset value will be recorded in option byte **OB** flash memory during manufacture and will be loaded to **CFG_OP0_OFFT** register after power on or chip cold reset.

The **CFG_OP0_OFFT** register provides the default offset calibration value and will be loaded to this register after chip cold reset. User can calibrate the input offset error and update the **OPA_OP0_OFFT** register after OPA offset calibration.

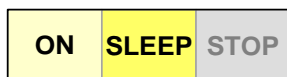
26.7.3. Operational Amplifier Output

- **Output Pin Control**

There is one output pin of **OPn_P0** to do as the operational amplifier OPn output to external application. (n= {0})

27. ADC (Analog-to-Digital Converter)

27.1. Introduction



The module can be running in ON and SLEEP modes only.

The chip builds in one ADC0 module which embeds one 12-bit successive approximation ADC (analog-to-digital converter), one PGA (programmable gain amplifier) with gain 1~4 or 1~128 and digital logic for output code control. It supports the configurable multiplexed channels those include maximum 16 external and multiple internal sources. The analog-to-digital conversion can be performed in One-Shot, One-Continue, Scan-One, Scan-Continue or Scan-Loop modes.

Notify: The sign of (x = module index; n, p = input channel number) is using for Registers, Signals and Pins/Ports in the descriptions of this chapter. [EX]: **ADCx_CONV_MDS**, **ADCx_SUMn** ~ x indicates module index number; n indicates channel number.

27.2. Features

- 12-bit SAR ADC
- Provide max. external 16 channels and internal 8 channels input
- Configurable resolution : 12/10/8-bit
- Configurable sampling time
- Support auto-sampling and trigger by external pin , internal events and software bit
- Data alignment for output code left/right justify
- Built-in input buffer stage with bypass option
 - Programmable gain : 1~128
 - Programmable gain : 1~4 (for MG32F02A128/U128/A064/U064/V032 only)
- Provide internal voltage source VBUF 1.4V
- Optional ADC top voltage reference from external VREF+, internal VDD or internal IVR24
- Interrupt generation at the end of sampling, end of conversion, end of scan conversion
- Support voltage window detect and output code limit
 - Two level programmable window threshold
- Built-in three channel independent hardware accumulators for ADC output code
- Support one-shot/channel scan/loop scan
- Support hardware calibration to minimize conversion error
- ADC data are buffered with DMA capability
- Support wait mode
 - Prevents ADC overrun in application with low frequency
- Support auto off mode
 - ADC auto power off except during the active conversion phase
- Built-in a temperature sensor
 - Temperature resolution : +/- 2 °C (Typical)
 - Temperature operation range : -40°C ~ 125°C

27.3. Implementation

27.3.1. Chip Implementation

The following tables are showing the implementation of ADC module.

Table 27-1. ADC Implementation-1

Chip	ADC Input	ADC Macro				ADC Vref
	Max. External Channels	Conversion Rate	Hardware Calibration	IVR24 Reference	PGA Gain Ratio	Voltage Source
MG32F02A128/A064 MG32F02U128/U064	16	1.5Msps/12bit	V	V	1 ~ 4	VREF+/IVR24
MG32F02V032	8	1.0Msps/12bit	V	-	-	VDD
MG32F02N128/N064 MG32F02K128/K064	16	1.5Msps/12bit	V	V	1 ~ 128	VREF+/IVR24

<Note>

V: Implemented; IVR24: ADC internal reference top voltage
VREF+, VDD: Voltage source from VREF+ pin, internal VDD by package type

Table 27-2. ADC Implementation-2

Chip	Package	ADC External Input					External Vref
		Input Channels	ADC_I[3:0]	ADC_I[7:4]	ADC_I[11:8]	ADC_I[15:12]	VREF+
MG32F02A128/U128 MG32F02N128/K128	LQFP80	16	V	V	V	V	V
MG32F02A128/U128 MG32F02A064/U064 MG32F02N128/N064 MG32F02K128/K064	LQFP64	16	V	V	V	V	V
MG32F02A064/U064 MG32F02N064/K064	LQFP48	12	V	-	V	V	V
MG32F02V032	LQFP32	8	V	-	V	-	VDD
MG32F02V032	QFN32	8	V	-	V	-	VDD
MG32F02V032	TSSOP20	2	ADC_I8 , ADC_I10				VDD

<Note>

V: Implemented; VDD: External ADC voltage reference from internal VDD and not support VREF+ pin

27.3.2. Modules' Functions

The following table is showing the implemented functions of ADC module.

Table 27-3. ADC Module Functions

Module	ADC0			Comment
Chip Module Functions	MG32F02A128	MG32F02V032	MG32F02N128	
	MG32F02U128		MG32F02K128	
	MG32F02A064		MG32F02N064	
	MG32F02U064		MG32F02K064	
ADC Bit Resolution	12-bit	12-bit	12-bit	
ADC Max. Conversion Rate	1.5Msps	1.0Msps	1.5Msps	
External Input Channels	16	8	16	
Internal Input Channels	8	4	7	
ADC Reference Top	VREF+/IVR24	VREF+	VREF+/IVR24	IVR24: internal reference voltage, VREF+: external voltage
VBUF, AVSS as ADC Input	yes	yes	yes	internal VBG buffer output, ADC ground signal
VCAP Out as ADC Input	yes	yes	yes	internal LDO output
DAC Out as ADC Input	yes	-	-	internal DAC output
ADC VREF as ADC Input	-	-	-	internal ADCx_IVREF reference voltage
Divided VDD as ADC Input	1/2 VDD	-	1/4 VDD	internal 1/2 VDD or 1/4 VDD
VPD as ADC Input	yes	-	-	internal VPD voltage
V33 Out as ADC Input	yes	-	-	internal V33 LDO output
TSO as ADC Input	yes	yes	yes	internal temperature sensor output
LCD_V1 as ADC Input	-	-	yes	internal LCD_V1 voltage
OPA out as ADC Input	-	-	yes	internal OPA out voltage
PGA with Gain	1~4	-	1~128	input buffer with gain ratio 1~4
Configurable sampling time	0~255	0~255	0~255	sampling clock time
Channel Scan Conversion	yes	yes	yes	
Hardware accumulators	3	3	3	
Voltage Window Detect	yes	yes	yes	detect code by the high/low threshold
Output Code Limitation	yes	yes	yes	clamp output code or skip the code
Code Left/Right Justify	yes	yes	yes	data alignment for output code
Signed Code Conversion	-	-	-	ADC output unsigned code
Wait Mode	yes	yes	yes	
Auto-Off Mode	yes	yes	yes	
ADC Calibration	Offset	Offset	Offset	
DMA request capability	yes	yes	yes	
DMA transfer Data Pack	32 or 16-bit	32 or 16-bit	32 or 16-bit	16-bit ADC code; 16-bit channel data

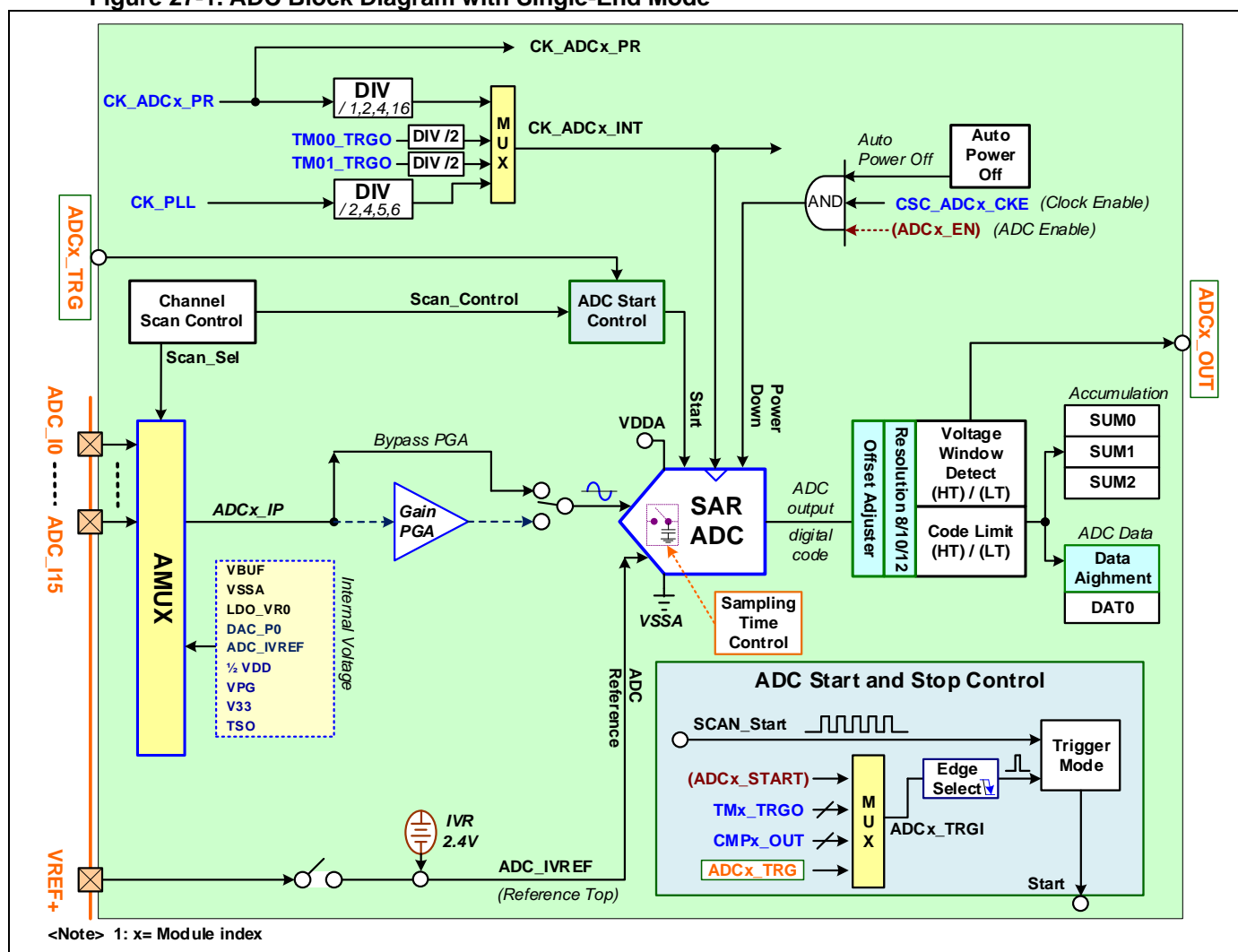
27.4. Control Block

The ADC control block consists of an analog multiplexer (AMUX) with maximum 16 external input channels, a 12-bit SAR (successive-approximation-register) ADC, reference voltage circuit, ADC conversion trigger start control block and change scan control block.

Refer the table of “[ADC Channel Definitions](#)” for more information about the supported internal channels of chip.

The following diagram is showing the ADC control block with single-end mode.

Figure 27-1. ADC Block Diagram with Single-End Mode



27.5. IO Lines

27.5.1. IO Signals

- **ADC_I[0..15]**

They are the ADC analog signals of input channel 0 ~ 15.

- **ADCx_TRG**

It is the external trigger start input signal for ADC data conversion.

- **ADCx_OUT**

It is the ADC state output signal of voltage window compare result.

- **VREF+**

It is the external voltage input as internal ADC reference top voltage.

[Notify]: This VREF+ pin is not supported for some chip package. Refer the related chip Data Sheet about this pin support.

27.5.2. IO Configure

User must configure the related IO pins in order to use the IO lines of this module. The IO operation modes, output high speed option, pull-high option, output drive strength, IO deglitch filter and input inverse selection are programmable by pin independent. The using of analog input **ADC_In** must set the IO mode to AIO mode for related IO pins. Please refer the section of "[IO Mode](#)" in GPIO chapter for more detail descriptions of IO mode configuration.

Each IO signal may be mapped and selected on several IO pins by configuring the IO AFS matrix. Please refer the section of "[Alternate Function Select](#)" in GPIO chapter for more detail descriptions of IO AFS configuration. About the actual IO pin AFS information, please refer the section of "[Pin Alternate Functions Selected Table](#)" in Pin Description chapter of the chip Data Sheet.

27.6. Power and Clock

27.6.1. ADC Power Control

The operation power VDDA of ADCx analog macros is from the IO power **VDD** pin. The ADC analog macro is enabled only when the **ADCx_EN** bit is set to logic 1. The ADC analog macro is in power down when this bit is logic 0.

User can set the **ADCx_EN** bit to logic 1 or 0 before chip entering **SLEEP** mode and force the CPU entering sleep. Then the chip will enter **SLEEP** mode and the ADC analog macros will be power-on or power-off by the register setting of **ADCx_EN**. The ADC module is not supported in **STOP** mode. Refer the System Power chapter for more information. (n= {0, 1, 2, 3})

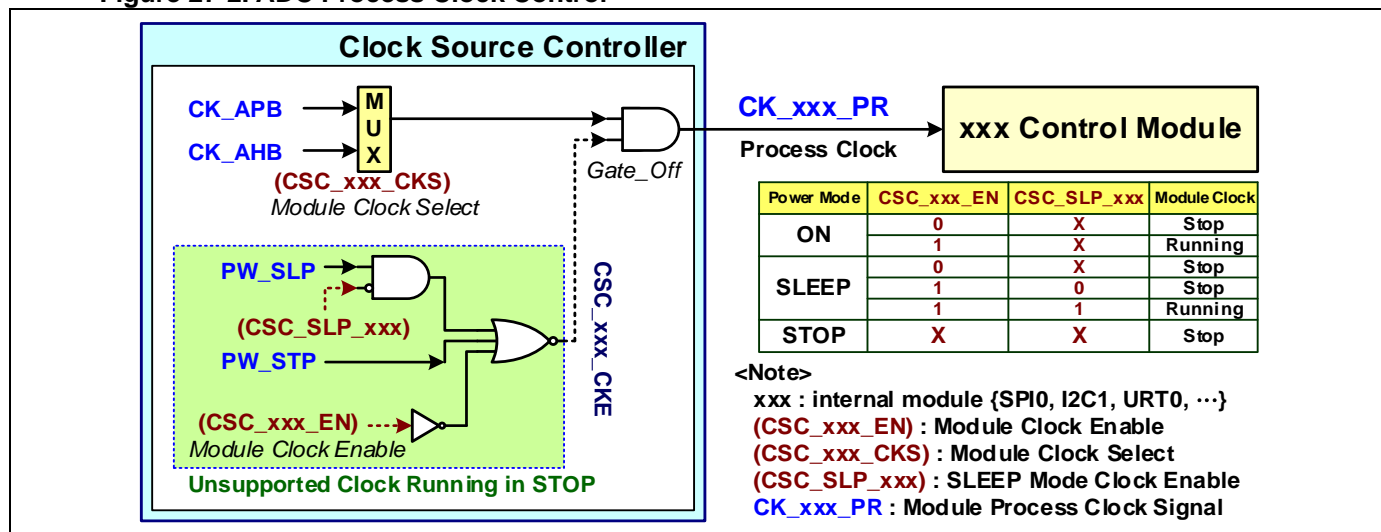
27.6.2. ADC Clock Control

- **Module Process Clock**

The module process clock of **CK_ADCx_PR** is using for the interface control logic between APB bus and the module. It is coming from CSC (Clock Source Controller) module. It can be enabled in **CSC_ADCx_EN** register and select the clock source from APB clock or AHB clock in **CSC_ADCx_CKS** register. User can plan the module clock is running or not beforehand for chip entering **SLEEP** mode by setting **CSC_SLP_ADCx** register. Refer the System Clock chapter for more information.

The following diagram is showing the ADC process clock control block on CSC module.

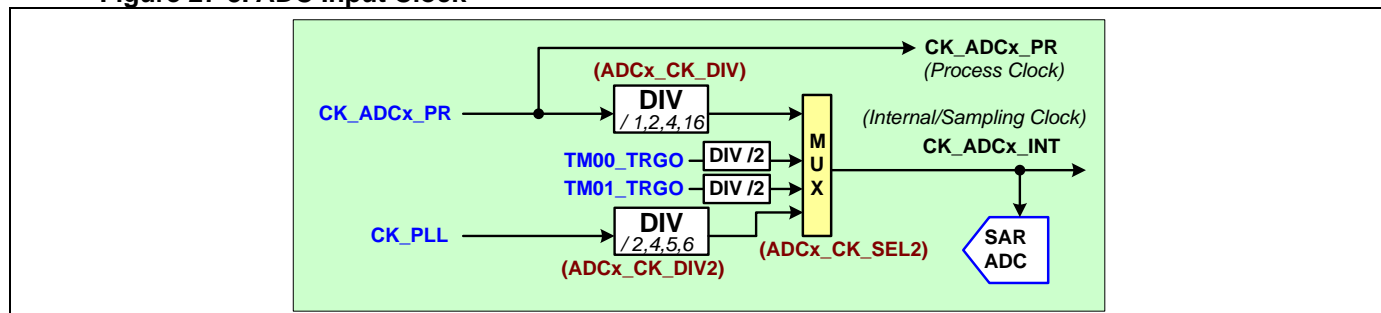
Figure 27-2. ADC Process Clock Control



● Module Internal Clock

The following diagram is showing the ADC input clock.

Figure 27-3. ADC Input Clock



ADC has a maximum conversion speed of 400Ksps. The ADC internal clock or conversion sampling clock is selected from the clock source of module process clock (AHB clock **CK_AHB**, APB clock **CK_APB**), timer trigger output signal (**TM00_TRGO** or **TM01_TRGO**) or PLL output clock **CK_PLL** by setting **ADCx_CK_SEL2** register. The ADC conversion clock should be no more than 12 MHz.

The module process clock can be divided by 1/2/4/16 to do as the internal clock in **ADCx_CK_DIV** register. The **CK_PLL** clock can be divided by 2/4/5/6 to do as the internal clock in **ADCx_CK_DIV2** register.

[Notify]: Usually the internal clock frequency needs slow down at least 1/2 than module process clock.

27.7. Interrupt and Event

There are two signals of **INT_ADC** and **RST_ADC** to be generated in this ADC control module. **INT_ADC** sends to External Interrupt Controller (EXIC) to do as an interrupt event. **RST_ADC** sends to Reset Source Controller to do as a reset event.

27.7.1. ADC Interrupt and Reset Event

- **Interrupt Events**

INT_ADC signal sends to External Interrupt Controller (EXIC) to do as an interrupt event. These interrupt flags are using for the interrupt service routine (ISR) flow control. Generally these flags are set by hardware and software clears them when the serviced job is completed in the related ISR. Each interrupt flag has one interrupt enable bit. User can enable or disable it. There is one interrupt all enable bit of **ADCx_IEA** to enable or disable all the interrupt sources for this module.

- **Reset Events**

RST_ADC signal sends to Reset Source Controller (RST) to do as the warm reset events or cold reset events. These reset events can be enabled to reset the chip by setting the registers in RST.

Refer the descriptions of System Reset chapter for more information about the reset events and control.

27.7.2. ADC Interrupt Flags

Generally these interrupt flags are set by hardware and clear by software writing 1. Refer the register descriptions for more information about the related interrupt flags and interrupt enable bit.

- **ESMPF**

ADC sampling end flag (**ADCx_ESMPF**). This flag is set at the end of the sampling phase. There is a related interrupt enable register bit of **ADCx_ESMP_IE**.

- **E1CNVF**

ADC one-time conversion end flag (**ADCx_E1CNVF**). This flag is set at the end of each conversion of a channel and a new data result is available in the **ADCx_DAT0** register. When clears this flag, also it clears the **ADCx_DAT0** flag and ready to receive next data. There is a related interrupt enable register bit of **ADCx_ESCNV_IE**.

- **ESCNVF**

ADC channel scan conversion end flag (**ADCx_E1CNVF**). This flag is set at the end of the conversion of a sequence channel scan. There is a related interrupt enable register bit of **ADCx_ESCNV_IE**.

- **OVRF**

ADC conversion overruns event flag (**ADCx_OVRF**). When clears this flag, also it clears the **ADCx_DAT0_OVRF** flag which is in the **ADCx_DAT0** register and is the same as the OVRF flag. There is a related interrupt enable register bit of **ADCx_OVR_IE**.

- **WDLF / WDIF / WDHf**

ADC voltage window detect outside low, inside and outside high event flags (**ADCx_WDLF**, **ADCx_WDIF** and **ADCx_WDHf**). There are three related interrupt enable register bits of **ADCx_WDL_IE**, **ADCx_WDI_IE** and **ADCx_WDH_IE**.

- **SUMOF**

ADC data sum-0, 1, 2 accumulation overflow or underflow flag (**ADCx_SUMOF**). When clears this flag, also it clears all the **ADCx_SUMn_OF** flags (n=0~2). There is a related interrupt enable register bit of **ADCx_SUMO_IE**.

- **SUMCF**

ADC data sum-0, 1, 2 accumulation complete flag (**ADCx_SUMCF**). When clears this flag, also it clears all the **ADCx_SUMn_CF** flags (n=0~2). There is a related interrupt enable register bit of **ADCx_SUMC_IE**.

- **SUMOVRF**

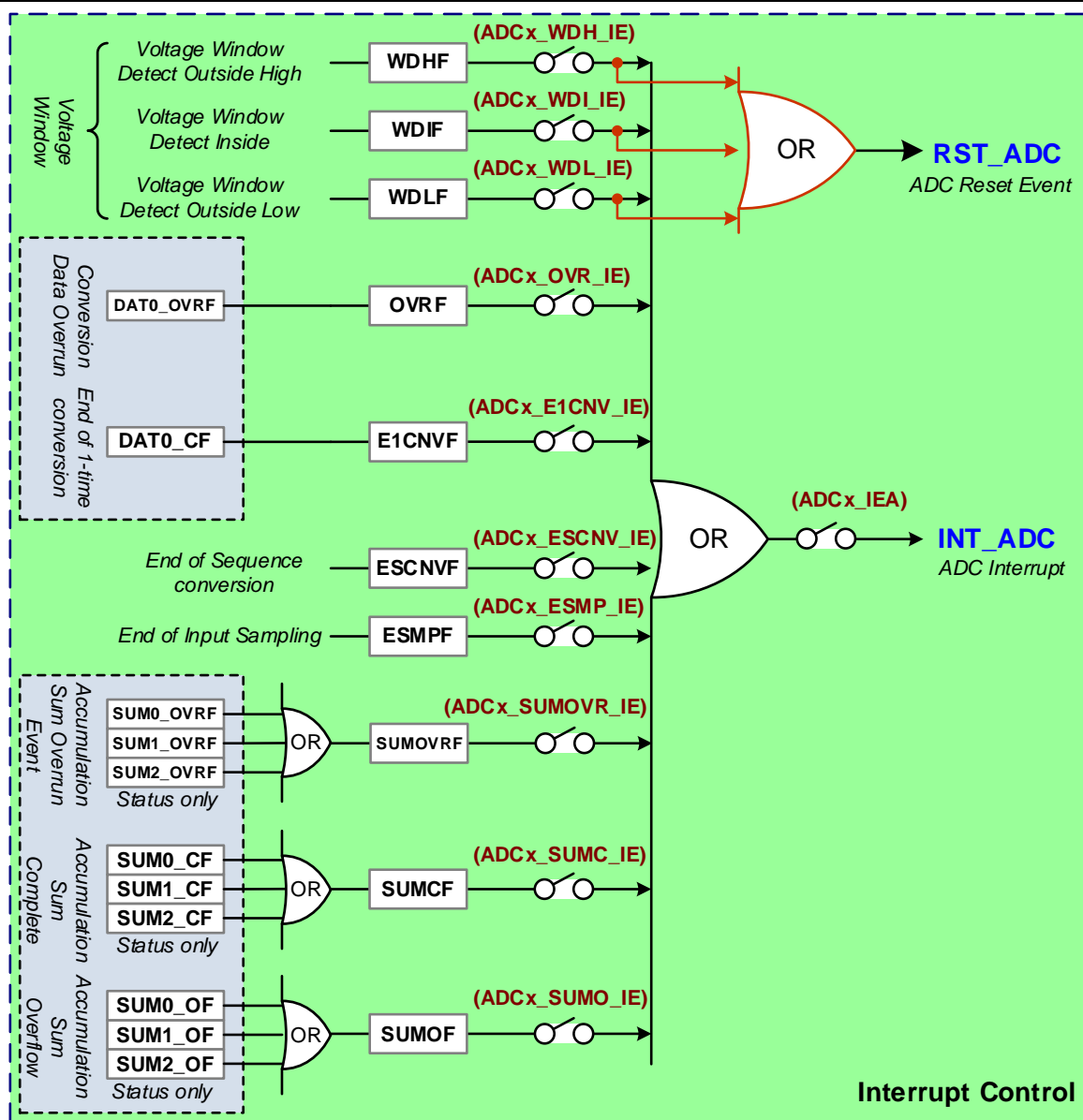
ADC data sum-0, 1, 2 register overrun flag (**ADCx_SUMOVRF**). When clears this flag, also it clears all the **ADCx_SUMn_OVRF** flags (n=0~2). There is a related interrupt enable register bit of **ADCx_SUMOVR_IE**.

- **SOCF**

ADC conversion status (**ADCx_SOCF**). This bit will be active during ADC start conversion to ADC conversion ready period.

The following diagram is showing the ADC interrupt control block.

Figure 27-4. ADC Interrupt Control



<Note> 1: x= Module index

27.8. ADC Operation

27.8.1. Single-End Mode

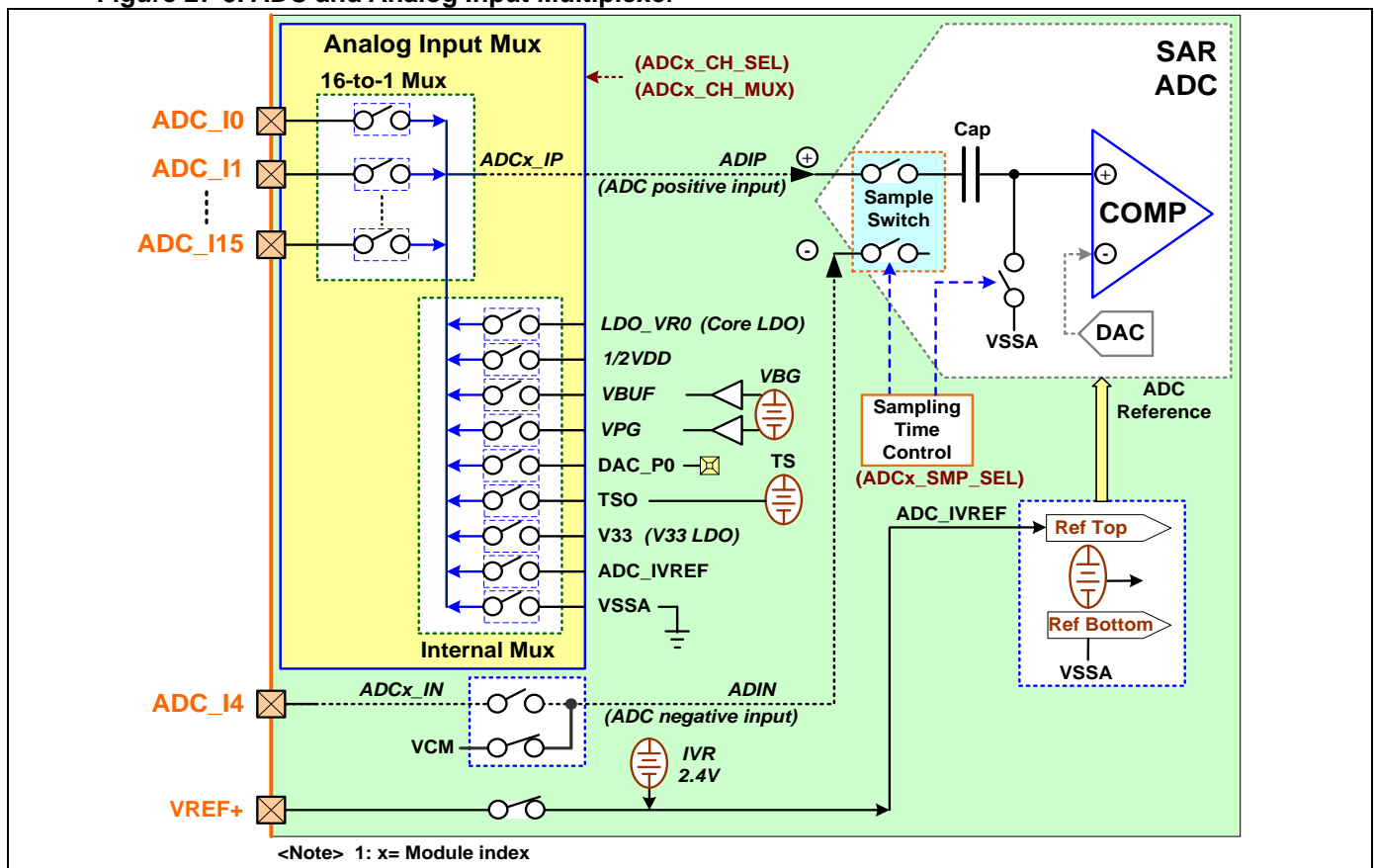
The ADC is supported the single-end operation mode. It can convert the ADC analog input and output to unsigned code. Refer the section “[Signed Code Converter](#)” for more information.

27.8.2. ADC Input Channels

The analog multiplexer (AMUX) selects the inputs to the ADC, allowing any of the input pins to be measured in single-ended mode. The AMUX can be configured by **ADCx_CH_SEL** and **ADCx_CH_MUX** registers. The selected pin is measured with respect to VSSA (internal ground).

Refer the table of “[ADC Channel Definitions](#)” for more information about the supported internal channels of chip.

Figure 27-5. ADC and Analog Input Multiplexer



● I/O Pins Used with ADC Function

The analog input pins used for the A/D converters also have its I/O pins for digital input and output function. In order to give the proper analog performance, a pin that is being used with the ADC should have its digital output as disabled. It is done by putting the port pin into the input-only mode. And when an analog signal is applied to the **ADC_In** pin and the digital input from this pin is not needed, software could set the corresponding pin to analog-input-only in **PA_IOMn** (n= {0~15}) register to turn off the digital input buffer to reduce power consumption.

- **ADC Input Channel Definitions**

The following table is showing register setting for ADC channel definitions.

Table 27-4. ADC Channel Definitions

External/Internal Channel Select			External / Internal Channel Definitions	Chip Support			
CH_SEL	CH_MUX	Channel Number	Descriptions	MG32F02A132 MG32F02A072	MG32F02A032	MG32F02A128 MG32F02U128 MG32F02A064 MG32F02U064	MG32F02V032
External	0x0	0	External Channel 0 (ADC_I0 pin input)	V	V	V	V
	0x1	1	External Channel 1 (ADC_I1 pin input)	V	V	V	V
	0x2	2	External Channel 2 (ADC_I2 pin input)	V	V	V	V
	0x3	3	External Channel 3 (ADC_I3 pin input)	V	V	V	V
	0x4	4	External Channel 4 (ADC_I4 pin input)	V		V	
	0x5	5	External Channel 5 (ADC_I5 pin input)	V		V	
	0x6	6	External Channel 6 (ADC_I6 pin input)	V		V	
	0x7	7	External Channel 7 (ADC_I7 pin input)	V		V	
	0x8	8	External Channel 8 (ADC_I8 pin input)	V	V	V	V
	0x9	9	External Channel 9 (ADC_I9 pin input)	V	V	V	V
	0xA	10	External Channel 10 (ADC_I10 pin input)	V	V	V	V
	0xB	11	External Channel 11 (ADC_I11 pin input)	V	V	V	V
	0xC	12	External Channel 12 (ADC_I12 pin input)	V	V	V	
	0xD	13	External Channel 13 (ADC_I13 pin input)	V	V	V	
	0xE	14	External Channel 14 (ADC_I14 pin input)	V	V	V	
	0xF	15	External Channel 15 (ADC_I15 pin input)	V	V	V	
Internal	0x0	0	Internal Channel 0 (VSSA voltage)	V	V	V	V
	0x1	1	Internal Channel 1 (IVREF voltage source)	V	V		
	0x2	2	Internal Channel 2 (DAC_P0 DAC voltage output)	V		V	
	0x3	3	Internal Channel 3 (VBUF 1.4V voltage source)	V	V	V	V
	0x4	4	Undefined Channel (input Mux output Hi-Z)				
	0x5	5	Undefined Channel (input Mux output Hi-Z)				
	0x6	6	Undefined Channel (input Mux output Hi-Z)				
	0x7	7	Undefined Channel (input Mux output Hi-Z)				
	0x8	8	Internal Channel 8 (LDO_VR0 output)		V	V	V
	0x9	9	Internal Channel 9 (TSO Output)			V	V
	0xA	10	Internal Channel 10 (1/2 VDD output)			V	
	0xB	11	Internal Channel 11 (VPG voltage)			V	
	0xC	12	Internal Channel 12 (V33 voltage)			V	
	0xD	13	Undefined Channel (input Mux output Hi-Z)				
	0xE	14	Undefined Channel (input Mux output Hi-Z)				
	0xF	15	Reserved for internal using				

<Note> **CH_MUX: (ADCx_CH_MUX)** ~ Channel Mux. number select

CH_SEL: (ADCx_CH_SEL) ~ External or Internal channels select

<Sign> "V" : channel is supported, " " : channel is not supported and the input Mux output Hi-Z

- **ADC Input from VBUF and VSSA**

The **VBUF** is the ADC internal voltage source which is generation from the internal bandgap voltage buffer. The **VSSA** is the internal ground of ADC macros. Usually user can input these two voltages to calibrate the ADC offset error and gain error.

- **ADC Input from ADC_IVREF**

The **ADC_IVREF** is the ADC internal reference top voltage which can input from the **VREF+** pin.

- **ADC Input from LDO_VR0**

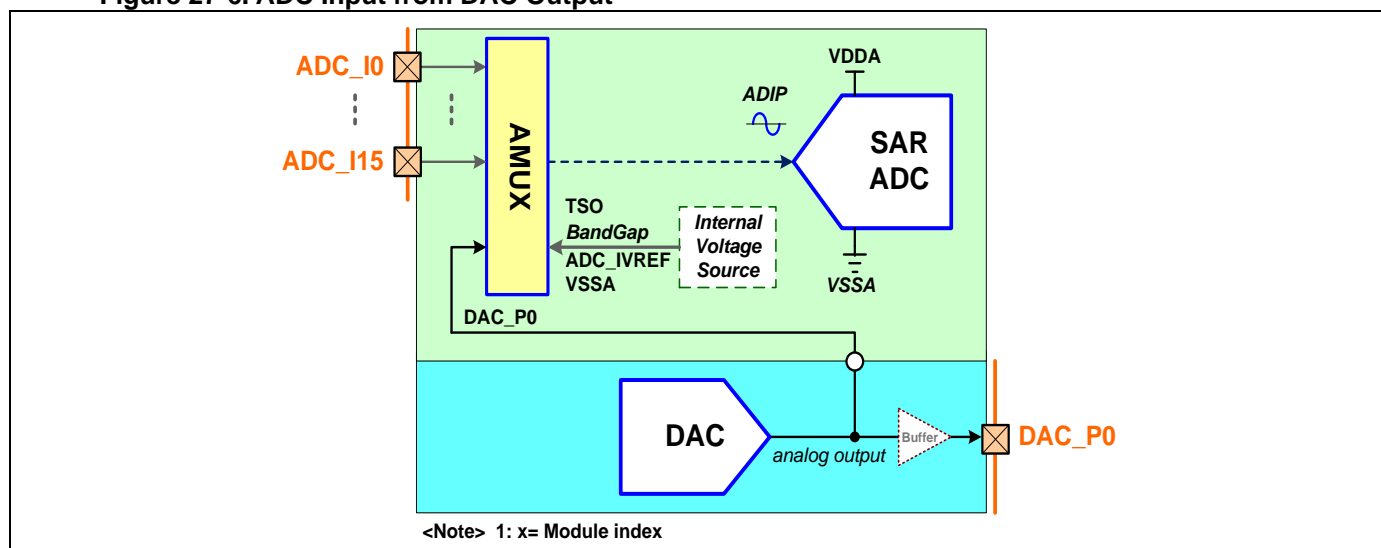
The **LDO_VR0** is the internal core logic power regulator output voltage from the **VR0/VCAP** pin. Usually user can input this voltage to monitor it or do as reference voltage for external ADC input voltage measurement.

- **ADC Input from DAC**

User can set **ADCx_CH_MUX** register to select the analog input from internal DAC output. It is useful to measure the DAC output by using the ADC. As the DAC output is current mode, an external resistor load R_{Load} is necessary to convert the current output to voltage level.

The following diagram is showing the internal connection of ADC input from DAC output.

Figure 27-6. ADC Input from DAC Output



- **ADC Input from 1/2VDD and V33**

The **1/2VDD** is the 1/2 voltage of the internal IO power from the **VDD** pin. The **V33** is the internal USB power regulator output voltage. Usually user can input these two voltages through ADC data conversion to measure and monitor their voltage level.

- **ADC Input from VPG**

The **VPG** is the ADC internal voltage source which is generation from the internal bandgap voltage buffer. The VPG is about 0.8 volt and usually user can input this voltage to calibrate the ADC PGA gain ratio. For example, set PGA gain x1 and x4 separated to get ADC codes to calculate the gain ratio error.

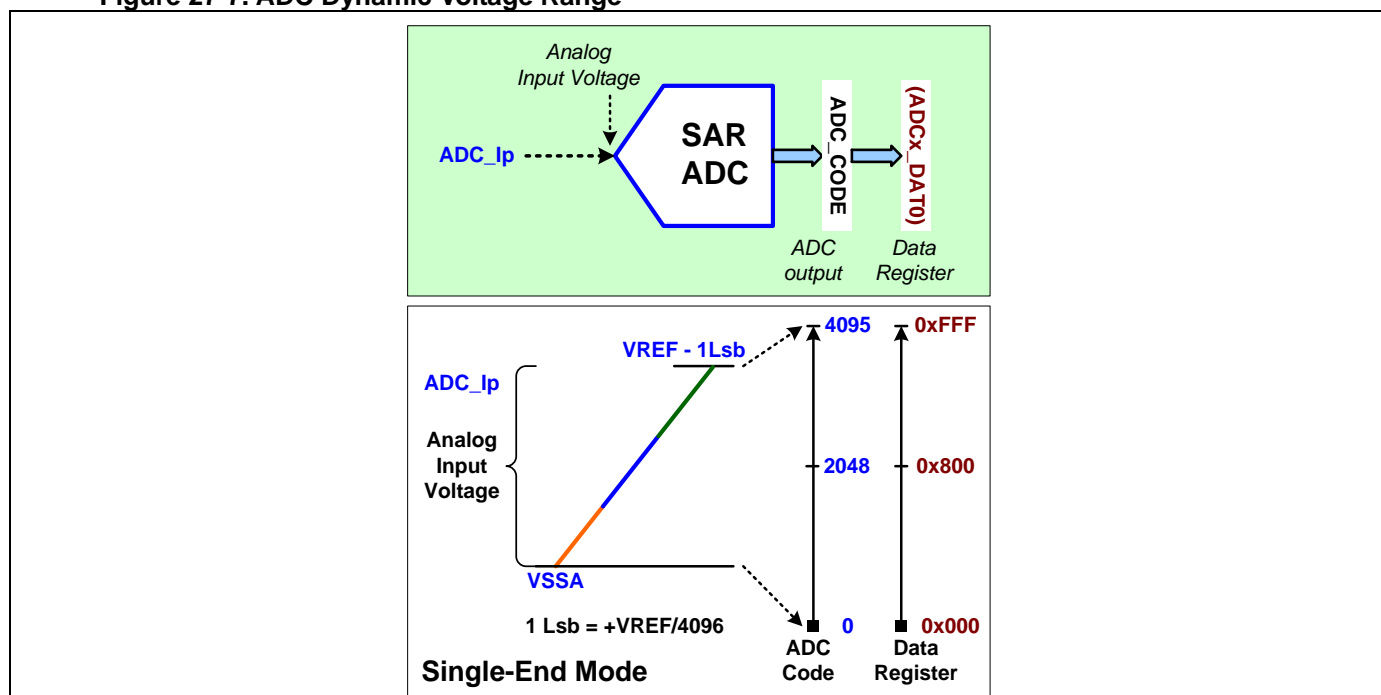
- **ADC Input from TSO**

User can input the TSO (temperature sensor output) voltage and measure the internal junction thermal of chip. Refer the section of "Temperature Sensor" for more information.

27.8.3. Input Dynamic Voltage Range and Code Range

The following diagram is showing the dynamic voltage range of the ADC for single-end mode.

Figure 27-7. ADC Dynamic Voltage Range



The ADC macro is embedded an internal reference voltage source **IVR24**. User can select the internal **IVR24** voltage or external **VREF** (**VREF+** pin voltage) to do as the ADC reference top voltage by setting **ADCx_IVREF_SEL** register. User can enable and power up the **IVR24** voltage source in **ADCx_IVR_EN** register.

The ADC input dynamic voltage range is about VSSA (very close 0 volt) to **VREF** (**VREF+** pin voltage) for single-end mode and $-\frac{1}{2}VREF$ to $+\frac{1}{2}VREF$ for differential mode if the ADC reference top voltage is selected **VREF**.

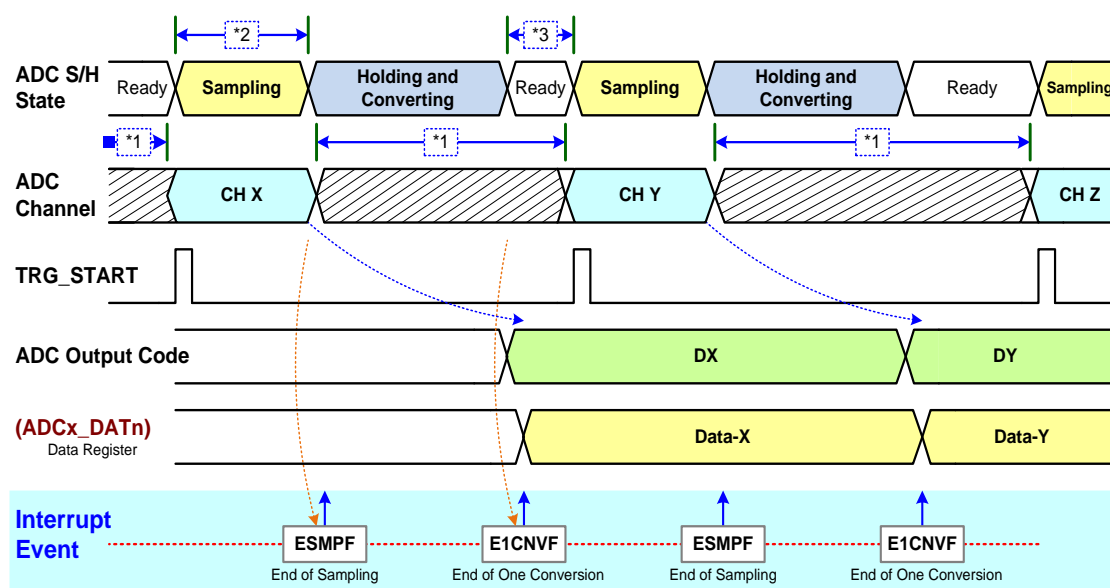
27.8.4. ADC Conversion

The ADC total conversion time is including of analog signal sampling time and analog signal-to-digital code conversion time.

● ADC Conversion Timing

User can select the appropriate conversion speed according to the frequency of the analog input signal. User can configure the analog signal-to-digital code conversion time for 24 or 30 ADC sampling clocks mode by setting **ADCx_CONV_TIME** register. When the ADC sampling time is set zero in **ADCx_SMP_SEL** register, the minimum ADC total conversion time will be 24 or 30 ADC sampling clocks by **ADCx_CONV_TIME** register setting. Refer the section of '[ADC Sampling Time](#)' about the ADC sampling time adjustment.

Figure 27-8. ADC Conversion Sequence



<Note-1> Software set ADC channel by setting (**ADCx_CH_MUX**) .

<Note-2> Programmable sampling time by setting (**ADCx_SMP_SEL**) register .
sampling cycle = (**ADCx_SMP_SEL**) +4 ~ ADC sampling clock time

<Note-3> ADC ready and wait to change channel for next conversion trigger.

<Note-4> {X,Y,...,Z} indicate the enabled channels those are following CH(channel) and D(Data) .

The ADC conversion is calculation by the formula:

$$\text{ADC Conversion Rate} = \frac{\text{Sampling Clock Frequency}}{\text{Sampling Clocks/Each Sample}}$$

For examples to 30 ADC sampling clocks mode, the ADC conversion rates can be 400Ksps, 800Ksps and 1.2Msps if the ADC input sampling clock frequencies are 12MHz, 24MHz and 36MHz. For examples to 24 ADC sampling clocks mode, the ADC conversion rates can be 500Ksps, 1Msps and 1.5Msps if the ADC input sampling clock frequencies are 12MHz, 24MHz and 36MHz.

● ADC Sampling Time

For input signal quality and conversion speedy issue, user can adjust and extend the ADC sampling time in **ADCx_SMP_SEL** register. Usually increase the ADC sampling time to get more stable voltage and better ADC performance if the conversion rate and signal bandwidth are reasonable and valid for actual application.

User can set the sampling time from 0 to 255 ADC sampling clock(s) by setting **ADCx_SMP_SEL** register. It also increases the total sampling clocks and conversion time.

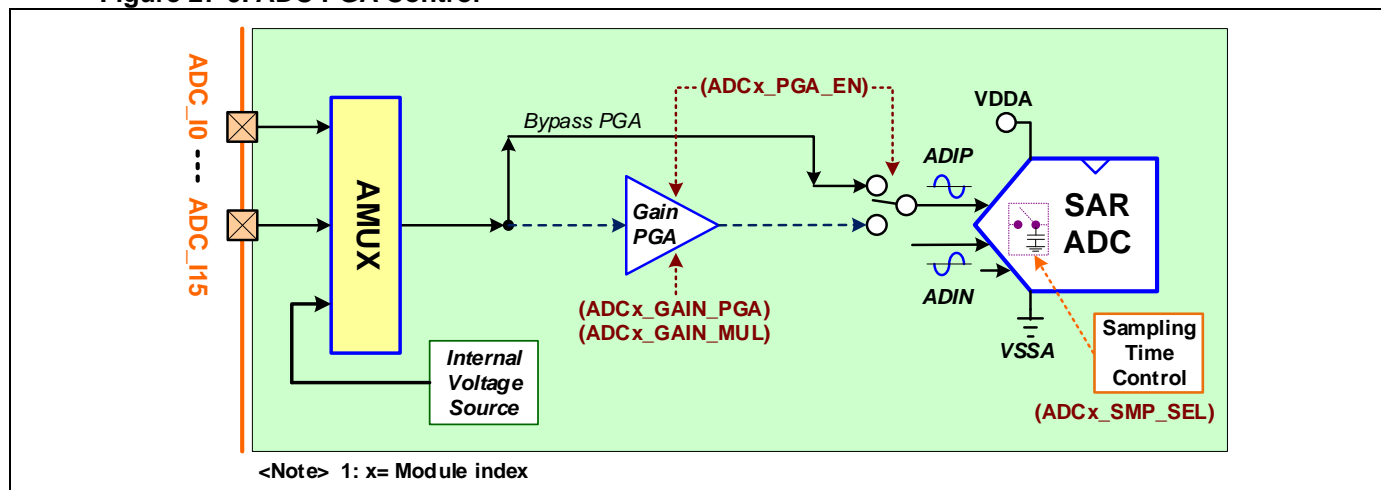
27.8.5. ADC PGA Control

● PGA Gain

The following diagram is showing the ADC PGA control block and related setting registers.

[Notify]: The **ADCx_GAIN_MUL** register is not supported for MG32F02A128/U128/A064/U064.

Figure 27-9. ADC PGA Control



The ADC module is built in a PGA (programmable gain amplifier) to improve the ADC input quality and amplify the input voltage range. The PGA gain is useful for measuring low-level signals. Increasing the PGA gain can reduce the input-referred noise. User can enable the PGA by setting **ADCx_PGA_EN** register.

— MG32F02A128/U128/A064/U064

User can set the gain value from 1 to 4 by setting **ADCx_GAIN_PGA** register.

The PGA gain value is set by the formula:

$$\text{PGA Gain} = \frac{\text{ADCx_GAIN_PGA} * 3}{63 + (63 - \text{ADCx_GAIN_PGA}) * 3} + 1$$

The following table is showing the ADC PGA gain ratio by register setting.

Table 27-5. ADC Gain Calculation - x1 ~ x4

GAIN_PGA	Gain Ratio	GAIN_PGA	Gain Ratio	GAIN_PGA	Gain Ratio	GAIN_PGA	Gain Ratio
0	1.000	16	1.235	32	1.615	48	2.333
1	1.012	17	1.254	33	1.647	49	2.400
2	1.024	18	1.273	34	1.680	50	2.471
3	1.037	19	1.292	35	1.714	51	2.545
4	1.050	20	1.313	36	1.750	52	2.625
5	1.063	21	1.333	37	1.787	53	2.710
6	1.077	22	1.355	38	1.826	54	2.800
7	1.091	23	1.377	39	1.867	55	2.897
8	1.105	24	1.400	40	1.909	56	3.000
9	1.120	25	1.424	41	1.953	57	3.111
10	1.135	26	1.448	42	2.000	58	3.231
11	1.151	27	1.474	43	2.049	59	3.360
12	1.167	28	1.500	44	2.100	60	3.500
13	1.183	29	1.527	45	2.154	61	3.652
14	1.200	30	1.556	46	2.211	62	3.818
15	1.217	31	1.585	47	2.270	63	4.000

GAIN_PGA : **ADCx_GAIN_PGA** register

Gain Ratio = (GAIN_PGA*3/(63+(63-GAIN_PGA)*3))+1

— MG32F02N128/K128/N064/K064

User can set the gain value from 1 to 128 by setting **ADCx_GAIN_PGA** and **ADCx_GAIN_MUL** registers. The **ADCx_GAIN_PGA** register can be set gain value 1, 2, 3, 4, 5, 6, 7 and 8. The **ADCx_GAIN_MUL** register can be set gain value 1 and 16.

The PGA gain value is set by the formula:

$$\text{PGA Gain} = \text{ADCx_GAIN_PGA} * \text{ADCx_GAIN_MUL}$$

The following table is showing the ADC PGA gain ratio by register setting.

Table 27-6. ADC Gain Calculation - x1 ~ x128

GAIN_MUL	GAIN_PGA	Gain Ratio	GAIN_MUL	GAIN_PGA	Gain Ratio
x1	x1	x1	x16	x1	x16
	x2	x2		x2	x32
	x3	x3		x3	x48
	x4	x4		x4	x64
	x5	x5		x5	x80
	x6	x6		x6	x96
	x7	x7		x7	x112
	x8	x8		x8	x128

GAIN_MUL, GAIN_PGA: **ADCx_GAIN_MUL**, **ADCx_GAIN_PGA** register

Gain Ratio = GAIN_MUL * GAIN_PGA

● PGA Offset Calibration

There is one **ADCx_OFFT_PGA** register to adjust and calibrate the input offset of PGA. User can enable the PGA offset calibration in **ADCx_CAL_POFFT** register. When enables, use can set a 0x20 value to this register and get the PGA offset calibration status bit in **ADCx_POF** register. Increase or decrease the value continuously until the PGA offset calibration status bit is changing. Set the final setting value to **ADCx_OFFT_PGA** register and the input offset of PGA calibration is complete.

● PGA Bias Control

There is one register of **ADCx_BUF_BIAS** for PGA bias current setting. The register is default "0" for normal condition. According to user application, user can set this register to "1" to increase the bias current and input bandwidth of the PGA. So the PGA operation speed is able to be raised for higher frequency of input ADC signal.

[Notify]: The **ADCx_BUF_BIAS** register is only supported for MG32F02A128/U128/A064/U064.

27.8.6. ADC Calibration

For user application and environment, the operation voltage and temperature will affect the ADC characters.

The chip is built-in the offset calibration function for ADC macro and user can calibrate the ADC conversion offset error by setting **ADCx_OFFT_ADC** register. This register is used as an offset code to adjust the ADC reference bottom voltage in order to calibrate the ADC output code. ADC output code is equal ADC conversion code minus this register offset code. Value 0x00, 0x01 to 0x0E, 0x0F are adjusted offset -31LSB, -29LSB to -3LSB, -1LSB. Values 0x10, 0x11 to 0x1E, 0x1F are adjusted offset 1LSB, 3LSB to 29LSB, 31LSB.

For the ADC offset calibration, user needs to input the voltage of **VSSA** and sets the **ADCx_OFFT_ADC** register to 0 (-31LSB). Usually user can get a zero ADC code and set the **ADCx_OFFT_ADC** register from -29LSB to 31LSB in sequence until the ADC code is changing to non-zero. Then user can fix the setting of **ADCx_OFFT_ADC** register and finish the ADC offset calibration.

27.8.7. ADC Operation in SLEEP

If the ADC is turned on in **SLEEP** mode, it will consume a little power. The power consumption can be reduced by turning off the ADC hardware (**ADCx_EN=0**) before entering **SLEEP** mode.

If software triggers the ADC operation in **SLEEP** mode, the ADC will finish the conversion and set the ADC interrupt flag. When the ADC interrupt enable (**ADCx_E1CNV_IE** or **ADCx_ESCNV_IE**) is set, the ADC interrupt will wake up CPU from **SLEEP** mode.

27.9. ADC Conversion Sequence

27.9.1. ADC Conversion Configuration and Sequence

- **Prepare ADC Conversion**

Prior to using the ADC function, the user should:

- Turn on the ADC hardware by setting the **ADCx_EN** bit
- Configure the ADC resolution by **ADCx_RES_SEL** bits
- Configure the ADC input clock by **ADCx_CK_SEL** and **ADCx_CK_DIV** bits
- Select the analog input channel by **ADCx_CH_SEL** and **ADCx_CH_MUX** bits
- Configure the selected input to the Analog-Input-Only mode by **PA_IOMn** registers (n={0~15})
- Configure ADC result arrangement by **ADCx_ALIGN_SEL** bit.

- **ADC Input Channel Selection**

User can set **ADCx_CH_SEL** and **ADCx_CH_MUX** registers to select the ADC input channel source. Refer the section of "[ADC Input Channels](#)" for more information.

- **ADC Conversion Mode**

The ADC support auto-sampling and trigger by external pin, internal events and software bit. User can set **ADCx_CONV_MDS** and **ADCx_TRG_CONT** registers to configure the ADC conversion mode. Refer the section of "[ADC Conversion Mode](#)" for more information.

- **ADC Conversion Start**

When the ADC setting of conversion is configuration complete, user needs to set **ADCx_START_SEL** register to configure the ADC trigger source and starts the ADC conversion. The ADC start trigger source is including of (1) software control register bit **ADCx_START** (2) external trigger pin **ADC0_TRG** (3) internal module trigger event **TM00_TRGO**, **CMP0_OUT**, **CMP1_OUT**, **TM01_TRGO**, **TM20_TRGO** and **TM36_TRGO**.

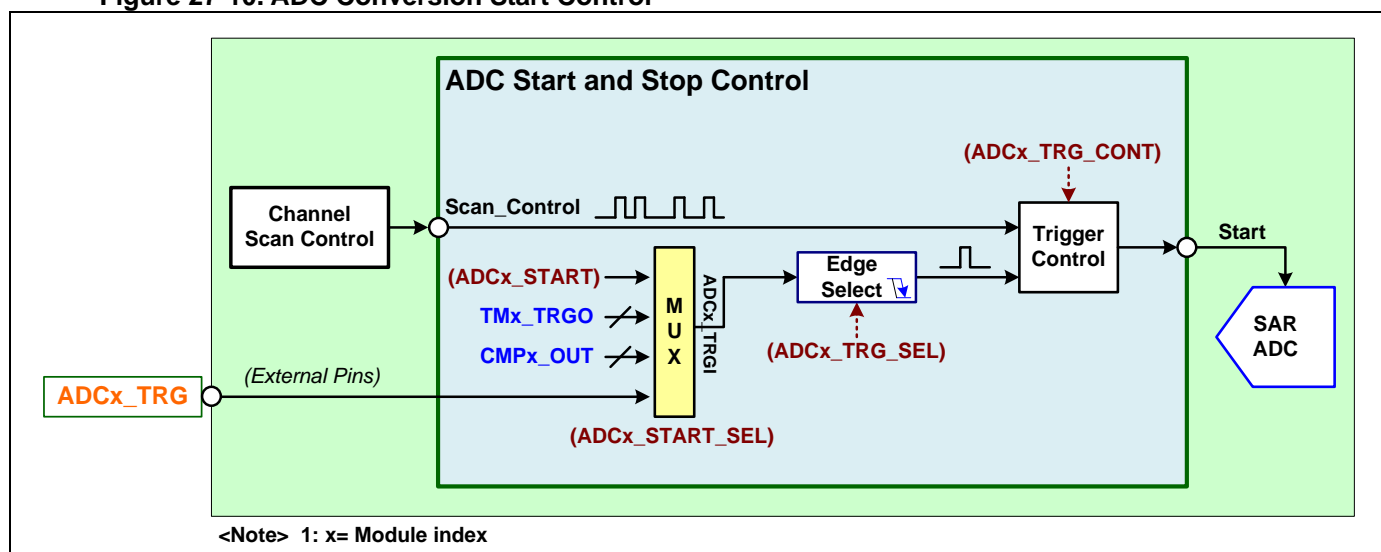
[Notify]: The TM20_TRGO is not supported as ADC start trigger signal for MG32F02A032.

User can trigger the ADC conversion by software register of **ADCx_START**, internal signals of timer trigger output **TMx_TRGO** or internal signals of analog comparator's result output **CMPx_OUT**.

When selects trigger by software register, the **ADCx_START** register is set by software to start ADC conversion and clear by hardware after the ADC conversion is complete.

When selects trigger by internal trigger signal, the **ADCx_TRG_SEL** register is used to set the trigger edge of rising edge, falling edge or dual edge.

Figure 27-10. ADC Conversion Start Control



27.9.2. ADC Conversion Hold

During the ADC conversion, user can set the **ADCx_HOLD** register to hold the ADC conversion and clear the **ADCx_HOLD** register to keep continuous for the ADC conversion.

27.9.3. ADC Conversion Overrun

When an ADC conversion is complete and the previous conversation ADC data is not yet to take out by software, the ADC conversion data is overrun. User can configure to overwritten the data register by new conversion ADC data or keep previous ADC conversion data by setting the **ADCx_OVR_MDS** register beforehand for the overrun condition.

Also user can enable the hardware “ADC Wait” function to avoid the conversion overrun condition. Refer the section of “[ADC Wait and Auto-Off](#)” for more information.

27.10. ADC Conversion Mode

The ADC is supported five conversion modes of One-Shot, One-Continue, Channel Scan-One, Channel Scan-Continue and Scan-Loop. User can set **ADCx_CONV_MDS** and **ADCx_TRG_CONT** registers to configure the ADC conversion mode.

Table 27-7. ADC Conversion Mode Control

ADC Conversion Mode	ADC Register		Conversion Function
	CONV_MDS	TRG_CONT	
One-Shot Mode	One	Disable	Convert one ADC data from single channel after one start trigger and then stop conversion.
One-Continue Mode	One	Enable	Convert one ADC data from single channel after one start trigger and then hardware automatically start next ADC conversion continuously and does not stop conversion.
Scan-One Mode	Scan	Disable	Convert one ADC data from one selected channel after one start trigger, then stops conversion. To follow and convert next ADC data from next selected channel after next start trigger. Run the same steps for all selected channels.
Scan-Continue Mode	Scan	Enable	Convert multiple ADC data continuously from multiple selected channels after one start trigger and then stop conversion when scan finished for all selected channels.
Scan-Loop Mode	Loop	X	When ADC conversion start by one start trigger, hardware does the channel scan ADC conversion one-by-one until finish the conversion for all selected channels. And hardware automatically starts next channel scan ADC conversion continuously and does not stop.

[Notify]: The TM20_TRGO is not supported as ADC start trigger signal for MG32F02A032.

27.10.1. One Shot and Continuous Conversion

● Starting One-Shot Conversion

User needs to select One-Shot conversion in **ADCx_CONV_MDS** and **ADCx_TRG_CONT** registers.

Now, user can set the **ADCx_START** bit to start the A-to-D conversion. The conversion time is controlled by bits **ADCx_CK_SEL** and **ADCx_CK_DIV**. Once the conversion is completed, the hardware will automatically clear the **ADCx_E1CNVF** bit, set the interrupt flag **ADCx_E1CNVF** and load the 12 bits of conversion result into **ADCx_DAT0** (according to **ADCx_ALIGN_SEL** bit) simultaneously.

As described above, the interrupt flag **ADCx_E1CNVF**, when set by hardware, shows a completed conversion.

Thus two ways may be used to check if the conversion is completed: (1) Always polling the interrupt flag **ADCx_E1CNVF** by software; (2) Enable the ADC interrupt by setting bits **ADCx_E1CNV_IE**, and then the CPU will jump into its Interrupt Service Routine when the conversion is completed. Regardless of (1) or (2), the **ADCx_E1CNVF** flag should be cleared by software before next conversion.

● Starting One-Continuous Conversion

User needs to select One-Continue conversion in **ADCx_CONV_MDS** and **ADCx_TRG_CONT** registers. If user selects the ADC trigger mode register **ADCx_START_SEL** to free-run or other trigger source except manual mode by software **ADCx_START** bit, then the ADC will keep conversion continuously unless **ADCx_EN** is cleared or configure ADC to manual mode.

● Conversion Result

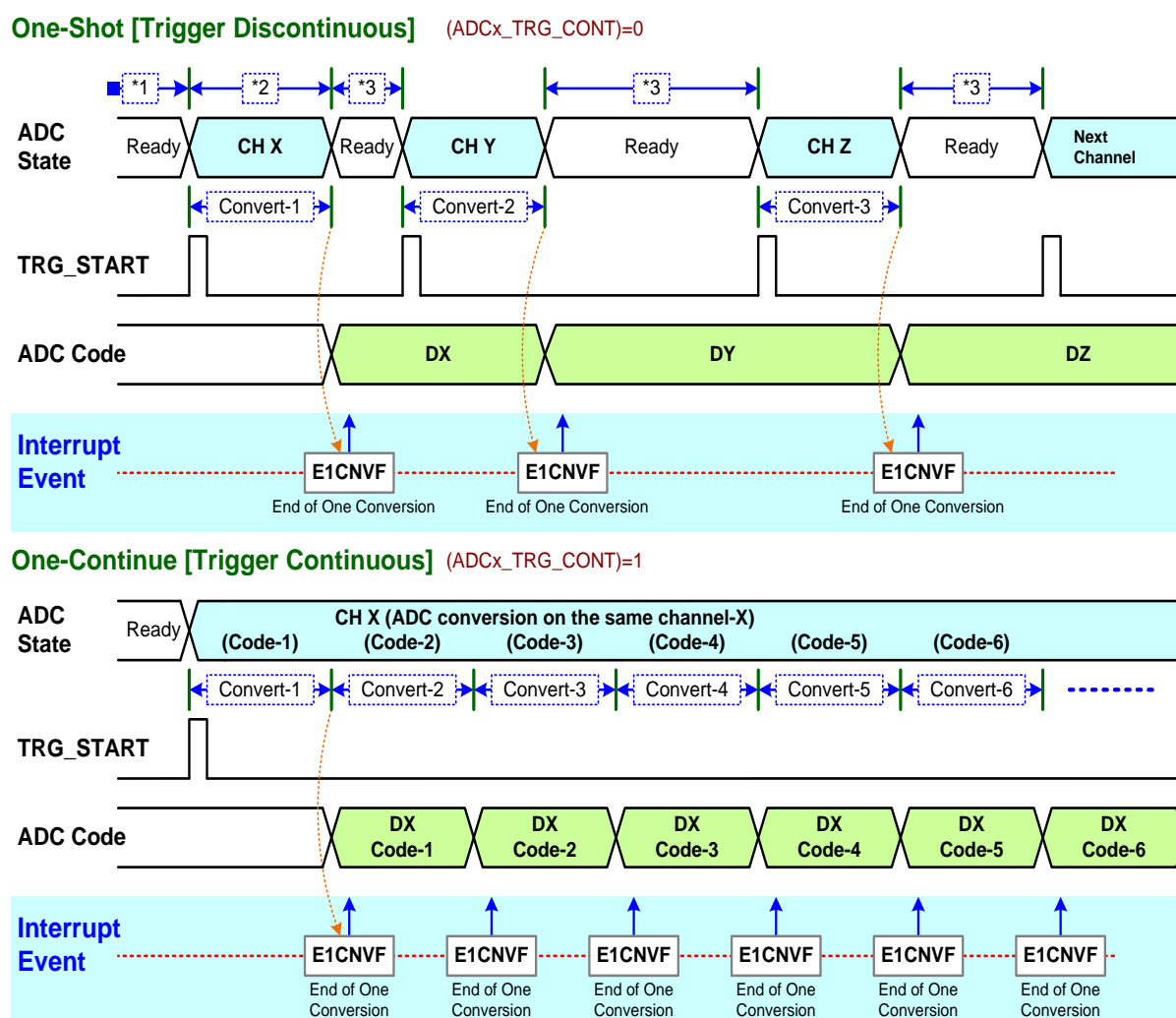
The result of the ADC is stored in a left-aligned or right-aligned 16-bit data register **ADCx_DAT0**.

After the conversion is complete and the **ADCx_E1CNVF** change high, the ADC conversion is complete and output the ADC code. For single ended conversion, the ADC output code is:

$$\text{ADC Code} = \frac{\text{VIN} * 4096}{\text{VREF}}$$

The ADC output code will be adjusted by the ADC output control blocks and store the conversion result data to the **ADCx_DAT0** register. Refer the section of “[ADC Output Control](#)” for more information.

Figure 27-11. ADC Conversion Mode – One Channel



<Note-1> Software set ADC channel by setting (**ADCx_CH_MUX**) .

<Note-2> Input signal sample and hold for ADC conversion.

<Note-3> ADC ready and wait to change channel for next conversion trigger.

<Note-4> {X,Y,...,Z} indicate the enabled channels those are following CH(channel) and D(Data) .

27.10.2. Channel Scan Conversion

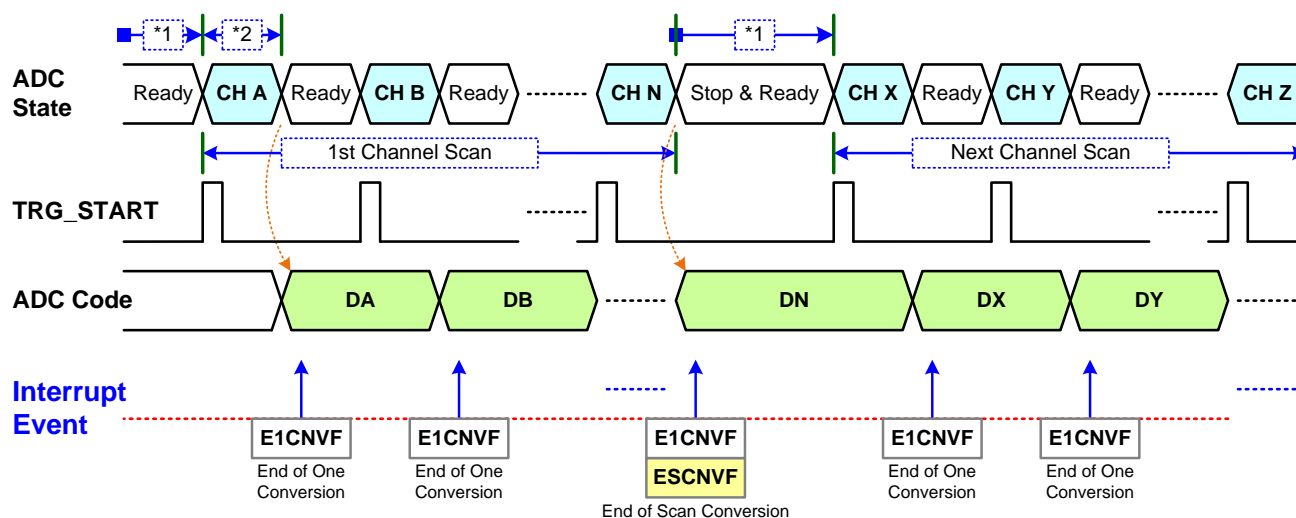
User can select channel Scan-One or Scan-Continue conversion in **ADCx_CONV_MDS** and **ADCx_TRG_CONT** registers. The **ADCx_CH_MSK** register sets the ADC channel selection mask for sequence channel scan. When selects mask, the related channel is masked and disabled from the channel scan loop.

Also user can set the **ADCx_START** bit to start the A-to-D conversion. Also the **ADCx_E1CNVF** is asserted after the single conversion of each channel has completed, the conversion result can be found in **ADCx_DAT0** register and the **ADCx_DAT_CH** indicates the activated channel.

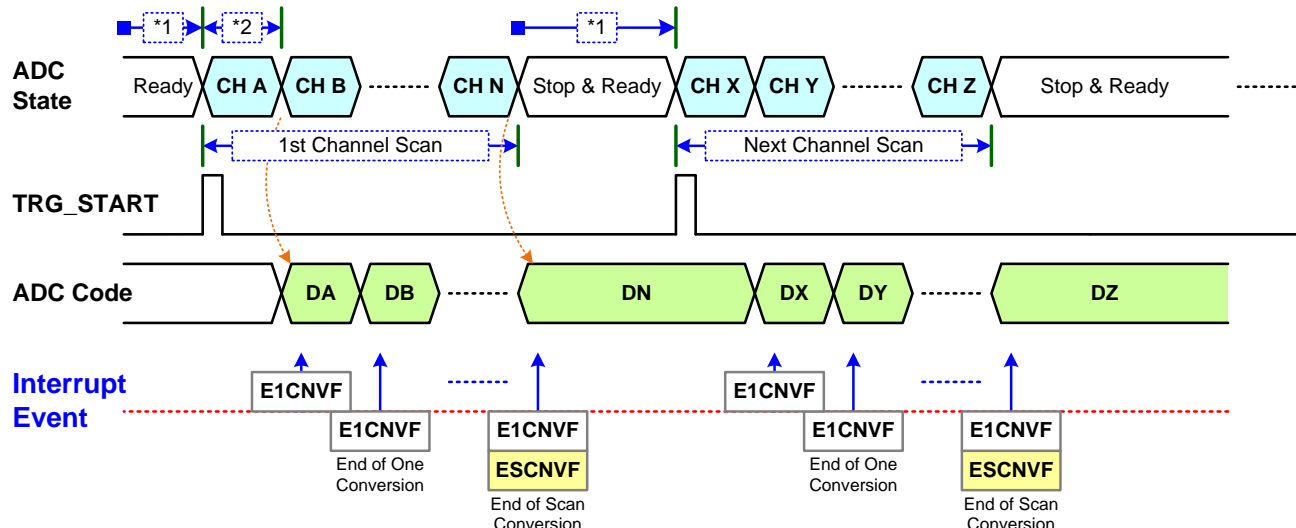
After the one-loop sequence scan has completed for all enabled channels, the ADC conversion is stop and **ADCx_ESCNVF** flag is asserted. User can directly poll this flag or run firmware service by interrupt trigger if **ADCx_ESCNV_IE**=1.

Figure 27-12. ADC Conversion Mode – Channel Scan

Scan-One [Trigger Discontinuous] (**ADCx_TRG_CONT**)=0



Scan-Continue [Trigger Continuous] (**ADCx_TRG_CONT**)=1



<Note-1> Software set ADC channels by setting (**ADCx_CH_MSKn**) before channel scan ADC conversion .

<Note-2> Input signal sample and hold for ADC conversion.

<Note-3> {A,B,...,N}/{X,Y,...,Z} indicate the enabled channels those are following CH(channel) and D(Data) .

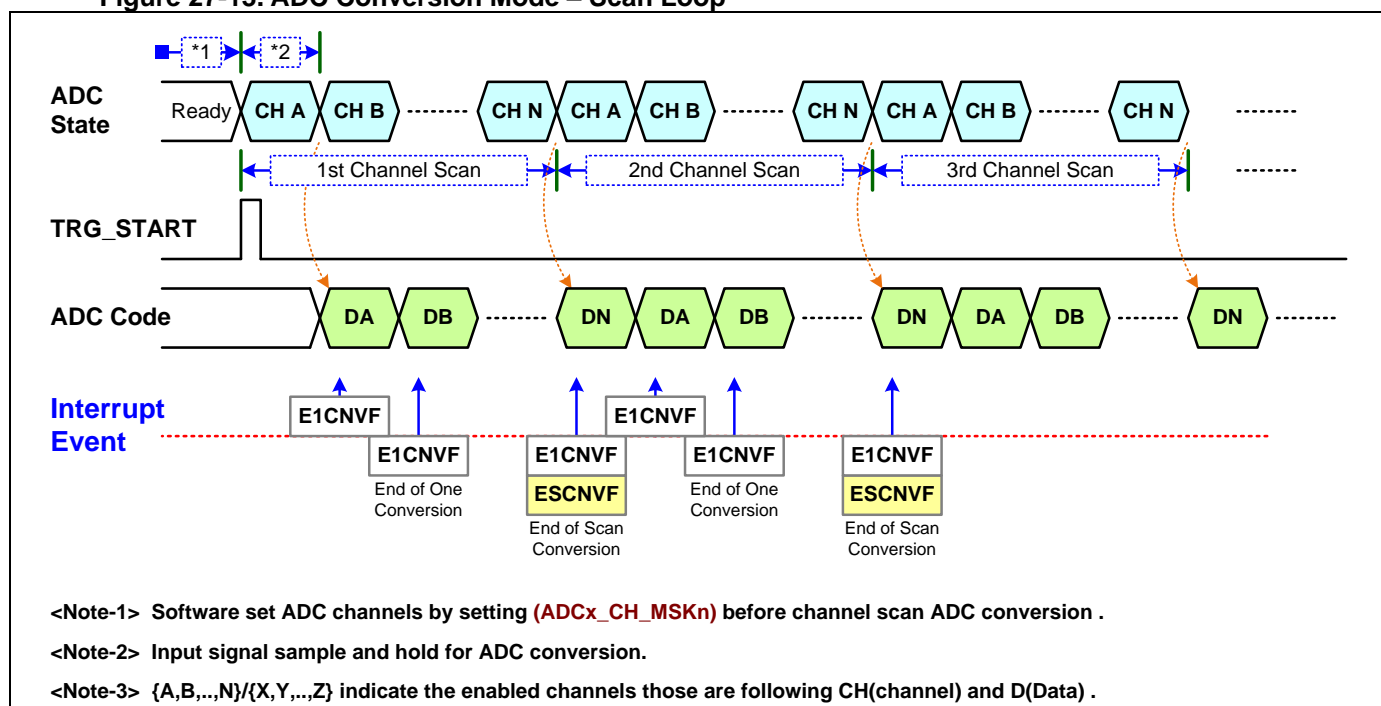
27.10.3. Scan Loop Conversion

User can select Scan-Loop conversion in **ADCx_CONV_MDS** register. Same as channel Scan-One or Scan-Continue conversion, user needs to set **ADCx_CH_MSK** registers.

The same, user can set the **ADCx_START** bit to start the A-to-D conversion and go the following procedures. After the one-loop sequence scan has completed, the ADC conversion is not stop and go to convert data continuously from defined first channel until sets **ADCx_START**=0 or stops the trigger-start signal for other source selection in **ADCx_START_SEL** register.

The following diagram is showing the ADC sequence scan hardware control flow.

Figure 27-13. ADC Conversion Mode – Scan Loop



27.10.4. Input Channel Change

When selects One Shot mode, user can change the input channel multiplexer at the time of one ADC data conversion end or one ADC data sampling end by software.

User can enable the interrupt of one-time conversion end in **ADCx_ESCNV_IE** register. The chip will assert the E1CNVF flag (**ADCx_E1CNVF**) at the end of each conversion of a channel and user can change the input channel at the time. Also user can enable the interrupt of ADC sampling end flag in **ADCx_ESMPF_IE** register. The chip will assert the ESMPF flag (**ADCx_ESMPF**) at the end of the sampling phase and user can change the input channel at the time.

When selects Channel Scan or Loop Scan modes, hardware will auto change the input channel to next input channel number. The ADC will change the input channel multiplexer at the time of one ADC data conversion end or one ADC data sampling end by setting **ADCx_CH_CHG** register.

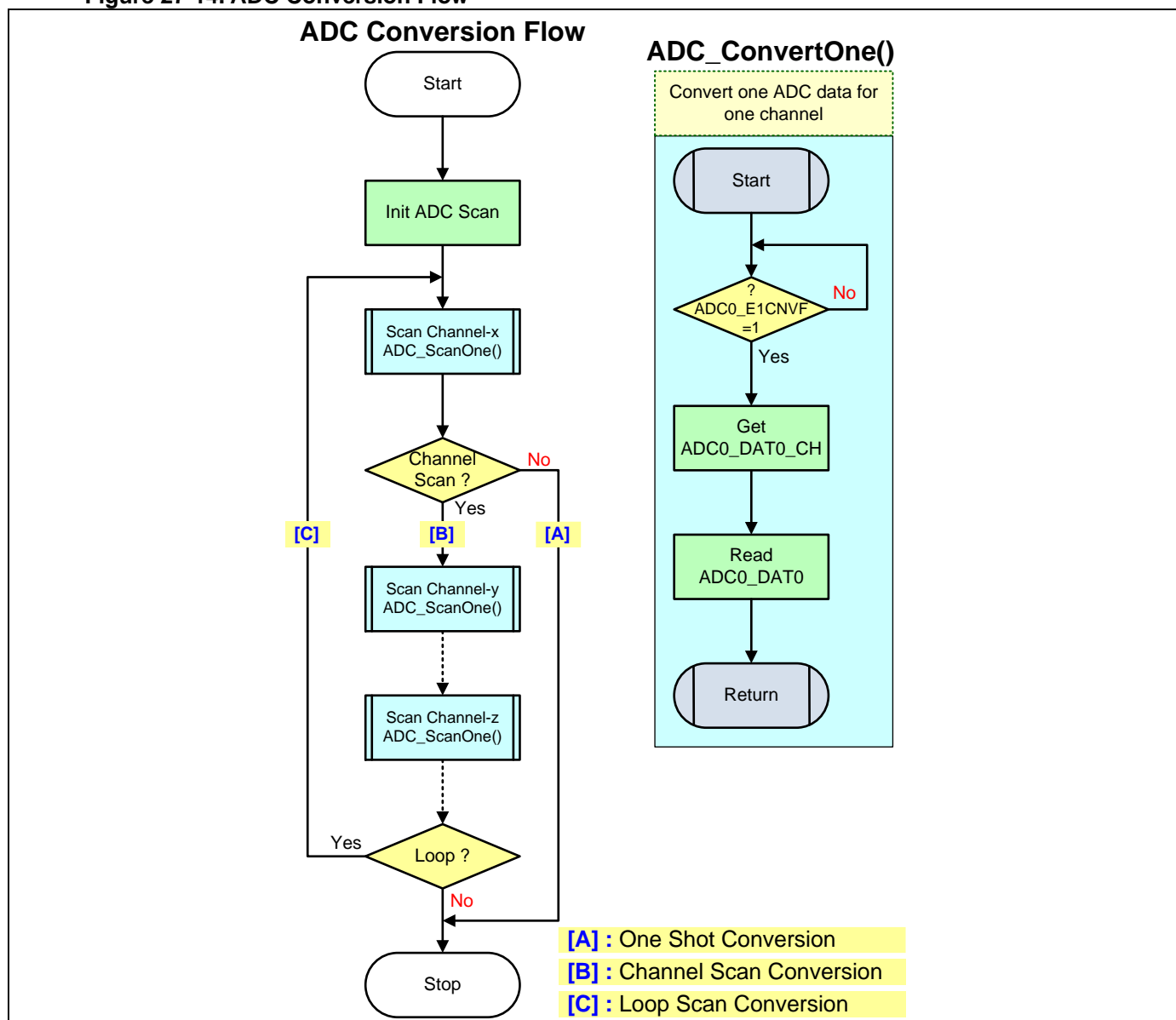
The next input channel number is controlled by the sequential channel numbers which are defined by setting **ADCx_CH_MSK** registers. For example, the channel 2~8 and 11~13 are masked. Then the channel scan will process by channel sequence of 0,1,9,10,14 and 15.

27.10.5. ADC Conversion Flow

The following diagram is showing the ADC conversion flow. The [A] flow is indicated for One Shot ADC conversion. The [B] flow is indicated for Channel Scan ADC conversion. The [C] flow is indicated for Loop Scan ADC conversion.

The ADC_ConverOne () flowchart is an example of software flow for one ADC conversion data control.

Figure 27-14. ADC Conversion Flow



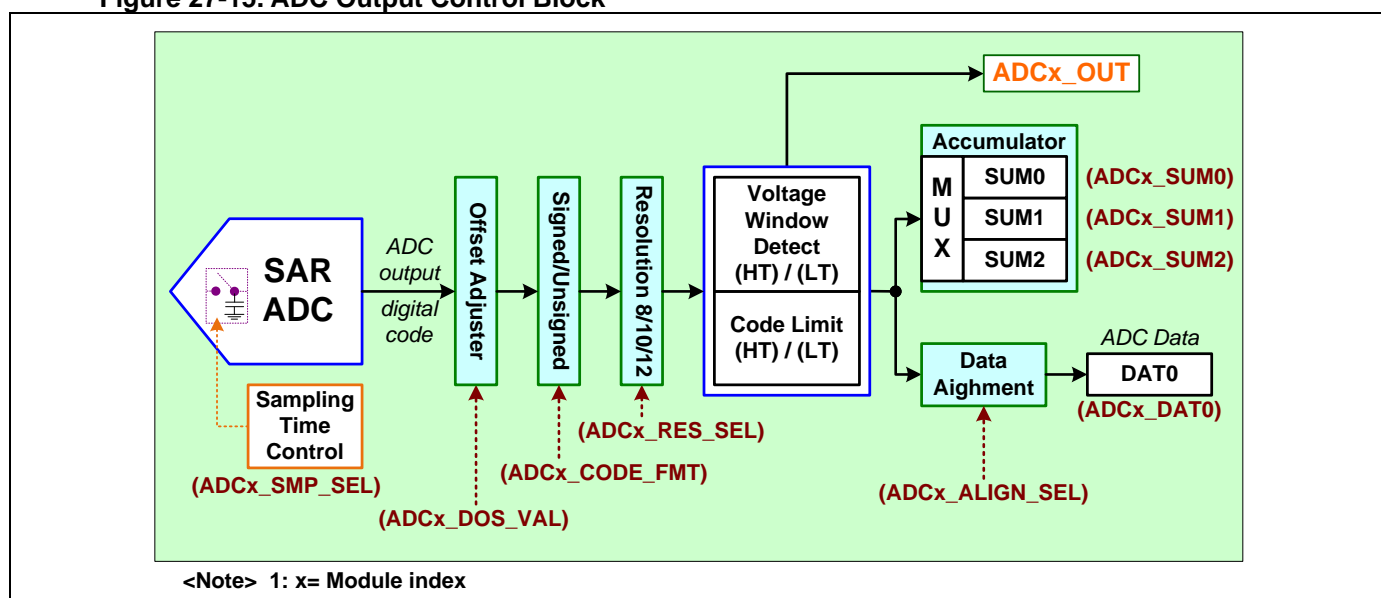
27.11. ADC Output Control

When an ADC conversion is complete, the ADC raw code is generated and sends to the ADC output control blocks those are including of Digital Offset Adjuster, Signed Code Converter, Digital Resolution Adjuster, Voltage Window Detector, Code Limiter and Data Alignment Adjuster.

The ADC output code will be adjusted by the ADC output control blocks and store the conversion result data to the **ADCx_DAT0** register. Refer the section of “[ADC Data Register](#)” for more information.

An extra accumulator is embedded to accumulate the sequential ADC data with programmable data number and records the sum to the summary registers. Refer the section of “[Data Sum](#)” for more information.

Figure 27-15. ADC Output Control Block



27.11.1. Digital Offset Adjuster

The ADC is built-in a digital code offset adjuster, user can set a 2s complement value by setting **ADCx_DOS_VAL** register for code offset adjustment. So the ADC output code will be adjusted by the setting value and output to Signed Code Converter.

Table 27-8. ADC Digital Offset Process

Digital Offset Process Output	Register				
	ADCx_DOS_VAL				
ADC Code+15	0	1	1	1	1
ADC Code+14	0	1	1	1	0
....				
ADC Code+2	0	0	0	1	0
ADC Code+1	0	0	0	0	1
ADC Code+0	0	0	0	0	0
ADC Code-1	1	1	1	1	1
ADC Code-2	1	1	1	1	0
....				
ADC Code-15	1	0	0	0	1
ADC Code-16	1	0	0	0	0

<Note> ADCx_DOS_VAL Msb = Sign bit

27.11.2. Signed Code Converter

For application, the ADC code can be adjusted to unsigned code for single-end mode.

- **Code Definition for Single-End Mode**

The following table is showing the signed and unsigned data formats of ADC code for Single-End mode.

Table 27-9. ADC Single-End Data Format Definitions

Format	ADC Input Level (LSB)	ADC Signed/Unsigned output Code											
		B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
Unsigned	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0	0	0	0	0	1
											
	2047	0	1	1	1	1	1	1	1	1	1	1	1
	2048	1	0	0	0	0	0	0	0	0	0	0	0
											
	4094	1	1	1	1	1	1	1	1	1	1	1	0
	4095	1	1	1	1	1	1	1	1	1	1	1	1

<Note> 1 LSB = VREF Voltage / 4096

- **Code Definition for Differential Mode**

The following table is showing the signed and unsigned data formats of ADC code for Differential mode.

Table 27-10. ADC Differential Data Format Definitions

Format	ADC Input Level (LSB)	ADC Signed/Unsigned output Code											
		B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
2's Complement (Signed)	-2048	1	0	0	0	0	0	0	0	0	0	0	0
	-2047	1	0	0	0	0	0	0	0	0	0	0	1
											
	-1	1	1	1	1	1	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0	0	0	0	0	1
											
	2046	0	1	1	1	1	1	1	1	1	1	1	0
	2047	0	1	1	1	1	1	1	1	1	1	1	1

<Note> 1 LSB = VREF Voltage / 4096
B11 = Sign bit for 2's Complement mode

27.11.3. Resolution and Data Alignment

User can set **ADCx_RES_SEL** register to select the ADC output code resolution. Also user can set **ADCx_ALIGN_SEL** registers to select the left-alignment or right-alignment for code format of ADC data register.

The following table is showing data alignment format for data register **ADCx_DATn**.

Table 27-11. ADC Data Alignment Definitions

Alignment	ADCx_DATn (ADC data register)															
	B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
12bit-Right	Z	Z	Z	Z	S ADC Output Code[11:0]											
10bit-Right	Z	Z	Z	Z	Z	Z	S ADC Output Code[9:0]									
8bit-Right	Z	Z	Z	Z	Z	Z	Z	Z	S ADC Output Code[7:0]							
12bit-Left	S ADC Output Code[11:0]												x	x	x	x
10bit-Left	S ADC Output Code[9:0]										x	x	x	x	x	x
8bit-Left	S ADC Output Code[7:0]								x	x	x	x	x	x	x	x

ADC Output Code : ADC output through Digital Offset and Signed/Unsigned adjuster output code

x: unknown

S: Sign bit for signed(2's complement) mode or Msb for unsigned mode

Z: Sign bit for signed(2's complement) mode or 0 for unsigned mode

27.11.4. ADC Data Register

When an ADC conversion is complete, the ADC output code will be adjusted by the ADC output control blocks and store the conversion result data to the **ADCx_DAT0** register.

There are one read only register of **ADCx_DAT0_CH** to indicate the input channel index for the current ADC data. Also one complete flag in **ADCx_DAT0_CF** register bit and one overrun flag in **ADCx_DAT0_OVRF** register bit for this current ADC data.

Others, there are three extra flags of voltage window detection in **ADCx_DAT0_WDLF**, **ADCx_DAT0_WDIF** and **ADCx_DAT0_WDLF** register bits for this current ADC data.

The following table is the ADC code range example for ADC 12-bit resolution output.

[Notify]: The ADC signed code is not supported for MG32F02A032.

Table 27-12. ADC Data Code Example

Alignment	ADCx_DATn (ADC data register)																Value
	B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0	
12bit-Right (Unsigned)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	2047
	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	2048
	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	4095
12bit-Right (Signed) (*1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	2047
	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	-2048
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-1
12bit-Left (Unsigned)	0	0	0	0	0	0	0	0	0	0	0	0	x	x	x	x	0
	0	1	1	1	1	1	1	1	1	1	1	1	x	x	x	x	2047
	1	0	0	0	0	0	0	0	0	0	0	0	x	x	x	x	2048
	1	1	1	1	1	1	1	1	1	1	1	1	x	x	x	x	4095
12bit-Left (Signed) (*1)	0	0	0	0	0	0	0	0	0	0	0	0	x	x	x	x	0
	0	1	1	1	1	1	1	1	1	1	1	1	x	x	x	x	2047
	1	0	0	0	0	0	0	0	0	0	0	0	x	x	x	x	-2048
	1	1	1	1	1	1	1	1	1	1	1	1	x	x	x	x	-1

ADC Output Code : ADC output through Digital Offset and Signed/Unsigned adjuster output code

x: unknown

*1: Only support for MG32F02A132/072

27.11.5. Voltage Window Detect

The ADC can compare the input voltage by a threshold window. User can enable the voltage window detect function in **ADCx_WIND_EN** register and select “Single” or “All” mode by setting **ADCx_WIND_MDS** register. When selects “Single” mode, the selection channel in **ADCx_CH_MUX** register which data is detected for the voltage window detector. When selects “All” mode, the all selection channel data are detected for the voltage window detector.

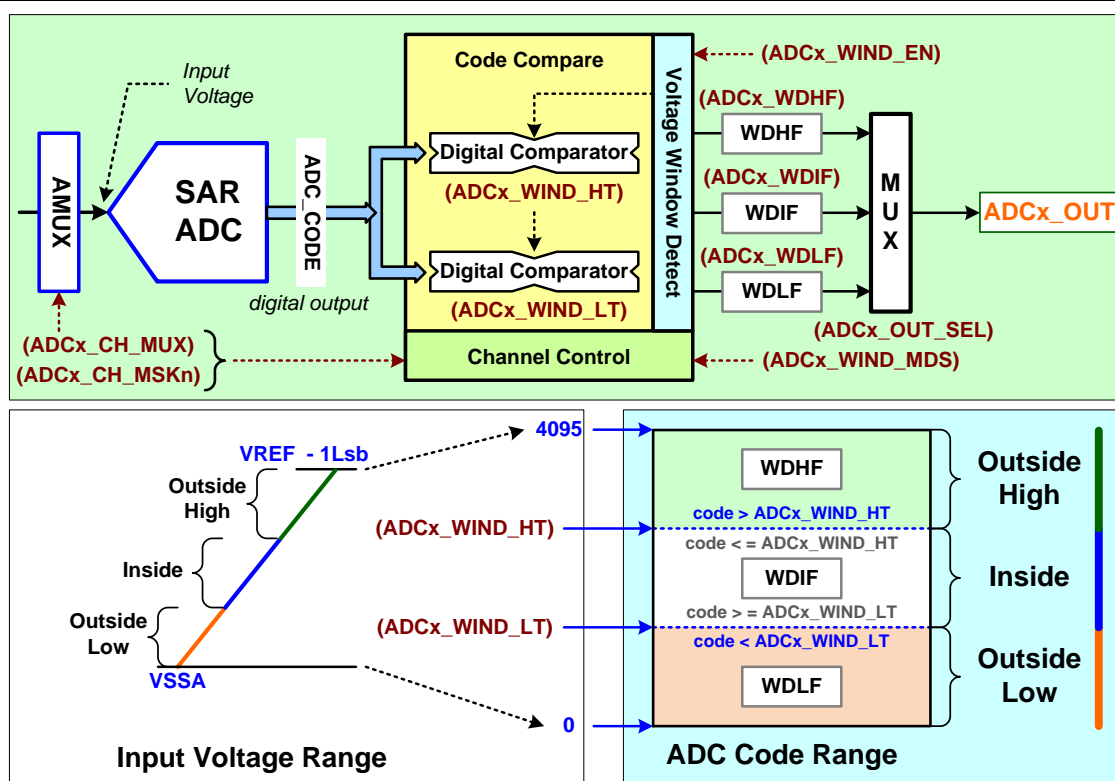
The threshold windows are defined and separated to three areas by **ADCx_WIND_LT** and **ADCx_WIND_HT** registers. The comparison result is reflected to the status flags of WDLF, WDIF, WDHF (**ADCx_WDLF**, **ADCx_WDIF** and **ADCx_WDHF**) and indicates outside low, inside, outside high areas. When the compared ADC code is large or equal to the threshold value of **ADCx_WIND_HT** register, the WDHF flag will be asserted. When the compared ADC code is small or equal to the threshold value of **ADCx_WIND_LT** register, the WDLF flag will be asserted. Other ADC code will cause to assert WDIF flag.

It also can do like as an analog watchdog to generation a reset event to Reset Source Controller. The analog watchdog feature allows the application to detect if the input voltage goes outside the user-defined higher or lower thresholds. See the “ADC interrupt control” for more information.

There is an output selection register of **ADCx_OUT_SEL** to select the window detected status (outside low, inside, outside high) or the internal data ready signal (**ADCx_RDY**) to output from **ADCx_OUT** port.

The following diagram is showing the ADC Voltage Window Detect block.

Figure 27-16. ADC Voltage Window Detect



<Note> 1: x= Module index

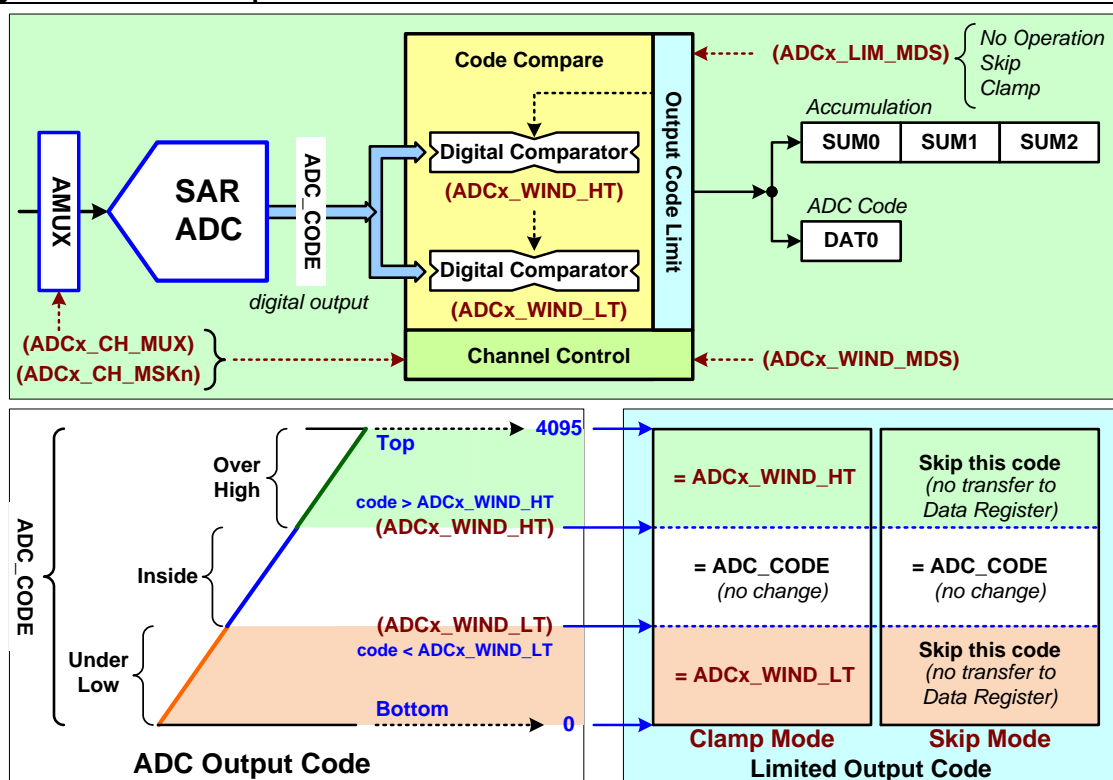
27.11.6. Output Code Limit

The ADC output code can be compared by a code limit area. User can select “Single” or “All” mode by setting **ADCx_WIND_MDS** register. When selects “Single” mode, the selection channel in **ADCx_CH_MUX** register which data is detected for the voltage window detector. When selects “All” mode, the all selection channel data are detected for the voltage window detector.

The code limit area is also defined by **ADCx_WIND_LT** and **ADCx_WIND_HT** registers. The output code limit function can select (1) No operation (2) Skip (3) Clamp and set by **ADCx_LIM_MDS** register.

The following diagram is showing the ADC Output Code Limit block.

Figure 27-17. ADC Output Code Limit



<Note> 1: x= Module index

27.12. ADC Data Sum

27.12.1. ADC Data Sum Accumulate

The ADC built-in one hardware accumulator for ADC output code. The accumulator is used to accumulate the sequential ADC data with programmable data number and records the sum to the summary registers. User can set the accumulated ADC data number in **ADCx_SUM_NUM** register. The ADC is supported three sum data registers of **ADCx_SUMn** ($n = \{0, 1, 2\}$) and user can get the accumulated sum from these registers.

User can select “Single” or “All” accumulation mode by setting **ADCx_SUM_MDS** register.

When selects “Single” mode for ADC one shot conversion mode, the **ADCx_SUM0_MUX** selection channel data is accumulated into **ADCx_SUM0**. When selects “All” mode, the all selection channel data are accumulated one-by-one into **ADCx_SUM0** only. When selects “Single” mode for ADC channel scan conversion mode, the **ADCx_SUM1_MUX/ADCx_SUM2_MUX** selection channel data are also separately accumulated into **ADCx_SUM1/ADCx_SUM2**. These registers can preset an initial value by user and accumulate the sum by ADC conversion data.

The following table is showing ADC Data Sum and Conversion Mode Control.

Table 27-13. ADC Data Sum and Conversion Mode Control

Conversion Mode		Separated Sum Function	All Data Sum Function
CONV_MDS	TRG_CONT	SUM_MDS = Single	SUM_MDS = All
One	Disable	ADC output data of the ADCx_SUM0_MUX selection channel is accumulated into ADCx_SUM0_DAT only after every ADC conversion finished. Also the data sum counter is increased by 1 after every ADC conversion finished until the counter value equals ADCx_SUM_NUM .	
One	Enable	ADC output data of the ADCx_SUM0_MUX selection channel is accumulated into ADCx_SUM0_DAT only after every ADC conversion finished. Also the data sum counter is increased by 1 after every ADC conversion finished until the counter value equals ADCx_SUM_NUM .	ADC output data of all selection channels is one-by-one accumulated into ADCx_SUM0_DAT only after every ADC conversion finished. Also the data sum counter is increased by 1 after every ADC conversion finished until the counter value equals ADCx_SUM_NUM .
Scan	Disable	ADC output data of the separated ADCx_SUMn_MUX selection channel is separately accumulated into ADCx_SUM0 , ADCx_SUM1 and ADCx_SUM2 after every ADC conversion finished. Also the data sum counter is increased by 1 after every ADC conversion finished until the counter value equals ADCx_SUM_NUM .	
Scan	Enable	ADC output data of the separated ADCx_SUMn_MUX selection channel is separately accumulated into ADCx_SUM0 , ADCx_SUM1 and ADCx_SUM2 after every ADC conversion finished. Also the data sum counter is increased by 1 after every time of channel scan finished until the counter value equals ADCx_SUM_NUM .	ADC output data of all selection channels is one-by-one accumulated into ADCx_SUM0_DAT only after every ADC conversion finished. Also the data sum counter is increased by 1 after every time of channel scan finished until the counter value equals ADCx_SUM_NUM .
Loop	X		

ADCx_SUMn_DAT: the data sum of ADC output codes ($x = \text{ADC module index}$, $n = \text{the data sum register index}$)

ADCx_SUMn_MUX: the selection of ADC channel multiplexer for the data sum **ADCx_SUMn_DAT**

ADCx_SUM_NUM: the total data count of **ADCx_SUMn_DAT**

The following table is showing ADC Data Sum Flags vs. Conversion Mode.

Table 27-14. ADC Data Sum Flags vs. Conversion Mode

Mode	Sum Complete Flag	Sum Overrun Flag	Sum Overflow Flag
CONV_MDS	SUMCF / SUMn_CF	SUMOVRF / SUMn_OVRF	SUMOF / SUMn_OF
One	Assert both ADCx_SUMCF and ADCx_SUMn_CF if related channel-n is accumulation complete.	Assert both ADCx_SUMOVRF and ADCx_SUMn_OVRF if related channel-n is accumulation overrun.	Assert both ADCx_SUMOF and ADCx_SUMn_OF if related channel-n is accumulation overflow.
Scan	Assert ADCx_SUMCF and all of ADCx_SUMn_CF if related channels are accumulation complete.	Assert ADCx_SUMOVRF and all of ADCx_SUMn_OVRF if related channels are accumulation overrun.	Assert ADCx_SUMOF and all of ADCx_SUMn_OF if related channels are accumulation overflow.
Loop			

27.12.2. ADC Data Sum Register

The accumulator accumulates the ADC data and stores the sum to **ADCx_SUMn** registers ($n = \{0, 1, 2\}$).

For each data sum register, there are one complete flag in **ADCx_SUMn_CF** register bit and one overrun flag in **ADCx_SUMn_OVRF** register bit. Also there are two flags of accumulation overflow and underflow in **ADCx_SUMn_OF** and **ADCx_SUMn_UF** register bits.

The following table is showing ADC Sum value range of unsigned format for ADC single-end mode and Signed format for ADC differential mode.

[Notify]: The ADC signed code is not supported for MG32F02A032.

Table 27-15. ADC SUM Value Range

Format	ADCx_SUMn (ADC accumulator sum registers)																Value
	B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0	
Unsigned	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	32767
	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	32768
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	65535
Signed (*1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	32767
	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-32768
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-1

<Note> Unsigned : using for Single-end mode ; Signed : using for Differential mode

*1 : Only support for MG32F02A132/072

27.12.3. ADC Data Sum Overrun

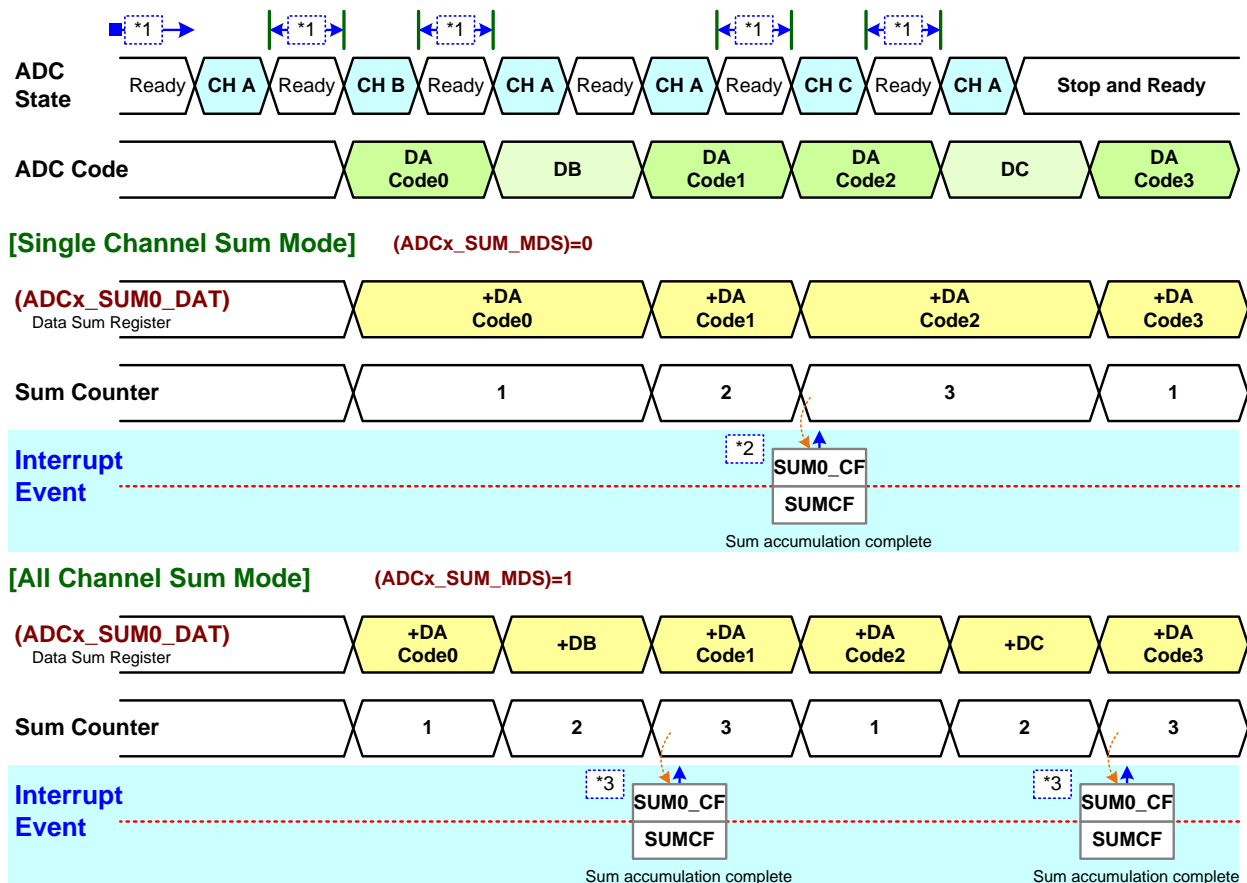
When an ADC sum accumulation is complete and the previous sum data is not yet to take out by software, the ADC data sum register is overrun. User can configure to overwritten the data register by new sum data or keep previous sum data by setting the **ADCx_SOVR_MDS** register beforehand for the overrun condition.

27.12.4. ADC Conversion Timing of Data Sum

- **ADC Data Sum Mode – One Shot or Channel Scan Discontinuous**

The following diagram is showing the control timing of “One Shot” or “Channel Scan Discontinuous” of ADC data sum mode.

Figure 27-18. ADC Data Sum Mode – One Shot or Channel Scan Discontinuous



<Note-1> Software set ADC channel by setting (ADCx_CH_MUX) .

<Note-2> The condition is (ADCx_SUM_NUM)=3 and (ADCx_SUM0_MUX)=A for example.

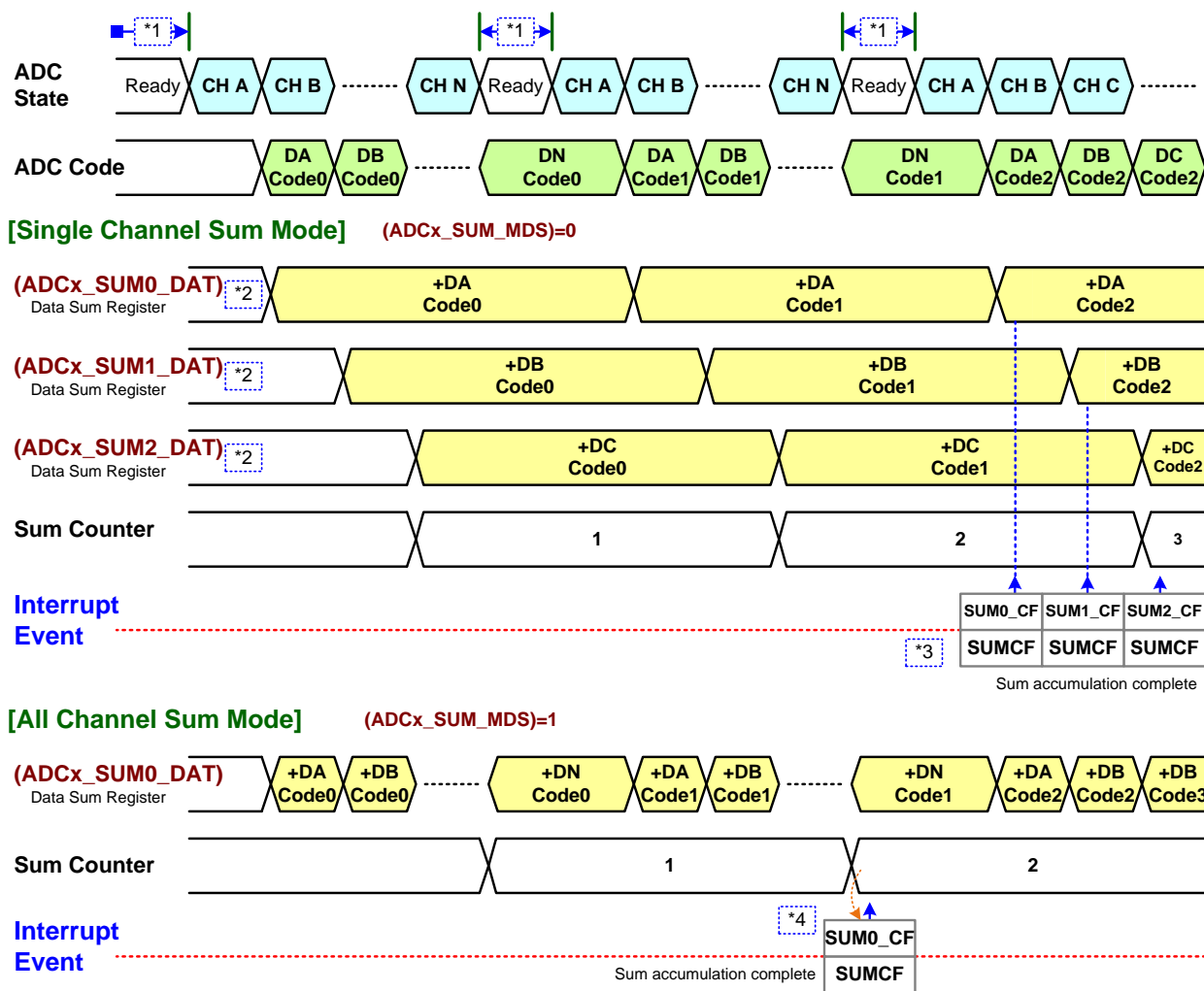
<Note-3> The condition is (ADCx_SUM_NUM)=3 and (ADCx_SUM0_MUX)=do not care for example.

<Note-4> {A,B,C,...} indicate the enabled channels those are following CH(channel) and D(Data) .

● ADC Data Sum Mode –Channel/Loop Scan

The following diagram is showing the control timing of “Channel Scan” or “Loop Scan” of ADC data sum mode.

Figure 27-19. ADC Data Sum Mode – Channel/Loop Scan



<Note-1> Software set ADC channels by setting (ADCx_CH_MSKn) before channel scan ADC conversion .

<Note-2> (ADCx_SUM0_MUX)=A , (ADCx_SUM1_MUX)=B , (ADCx_SUM2_MUX)=C for example.

<Note-3> The condition is (ADCx_SUM_NUM)=3 for example.

<Note-4> The condition is (ADCx_SUM_NUM)=2 and (ADCx_SUM0_MUX)=do not care for example.

<Note-5> {A,B,C...,N} indicate the enabled channels those are following CH(channel) and D(Data) .

27.13. ADC Wait and Auto-Off

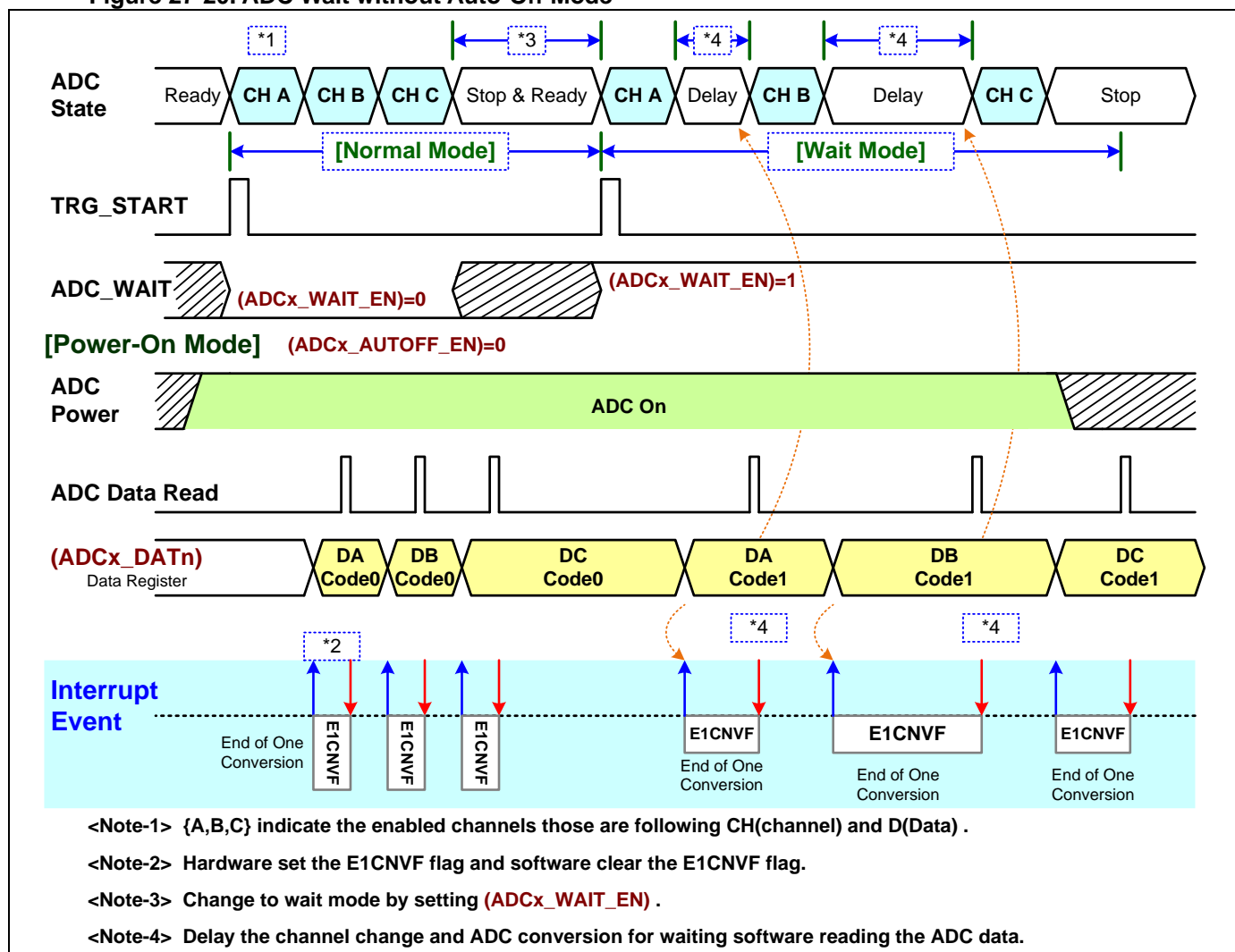
The ADC supports a wait mode function and enables by setting **ADCx_WAIT_EN** register. Usually, this function is able to prevent ADC overrun in application with low frequency ADC sampling clock.

Also ADC supports an auto-off mode function and enables by setting **ADCx_AUTOFF_EN** register. This function can force the ADC auto to enter power-off except during the active conversion phase. The ADC will automatically power on and recover to do a conversion of the input signal if detects the next conversion trigger start. This mode provides power savings in chip when there are long idle periods between ADC conversions.

One ADC power-on start time counter (SCNT) is built in for ADC auto-off mode. When user enables the ADC auto-off mode, the SCNT will be starting after ADC is triggered starting to wish ADC conversion. After the SCNT is counting underflow, the ADC will be starting to do data conversion. User can select the clock divider to divide the input APB clock to output as the SCNT counter clock by setting **ADCx_CK_SDIV** register. Also user can set the SCNT counter in **ADCx_SCNT** register value and make the counting time to fix the requested ADC enable time. Refer the chip Data Sheet about the ADC power-on enable time which is typical less than 5us.

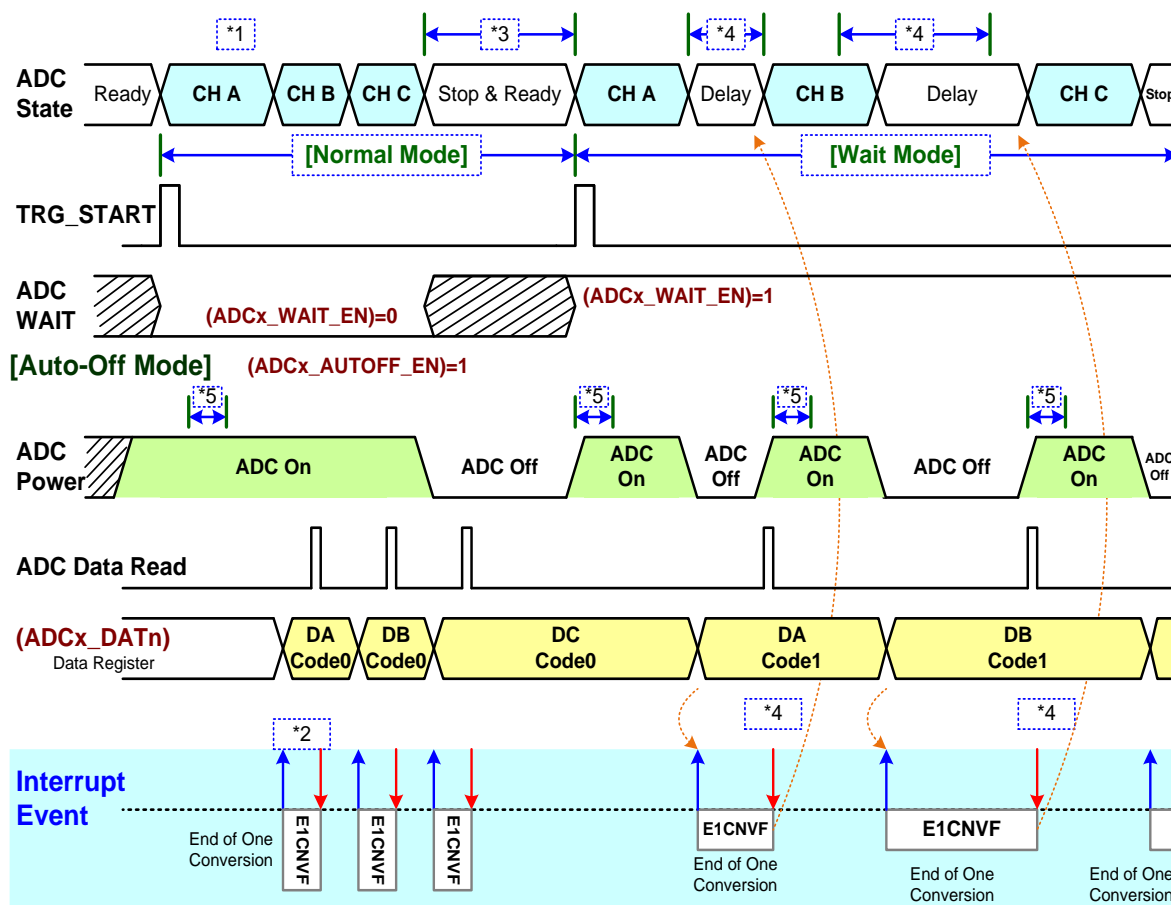
The following diagram is showing the ADC Wait without Auto-Off mode timing. The ADC data conversion is going on until the ADC data was reading by software or DMA hardware and clearing the E1CNVF flag.

Figure 27-20. ADC Wait without Auto-Off Mode



The following diagram is showing the ADC Wait with Auto-Off mode timing. The ADC macro is always power-off after ADC conversion is finished until next time ADC conversion is starting again.

Figure 27-21. ADC Wait with Auto-Off Mode Timing



<Note-1> {A,B,C} indicate the enabled channels those are following CH(channel) and D(Data) .

<Note-2> Hardware set the E1CNVF flag and software clear the E1CNVF flag.

<Note-3> Change to wait mode by setting ($ADCx_WAIT_EN$) .

<Note-4> Delay the channel change and ADC conversion for waiting software reading the ADC data.

<Note-5> ADC power-on start up time.

The following table is showing ADC auto-off mode start-up cycle by ADC conversion mode and Wait mode setting.

Table 27-16. ADC Auto-Off Mode Start-Up Cycle vs. Conversion Mode

ADC Conversion Mode			Wait Mode	Auto-Off Mode
Mode	CONV_MDS	TRG_CONT	WAIT_EN	Start-Up Cycle
One-Shot Mode	One	Disable	x	Every
One-Continue Mode	One	Enable	0	Once
			1	Every
Scan-One Mode	Scan	Disable	x	Every
Scan-Continue Mode	Scan	Enable	0	Once
			1	Every
Scan-Loop Mode	Loop	X	0	Once
			1	Every

<Sign> "Every" : Insert Start-Up cycle before every ADC conversion

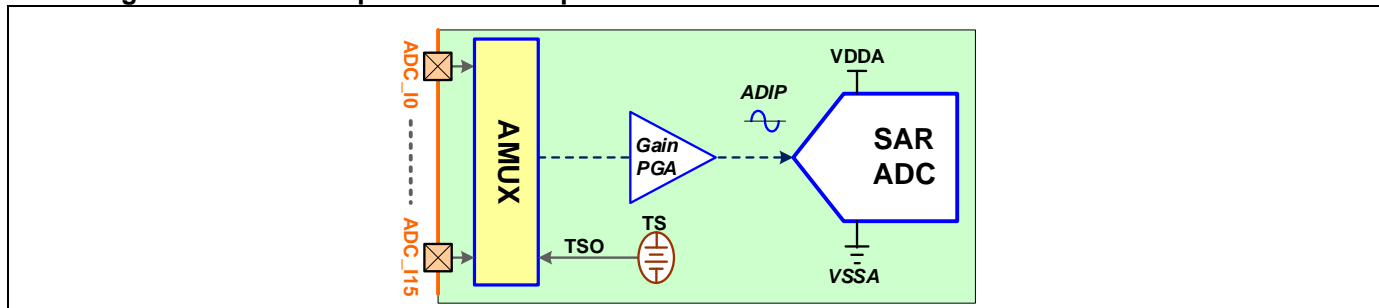
"Once" : Insert Start-Up cycle in first ADC conversion until stop this continuous ADC conversions

27.14. Temperature Sensor

The ADC is embedded one temperature sensor to measure the internal junction temperature of chip for product application. The temperature sensor is connected to the ADC internal channel TSO (Temperature Sensor Output). User can convert the TSO voltage to ADC code in order to calculate the temperature value.

User can enable the temperature sensor in ADCx_TS_EN register and select the analog input multiplex to internal ADC channel TSO by setting ADCx_CH_SEL and ADCx_CH_MUX registers. Refer the section of “ADC Input Channels” for more information about the ADC internal channel. It has an internal reference IVR24 auto select function by enabling in ADCx_TS_AUTO register. When internal channel TSO is selected in ADCx_CH_MUX and this bit is set 0, the ADC reference is force to select internal reference IVR24. Also the ADC reference will be set back to the register setting of ADCx_IVREF_SEL when ADC0_CH_MUX is set to other channel.

Figure 27-22. ADC Input from TS Output



As the junction temperature is measurement by the ADC, the ADC sampling time must be increased over the temperature sensor enable time which is typical about 50us. Refer the chip Data Sheet about the temperature sensor enable time.

There are two ADC codes those are recorded the voltages of temperature sensor at two temperatures in the read only registers of ADCx_TCAL0 and ADCx_TCAL1. These two temperature sensor data codes are loaded from the on-chip option bytes (OB) flash memory after chip reset. They are calibrated and stored in OB flash memory during the chip is manufactured from megawin manufactory. There are two chip configuration option registers (OR) of CFG_TEMP_CAL1 and CFG_TEMP_CAL1 in CFG module those are aliased to ADC0_TCAL1 and ADC0_TCAL1 registers.

Refer the section of “[Manufacturer Temperature Sensor Calibration Value](#)” in Hardware Option chapter for more information.

User can read these two codes' value and the ADC output code at current temperature, so user can calculate them to get the current temperature by following formula.

$$\text{Temperature (}^{\circ}\text{C)} = \frac{(T1 - T0) * (\text{ADCx_DAT0} - \text{CFG_TEMP_CAL0})}{(\text{CFG_TEMP_CAL1} - \text{CFG_TEMP_CAL0})} + T0$$

T0 / T1 : Temperature Value-0 / Value-1 recording during chip manufacture

ADCx_DAT0 : ADC code value of current temperature

CFG_TEMP_CAL0 / CFG_TEMP_CAL1 : ADC code value of T0 / T1

27.15. ADC DMA Operation

27.15.1. DMA Module Configure

When the chip supports a DMA (direct memory access) controller, user can configure the DMA setting of transferred source/destination devices, channel request arbitration and others in the DMA module before a DMA data transaction. The DMA source and the destination can be memory or peripheral.

Refer the DMA chapter for more detail information about the DMA module configuration.

27.15.2. ADC DMA Control

After DMA configuration is finished, user needs to set the SPI module DMA enable bit of **ADCx_DMA_EN**.

Finally, the related channel request start bit of **DMA_CHn_REQ** is necessary to be set to start the ADC DMA transaction.(n = DMA channel index) Then the transferred source/destination devices will assert the request signal to DMA controller and the DMA controller will assert the acknowledge signal to the request source/destination devices. At the time, the data transferred connection is built for DMA transaction.

- **ADC Input to DMA**

The **ADCx_DMA_EN** register bit is used to enable DMA data transfer from ADC conversion input to DMA destination.

During the DMA transaction cycle, the data ready flags of **ADCx_ESMPF**, **ADCx_E1CNVF** and **ADCx_ESCNVF** are masked by hardware.

The ADC DMA transfer packet data size can be configured to 16-bit or 32-bit by setting **ADCx_DMA_DSIZE** register. When selects '16Bit', chip will transfer the bit[15:0] of **ADCx_DAT0** which is only the ADC conversion 12-bit, 10-bit or 8-bit data code for DMA transmission. When selects '32Bit', chip will transfer all 32-bit of **ADCx_DAT0** which is including of the ADC conversion data code, three ADC voltage window detect event flags, ADC conversion complete/overrun status bits and ADC data conversion channel number for DMA transmission.

When the ADC DMA transfer packet data size is configured to 32-bit, the DMA operation can only allow running peripheral-to-memory transfer type in actual application. It is not applicable to run peripheral-to-peripheral DMA transfer type as the ADC 32-bit data is including of others those are not ADC conversion data code. When the ADC DMA transfer packet data size is configured to 16-bit and the DMA operation is running peripheral-to-peripheral transfer type with DAC module, the destination peripheral DAC must set the receiving data width to 10-bit or 12-bit by setting **DAC_RES0_SEL** register. If this register is set 8-bit, the DAC output will be not normal.

- **ADC Sampling Clock Limit**

When the ADC DMA function is enabled, the ADC sampling clock frequency must equal or be slow under 1/4 ratio of AHB clock frequency for normal DMA transaction.

27.15.3. ADC Interrupt Flag Control during DMA

During DMA operation cycle, the module's interrupt flags will control and act three types.

One is masked during the DMA process (ESMPF, E1CNVF and ESCNVF flags). Another is to disable the DMA function after the flag (OVRF, WDLF, WDIF and WDHf flags) has asserted. At the time, hardware will clear the **ADCx_DMA_EN** bit in this condition. Others are normally as same as the action of not processing in DMA operation.

The following table is showing ADC interrupt flag control of DMA function.

Table 27-17. ADC Interrupt Flag Control for DMA Function

Action	Mask the Flag during DMA Process	DMA Disable after the Flag asserted (*1)	Normal Control
Peripheral	(Data flow flags)	(Error/Detect flags)	(Other flags)
ADCx	ADCx_ESMPF ADCx_E1CNVF(*3) ADCx_ESCNVF(*2)	ADCx_OVRF ADCx_WDLF ADCx_WDIF ADCx_WDHf	ADCx_SUMOF ADCx_SUMCF ADCx_SUMOVRF

Note-1 : When the flag is asserted, it will not force peripheral DMA disabled if the related interrupt enable bit is not enabled.

Note-2 : This flag will be asserted after DMA TX complete.

Note-3 : When ADCx_DMA_MDS is disabled, the ADCx_E1CNVF flag is masked during DMA process. When ADCx_DMA_MDS is "Keep", the ADCx_E1CNVF flag is normal and does not mask during DMA process.

User can set the **ADCx_DMA_MDS** register to select E1CNVF flag asserted mode during DMA process. When selects 'Disable', the E1CNVF flag will be masked after ADC conversion end. When selects 'Keep', the E1CNVF flag will be asserted after ADC conversion end. The following table is showing ADC conversion mode support by **ADCx_DMA_MDS** register setting.

Table 27-18. ADC DMA Mode vs. Conversion Mode

ADC Conversion Mode			DMA Mode	Function Supported
Mode	CONV_MDS	TRG_CONT	DMA_MDS	
One-Shot Mode	One	Disable	0	V
			1	V
One-Continue Mode	One	Enable	0	V
			1	-
Scan-One Mode	Scan	Disable	0	V
			1	V
Scan-Continue Mode	Scan	Enable	0	V
			1	-
Scan-Loop Mode	Loop	X	0	V
			1	-

<Sign> "V" : supported, "-" : not supported

27.16. ADC Application Circuit

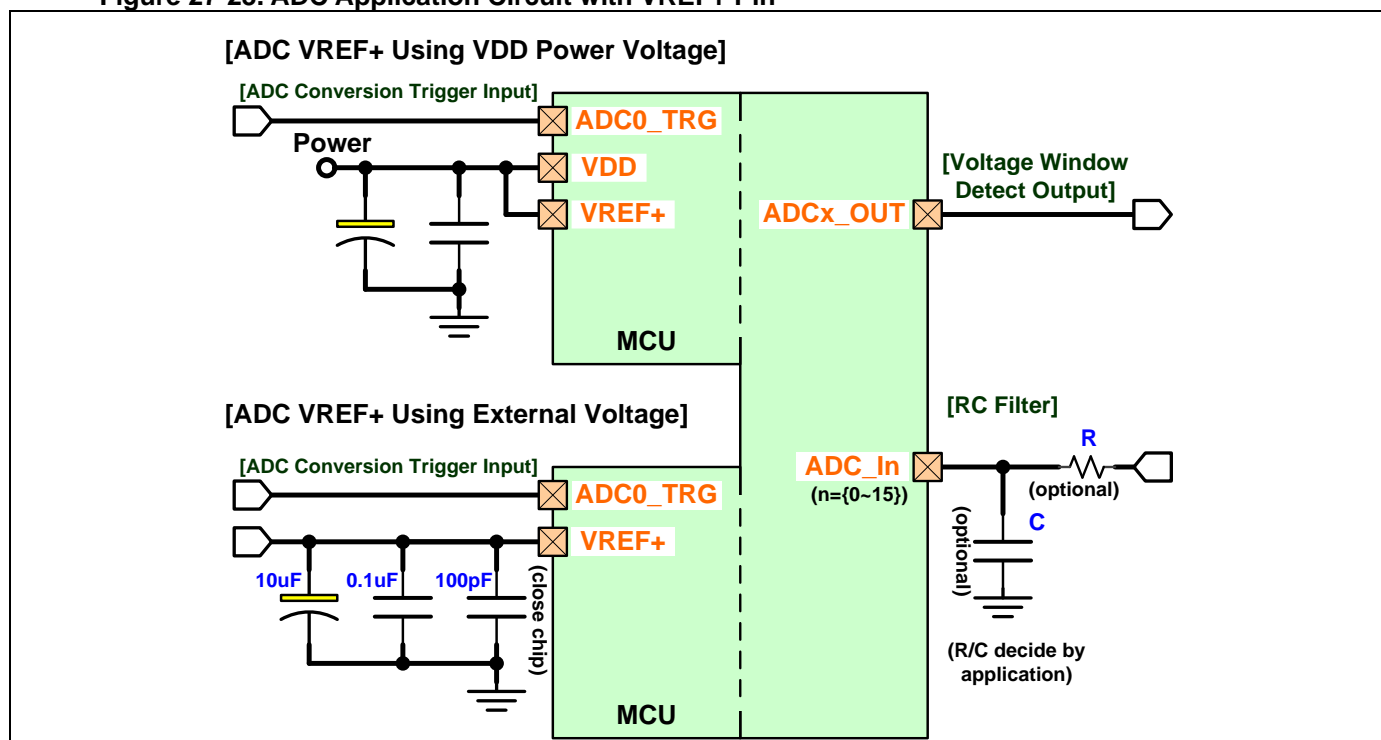
The ADC reference voltage source can be from

- (1) VDD power by connecting **VREF+** pin to **VDD** pin directly
- (2) External quiet reference voltage source

When uses the VDD power as the ADC reference voltage, it must connect **VREF+** pin trace to the point which is at current flow behind the power capacitor(s). When uses the external reference voltage source as the ADC reference voltage, it must add some decoupling and bypass capacitors, as shown in following diagram.

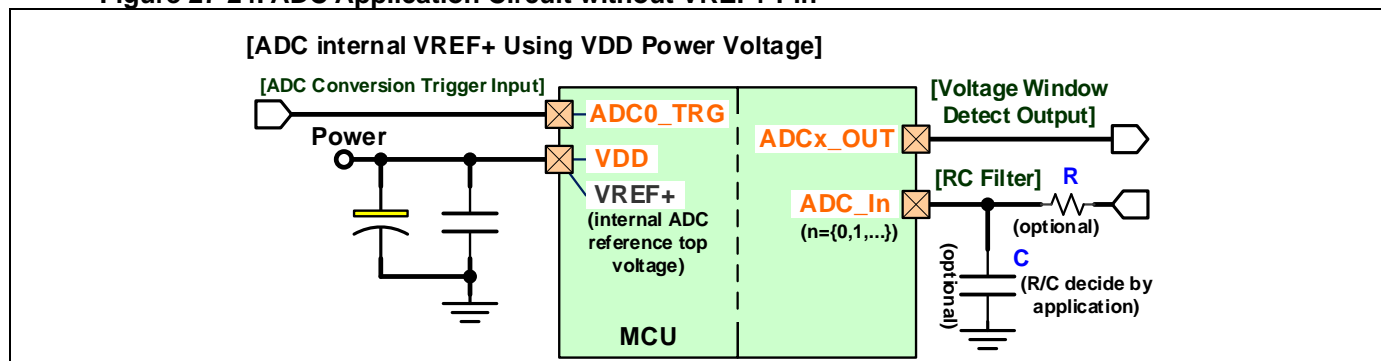
An optional **ADCx_TRG** pin is able to input the trigger signal for ADC input conversion and an optional **ADCx_OUT** pin is used to output the internal ADC window detection status.

Figure 27-23. ADC Application Circuit with VREF+ Pin



The **VREF+** (ADC reference voltage) will be internally connected with VDD power if the chip package is without **VREF+** pin. The following diagram is showing the ADC application circuit without **VREF+** Pin.

Figure 27-24. ADC Application Circuit without VREF+ Pin



- **Analog Input Filter**

The analog input filter is used as the anti-aliasing filter to eliminate the aliasing effect during the ADC sampling. The aliasing will occur when the unwanted signals or noise are present in the input signal those frequency are higher than half the ADC sampling frequency (Nyquist frequency). These unwanted signals will fold into the in-band frequency of original conversion signal. These noises are very hard to filter out by back-end digital filter. So strongly suggestion, user needs to implement the external anti-aliasing filter to remove these unwanted signals for best ADC conversion.

Generally, one first-order resistor-capacitor low-pass filter is used to filter out high frequency noise and reduce the effect of aliasing. The frequency bandwidth of low-pass filter can be design at the half of the ADC sampling frequency or smaller.

28. CMP (Analog Comparator)

28.1. Introduction



The module can be running in ON and SLEEP modes but can wakeup from STOP mode.

The chip builds in one CMP module which embeds four general purpose analog comparators with flexible input multiplexer, two internal voltage references of R-ladder and independent digital synchronized filter for each analog comparator. These analog comparators can be configured to four standalone comparators or a combined window comparator. The module provides the comparator output result status bit and the interrupt flags of rising edge or falling edge change. Also the output result can be output to external pin or internal other modules for trigger event.

The module can be running in ON and SLEEP mode only but can be wakeup MCU from STOP mode by the input comparison change detection.

Notify: The sign of (n= analog comparator macro index number) is using for Registers, Signals and Pins/Ports in the descriptions of this chapter.

28.2. Features

- Provide max. 2 fast comparators
- Provide external total max. 10 channels input for all comparators
- Programmable 64-step threshold of internal voltage reference
 - Selectable top reference voltage from VDD or LDO_VCAP(LDO_VR0)
- Provide flexible 6 channels input for each +/- input path selection
 - Two common and two independent external channels, two internal channels
- Programmable response time for optimal current consumption
- Combined window comparator from two comparators
- Selectable compare output polarity
- Support wakeup from SLEEP and STOP modes
- Compare output to I/O , interrupt or as internal module trigger event
 - Timer internal trigger, Capture events, or Break events
- Support analog watch dog as a reset source

28.3. Implementation

28.3.1. Chip Implementation

The following table is showing the implemented CMP analog comparators of chips.

Table 28-1. CMP Implementation

Chip	CMP Analog Comparator			Analog Comparator Macro		
	COMP0,1	COMP2,3	Total Channels	Internal Channel	IVREF/IVREF2 Top Voltage	Programmable Input Hysteresis
MG32F02A128/A064 MG32F02U128/U064	V	-	6	IVREF, DAC Out	VDD, LDO Out	V
MG32F02N128/N064 MG32F02K128/K064	V	-	6	IVREF, LDO Out, OP0_P0	VDD, LDO Out	V
MG32F02V032	Not Implemented					

<Note> V: Implemented

28.3.2. Modules' Functions

The following table is showing the implemented functions of CMP module.

Table 28-2. Analog Comparator Modules' Functions

Module	CMP			Comment
Chip	MG32F02A128 MG32F02U128 MG32F02A064 MG32F02U064	MG32F02V032	MG32F02N128 MG32F02K128 MG32F02N064 MG32F02K064	
Module Functions				
Comparator Units	2	0	2	Rail-to-rail comparators
Input Channels of +/- path	6	-	6	Input channel number for each comparator +/- input path
Total External Channels	6	-	6	Total external channels of all analog comparators
IVREF DAC	IVREF/IVREF2	-	IVREF/IVREF2	Two internal voltage reference R-DAC
IVREF DAC Bit Resolution	6 (64 levels)	-	6 (64 levels)	
IVREF DAC Top Voltage	VDD/LDO_VR0	-	VDD/LDO_VCAP	LDO_VCAP : internal LDO output voltage
LDO out as Internal Input	-	-	yes	
DAC out as Internal Input	yes	-	-	
OPA out as Internal Input	-	-	yes	
IVREF as Internal Input	yes	-	yes	
IVREF out to Pin	yes	-	yes	IVREF voltage output to CMPn_I0
Hysteresis Voltage	0, 10mV	-	0, 10mV	
Response time	200ns , 10us	-	200ns , 10us	optimal current consumption
Wakeup from SLEEP/STOP	yes	-	yes	
Output as Reset Source	yes	-	yes	Analog Watch Dog as a reset source
Compare output to I/O	yes	-	yes	Compare output to I/O, interrupt or as internal module trigger event (Timer internal trigger , Capture events , or Break events)

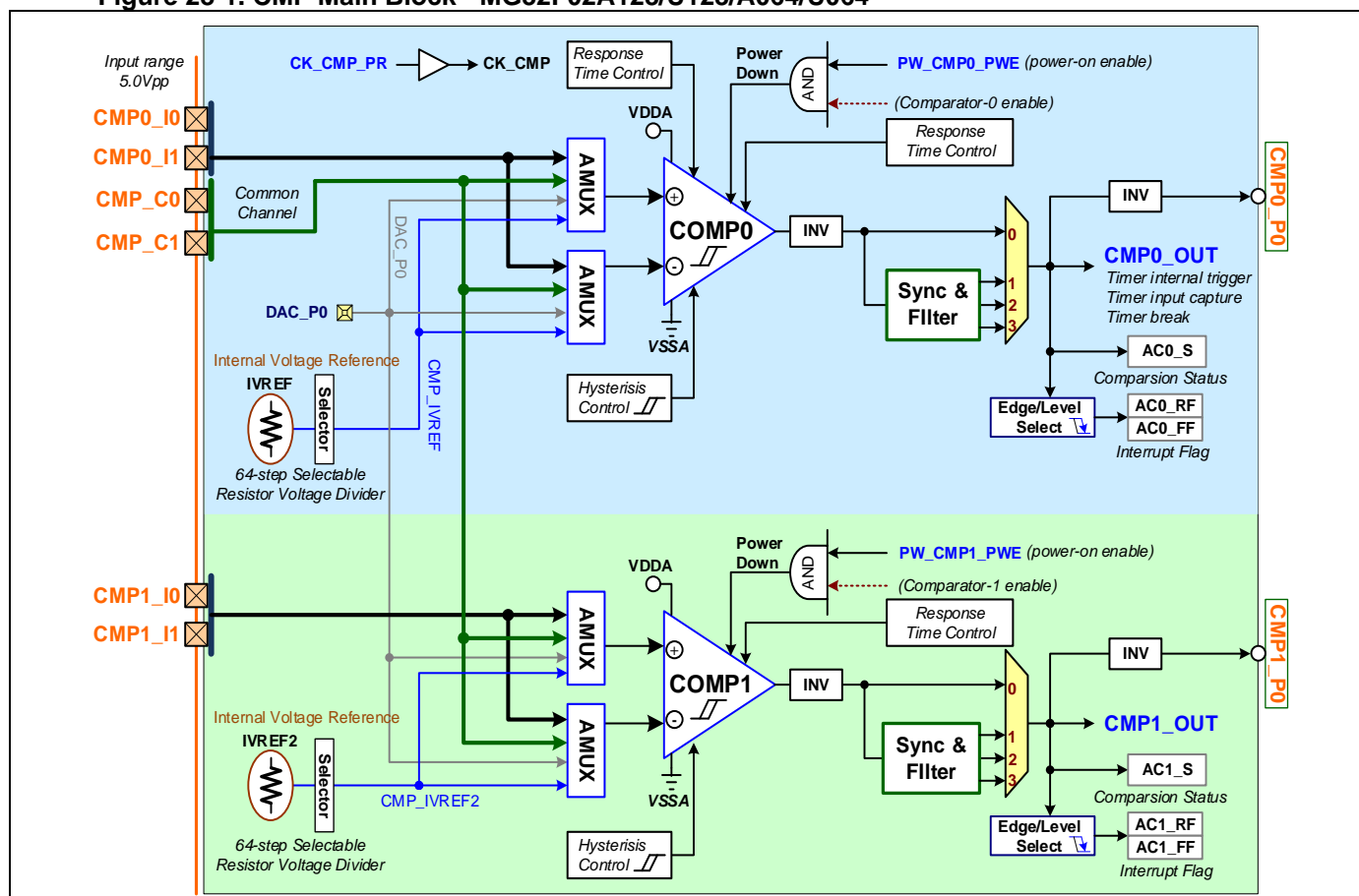
28.4. Control Block

The CMP module includes maximum four general purpose analog comparators CMP0~3 by the same design structure and two internal voltage references IVREF/IVREF2 by R-ladder structure. Each one is with the independent input multiplexer, digital synchronized filter and digital output circuit. The IVREF is using for CMP0 and the IVREF2 is using for CMP1. Refer the section of “Chip Implement” about the supported analog comparators and the optional internal voltage source LDO output **LDO_VCAP(LDO_VR0)** and DAC output **DAC_P0**.

- **MG32F02A128/U128/A064/U064**

The following diagram is showing the analog comparator controller block of MG32F02A128/U128/A064/U064.

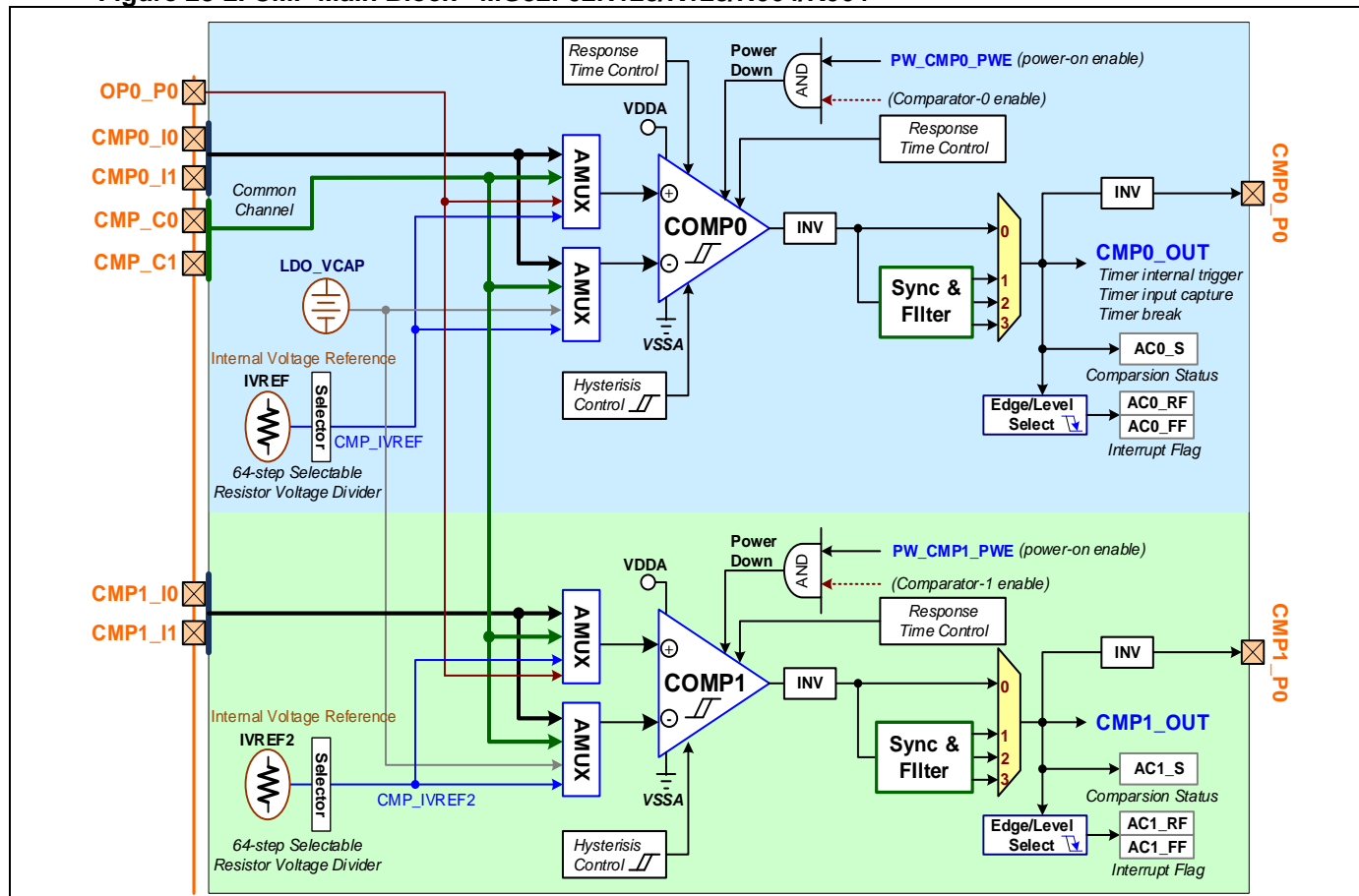
Figure 28-1. CMP Main Block - MG32F02A128/U128/A064/U064



- MG32F02N128/K128/N064/K064

The following diagram is showing the analog comparator controller block of MG32F02N128/K128/N064/K064.

Figure 28-2. CMP Main Block - MG32F02N128/K128/N064/K064



28.5. IO Lines

28.5.1. IO Signals

- **CMP_Cn**

It is the comparator analog input signal of common channel-n which is able to use for plus input or minus input of all analog comparators.

- **CMPn_I[0..1]**

They are the comparator analog input signal for analog comparator **COMPn** which is able to use for plus input or minus input.

- **CMPn_P0**

It is the compared data result output for analog comparator **COMPn**.

28.5.2. IO Configure

User must configure the related IO pins in order to use the IO lines of this module. The IO operation modes, output high speed option, pull-high option, output drive strength, IO deglitch filter and input inverse selection are programmable by pin independent. The using of analog input **CMP_Cn** and **CMPn_I[0..1]** must set the IO mode to AIO mode for related IO pins. Please refer the section of “[IO Mode](#)” in GPIO chapter for more detail descriptions of IO mode configuration.

Each IO signal may be mapped and selected on several IO pins by configuring the IO AFS matrix. Please refer the section of “[Alternate Function Select](#)” in GPIO chapter for more detail descriptions of IO AFS configuration. About the actual IO pin AFS information, please refer the section of “[Pin Alternate Functions Selected Table](#)” in Pin Description chapter of the chip Data Sheet.

28.6. Power and Clock

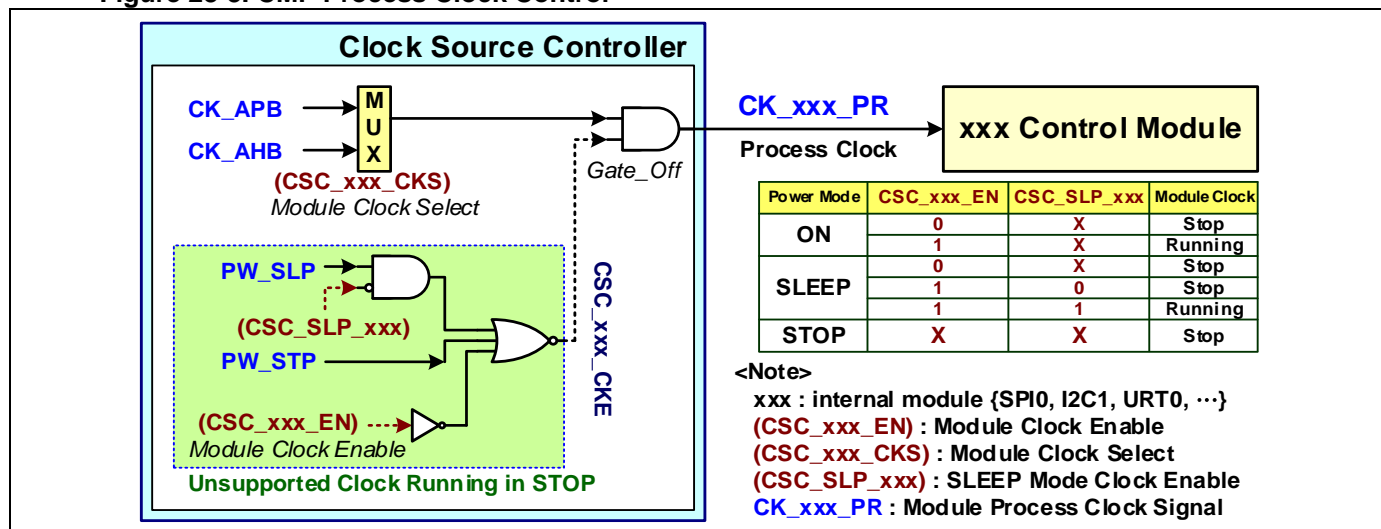
28.6.1. CMP Power Control

The operation power VDDA of CMP analog macros is from the IO power **VDD** pin. Four analog comparators are with independent power on enable bits of **CMP_ACn_EN**. Each of CMP analog comparator is enabled only when the **CMP_ACn_EN** bit is set to logic 1. Each one can be disabled and powered down independently for power saving. User can plan the power control of each analog comparators independently is on or off beforehand for chip entering **SLEEP** or **STOP** mode by setting **PW_SLP_CMPn** or **PW_STP_CMPn** registers. Refer the System Power chapter for more information. (n= {0, 1})

28.6.2. CMP Clock Control

The module process clock of **CK_CMP_PR** is using for the interface control logic between APB bus and the module. It is coming from CSC (Clock Source Controller) module. It can be enabled in **CSC_CMP_EN** register and select the clock source from APB clock or AHB clock in **CSC_CMP_CKS** register. User can plan the module clock is running or not beforehand for chip entering **SLEEP** mode by setting **CSC_SLP_CMP** register. Refer the System Clock chapter for more information.

Figure 28-3. CMP Process Clock Control



28.7. Interrupt and Event

28.7.1. CMP Interrupt Control

There are three types' signals of **INT_CMP**, **RST_CMPn** and **WUP_CMPn** to be generated in this Analog Comparator control module. (n= {0, 1})

- **Interrupt Events**

INT_CMP signal sends to External Interrupt Controller (EXIC) to do as an interrupt event.

These interrupt flags are using for the interrupt service routine (ISR) flow control. Generally these flags are set by hardware and software clears them when the serviced job is completed in the related ISR. Each interrupt flag has one interrupt enable bit. User can enable or disable it. There is one interrupt all enable bit of **CMP_IEA** to enable or disable all the interrupt sources for this module.

- **Reset Events**

RST_CMPn signal sends to Reset Source Controller (RST) to do as the warm reset events or cold reset events. These reset events can be enabled to reset the chip by setting the registers in RST.

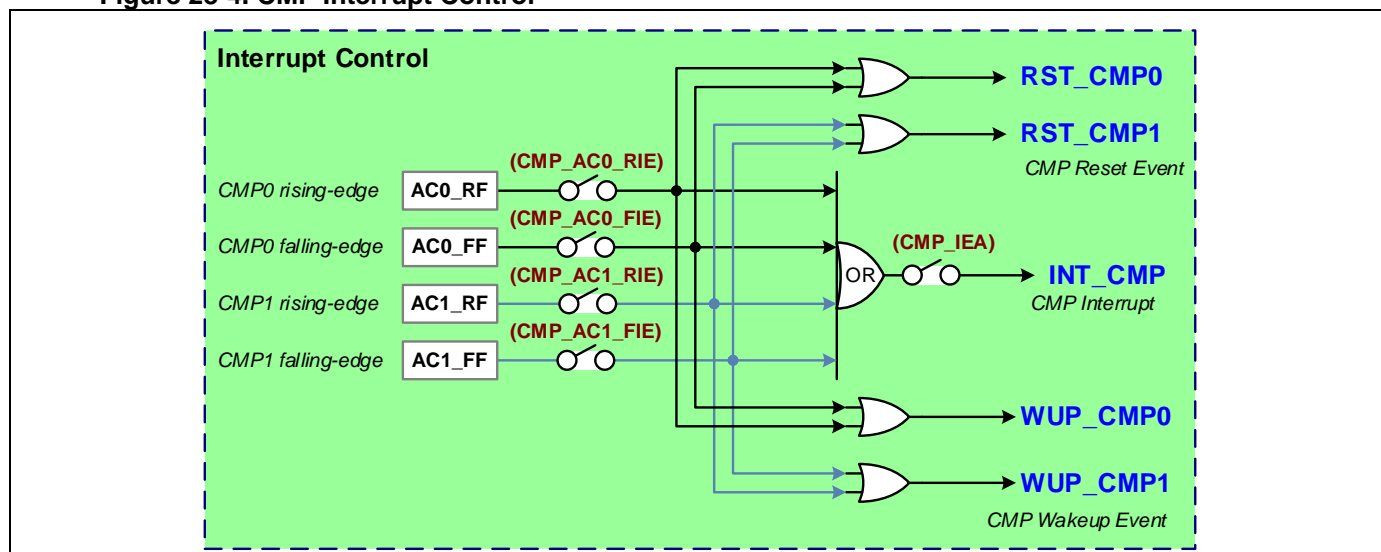
Refer the descriptions of System Reset chapter for more information about the reset events and control.

- **Wakeup Events**

WUP_CMPn signal sends to Power Controller (PW) to do as the system wakeup events. These wakeup events can be enabled to wakeup the chip when it is in SLEEP or STOP mode by setting the registers in PW. The Power Controller manages these signals and sends the wakeup signal to EXIC and NVIC for chip wakeup control.

Refer the descriptions of System Power and Interrupt chapters for more information about the wakeup events and control.

Figure 28-4. CMP Interrupt Control



28.7.2. CMP Interrupt Flags

Generally these interrupt flags are set by hardware and clear by software writing 1. Refer the register descriptions for more information about the related interrupt flags and interrupt enable bit.

- **ACn_RF**

Analog comparator **CMPn** rising edge interrupt flag (**CMP_ACn_RF**). There is a related interrupt enable register bit of **CMP_ACn_RIE**.

- **ACn_FF**

Analog comparator **CMPn** falling edge interrupt flag (**CMP_ACn_FF**). There is a related interrupt enable register bit of **CMP_ACn_FIE**.

28.8. CMP Analog Comparator

28.8.1. Input Channels

- **Analog Input Multiplexer**

Each the analog multiplexer (AMUX) of the positive or negative input of any analog comparator has flexible 6 channels input. These channels are including of two common external channels, two comparator independently external channels and two common internal channels. The AMUX can be configured by **CMP_ACn_PMUX**, **CMP_ACn_NMUX** registers. (n= {0, 1})

The two common external channels are from **CMP_C0** and **CMP_C1** pins those can input to all analog comparators. The two independent external channels are from **CMPn_I0** and **CMPn_I1** pins those can input to the analog comparator-n. The two common internal channels are from internal voltage reference of **IVREF** or **IVREF2** and internal voltage source of **LDO_VR0** or **DAC_P0** those can input to all analog comparators. The reference voltage source of **LDO_VCAP(LDO_VR0)** or **DAC_P0** is optional by chip implemented. Refer the section of "[Chip Implement](#)" about the implemented internal voltage source.

- **I/O Pins Used with Comparator Function**

The analog input pins used for the comparators also have its I/O port digital input and output function. In order to give the proper analog performance, a pin that is being used should have its digital output as disabled. It is done by putting the port pin into the input-only mode. And when an analog signal is applied to the analog input pin and the digital input from this pin is not needed, software could set the corresponding pin to analog-input-only in **PX_IOMn** (X={A,B},n={0~15}) to reduce power consumption in the digital input buffer.

28.8.2. Internal Voltage Reference and Source

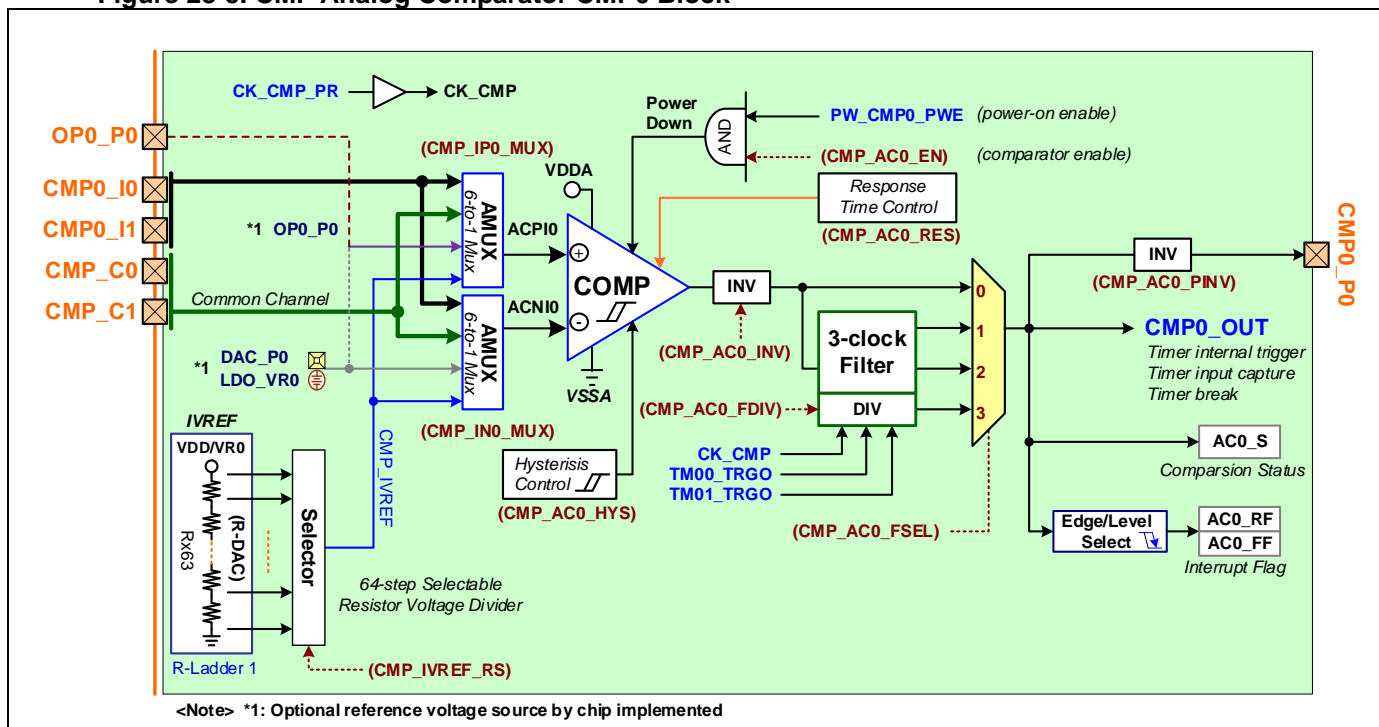
The analog comparator is built-in two internal voltage references – **IVREF** and **IVREF2** with 64-steps R-ladder structure. They can use as one of the analog comparator input and compare with another input from external source. The **IVREF** is use only for analog comparator COMP0 and the **IVREF2** is use only for analog comparator COMP1/2/3. Refer "CMP Internal Voltage Reference" section for more information.

Others, the analog comparator provides one internal voltage source of **LDO_VR0** or **DAC_P0** to do as the comparator internal channel input. The reference voltage source of **LDO_VR0** or **DAC_P0** is optional by chip implemented. Refer the section of "[Chip Implement](#)" about the implemented internal voltage source.

28.8.3. Analog Comparator-0

A built-in internal voltage references – **IVREF** is able to use as the comparator input and it is used for CMP0 only. The following diagram is showing the Analog Comparator-0 controller block.

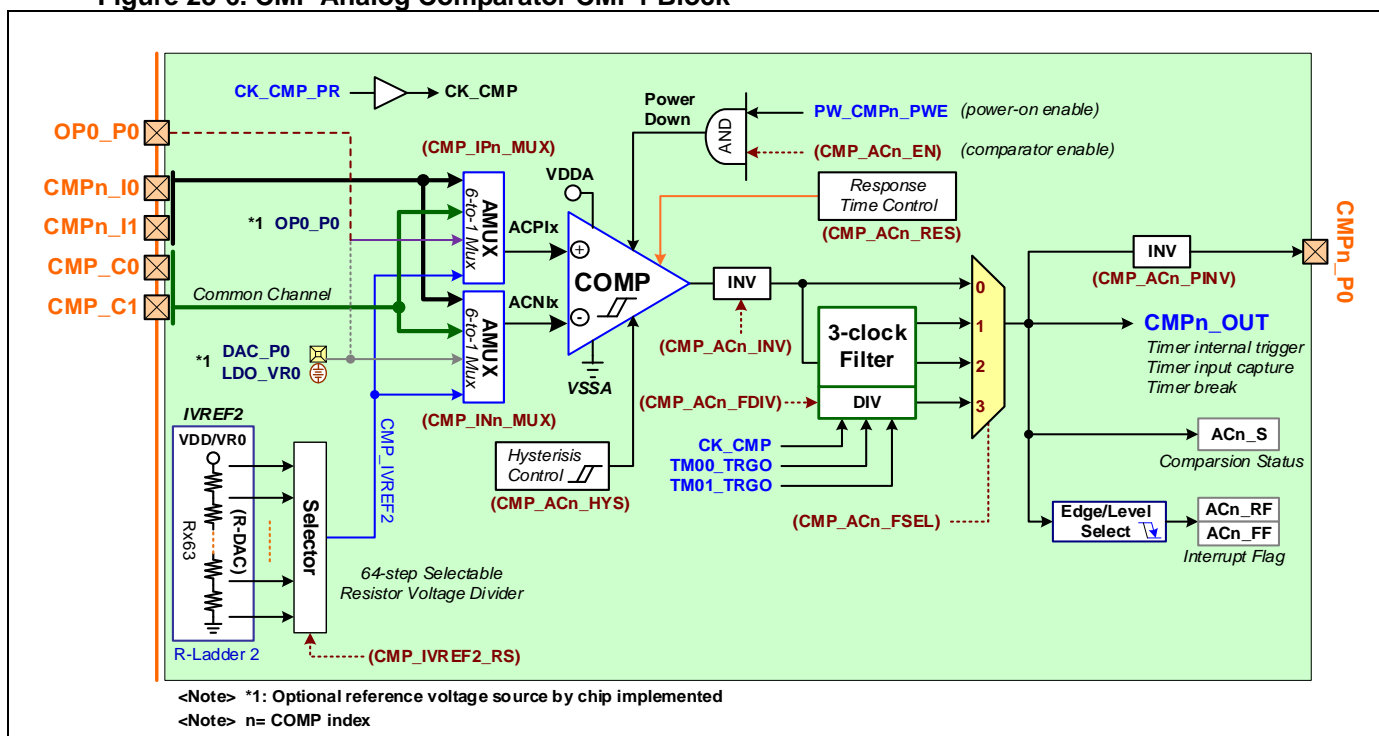
Figure 28-5. CMP Analog Comparator CMP0 Block



28.8.4. Analog Comparator-1

The analog comparators CMP1 are the same structure as CMP0. A built-in internal voltage references – **IVREF2** is shared to use for CMP1. The following diagram is showing the Analog Comparator-1 controller block.

Figure 28-6. CMP Analog Comparator CMP1 Block



28.8.5. Response Time Select

The analog comparator can select the response time independently by **CMP_ACn_RES** bit for different applications. It is also control the operation current of comparator for optimal current consumption. (n= {0 ~ 1})

28.8.6. Comparator Output

- **Comparator Output Result**

The analog comparator also has an adjustable output polarity by setting **CMP_ACn_INV** register bit. The result of output are storing in **CMP_ACn_S**. The **CMPn_OUT** signal of each comparator is able to output to other internal modules. For example, it can send to internal timer modules as timer trigger input, timer capture input or timer break input.

When the output is changed, the related flag of **CMP_ACn_RF** or **CMP_ACn_FF** is asserted for 0-to-1 rising change or 1-to-0 falling change. (n= {0, 1})

- **Comparator Output Filter**

The analog comparator output is selectable independently with a digital synchronized filter or bypasses the filter by setting **CMP_ACn_FSEL** register. Also this register can enable the 3-clock synchronized filter and select the filter clock source from module internal clock **CK_CMP**, timer trigger output signal of **TM00_TRGO** and **TM01_TRGO**. When enables the synchronized filter, user can select the synchronized filter clock divider in **CMP_ACn_FDIV** register.

- **Output Pin Control**

There is one output pin of **CMPn_P0** to output the analog comparator result to external for each comparator. One extra inverse control bits is selectable independently by **CMP_ACn_PINV** on output pins of **CMPn_P0**.

28.9. CMP Internal Voltage Reference

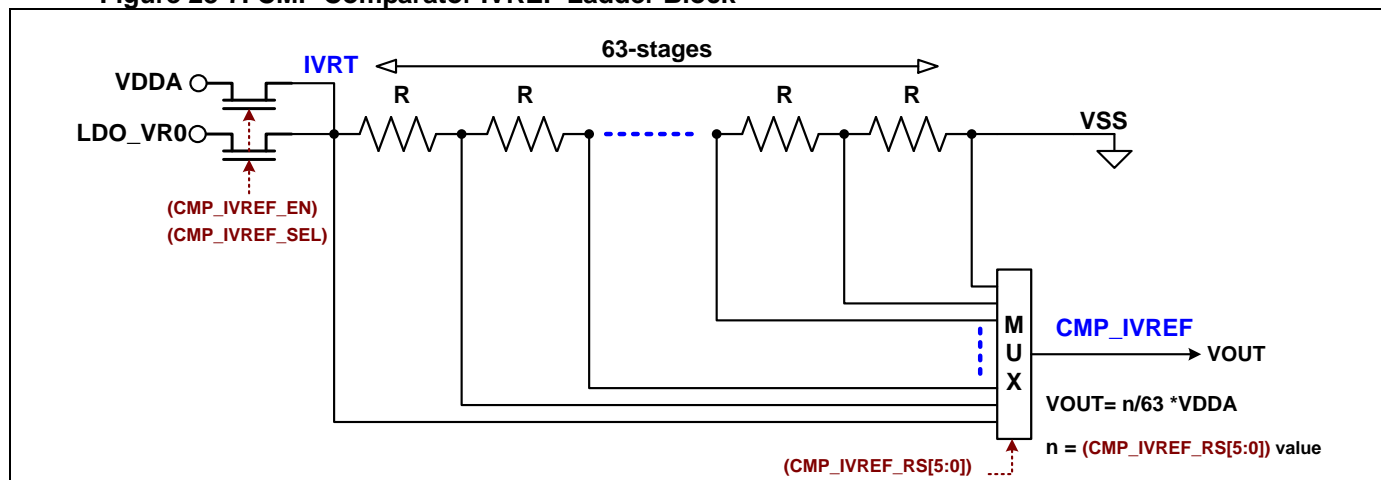
28.9.1. IVREF and IVREF2

The analog comparator is built-in two internal voltage references – **IVREF** and **IVREF2**. The **IVREF** and **IVREF2** are the same as R-DAC structure with 64-steps linear R-ladder. The **IVREF** is used for CMP0 only and **IVREF2** is shared to use for CMP1/CMP2/CMP3. They can use as one of the analog comparator input and compare with another input from external source.

There is an enable register bit of **CMP_IVREF_EN** to enable the **IVREF**. Also there is an enable register bit of **CMP_IVREF2_EN** to enable the **IVREF2**. User can disable the unused internal voltage reference for power saving. For the top voltage **IVRT** of **IVREF** or **IVREF2** R-ladder, user can select the voltage source from internal analog power **VDDA** or core power regulator output **LDO_VR0** by setting **CMP_IVREF_SEL** or **CMP_IVREF_SEL2** register.

The following diagram is showing the Internal Voltage Reference R-Ladder block.

Figure 28-7. CMP Comparator IVREF Ladder Block



28.9.2. IVREF Output Voltage

There are two registers of **CMP_IVREF_RS** and **CMP_IVREF2_RS** those are used to select voltage level of R-Ladder **IVREF** or **IVREF2**. User can calculate the output voltage of **IVREF** or **IVREF2** by following formula:

$$\text{IVREF_OUT (volt)} = \frac{\text{IVRT} * \text{CMP_IVREF_RS}}{63}$$

IVRT is the top voltage of **IVREF** or **IVREF2** R-ladder.

The following table is showing the R-Ladder Output of **IVREF** or **IVREF2** by **CMP_IVREF_RS** or **CMP_IVREF2_RS** register setting.

Table 28-3. CMP Comparator IVREF R-Ladder Output

Register	IVREF Output			Register	IVREF Output		
IVREFx_RS	Output Level	IVRT (volt) =VDD	IVRT (volt) =LDO_VCAP	IVREFx_RS	Output Level	IVRT (volt) =VDD	IVRT (volt) =LDO_VCAP
		5	3.3			5	3.3
0	0/63 IVRT	0.000 V	0.000 V	32	32/63 IVRT	2.540 V	1.676 V
1	1/63 IVRT	0.079 V	0.052 V	33	33/63 IVRT	2.619 V	1.729 V
2	2/63 IVRT	0.159 V	0.105 V	34	34/63 IVRT	2.698 V	1.781 V
3	3/63 IVRT	0.238 V	0.157 V	35	35/63 IVRT	2.778 V	1.833 V
4	4/63 IVRT	0.317 V	0.210 V	36	36/63 IVRT	2.857 V	1.886 V
5	5/63 IVRT	0.397 V	0.262 V	37	37/63 IVRT	2.937 V	1.938 V
6	6/63 IVRT	0.476 V	0.314 V	38	38/63 IVRT	3.016 V	1.990 V
7	7/63 IVRT	0.556 V	0.367 V	39	39/63 IVRT	3.095 V	2.043 V
8	8/63 IVRT	0.635 V	0.419 V	40	40/63 IVRT	3.175 V	2.095 V
9	9/63 IVRT	0.714 V	0.471 V	41	41/63 IVRT	3.254 V	2.148 V
10	10/63 IVRT	0.794 V	0.524 V	42	42/63 IVRT	3.333 V	2.200 V
11	11/63 IVRT	0.873 V	0.576 V	43	43/63 IVRT	3.413 V	2.252 V
12	12/63 IVRT	0.952 V	0.629 V	44	44/63 IVRT	3.492 V	2.305 V
13	13/63 IVRT	1.032 V	0.681 V	45	45/63 IVRT	3.571 V	2.357 V
14	14/63 IVRT	1.111 V	0.733 V	46	46/63 IVRT	3.651 V	2.410 V
15	15/63 IVRT	1.190 V	0.786 V	47	47/63 IVRT	3.730 V	2.462 V
16	16/63 IVRT	1.270 V	0.838 V	48	48/63 IVRT	3.810 V	2.514 V
17	17/63 IVRT	1.349 V	0.890 V	49	49/63 IVRT	3.889 V	2.567 V
18	18/63 IVRT	1.429 V	0.943 V	50	50/63 IVRT	3.968 V	2.619 V
19	19/63 IVRT	1.508 V	0.995 V	51	51/63 IVRT	4.048 V	2.671 V
20	20/63 IVRT	1.587 V	1.048 V	52	52/63 IVRT	4.127 V	2.724 V
21	21/63 IVRT	1.667 V	1.100 V	53	53/63 IVRT	4.206 V	2.776 V
22	22/63 IVRT	1.746 V	1.152 V	54	54/63 IVRT	4.286 V	2.829 V
23	23/63 IVRT	1.825 V	1.205 V	55	55/63 IVRT	4.365 V	2.881 V
24	24/63 IVRT	1.905 V	1.257 V	56	56/63 IVRT	4.444 V	2.933 V
25	25/63 IVRT	1.984 V	1.310 V	57	57/63 IVRT	4.524 V	2.986 V
26	26/63 IVRT	2.063 V	1.362 V	58	58/63 IVRT	4.603 V	3.038 V
27	27/63 IVRT	2.143 V	1.414 V	59	59/63 IVRT	4.683 V	3.090 V
28	28/63 IVRT	2.222 V	1.467 V	60	60/63 IVRT	4.762 V	3.143 V
29	29/63 IVRT	2.302 V	1.519 V	61	61/63 IVRT	4.841 V	3.195 V
30	30/63 IVRT	2.381 V	1.571 V	62	62/63 IVRT	4.921 V	3.248 V
31	31/63 IVRT	2.460 V	1.624 V	63	63/63 IVRT	5.000 V	3.300 V

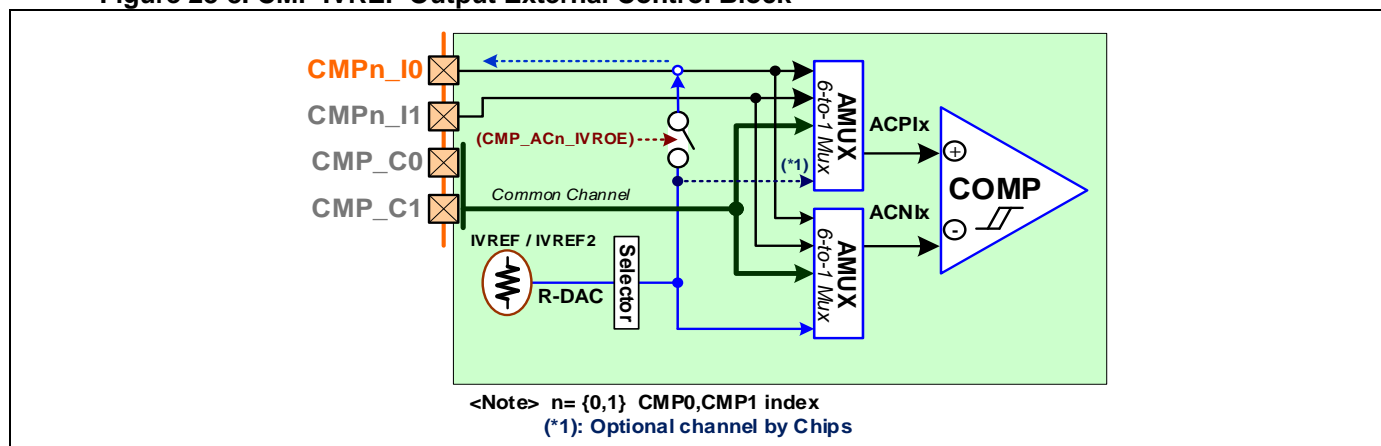
<Note> IVREFx_RS = CMP_IVREF_RS or CMP_IVREF2_RS register bits

IVRT : IVREF Top Voltage

● IVREF Output External

The internal reference voltages (R-DAC output) of **IVREF** and **IVREF2** are able to output to external pin **CMP0_I0** and **CMP1_I0** independently by enabling in **CMP_AC0_IVROE** and **CMP_AC1_IVROE** registers. The **IVREF** and **IVREF2** outputs provide weak voltage level without output buffer. For application using suggestion, the output capacitance load is under 5pF and user needs additional output buffer on external PCB.

Figure 28-8. CMP IVREF Output External Control Block



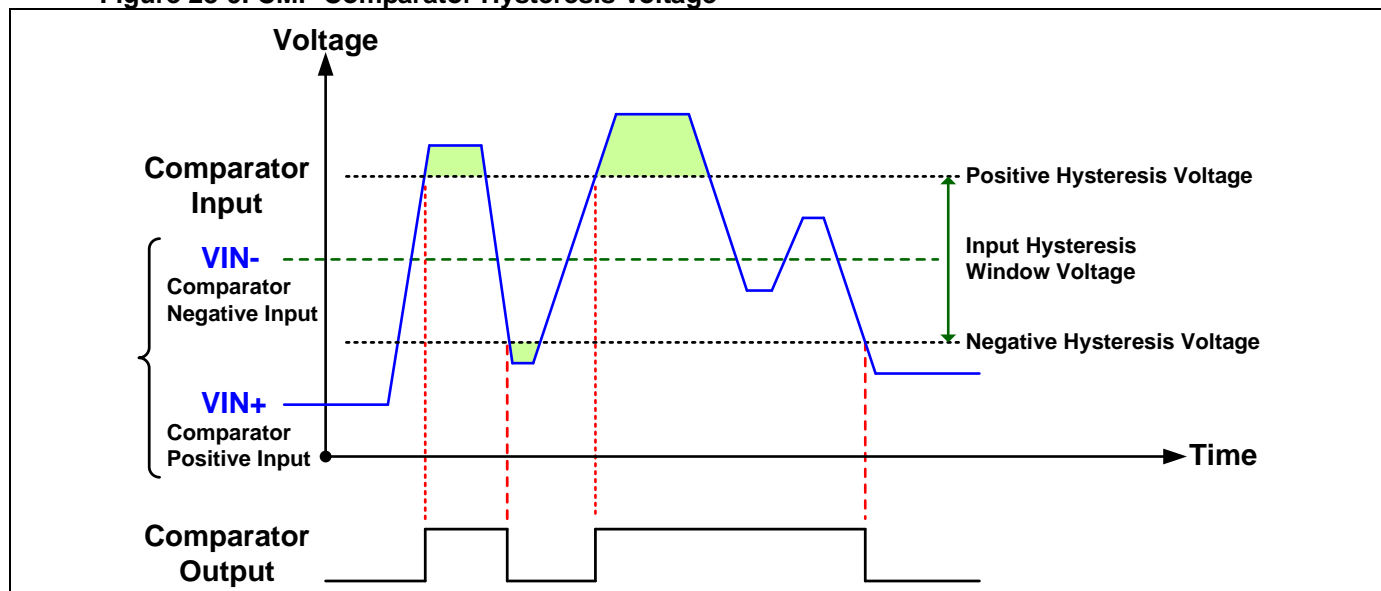
28.10. CMP Input Hysteresis Voltage

When the positive input is in the voltage range of hysteresis which is referred to the negative input voltage, the analog comparator output is no change and keeps previous result.

User can configure the input hysteresis window voltage by setting **CMP_ACn_HYS** register. When selects 'No', the hysteresis voltage range is about 0mV. When selects 'Low', the hysteresis voltage range is set to low hysteresis. Please refer the related chip Data Sheet about the hysteresis voltage range information.

The following diagram is showing the hysteresis voltage of the analog comparator input.

Figure 28-9. CMP Comparator Hysteresis Voltage



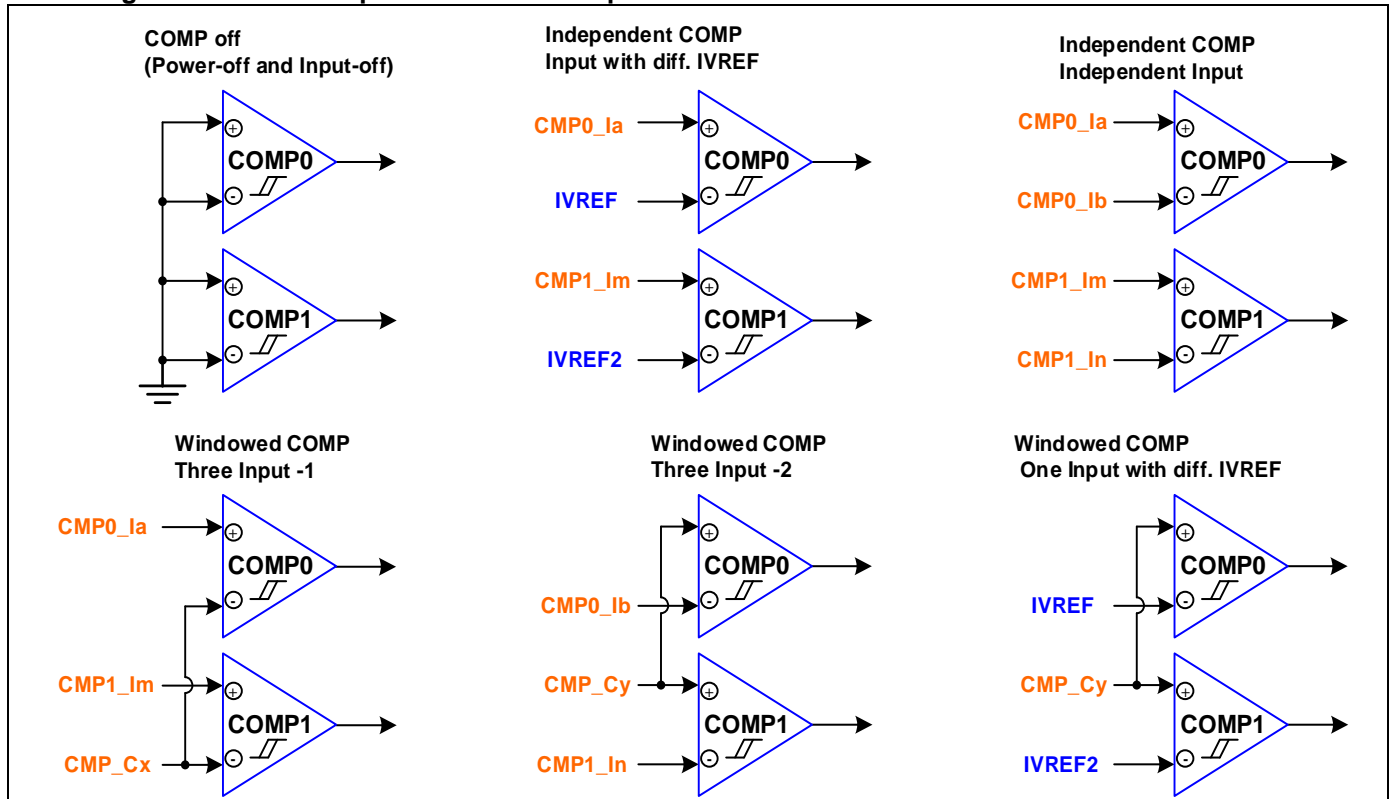
28.11. CMP Input Networks

User can flexibly connect these analog comparator inputs for user's application.

Notify: The sign of (COMPx = analog comparator index x) is using for the following descriptions.

The following diagram is showing the comparator input network connections for different applications.

Figure 28-10. CMP Input Networks Example



- **COMP off**

Short all of positive inputs and negative inputs. The analog comparator will halt as power off state.

- **Independent COMP Input with different IVREF**

Connect **IVREF** voltage to COMP0 negative input as the reference voltage to compare with external input voltage **CMP0_Ia**. Also connect **IVREF2** voltage to COMP1 negative input as the reference voltage to compare with external input voltage **CMP0_Im**.

- **Independent COMP Independent Input**

Normally, all the COMP can use independent. Both positive input and negative input of all COMP connect with different external voltage source.

- **Windowed COMP Three Input-1**

Connect the same input voltage source of **CMP_Cx** to the negative input of both COMP0 and COMP1. Input two different input voltage sources of **CMP0_Ia** and **CMP1_Im** to the positive input of COMP0 and COMP1. User can input two level voltages on the two positive inputs to compare with the same voltage to do like as a two-bit ADC.

- **Windowed COMP Three Input-2**

It is same as "Windowed COMP Three Input-1". The different is that the common input voltage **CMP_Cy** is connected to the positive input of both COMP0 and COMP1.

- **Windowed COMP Two Input and One IVREF**

It is same as "Windowed COMP Three Input-1". The different is that one of the positive voltage input is changed to **IVREF2** voltage.

- **Windowed COMP Two Input with same IVREF**

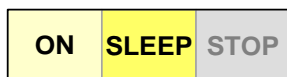
It is same as "Windowed COMP Three Input-2". The different is from that the common voltage is changed to **IVREF2** voltage.

- **Windowed COMP One Input with different IVREF**

It is same as “Windowed COMP Three Input-2”. The different is from that the positive voltage inputs are changed to **IVREF** voltage and **IVREF2** voltage for COMP0 and COMP1. User can set **IVREF** and **IVREF2** to different voltage level to compare with the same voltage to do like as a two-bit ADC.

29. DAC (Digital-to-Analog Converter)

29.1. Introduction



The module can be running in ON and SLEEP modes only.

The chip builds in one DAC module which is embedded one 12-bit voltage mode DAC and digital logic for input code control. The digital-to-analog conversion can be performed and start trigger by data register written, events (external pin input or internal events). The voltage DAC can output with an output buffer under the conversion rate up to 1 MHz.

29.2. Features

- Conversion start trigger by register written, external pin and internal events
- Data alignment for input code left/right justify
- Output data are buffered with DMA capability
- One 12-bit voltage DAC
 - Maximum conversion rate is 1Msps
 - DAC analog output to ADC and Analog Comparator internal channels
- Build in internal output buffer
 - Bypass output buffer option
- Configurable code width : 12/10/8-bit

29.3. Implementation

29.3.1. Chip Implementation

The following table is showing the implemented DAC analog parameters.

Table 29-1. DAC Implementation

Chip	DAC Macro			
	DAC Type	Conversion Rate	Output Buffer	Full-Scale Range (Volt)
MG32F02A128/A064 MG32F02U128/U064	Voltage DAC	1Msps/12bit	V	0.2 ~ VDD-0.2
MG32F02V032 MG32F02N128/N064 MG32F02K128/K064	Not Implemented			

<Note> V: Implemented

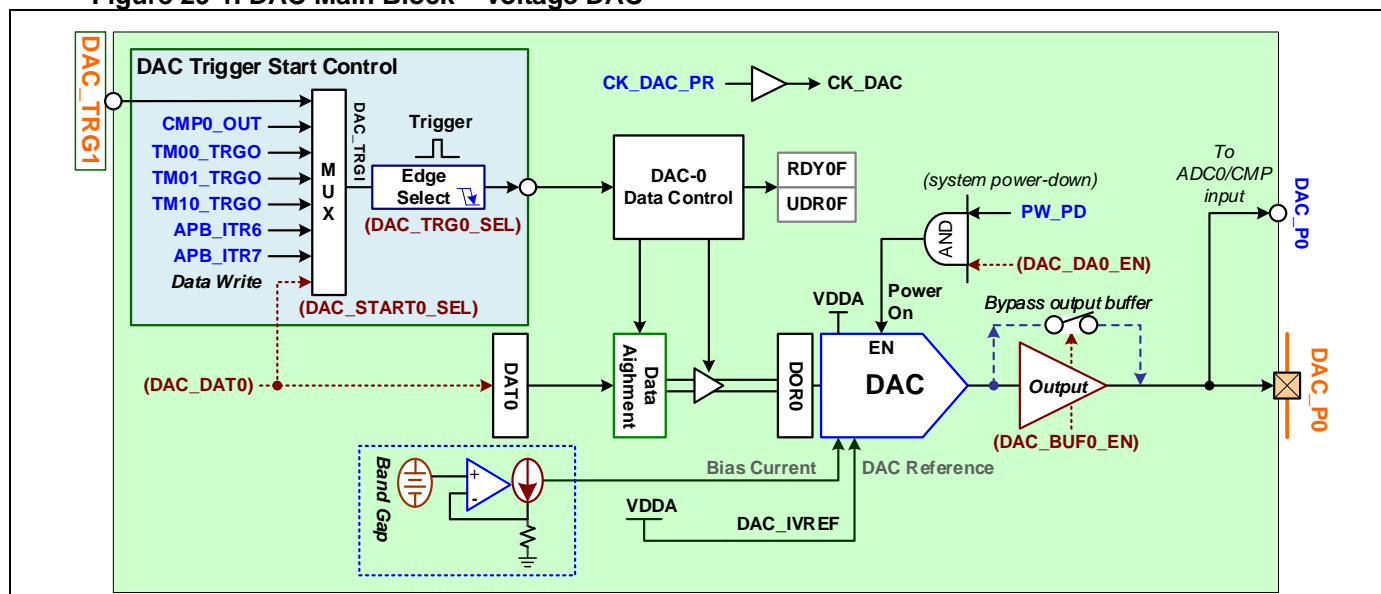
29.4. Control Block

29.4.1. Voltage DAC Control Block

The voltage DAC control block consists of a 1Msps/12-bit voltage mode DAC, output buffer, reference voltage circuit, a DAC data code register, a DAC conversion output register (DOR0) and DAC conversion trigger start control block.

The following diagram is showing the voltage DAC control block.

Figure 29-1. DAC Main Block – Voltage DAC



29.5. IO Lines

29.5.1. IO Signals

- **DAC_P0**

It is the DAC analog output of channel 0.

- **DAC_TRG**

It is the external trigger start input signal for DAC data conversion.

29.5.2. IO Configure

User must configure the related IO pins in order to use the IO lines of this module. The IO operation modes, output high speed option, pull-high option, output drive strength, IO deglitch filter and input inverse selection are programmable by pin independent. The using of analog output **DACP_0** must set the IO mode to AIO mode for related IO pin. Please refer the section of “[IO Mode](#)” in GPIO chapter for more detail descriptions of IO mode configuration.

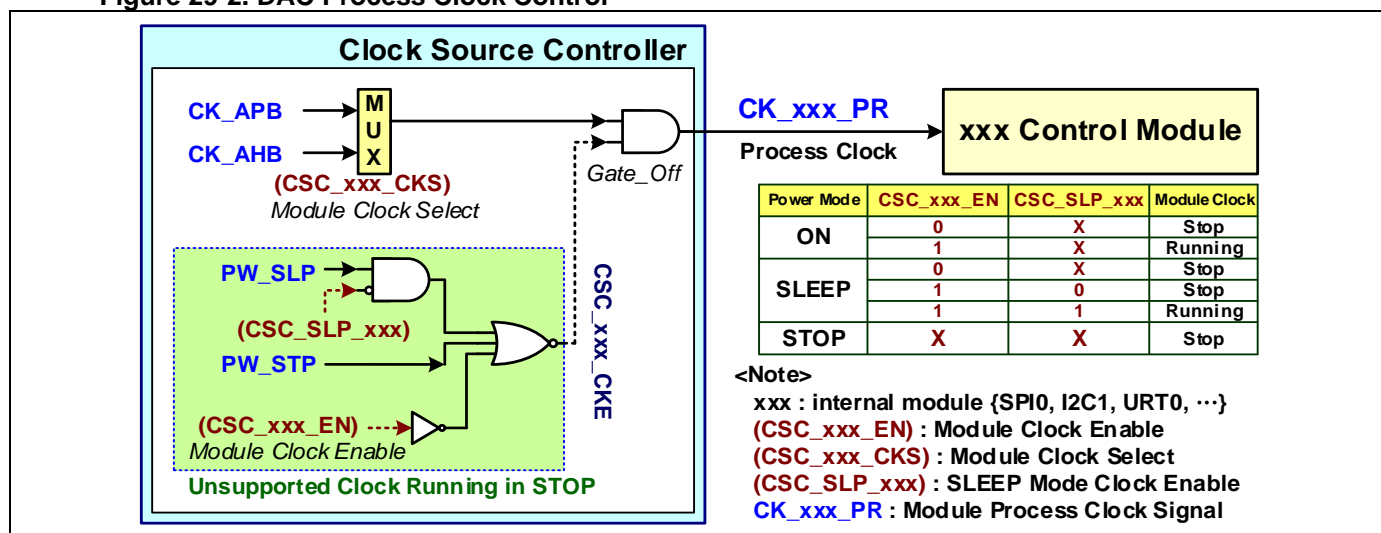
Each IO signal may be mapped and selected on several IO pins by configuring the IO AFS matrix. Please refer the section of “[Alternate Function Select](#)” in GPIO chapter for more detail descriptions of IO AFS configuration. About the actual IO pin AFS information, please refer the section of “[Pin Alternate Functions Selected Table](#)” in Pin Description chapter of the chip Data Sheet.

29.6. Power and Clock

The operation power VDDA of DAC analog macro is from the IO power **VDD** pin. The DAC analog macro is power enabled only when the **DAC_DAC0_EN** bit is set to logic 1. The DAC analog macro is power down when this bit is logic 0.

The module process clock of **CK_DAC_PR** is using for the interface control logic between APB bus and the module. It is coming from CSC (Clock Source Controller) module. It can be enabled in **CSC_DAC_EN** register and select the clock source from APB clock or AHB clock in **CSC_DAC_CKS** register. User can plan the module clock is running or not beforehand for chip entering **SLEEP** mode by setting **CSC_SLP_DAC** register. Refer the System Clock chapter for more information.

Figure 29-2. DAC Process Clock Control



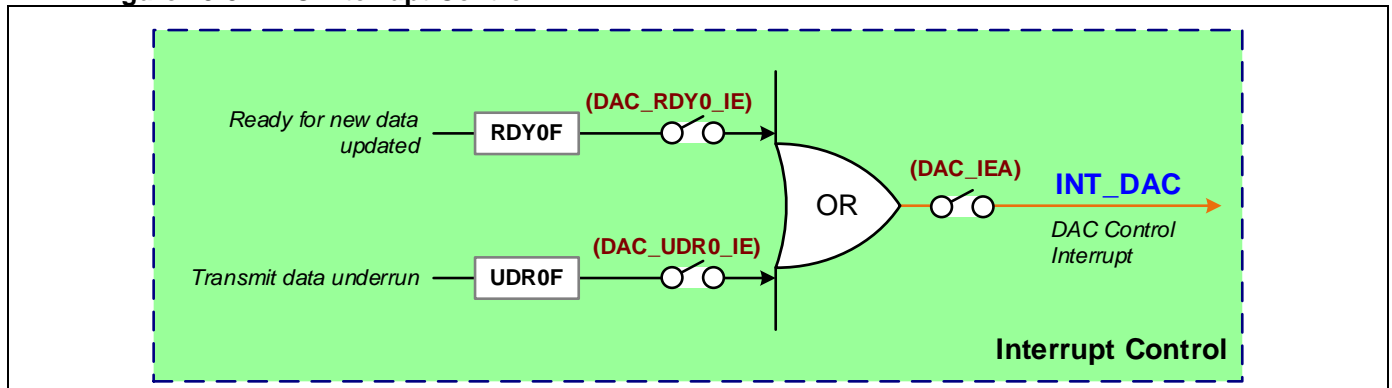
29.7. Interrupt and Event

29.7.1. DAC Interrupt Control

There is one signal of **INT_DAC** to be generated in this DAC module. **INT_DAC** sends to External Interrupt Controller (EXIC) to do as an interrupt event.

These interrupt flags are using for the interrupt service routine (ISR) flow control. Generally these flags are set by hardware and software clears them when the serviced job is completed in the related ISR. Each interrupt flag has one interrupt enable bit. User can enable or disable it. There is one interrupt all enable bit of **DAC_IEA** to enable or disable all the interrupt sources for this module.

Figure 29-3. DAC Interrupt Control



29.7.2. DAC Interrupt Flags

Generally these interrupt flags are set by hardware and clear by software writing 1. Refer the register descriptions for more information about the related interrupt flags and interrupt enable bit.

- **UDR0F**

It is DAC-0 conversion underrun event flag (**DAC_UDR0F**). There is a related interrupt enable register bit of **DAC_UDR0_IE**.

- **RDY0F**

It is DAC-0 ready flag (**DAC_RDY0F**) to update new data to data register. There is a related interrupt enable register bit of **DAC_RDY0_IE**.

29.8. DAC Output Voltage

29.8.1. Voltage DAC Output

The voltage DAC is embedded an output buffer and user can enable it by setting **DAC_BUF0_EN** register. The output voltage V_{out} is calculated by the formula in following.

The voltage DAC output is calculation by the formula:

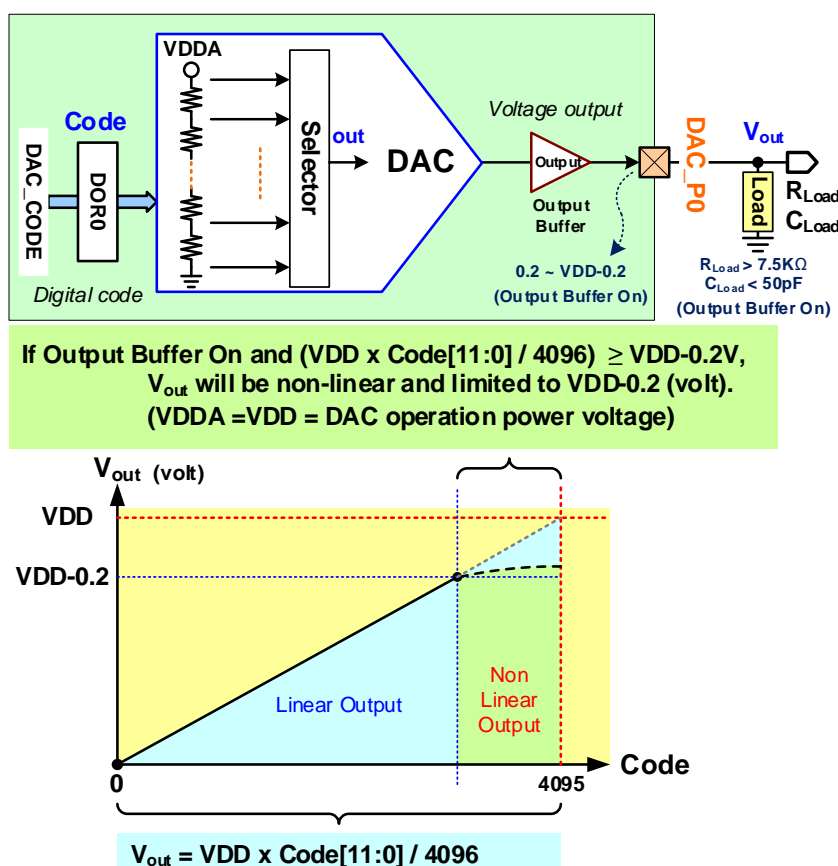
$$V_{out} = \frac{VDD * Code[11:0]}{4096}$$

As the DAC is voltage mode, the DAC is directly output the programmable voltage under the valid resistive load. Refer the related chip Data Sheet for the detail information about the output resistive load range with output buffer or not.

The DAC output is linear as the formula in following diagram when the output voltage is below the boundary voltage “DAC operation power voltage – 0.2 volt”. When the output is with buffer and the output voltage is over the boundary voltage “DAC operation power voltage – 0.2 volt”, the output is non-linear.

The following diagram is showing the DAC output block and output voltage curve.

Figure 29-4. DAC Output Voltage – Voltage DAC



29.9. DAC Conversion

The DAC conversion can be performed and start trigger by data register written, events. User can set the start trigger control source in **DAC_START0_SEL** register. For trigger by events, user can select external pin input or internal events and select the trigger signal edge for rising edge, falling edge or dual edge in **DAC_TRG0_SEL** register.

The DAC inputs code from a DAC data register **DAC_DAT0** and sends to the conversion output register **DAC_DOR0** for output conversion.

For the current DAC, the maximum conversion rate is 100 KHz and the DAC code updated interval must large 10us. For the voltage DAC, the maximum conversion rate is 1 MHz and the DAC code updated interval must large 1us.

- **Prepare DAC Conversion**

Prior to using the DAC function, the user should:

- Configure the DAC resolution by **DAC_RES0_SEL** bit.
- Configure the DAC code arrangement by **DAC_ALIGN0_SEL** bit.
- Initial DAC output voltage by setting **DAC_DAT0** register.
- Configure the selected input to the Analog-Input-Only mode by **PA_IOMz** register. (z=IO pin index)
- Turn on the DAC macro by setting the **DAC_DAC0_EN** bit.
- Turn on the DAC output buffer by setting the **DAC_BUF0_EN** bit.

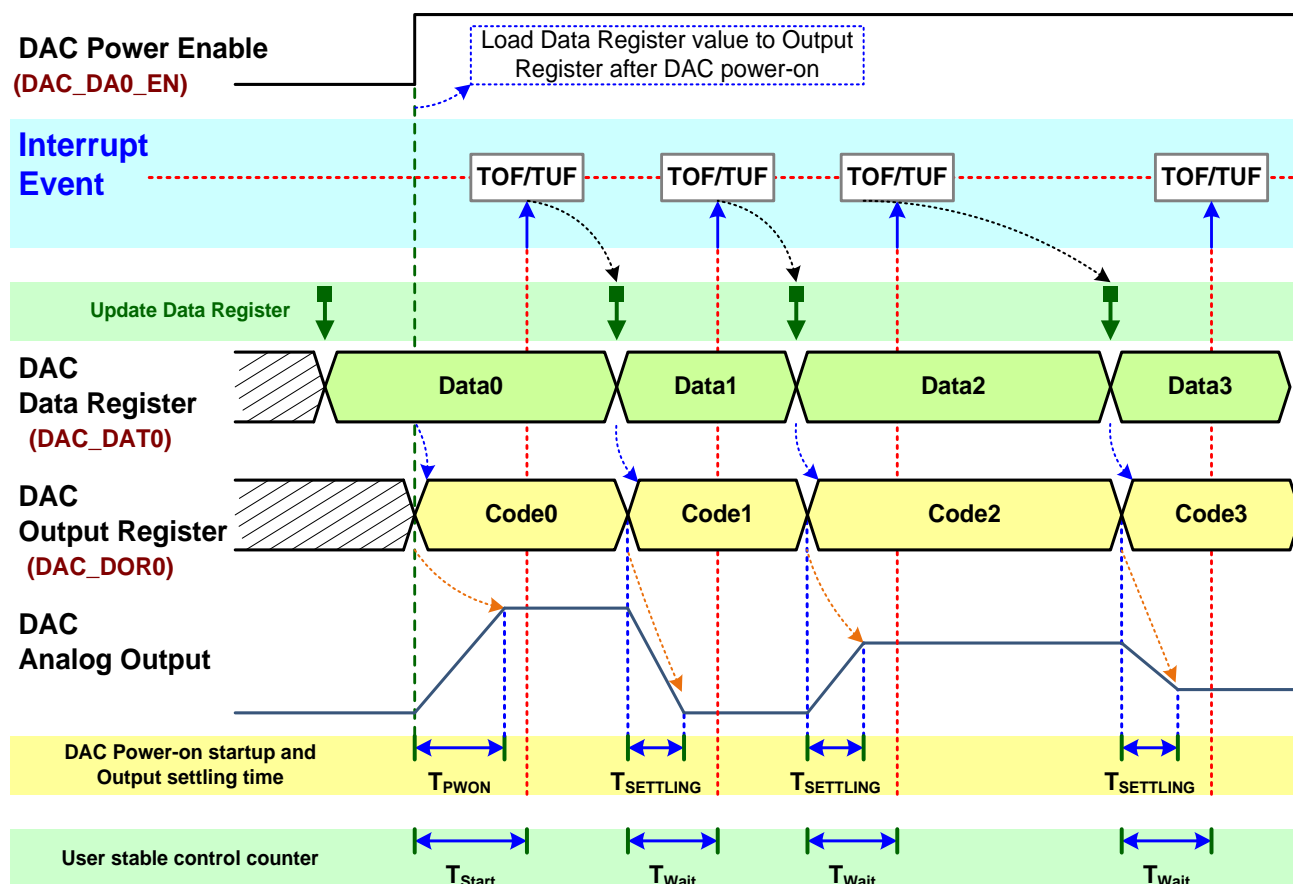
29.9.1. DAC Data Register Written

The DAC conversion can be performed and start trigger directly by data register written of every time. The DAC will directly copy the data register to DOR after data register is written. The DAC will automatically copy the data register to DOR after DAC power-on.

User can use the internal timer to control the DAC data code updated interval by using the timer overflow or underflow interrupt.

The following diagram is showing the example that start trigger by internal event of TMx timer overflow or underflow.

Figure 29-4. DAC Conversion – Output Update by Data Register Written



<Note-1> TOF/TUF : TMx overflow/underflow flag (x=Timer module index)

<Note-2> T_{PWON} : DAC Power On Startup Time , $T_{SETTLING}$: DAC Output Settling Time

<Note-3> T_{Start} : User define the power-on stable time by setting TMx timer. ($T_{Start} > T_{PWON}$)

<Note-4> T_{Wait} : User define the conversion stable time by setting TMx timer. ($T_{Wait} > T_{SETTLING}$)

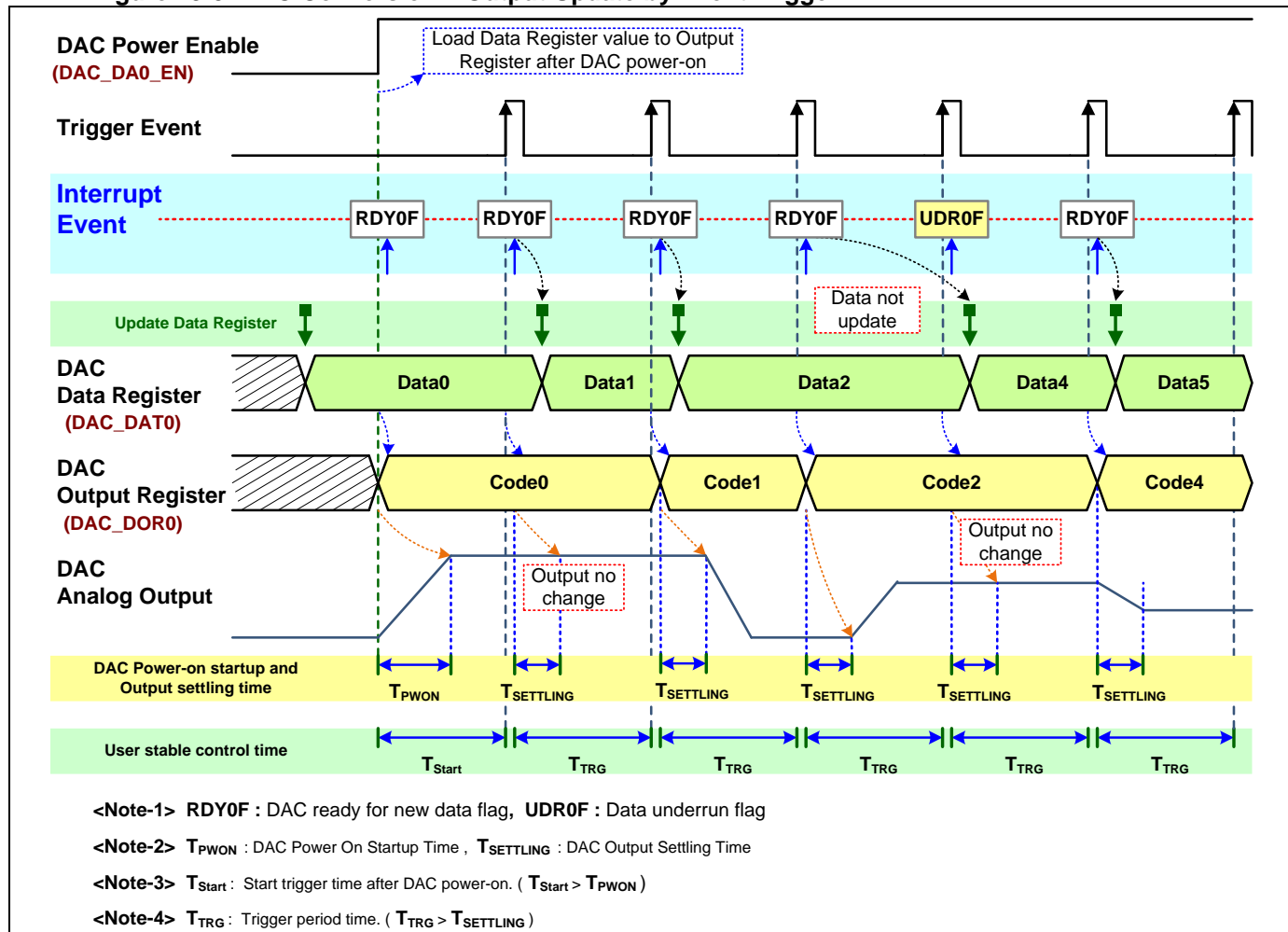
<Note-5> User can use any TMx timer, other available timer or software timer to control the DAC data update flow.

29.9.2. DAC Event Trigger

The DAC conversion can be performed and start trigger by external pin input (**DAC_TRG0** pin) or internal events. **DAC_RDY0F** is asserted at the selected edge of trigger event and the DAC will copy the data of data code register to DOR. Also the DAC will automatically copy the data register to DOR after DAC power-on.

DAC_UDR0F will be asserted if the next trigger edge is happened and data register has not yet updated.

Figure 29-5. DAC Conversion – Output Update by Event Trigger



29.9.3. DAC Data Resolution and Alignment

User can configure the code width of data register **DAC_DAT0** by 8-bit, 10-bit or 12-bit in **DAC_RES0_SEL** register. Also user can set data alignment in **DAC_ALIGN0_SEL** register for input code left/right justify. The following table is showing the DAC code format definition for data alignment and code width.

Table 29-2. DAC Data Alignment Definitions

Alignment	DAC_DAT0 (DAC data register)															
	B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
12bit-Right	x	x	x	x	M DAC Output Code[11:0]											
10bit-Right	x	x	x	x	x	x	M DAC Output Code[9:0]									
8bit-Right	x	x	x	x	x	x	x	x	M DAC Output Code[7:0]							
12bit-Left	M DAC Output Code[11:0]												x	x	x	x
10bit-Left	M DAC Output Code[9:0]										x	x	x	x	x	x
8bit-Left	M DAC Output Code[7:0]								x	x	x	x	x	x	x	x

<Note> MG32F02A132/072 are not supported 12-bit resolution

<Sign> M: MSb bit , x: don't care

29.10. DAC DMA Operation

29.10.1. DMA Module Configure

When the chip supports a DMA (direct memory access) controller, user can configure the DMA setting of transferred source/destination devices, channel request arbitration and others in the DMA module before a DMA data transaction. The DMA source and the destination can be memory or peripheral.

Refer the DMA chapter for more detail information about the DMA module configuration.

29.10.2. DAC DMA Control

After DMA configuration is finished, user needs to set the DAC module DMA enable bit of **DAC_DMA_EN**.

Finally, the related channel request start bit of **DMA_CHn_REQ** is necessary to be set to start the DAC DMA transaction.(n = DMA channel index) Then the transferred source/destination devices will assert the request signal to DMA controller and the DMA controller will assert the acknowledge signal to the request source/destination devices. At the time, the data transferred connection is built for DMA transaction.

- **DAC Out from DMA**

The **DAC_DMA_EN** register bit is used to enable DMA data transfer from DMA source to DAC transmitted output. During the DMA transaction cycle, the data ready flag of **DAC_RDY0F** is masked by hardware.

When the DAC DMA operation is running peripheral-to-peripheral transfer type with ADC module, the DAC must set the data resolution to 10-bit or 12-bit by setting **DAC_RES0_SEL** register and the ADC transfer packet data size must configure to 16-bit by setting **ADCx_DMA_DSIZE** register. The DAC output will be not normal if the DAC data resolution is set to 8-bit or the ADC transfer packet data size is set 32-bit.

29.10.3. DAC Interrupt Flag Control during DMA

During DMA operation cycle, the module's interrupt flags will control and act three types as following table.

One is masked during the DMA process (RDY0F flag). Another is to disable the DMA function after the flag (UDR0F flag) has asserted. At the time, hardware will clear the **DAC_DMA_EN** bit in this condition. Others are normally as same as the action of not processing in DMA operation.

Table 29-3. DAC Interrupt Flag Control for DMA Function

Action	Mask the Flag during DMA Process	DMA Disable after the Flag asserted (*1)	Normal Control
Peripheral	(Data flow flags)	(Error/Detect flags)	(Other flags)
DAC	DAC_RDY0F (*2)	DAC_UDR0F	

Note-1 : When the flag is asserted, it will not force peripheral DMA disabled if the related interrupt enable bit is not enabled.

Note-2 : **DAC_RDY0F** will be asserted after DMA TX complete.

29.11. DAC Application Circuit

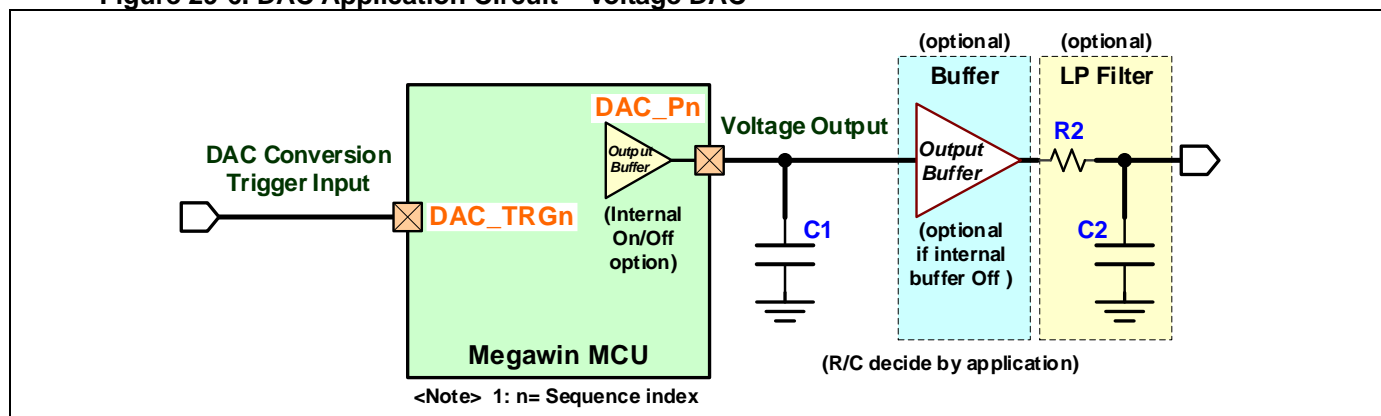
An optional **DAC_TRG0** pin is able to input the trigger signal for DAC output conversion.

29.11.1. Voltage DAC Application

The embedded output buffer is able to enable by application output resistive load for user request. Usually, the capacitor (**C1**) is necessary to use storing the DAC output and stabling the output voltage. An external low-pass filter (**R2/C2**) is optional to be implemented in the application circuit for more better outputting performance.

The following diagram is showing the voltage DAC application circuit.

Figure 29-6. DAC Application Circuit – Voltage DAC



30. Appended Information

30.1. Word Glossary

This table gives a brief of abbreviations used in this document.

Table 30-1. Word Glossary

Item	Descriptions
ON	ON mode ~ Chip in normal operation
SLEEP	SLEEP mode ~ Chip in sleep operation, CPU sleep
STOP	STOP mode ~ Chip in stop operation, CPU in Deepsleep, AHB/APB clock stop
Byte	data of 8-bit length
Half-Word	data of 16-bit length
Word	data of 32-bit length
Module	peripheral module for special function control
ADC	Analog-to-Digital Converter
AHB	Advanced High-performance Bus
APB	Advanced Peripheral Bus
ARGB	Addressable RGB LED
BOD	Brown-Out Detection
CAN	Controller Area Network
CCL	Configurable Custom Logic
CCP, IC/OC/PWM	Input Capture / Output Compare / Pulse Width Modulation
CMP / ACMP	Analog Comparator
CRC	Cyclic Redundancy Check
DAC	Digital-to-Analog Converter
DAP	ARM Cortex-Mx Debug Access Port
DMA	Direct Memory Access
EXTCK	External input clock
FIFO	First-In-First-Out
GPIO	General Purpose Input/Output
I2C	Inter Integrated Circuit
IAP	In-Application Programming (service by user application program code)
ICP	In-Circuit Programming (service by SWD/JTAG interface)
ISP	In-System Programming (service by boot load program code)
IHRCO	Internal high frequency RC oscillator
ILRCO	Internal low frequency RC oscillator
IrDA	Infrared Data Association
KBI	Key Board Interrupt ; Key pad Interrupt
LDO	Low-dropout regulator
LIN	Local Interconnect Network
MUX	Multiplexer or Data Selector
MCD	Missing clock detector
NCO	Numerically Controlled Oscillator
NVIC	Nested Vectored Interrupt Controller
OB	Hardware configuration Option Byte flash memory
OR	Hardware configuration Option Register
PLL	Phase-Locked Loop
SDT	State Detector
SMBus	System Management Bus (I2C protocol)
SPI	Serial Peripheral Interface
SWD	Serial Wire Debug
USB	Universal Serial Bus
VCAP / VR0	MCU core logic power and internal embedded LDO output pin (must place capacitors to be close this pin)
WIC	Wakeup Interrupt Controller
XOSC	Internal embedded crystal oscillator for external crystal circuit

<Note> This table gives a brief of abbreviations used in this document.

30.2. Modules Interconnection

The following table is showing the interconnection digital signals of the internal modules.

Table 30-2. Modules Interconnection Table

Module Output	Module Input Signals									
	ADC0	DAC	APB	APB	TMx (*1)	URT0/1/2	URT4/5/6/7	SPI0	I2Cx	EMB
ADC0_OUT	-	-	OBMn_BKS0	APB_ITR7	TM36_BKI					
CMP0_OUT	ADC0_TRGI	DAC_TRGI	OBMn_BKS0		TM36_BKI					
CMP1_OUT	ADC0_TRGI		OBMn_BKS1		TM36_BKI					
INT_BOD1			OBMn_BKS1		TM36_BKI					
INT_PA			OBMn_BKS0							
INT_PB			OBMn_BKS1	APB_ITR6	TM36_BKI					
INT_PC			OBMn_BKS0							
INT_PD			OBMn_BKS1	APB_ITR7						
INT_PE			OBMn_BKS2							
ICKO_INT			-	APB_ITR7						
RTC_OUT			-	APB_ITR7						
NCO_P0			-	APB_ITR7		CK_URT _x	CK_URT _x			
CCL_P0			OBMn_BKS0							
CCL_P1			OBMn_BKS1							
APB_ITR6		DAC_TRGI		-	TMx_ITR					
APB_ITR7		DAC_TRGI		-	TMx_ITR					
TM00_TRGO	ADC0_TRGI	DAC_TRGI	OBMn_BKS0	APB_ITR6		CK_URT _x	CK_URT _x	CK_SPI0	CK_I2Cx	
TM01_TRGO	ADC0_TRGI	DAC_TRGI	OBMn_BKS1	APB_ITR7		CK_URT _x _RX				
TM10_TRGO		DAC_TRGI	OBMn_BKS0	APB_ITR6		CK_URT _x _RX				
TM16_TRGO			OBMn_BKS1	APB_ITR7						
TM20_TRGO	ADC0_TRGI		OBMn_BKS0	APB_ITR6						
TM26_TRGO			OBMn_BKS1	APB_ITR7						
TM36_TRGO	ADC0_TRGI		OBMn_BKS2	APB_ITR6	-					
TM36_XOR				APB_ITR7	-					
TM20_OC00			OBMn_BKS0							
TM20_OC10			OBMn_BKS1							
TM36_OC2			OBMn_BKS0							
TM36_OC3			OBMn_BKS1							
TM10_CKO								SPI0_CLK		EMB_MOE EMB_MWE
TM16_CKO								SPI0_CLK		EMB_MOE EMB_MWE
TM20_CKO								SPI0_CLK		EMB_MOE EMB_MWE
URT0_TX			OBMn_BKS0			-				
URT0_RX			OBMn_BKS0			-				
URT1_TX			OBMn_BKS1			-				
URT1_RX			OBMn_BKS1			-				
URT2_TX			OBMn_BKS0			-				
URT2_RX			OBMn_BKS0			-				
URT1_BRO				APB_ITR7		-				
URT1_TMO				APB_ITR6		-				
URT2_BRO			OBMn_BKS2	APB_ITR6		-				
URT2_TMO			OBMn_BKS2	APB_ITR6		-				
SPI0_MOSI			OBMn_BKS2					-		
EMB_MOE								SPI0_CLK		-
EMB_MWE								SPI0_CLK		-

<Note> *1: TMx = {TM00, TM01, TM10, TM16, TM20, TM26, TM36}

31. Revision History

Version 5.2 (2025_1211)		Chapter
1	Add the table of "Table 3-6. LDO Control" and the descriptions of PW_LCTL_SLP and PW_LCTL_STP.	3.10.2
2	Change "APB Control" to "APX Control" in the figure of "Figure 15-1. APX Interrupt Control".	15.4.1
3	Update the table of "Table 15-2. CCL Input Mux Signal Selection" for MG32F02V032.	15.5.2
4	Update the figure of "Figure 17-11. UART Other IO Control" about RTS control.	17.8.1
5	Update the table of "Table 17-11. UART BR Baud-Rate Timer On/Off Control".	17.15.1
6	Update the table of "Table 21-2. Timer Modules' Functions" to support "Input duty capture" function for TM26 module.	21.3.2
7	Add the description of ADCx_GAIN_MUL register and update the description of ADCx_BUF_BIAS register. Add the table of "Table 27-6. ADC Gain Calculation - x1 ~ x128".	27.8.5
8	Remove the table of "Table 27-16. ADC Interrupt Flag Control for DMA Function – MG32F02A132/072".	27.15.3
9	Update the figure of "Figure 29-1. DAC Main Block – Voltage DAC" about bypass output buffer.	29.4.1
10	Update the DAC output voltage formula and the figure of "Figure 29-4. DAC Output Voltage – Voltage DAC".	29.8.1
11	Add the sequence list of prepare DAC conversion.	29.9
Version 5.1 (2025_0416)		Chapter
1	Update the table of "Table 2-1. Chip Implementation Table" in the section of "Chip Implementation Summary".	2.2.1
2	Add AHB clock divider limitation descriptions for embedded Flash as DMA source in the section of "12.8.1. DMA Source and Destination".	12.8.1
3	Update the formula of CK_LCD_FRM in the sections of "25.8.1. LCD Clock" and "25.9. LCD Drive Timing".	25.8.1 25.9.1
4	Update "OVD_DIS" value in the table of "Table 25-3. LCD Power Source and R-Ladder Control Mode".	25.8.2
5	Add the figure of "Figure 25-8. LCD CPRF Interrupt and PRDYF Status" and related descriptions.	25.8.3
6	Add a separated new section of "25.9.1. LCD Clock and Frame" from 25.9 section and add a new figure of "Figure 25-13. LCD Clock and Frame Timing".	25.9.1
7	Add new figure of "Figure 25-23. LCD Dead Timing – In Frame (Type-A)" in the section of "25.9.6. LCD Dead Timing".	25.9.6
8	Remove the descriptions of ADC differential mode function.	27.4 27.7
9	Update the figures of "Figure 28-5. CMP Analog Comparator CMP0 Block" and "Figure 28-6. CMP Analog Comparator CMP1 Block" to add OP0_P0 internal input channel.	28.8.3 28.8.4
10	Update the table of "Table 28-3. CMP Comparator IVREF R-Ladder Output" to add 5/3.3/1.55 voltage row.	28.9.2
Version 5.0 (2024_1101)		Chapter
1	Add and modify content about new additional MG32F02N128/K128/N064/K064 chip. Update the tables in the section of "Chip Implementation" and "Modules' Functions" for each module chapter.	
2	Remove MG32F02A132/A072/A032 chip and modify related content. Update the tables in the section of "Chip Implementation" and "Modules' Functions" for each module chapter.	
3	Update the table of "Table 2-1. Chip Implementation Table" in the section of "Chip Implementation Summary".	2.2.1
4	Add the section of "2.3.4. MG32F02N128/K128/N064/K064 Main Block" in Chip and System chapter.	2.3.3
5	Add descriptions about ADC PGA offset Calibration in the section of "Manufacturer ADC	8.3.2

	Calibration Value".	
6	Add "Manufacturer OPA Calibration Value" section in APB chapter.	8.3.4
7	Add "Multi-Function Signal" section in APB chapter.	14.9
8	Add "SDT Control" section in APX chapter.	15.7
Version 4.5 (2024_0619)		Chapter
1	Change the key words of "ARM" and "Cortex" to "ARM®" and "Cortex®".	
2	Add GPIO wakeup descriptions for STOP mode in the sections of "3.11.1. Wakeup Event Source" and "10.7.1. EXIC Interrupt Control".	3.11.1 10.7.1
3	Add DMA "upper 2K-bytes SRAM" descriptions in the sections of "7.8.3. On-Chip Data RAM" and "12.8.4. DMA SRAM Using".	7.8.3 12.8.4
4	Update the descriptions of "Erase mode" in the section of "7.9.1. Flash Memory Access".	7.9.1
5	Update URT0 to URTx in the tables of "Table 12-8. Peripheral Module Interrupt Flag Control for DMA – MG32F02A032", "Table 12-9. Peripheral Module Interrupt Flag Control for DMA – MG32F02A128/U128/A064/U064", "Table 12-10. Peripheral Module Interrupt Flag Control for DMA – MG32F02V032".	12.9.6
6	Remove "SDT Control" function in the APX chapter.	15
7	Update the figure of "Figure 20-37. Timer Break Input Source" to correct the strings of "TMx_BKI_ENz" and "TMx_BKE_ENz".	20.15.1
Version 4.4 (2023_1114)		Chapter
1	Update the IO number in the table of "Table 9-1. GPIO Implementation".	9.3.1
2	Rename "NCO0_P0" to "NCO_P0".	17.6.2
3	Remove the notice of "[Notify]: The SPIx_D[4..7] is not supported for MG32F02A032." and refer related chip implement information in the section of "18.3.1 Chip Implementation".	18.5.1
4	Update the note in the table of "Table 18-6. SPI Data Control Table".	18.12
5	Remove the notice of "[Notify]: The TM20/TM26 is not supported for MG32F02A032." and refer related chip implement information in the section of "20.3.1 Chip Implementation".	20.9.1
Version 4.3 (2023_0524)		Chapter
1	Update the descriptions about VREF+ in the section of "3.5. Power Supply".	3.5
2	Update the descriptions about DMA using in the section of "7.8.3. On-Chip Data RAM".	7.8.3
3	Add the table of "Table 17-3. UART Pin Swap Function Mapping" in the section of "17.8.1. UART IO Control".	17.8.1
4	Rename "Figure 24-1. ADC Block Diagram" to "Figure 24-1. ADC Block Diagram with Differential Mode" and add "Figure 24-2. ADC Block Diagram with Single-End Mode"	24.4
5	Update the descriptions about VPG in the paragraph of "ADC Input from VPG" of the section of "24.8.2. ADC Input Channels".	24.8.2
6	Update the descriptions about VREF+ and add the figure of "Figure 24-25. ADC Application Circuit without VREF+ Pin" in the section of "24.16. ADC Application Circuit".	24.16
Version 4.2 (2022_1214)		Chapter
1	Move the section of "1.4. Word Glossary" into the Appended Information chapter.	1.4
2	Update the table of "Table 2-1. Chip Implementation Table" in the section of "Chip Implementation Summary" about MG32F02A128/A064 and MG32F02U128/U064.	2.2.1
3	Update the figure of "Figure 3-1. Power Mode State" in the section of "3.4. Power Operation Mode" about the wakeup time from STOP.	3.4
4	Update the figure of "Figure 9-1. GPIO Control Block".	9.4
5	Separate the section of the "9.6.1. Analog IO and Digital Input Structure" to two sections of "9.6.1. Analog IO Structure" and "9.6.2. Digital Input Structure".	9.6
6	Update Flash as DMA source list in the table of "Table 12-1. DMA Implementation".	12.3

7	Update the table of “Table 12-2. DMA Memory Source Support” and the descriptions in the “12.8.1. DMA Source and Destination” section.	12.8.1
8	Separate the table of “Table 12-3. DMA Channel Source/Destination Request Selection” to two tables of “Table 12-3. DMA Channel Source Request Selection” and “Table 12-4. DMA Channel Destination Request Selection” in the section of the “12.8.1. DMA Source and Destination”.	12.8.1
9	Add NCO_P0 signal in the figure of “Figure 14-3. Timer Common Trigger/Clock Source Selection”.	14.5.2
10	Remove the dummy column of MG32F02V032 and change the table style in the table of “Table 16-2. I2C Modules' Functions”.	16.3.2
11	Update the descriptions about signal filter in the section of “16.5.2. IO Configure”.	16.5.2
12	Change the table style of the tables of “Table 17-2. UART Modules' Functions”, “Table 18-2. SPI Modules' Functions”, “Table 24-2. ADC Modules' Functions”, “Table 25-2. Analog Comparator Modules' Functions”.	17.3.2 18.3.2 24.3.2 25.3.2
13	Add the section of “18.11.4. SPI Received Data Frame Size” in SPI chapter.	18.11
14	Update the descriptions in the section of “19.5.1. IO Signals” and remove the figure of “Figure 19-2. USB Pin control”.	19.5.1
15	Update the descriptions in the section of “19.9.4. USB Buffer Mode” and the table of “Table 19-3. USB Buffer Mode Control”.	19.9.4
16	Remove “Chip” row in the table of “Table 20-2. Timer Modules' Functions” in Timer chapter.	20.3.2
17	Add ADCx_TRGI, ADCx_IP and ADCx_IN signals in the figure of “Figure 24-1. ADC Block Diagram”	24.4
18	Add DAC_TRGI signal in the figure of “Figure 26-1. DAC Main Block – Current DAC” and “Figure 26-2. DAC Main Block – Voltage DAC”	26.4.1 26.4.2
19	Add Appended Information chapter and the section of “27.2. Modules Interconnection”.	27
Version 4.1 (2022_0316)		Chapter
1	Add “VR0 / VCAP” in the table of “Table 1-1. Word Glossary”.	1.4
2	Merge the figures of “Figure 3-5. Power Voltage Detect - MG32F02A132/A072/A032” and “Figure 3-6. Power Voltage Detect - MG32F02A128/U128/A064/U064/V032” to the figure of “Figure 3-5. Power Voltage Detect” in the section of “3.7.3. Power Voltage Detection Threshold”.	3.7.3
3	Add the table of “Table 3-3. Voltage Detect Threshold” in the section of “3.7.3. Power Voltage Detection Threshold”.	3.7.3
4	Add the descriptions of VR0 (VCAP) pin in the section of “3.12. Chip Power Application Circuit”.	3.12
5	Update the figure of “Figure 4-5. Chip Reset Timing” in the section of “4.4.2. Power-On Reset and Chip Reset Timing”.	4.4.2
6	Update the table of “Table 7-1. Embedded Memory Implementation” in System Memory chapter.	7.3
7	Update the descriptions and figure of “Figure 12-7. DMA SRAM Using Suggestion” in the section of “12.8.4. DMA SRAM Using”.	12.8.4
8	Update the descriptions and figures in the “15.6 SDT Control” section.	15.6
9	Update the figure of “Figure 17-14. UART RX/TX IO Control – MG32F02A128/U128/A064/U064/V032” in the section of “17.8.1. UART IO Control”.	17.8.1
10	Remove the section “UART SDT Input” in the UART chapter.	17.8
11	Update the figures of “Figure 18-4. SPI Data IO Control” and “Figure 18-5. SPI Clock/NSS IO Control” in the section of “18.8.1. SPI IO Control”.	18.8.1
12	Remove the section “SPI SDT Input” in the SPI chapter.	18.8
Version 4.0 (2021_1230)		Chapter
1	Add and modify content about new additional MG32F02V032 chip. Update the tables in the section of “Chip Implementation” and “Modules' Functions” for each module chapter.	
2	Update the table of “Table 2-1. Chip Implementation Table” in the section of “Chip Implementation Summary”.	2.2.1

3	Add the section of “2.3.4. MG32F02V032 Main Block” in Chip and System chapter.	2.3.4
4	Add the descriptions about PLL VCO output frequency formula in the section of “5.7.3 PLL Clock”	5.7.3
5	Add the descriptions about five levels of output drive strength control and anti-current leakage IO in the sections of “9.5.2. IO Configuration”.	9.5.2
6	Add ASB function and the sections of “15.7. ASB Control” and “15.8. APX DMA Operation” in APX chapter.	15.7 15.8
7	Add the descriptions about NSSF flag in the sections of “17.7.3. UART Interrupt Flags”.	17.7.3
8	Add the descriptions about UART HFC connection in the sections of “17.9.1. UART Connect for UART/IrDA Mode”.	17.9.1
9	Add the new sections of “17.9.5. UART Connect for SYNC Mode”.	17.9.5
10	Update the descriptions about operation mode in sections of “17.11.2. UART Operation Mode Setting”.	17.11.2
11	Add the descriptions about HFC signals in sections of “17.24.1. UART Connection for Hardware Flow Control”.	17.24.1
12	Add the descriptions about SPI slave mode high speed operation in sections of “18.10.5. SPI High Speed Operation”.	18.10.5
13	Add the descriptions about IDCF flag in the sections of “20.7.3. Timer Interrupt Flags”.	20.7.3
14	Update the descriptions from “CMP0_OUT and CMP1_OUT” to “CMP2_OUT and CMP3_OUT” in the sections of “20.10.1. TM3x Timer Input/Output Channels Block” and “20.10.2. TM2x Timer Input/Output Channels Block”.	20.10.1 20.10.2
15	Update the figures of “Figure 20-22. Timer Input Capture Timing for Full-Counter Mode – MG32F02A132 /072” in the section of “20.12.3. Capture Control and Status”.	20.12.3
16	Add the section of “Duty Capture Control” in Timer chapter.	20.12.4
17	Add the descriptions about new timer input capture DMA enable bits in the sections of “20.17.2. Timer DMA Control”.	20.17.2
18	Add the new sections of “23.6.3. RTC Clock Status”.	23.6.3
19	Update the table of “Table 24-4. ADC Channel Definitions” in the section of “24.8.2. ADC Input Channels”.	24.8.2