BE-U1000

Reference Manual

Document ID: BE-U-RM#1740

BAIKAL ELECTRONICS, JSC. Registration code: 7707767484

Revision 1.3.0

August 25, 2026

Revision History

Revision	Date	Description
1.0.0	December 26, 2025	Initial Release
1.1.0	February 4, 2026	Added: • **13.4.1.6. Serial Master Operation Mode (QSPI/SPI Master only)** • **13.4.1.7. Serial-Slave Operation Mode (SPI Slave only)** Updated: • `CTRLR0.SLV_OE` bit description in the **13.4. SPI & QSPI** section • `SER.SER` bit description in the **13.4. SPI & QSPI** section
1.3.0	August 25, 2026	Added: • Value After Reset for the `PLLSET.TEST` bit in **5.2.2.2. PLL Control Register (`PLLSET`)** section • Value After Reset for the `TRACESEL` bit in **5.2.2.7. System Control Register 0 (`SYSCR0`)** section Updated: • **Figures 5-18**, **5-19** and **5-20** in **5.1.3.1. Alternate Functions** section • `CLKSEL.CANX2CLKSEL` bit values in **5.2.2.1. Clock Select Register (`CLKSEL`)** section • **11.1.2.3. Discontinuous Mode** section • `ASQR` register description in **11.2.2.5. ADC Sequence Register (`ASQR`)** section • `AFMRi` register description in **13.2.2.2.2.4. Acceptance Filter i Mask Register (`AFMRi`)** section • `IIR.IID` bit description in **13.3.2.2.6. Interrupt Identity Register (`IIR`)** section • **13.6.1.3.5. Receive Channel Interrupts** section • `TCR0.WLEN`, `RFCR0.RXCHDT` and `TFCR0.TXCHET` bit descriptions in **13.6 I2S** section • `DATA_TOGGLE` bit description in **14.1.2.2.15. Endpoint 0 Control and Status Register 1 (`CSR0H`)** and **14.1.2.2.16. Tx Endpoint #n Control and Status Register 1 (`TXCSRH`)** sections • **14.1.2.2.19. Rx Endpoint #n Control and Status Register 1 (`RXCSRH`)** section • **14.1.2.2.57. High Speed Timeout Increase Register (`C_T_HSBT`)** section Corrected: • CANFD number on the **Figure 5-14** in **5.1.1.2.6** `can_clk` **and** `can_clk_x2` **Clock Domains** section • Spelling of “ETP” to “ETR” input signal in **12.1.1.2.1. External Trigger Control** section

Revision	Date	Description
		• `WDT_CR` register default value "0x40" to "0x10" in **12.4.2.1. Register Map** section • `ESTAT` bit values in **13.2.2.2.1.7. Status Register (`SR`)** section • `TCR` register offset value "0x94" to "0x98" and spelling of “`RR0`" to “`RRi`" bit in **13.2.2.2.1.16. Tx Buffer Cancel Request (`TCR`)** section • `ET` bit values in **13.2.2.2.3.2. TXE FIFO TBi Data Length Code Register (`TXE_FIFO_TBi_DLC`)** section • Bit values in **13.6.2.2.18. Receive FIFO Configuration Register (`RFCR0`)** and **13.6.2.2.19. Transmit FIFO Configuration Register (`TFCR0`)** sections • Spelling of “`REQPKT`" to “`EP0`" bit in **14.1.1.13.2.2. IN Data Phase** section • Value After Reset for the `TRPIBCONTROL.TR_PIB_MODE` bit in **16. Tracer** section

Glossary

ADC	Analog-to-Digital Converter
BEU	Bus Error Unit
BI	Break Interrupt
BRP	Baud Rate Prescaler
CAN FD	Controller Area Network with Flexible Data-Rate
CLIC	Core-Local Interrupt Controller
CLINT	Core Local Interruptor
COM	Commutation Event
CRU	Control Registers Unit
CSRs	Control and Status Registers
DDR	Dual Data-Rate
DM	Debug Module
DMA	Direct Memory Access
DMI	Debug Module Interface
DPD	Deep Power Down
DTM	Debug Transport Module
EBT	Early Burst Termination
EEPROM	Electrically Erasable Programmable Read-Only Memory
eFLASH	Embedded FLASH
EOC	End of Conversion
FE	Framing Error
FP	Front Port
FSM	Finite State Machine
GPIO	General Purpose Input-Output
HMP	Hardware Performance Monitor
HNP	Host Negotiation Protocol
I2C	Inter-Integrated Circuit
I2S	Inter-IC Sound
IrDA	Infrared Data Association
ISR	Interrupt Service Routine
LDO	Low Dropout
LLI	Linked List Item
LSB	Least Significant Bit

MP	Memory Port
MSB	Most Significant Bit
NMIs	Non-Maskable Interrupts
PE	Parity Error
PEE	Protocol Exception Event
PFD	Phase-Frequency Detector
PIB	Pin Interface Block
PIO	Programmable Input/Output
PLL	Phase-Locked Loop
PMP	Physical Memory Protection
PP	Peripheral Port
PWM	Pulse Width Modulation
QSPI	Quad Serial Peripheral Interface
OTG	On-The-Go
RBMM	Rx Buffer Management Module
ROM	Read-Only Memory
SIR	Serial Infrared
SOC	Start of Conversion
SoC	System-on-Chip
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
SRP	Session Request Protocol
SSP	Secondary Sample Point
TBMM	Tx Buffer Management Module
TCM	Tightly Coupled Memory
TDC	Transmitter Delay Compensation
THRE	Transmit Holding Register Empty
UART	Universal Asynchronous Receiver/Transmitter
UEV	Update Event
VCO	Voltage Controlled Oscillator
WDT	Watchdog Timer

Table of contents

List of Tables ... 28
List of Figures ... 42
Glossary ... 5
1. Introduction ... 44
2. Architecture ... 45
3. Memory Map ... 46
4. Interrupt List ... 51
4.1. Core 0/1 Interrupts ... 51
4.2. Core 2 Interrupts ... 55
5. Control Registers Unit ... 56
5.1. CRU Functional Description ... 56
5.1.1. Clock Generation ... 56
5.1.1.1. Clock Management ... 56
5.1.1.1.1. Clock Sources ... 58
5.1.1.1.2. PLL ... 60
5.1.1.1.2.1. PLL Output Frequency ... 60
5.1.1.1.2.2. PLL Stability Tracking ... 61
5.1.1.1.3. Dividers ... 62
5.1.1.1.4. Output Clocks Control ... 62
5.1.1.2. Clock Domains ... 63
5.1.1.2.1. `cclk` Clock Domain ... 63
5.1.1.2.2. `pclk_0` Clock Domain ... 64
5.1.1.2.3. `pclk_1` Clock Domain ... 65
5.1.1.2.4. `pclk_2` Clock Domain ... 66
5.1.1.2.5. `hclk` Clock Domain ... 67
5.1.1.2.6. `can_clk` and `can_clk_x2` Clock Domains ... 67
5.1.1.2.7. `tclk` Clock Domain ... 68
5.1.2. Reset Sources ... 68
5.1.3. I/O Pins Configuration ... 69
5.1.3.1. Alternate Functions ... 70
5.1.3.2. Input Enable ... 71
5.1.3.3. Drive Strength ... 71
5.1.3.4. Pull-Up and Pull-Down Resistors ... 71
5.2. Control Registers ... 72
5.2.1. Register Map ... 72
5.2.2. Register Descriptions ... 73
5.2.2.1. Clock Select Register (`CLKSEL`) ... 73
5.2.2.2. PLL Control Register (`PLLSET`) ... 75
5.2.2.3. Clock Control Register 0 (`CLKCR0`) ... 76
5.2.2.4. `pclk0` Clock Gating and Reset Control Register (`PCLK0EN`) ... 78
5.2.2.5. `pclk1` Clock Gating and Reset Control Register (`PCLK1EN`) ... 82

5.2.2.6. pclk2 Clock Gating and Reset Control Register (PCLK2EN) 85
5.2.2.7. System Control Register 0 (SYSCR0) 87
5.2.2.8. System Control Register 1 (SYSCR1) 90
5.2.2.9. System Control Register 2 (SYSCR2) 90
5.2.2.10. Main Bus Matrix Prority Control Register 0 (PRIOR0) 90
5.2.2.11. Main Bus Matrix Prority Control Register 1 (PRIOR1) 91
5.2.2.12. I/O Pull-Up Control Register 0 (IOPUCR0) 93
5.2.2.13. I/O Pull-Up Control Register 1 (IOPUCR1) 94
5.2.2.14. I/O Pull-Down Control Register 0 (IOPDCR0) 94
5.2.2.15. I/O Pull-Down Control Register 1 (IOPDCR1) 95
5.2.2.16. I/O Alternate Function Control Register 0 (IOAFCR0) 96
5.2.2.17. I/O Alternate Function Control Register 1 (IOAFCR1) 98
5.2.2.18. I/O Alternate Function Control Register 2 (IOAFCR2) 99
5.2.2.19. I/O Alternate Function Control Register 3 (IOAFCR3) 101
5.2.2.20. I/O Alternate Function Control Register 4 (IOAFCR4) 103
5.2.2.21. I/O Alternate Function Control Register 5 (IOAFCR5) 105
5.2.2.22. I/O Drive Strength Control Register 0 (IODSCR0) 107
5.2.2.23. I/O Drive Strength Control Register 1 (IODSCR1) 108
5.2.2.24. I/O Drive Strength Control Register 2 (IODSCR2) 109
5.2.2.25. LDO Control Register (LDOCR) 110
5.2.2.26. USB PHY Control and Status Register (USBCR) 111
5.2.2.27. System Status Register (SYSSR0) 113
5.2.2.28. WDT Clear Key Register (WDTCLRKEY) 114
5.2.2.29. Interrupt Source Control Register (INTCR0) 114
5.2.2.30. Clock Control Register 1 (CLKCR1) 115
5.2.2.31. Flash NVR Control Register (FLASHNVRCR) 116
5.2.2.32. Core 1 Control Register (CORE1CR) 116
6. RISC-V Cores 118
6.1. BR-350 Core 118
6.1.1. BR-350 Core Functional Description 119
6.1.1.1. Non-Maskable Interrupts (int_nmi) 119
6.1.1.2. Physical Memory Protection 119
6.1.1.3. Hardware Performance Monitor 119
6.1.1.4. Debug 121
6.1.1.4.1. Core Debug Mode 122
6.1.1.4.2. Trigger Module 124
6.1.1.5. CLINT 124
6.1.1.6. Bus Error Unit 124
6.1.1.7. CLIC 124
6.1.2. BR-350 Core Registers 124
6.1.2.1. Control and Status Registers 125
6.1.2.1.1. CLIC CSRs 125
6.1.2.1.1.1. CSRs Map 125
6.1.2.1.2. Trigger Module CSRs 125

6.1.2.1.2.1. CSRs Map....125
6.1.2.1.3. Core Debug CSRs....126
6.1.2.1.3.1. CSRs Map....126
6.1.2.1.4. Core CSRs....126
6.1.2.1.4.1. CSRs Map....126
6.1.2.1.4.2. CSR Descriptions....127
6.1.2.1.4.2.1. Machine NMI Trap Handler Address (NMITVEC)....127
6.1.2.1.4.2.2. Core Configuration (XCCFG)....127
6.1.2.1.4.2.3. Instruction Cache Configuration (XICFG)....127
6.1.2.2. DTM Registers....128
6.1.2.2.1. Register Map....128
6.1.2.3. DM Registers (access via DMI)....128
6.1.2.3.1. Register Map....128
6.1.2.4. CLINT Registers....129
6.1.2.4.1. Register Map....129
6.1.2.4.2. Register Descriptions....130
6.1.2.4.2.1. Machine Software Interrupt Pending (MSIP0)....130
6.1.2.4.2.2. Machine Timer Compare (MTIMECMP0)....130
6.1.2.4.2.3. Machine Timer Value (MTIME)....130
6.1.2.5. Bus Error Unit Registers....131
6.1.2.5.1. Register Map....131
6.1.2.5.2. Register Descriptions....131
6.1.2.5.2.1. CPU Status Register (STATUS_CPU)....131
6.1.2.5.2.2. CPU Interrupt Control Register (CONTROL_CPU)....132
6.1.2.5.2.3. CPU Error Address Low Register (ADDR_LO_CPU)....132
6.1.2.5.2.4. CPU Error Address High Register (ADDR_HI_CPU)....132
6.1.2.6. CLIC Registers....133
6.1.2.6.1. Register Map....133
6.2. BM-310 Core....134
6.2.1. BM-310 Core Functional Description....134
6.2.1.1. Debug....134
6.2.1.1.1. Core Debug Mode....135
6.2.1.1.2. Trigger Module....136
6.2.1.2. CLINT....137
6.2.2. BM-310 Core Registers....137
6.2.2.1. Control and Status Registers....137
6.2.2.1.1. Trigger Module CSRs....137
6.2.2.1.1.1. CSRs Map....137
6.2.2.1.2. Core Debug CSRs....137
6.2.2.1.2.1. CSRs Map....137
6.2.2.1.3. Core CSRs....138
6.2.2.1.3.1. CSRs Map....138
6.2.2.1.3.2. CSR Descriptions....138
6.2.2.1.3.2.1. Core Configuration (XCCFG)....138

6.2.2.2. DTM Registers 139
6.2.2.2.1. Register Map 139
6.2.2.3. DM Registers (access via DMI) 139
6.2.2.3.1. Register Map 139
6.2.2.4. CLINT Registers 140
6.2.2.4.1. Register Map 140
6.2.2.4.2. Register Descriptions 140
6.2.2.4.2.1. Machine Software Interrupt Pending (MSIP0) 140
6.2.2.4.2.2. Machine Timer Compare (MTIMECMP0) 141
6.2.2.4.2.3. Machine Timer Value (MTIME) 141
7. Core 2 PIO 142
7.1. PIO Registers 142
7.1.1. Register Map 142
7.1.2. Register Descriptions 143
7.1.2.1. Interrupt Register (INT) 143
7.1.2.2. Input Data Register (PIO_IN) 143
7.1.2.3. Enable Register (PIO_EN) 143
7.1.2.4. Output Data Register (PIO_OU) 144
8. DMA Controller 145
8.1. DMA Functional Description 146
8.1.1. Handshaking Interface 146
8.1.2. Basic Interface Definitions 152
8.1.3. Memory Peripherals 153
8.1.4. Software Handshaking 153
8.1.5. Handshaking Interface Operation 154
8.1.5.1. Single Transaction Region 154
8.1.5.2. Early-Terminated Burst Transaction 155
8.1.5.3. Software Handshaking Operation 155
8.1.5.4. Single Transactions 156
8.1.6. Setting up Transfers 156
8.1.7. Peripheral Burst Transaction Requests 157
8.1.7.1. Transmit Watermark Level and Transmit FIFO Underflow 157
8.1.7.2. Choosing the Transmit Watermark Level 157
8.1.7.3. Selecting CTLx.DEST_MSIZE and Transmit FIFO Overflow 158
8.1.7.4. Receive Watermark Level and Receive FIFO Overflow 158
8.1.7.5. Choosing the Receive Watermark level 158
8.1.7.6. Selecting CTLx.SRC_MSIZE and Receive FIFO Underflow 159
8.1.8. Generating Requests for the Master Bus Interface 159
8.2. DMA Registers 160
8.2.1. Register Map 161
8.2.2. Register Descriptions 164
8.2.2.1. Channel Registers 164
8.2.2.1.1. Source Address Register (SARx) for Channel x 164
8.2.2.1.2. Destination Address Register (DARx) for Channel x 165

8.2.2.1.3. Hardware Realignment of SAR/DAR Registers ... 165
8.2.2.1.4. Linked List Pointer Register (LLPx) for Channel x ... 166
8.2.2.1.5. Control Register (CTLx) for Channel x ... 166
8.2.2.1.6. Configuration Register (CFGx) for Channel x ... 170
8.2.2.2. Interrupt Registers ... 174
8.2.2.2.1. Raw Status for IntTfr Interrupt (RAW_TFR) ... 174
8.2.2.2.2. Raw Status for IntBlock Interrupt (RAW_BLOCK) ... 174
8.2.2.2.3. Raw Status for IntSrcTran Interrupt (RAW_SRC_TRAN) ... 175
8.2.2.2.4. Raw Status for IntDstTran Interrupt (RAW_DST_TRAN) ... 175
8.2.2.2.5. Raw Status for IntErr Interrupt (RAW_ERR) ... 176
8.2.2.2.6. Status for IntTfr Interrupt (STAT_TFR) ... 176
8.2.2.2.7. Status for IntBlock Interrupt (STAT_BLOCK) ... 177
8.2.2.2.8. Status for IntSrcTran Interrupt (STAT_SRC_TRAN) ... 177
8.2.2.2.9. Status for IntDstTran Interrupt (STAT_DST_TRAN) ... 177
8.2.2.2.10. Status for IntErr Interrupt (STAT_ERR) ... 178
8.2.2.2.11. Mask for IntTfr Interrupt (MASK_TFR) ... 178
8.2.2.2.12. Mask for IntBlock Interrupt (MASK_BLOCK) ... 179
8.2.2.2.13. Mask for IntSrcTran Interrupt (MASK_SRC_TRAN) ... 179
8.2.2.2.14. Mask for IntDstTran Interrupt (MASK_DST_TRAN) ... 180
8.2.2.2.15. Mask for IntErr Interrupt (MASK_ERR) ... 180
8.2.2.2.16. Clear for IntTfr Interrupt (CLR_TFR) ... 181
8.2.2.2.17. Clear for IntBlock Interrupt (CLR_BLOCK) ... 181
8.2.2.2.18. Clear for IntSrcTran Interrupt (CLR_SRC_TRAN) ... 181
8.2.2.2.19. Clear for IntDstTran Interrupt (CLR_DST_TRAN) ... 182
8.2.2.2.20. Clear for IntErr Interrupt (CLR_ERR) ... 182
8.2.2.2.21. Status for Each Interrupt Type (STAT) ... 183
8.2.2.3. Software Handshaking Registers ... 184
8.2.2.3.1. Source Software Transaction Request Register (REQ_SRC) ... 184
8.2.2.3.2. Destination Software Transaction Request Register (REQ_DST) ... 184
8.2.2.3.3. Source Single Transaction Request Register (SGL_REQ_SRC) ... 185
8.2.2.3.4. Destination Single Transaction Request Register (SGL_REQ_DST) ... 185
8.2.2.3.5. Source Last Transaction Request Register (LST_SRC) ... 186
8.2.2.3.6. Destination Last Transaction Request Register (LST_DST) ... 187
8.2.2.4. Miscellaneous DMA Controller Registers ... 187
8.2.2.4.1. DMA Configuration Register (CFG) ... 187
8.2.2.4.2. DMA Channel Enable Register (CH_EN) ... 188
8.2.2.4.3. DMA Test Register (TEST) ... 188
8.3. Programming the DMA ... 189
8.3.1. Register Access ... 189
8.3.2. DMA Controller Transfer Types ... 189
8.3.2.1. Multi-Block Transfers ... 190
8.3.2.1.1. Block Chaining Using Linked Lists ... 190
8.3.2.1.2. Auto-Reloading of Channel Registers ... 191
8.3.2.1.3. Contiguous Address Between Blocks ... 191

8.3.2.1.4. Suspension of Transfers Between Blocks 192
8.3.2.2. Ending Multi-Block Transfers 192
8.3.3. Programming the Linked List Multi-Block Transfer 193
8.3.4. Programming a Channel 194
8.3.4.1. Programming Examples 195
8.3.4.1.1. Single-Block Transfer 195
8.3.4.1.2. Multi-Block Transfer with Linked List for Source and Linked List for Destination 196
8.3.5. Disabling a Channel Prior to Transfer Completion 198
8.3.5.1. Abnormal Transfer Termination 198
8.3.6. Defined-length Burst Support on DMA Controller 199
9. eFLASH Controller 200
9.1. eFLASH Controller Registers 201
9.1.1. Register Map 201
9.1.2. Register Descriptions 202
9.1.2.1. eFLASH Controller Control Register (EFLASH_CR) 202
9.1.2.2. eFLASH Controller Address Register (EFLASH_ADDR) 203
9.1.2.3. eFLASH Controller Write Data Register (EFLASH_WDATA) 203
9.1.2.4. eFLASH Controller Read Data Register (EFLASH_RDATA) 203
9.1.2.5. eFLASH Controller Status Register (EFLASH_SR) 204
9.1.2.6. eFLASH Controller Time Intervals Register 0 (EFLASH_TIME0) 204
9.1.2.7. eFLASH Controller Time Intervals Register 1 (EFLASH_TIME1) 204
9.1.2.8. eFLASH Controller Time Intervals Register 2 (EFLASH_TIME2) 205
9.1.2.9. eFLASH Controller Time Intervals Register 3 (EFLASH_TIME3) 206
9.1.2.10. eFLASH Controller Time Intervals Register 4 (EFLASH_TIME4) 206
9.1.2.11. eFLASH Controller Time Intervals Register 5 (EFLASH_TIME5) 207
9.1.2.12. eFLASH Controller Time Intervals Register 6 (EFLASH_TIME6) 208
9.2. Programming the eFLASH Controller 208
9.2.1. Time Intervals Configuration 208
9.2.2. Memory Configuration Operation (CFG) 210
9.2.3. Memory Write Operation (WRITE) 210
9.2.4. Memory Read Operation (READ) 210
9.2.5. Chip Erase Operation (CHIP_ERASE) 210
9.2.6. Block Erase Operation (BLOCK_ERASE) 211
9.2.7. Sector Erase Operation (SECTOR_ERASE) 211
9.2.8. Entering the DPD Mode Operation (ENTER_DPD) 211
9.2.9. Exiting the DPD Mode Operation (EXIT_DPD) 211
9.2.10. Direst Access to the eFLASH Memory 211
10. Mailbox 213
10.1. Mailbox Registers 213
10.1.1. Register Map 214
10.1.2. Register Descriptions 215
10.1.2.1. Messages Exchange Register 0 (MB**n**DATA0) 215
10.1.2.2. Message Ready Flag Register 0 (MB**n**FULL0) 215
10.1.2.3. Message Not Ready Flag Register 0 (MB**n**EMPTY0) 216

10.1.2.4. Messages Exchange Register 1 (MBnDATA1)..... 216
10.1.2.5. Message Ready Flag Register 1 (MBnFULL1)..... 216
10.1.2.6. Message Not Ready Flag Register 1 (MBnEMPTY1)..... 217
11. ADC..... 218
11.1. ADC Functional Description..... 218
11.1.1. ADC Pins..... 219
11.1.2. Operation Modes..... 219
11.1.2.1. Single Mode..... 220
11.1.2.2. Scan Mode..... 220
11.1.2.3. Discontinuous Mode..... 221
11.1.2.4. Temperature Sensor Mode..... 221
11.1.2.4.1. Measurement Without Trimming or Calibration..... 221
11.1.2.4.2. Measurement With Trimming and Calibration..... 222
11.1.3. Conversion on PWMA Event..... 222
11.1.4. Direct Clocking..... 223
11.2. ADC Registers..... 223
11.2.1. Register Map..... 224
11.2.2. Register Descriptions..... 224
11.2.2.1. ADC Control Register 1 (ADC_CR1)..... 224
11.2.2.2. ADC Set Register (ASER)..... 227
11.2.2.3. ADC Status Register (ASTR)..... 228
11.2.2.4. ADC Sample Time Register (ASMPR)..... 228
11.2.2.5. ADC Sequence Register (ASQR)..... 230
11.2.2.6. ADC Data Register (ADR)..... 231
12. Timers..... 232
12.1. PWMA..... 232
12.1.1. PWMA Functional Description..... 233
12.1.1.1. Inter-Block Trigger Connections..... 233
12.1.1.2. Control Unit..... 234
12.1.1.2.1. External Trigger Control..... 234
12.1.1.2.2. Trigger Selection..... 235
12.1.1.2.3. Slave Mode Controller..... 236
12.1.1.2.4. Master Mode Controller..... 237
12.1.1.3. Time-Base Unit..... 238
12.1.1.3.1. Counter..... 239
12.1.1.3.2. Prescaler..... 239
12.1.1.3.3. Repetition Counter..... 239
12.1.1.4. Counter Modes..... 240
12.1.1.4.1. Upcounting Mode..... 240
12.1.1.4.2. Downcounting Mode..... 240
12.1.1.4.3. Center-aligned Mode (Up/Down Counting)..... 241
12.1.1.5. Capture/Compare Channels..... 242
12.1.1.5.1. Input Stage..... 242
12.1.1.5.2. Output Stage..... 244

12.1.1.5.3. Capture/Compare Stage 246
12.1.1.6. XOR Function 247
12.1.1.7. Dead-time Insertion 247
12.1.1.8. Break Function 248
12.1.1.9. Shadow Registers 249
12.1.1.10. Quadrature Encoder 249
12.1.1.10.1. Operation Modes 249
12.1.1.10.1.1. Quadrature Mode 250
12.1.1.10.1.2. qa Rising/Falling Edge Mode 250
12.1.1.10.2. Initialization and Reset 250
12.1.2. PWMA Registers 250
12.1.2.1. Register Map 251
12.1.2.2. Register Descriptions 252
12.1.2.2.1. PWMA Control Register 1 (PWMA_CR1) 252
12.1.2.2.2. PWMA Control Register 2 (PWMA_CR2) 254
12.1.2.2.3. PWMA Slave Mode Control Register (PWMA_SMCR) 257
12.1.2.2.4. PWMA DMA/Interrupt Enable Register (PWMA_DIER) 260
12.1.2.2.5. PWMA Status Register (PWMA_SR) 262
12.1.2.2.6. PWMA Event Generation Register (PWMA_EGR) 266
12.1.2.2.7. PWMA Capture/Compare Mode Register 1 (PWMA_CCMR1) 269
12.1.2.2.8. PWMA Capture/Compare Mode Register 2 (PWMA_CCMR2) 274
12.1.2.2.9. PWMA Capture/Compare Enable Register (PWMA_CCER) 279
12.1.2.2.10. PWMA Counter (PWMA_CNT) 281
12.1.2.2.11. PWMA Prescaler (PWMA_PSC) 281
12.1.2.2.12. PWMA Auto-reload Register (PWMA_ARR) 281
12.1.2.2.13. PWMA Repetition Counter Register (PWMA_RCR) 282
12.1.2.2.14. PWMA Capture/Compare Register 0 (PWMA_CCR0) 282
12.1.2.2.15. PWMA Capture/Compare Register 1 (PWMA_CCR1) 283
12.1.2.2.16. PWMA Capture/Compare Register 2 (PWMA_CCR2) 284
12.1.2.2.17. PWMA Capture/Compare Register 3 (PWMA_CCR3) 284
12.1.2.2.18. PWMA Break and Dead-Time Register (PWMA_BDTR) 285
12.1.2.2.19. PWMA DMA Control Register (PWMA_DCR) 287
12.1.2.2.20. PWMA DMA Address for Full Transfer (PWMA_DMAR) 287
12.1.2.2.21. PWMA Quadrature Encoder Setting Register (PWMA_QESET) 288
12.1.2.2.22. PWMA Quadrature Encoder Filters Setting (PWMA_FILTSET) 290
12.1.2.2.23. PWMA Quadrature Encoder Max Count Value (PWMA_QEMAX) 291
12.1.2.2.24. PWMA Quadrature Encoder Init Value (PWMA_INITSET) 292
12.1.2.2.25. PWMA Quadrature Encoder Counter Value (PWMA_QECNT) 292
12.2. PWMG 292
12.2.1. PWMG Functional Description 293
12.2.1.1. Time-Base Unit 293
12.2.1.1.1. Counter 294
12.2.1.1.2. Prescaler 294
12.2.1.2. Counter Upcounting Mode 294

12.2.1.3. Capture/Compare Channel 295
12.2.1.3.1. Input Stage 295
12.2.1.3.2. Output Stage 296
12.2.1.3.3. Capture/Compare Stage 296
12.2.1.4. Shadow Registers 297
12.2.2. PWMG Registers 298
12.2.2.1. Register Map 298
12.2.2.2. Register Descriptions 298
12.2.2.2.1. PWMG Control Register 1 (PWMG_CR1) 298
12.2.2.2.2. PWMG Interrupt Enable Register (PWMG_IER) 300
12.2.2.2.3. PWMG Status Register (PWMG_SR) 300
12.2.2.2.4. PWMG Event Generation Register (PWMG_EGR) 301
12.2.2.2.5. PWMG Capture/Compare Mode Register 1 (PWMG_CCMR1) 302
12.2.2.2.6. PWMG Capture/Compare Enable Register (PWMG_CCER) 305
12.2.2.2.7. PWMG Counter (PWMG_CNT) 306
12.2.2.2.8. PWMG Prescaler (PWMG_PSC) 306
12.2.2.2.9. PWMG Auto-reload Register (PWMG_ARR) 306
12.2.2.2.10. PWMG Capture/Compare Register 0 (PWMG_CCR0) 306
12.3. TIM 307
12.3.1. TIM Functional Description 308
12.3.1.1. Timer Operation 308
12.3.1.1.1. Using a Timer 308
12.3.1.1.2. Enabling a Timer 308
12.3.1.1.3. Disabling a Timer 309
12.3.1.1.4. Loading a Timer Countdown Value 309
12.3.1.1.5. Working with Interrupts 309
12.3.1.1.6. Generating Toggled Outputs 310
12.3.1.1.6.1. Pulse Width Modulation of Toggle Outputs 310
12.3.1.1.6.2. Pulse Width Modulation with 0% and 100% Duty Cycles 310
12.3.2. TIM Registers 312
12.3.2.1. Register Map 312
12.3.2.2. Register Descriptions 314
12.3.2.2.1. Timer N Load Count Register (TIMER<N>LOAD_COUNT) 314
12.3.2.2.2. Timer N Load Count 2 Register (TIMER<N>LOAD_COUNT2) 314
12.3.2.2.3. Timer N Current Value Register (TIMER<N>CURRENT_VALUE) 314
12.3.2.2.4. Timer N Control Register (TIMER<N>CONTROL_REG) 315
12.3.2.2.5. Timer N End-of-Interrupt Register (TIMER<N>EOI) 316
12.3.2.2.6. Timer N Interrupt Status Register (TIMER<N>INT_STATUS) 316
12.3.2.2.7. Timers Interrupt Status Register (TIMERS_INT_STATUS) 316
12.3.2.2.8. Timers End-of-Interrupt Register (TIMERS_EOI) 317
12.3.2.2.9. Timers Raw Interrupt Status Register (TIMERS_RAW_INT_STATUS) 317
12.4. WDT 317
12.4.1. WDT Functional Description 318
12.4.1.1. Interrupts 319

12.4.1.2. Counter 319
12.4.1.3. System Resets 319
12.4.1.4. Reset Pulse Length 319
12.4.2. WDT Registers 319
12.4.2.1. Register Map 320
12.4.2.2. Register Descriptions 320
12.4.2.2.1. Control Register (WDT_CR) 321
12.4.2.2.2. Timeout Range Register (WDT_TORR) 321
12.4.2.2.3. Current Counter Value Register (WDT_CCVR) 322
12.4.2.2.4. Counter Restart Register (WDT_CRR) 322
12.4.2.2.5. Interrupt Status Register (WDT_STAT) 323
12.4.2.2.6. Interrupt Clear Register (WDT_EOI) 323
12.4.2.2.7. Component Parameters Register 5 (WDT_COMP_PARAM_5) 323
12.4.2.2.8. Component Parameters Register 4 (WDT_COMP_PARAM_4) 324
12.4.2.2.9. Component Parameters Register 3 (WDT_COMP_PARAM_3) 324
12.4.2.2.10. Component Parameters Register 2 (WDT_COMP_PARAM_2) 324
12.4.2.2.11. Component Parameters Register 1 (WDT_COMP_PARAM_1) 325
13. Low Speed Peripherals 327
13.1. GPIO 327
13.1.1. GPIO Functional Description 327
13.1.1.1. Data and Control Flow 327
13.1.1.1.1. Software Control 328
13.1.1.1.2. Reading External Signals 328
13.1.1.2. Interrupts 328
13.1.1.2.1. Debounce Operation 328
13.1.1.2.2. Level-sensitive Interrupts 329
13.1.2. GPIO Registers 329
13.1.2.1. Programming Considerations 329
13.1.2.2. Register Map 329
13.1.2.3. Register Descriptions 330
13.1.2.3.1. Data Output Register (GPIO_DR) 330
13.1.2.3.2. Data Direction Control Register (GPIO_DDR) 330
13.1.2.3.3. Interrupt Enable Register (GPIO_INTEN) 331
13.1.2.3.4. Interrupt Mask Register (GPIO_INTMSK) 331
13.1.2.3.5. Interrupt Level Register (GPIO_INTLVL) 331
13.1.2.3.6. Interrupt Polarity Register (GPIO_INTPOLARITY) 332
13.1.2.3.7. Interrupt Status Register (GPIO_INTSTAT) 332
13.1.2.3.8. Raw Interrupt Status Register (GPIO_INTSTATRAW) 332
13.1.2.3.9. Debounce Enable Register (GPIO_DEBOUNCE) 333
13.1.2.3.10. Clear Interrupt Register (GPIO_EOI) 333
13.1.2.3.11. External Port Register (GPIO_EXT) 334
13.1.2.3.12. Synchronization Level Register (GPIO_SYNC) 334
13.1.2.3.13. Interrupt Both Edge Type Register (GPIO_BOTHEDGE) 334
13.2. CAN FD 335

13.2.1. CAN FD Functional Description ... 336
13.2.1.1. Operating Modes and States ... 336
13.2.1.2. Interrupts ... 339
13.2.2. CAN FD Registers ... 339
13.2.2.1. Register Map ... 340
13.2.2.2. Register Descriptions ... 341
13.2.2.2.1. CAN FD Core Register Space ... 341
13.2.2.2.1.1. Software Reset Register (SRR) ... 341
13.2.2.2.1.2. Mode Select Register (MSR) ... 342
13.2.2.2.1.3. Arbitration Phase (Nominal) Baud Rate Prescaler Register (BRPR) ... 345
13.2.2.2.1.4. Arbitration Phase (Nominal) Bit Timing Register (BTR) ... 345
13.2.2.2.1.5. Error Count Register (ECR) ... 346
13.2.2.2.1.6. Error Status Register (ESR) ... 346
13.2.2.2.1.7. Status Register (SR) ... 348
13.2.2.2.1.8. Interrupt Status Register (ISR) ... 350
13.2.2.2.1.9. Interrupt Enable Register (IER) ... 354
13.2.2.2.1.10. Interrupt Clear Register (ICR) ... 357
13.2.2.2.1.11. Timestamp Register (TSR) ... 360
13.2.2.2.1.12. Data Phase Baud Rate Prescaler Register (DP_BRPR) ... 360
13.2.2.2.1.13. Data Phase Bit Timing Register (DP_BTR) ... 361
13.2.2.2.1.14. Tx Buffer Ready Request Register (TRR) ... 362
13.2.2.2.1.15. Interrupt Enable Tx Buffer Ready Request Served/Cleared (IETRS) ... 363
13.2.2.2.1.16. Tx Buffer Cancel Request (TCR) ... 363
13.2.2.2.1.17. Interrupt Enable Tx Buffer Cancellation Served/Cleared (IETCS) ... 364
13.2.2.2.1.18. Tx Event FIFO Status Register (TxE_FSR) ... 364
13.2.2.2.1.19. Tx Event FIFO Watermark Register (TxE_WMR) ... 365
13.2.2.2.1.20. Acceptance Filter (Control) Register (AFR) ... 366
13.2.2.2.1.21. Rx FIFO Status Register (FSR) ... 366
13.2.2.2.1.22. Rx FIFO Watermark Register (WMR) ... 367
13.2.2.2.2. CAN FD Tx Message Space ... 368
13.2.2.2.2.1. TBi Message ID Register (TBiID) ... 368
13.2.2.2.2.2. TBi Data Length Code Register (TBiDLC) ... 369
13.2.2.2.2.3. TBi Data Byte n Register (TBiDWn) ... 370
13.2.2.2.2.4. Acceptance Filter i Mask Register (AFMRi) ... 370
13.2.2.2.2.5. Acceptance Filter i ID registers Register (AFIRi) ... 372
13.2.2.2.3. CAN FD Tx Event FIFO Status Register Space ... 372
13.2.2.2.3.1. TXE FIFO TBi Message ID Register (TXE_FIFO_TBi_ID) ... 372
13.2.2.2.3.2. TXE FIFO TBi Data Length Code Register (TXE_FIFO_TBi_DLC) ... 374
13.2.2.2.4. CAN FD Rx Message Space ... 374
13.2.2.2.4.1. RBi Message ID Register (RBiID) ... 374
13.2.2.2.4.2. RBi Data Length Code Register (RBiDLC) ... 376
13.2.2.2.4.3. RBi Data Byte n Register (RBiDWn) ... 376
13.3. UART ... 377
13.3.1. UART Functional Description ... 378

13.3.1.1. Programmable THRE Interrupt 378
13.3.1.2. UART Serial Protocol 378
13.3.1.3. 9-bit Data Transfer 379
13.3.1.3.1. Transmit Mode 379
13.3.1.3.1.1. Transmit Mode 0 379
13.3.1.3.1.2. Transmit Mode 1 380
13.3.1.3.2. Receive Mode 380
13.3.1.3.2.1. Hardware Address Match Receive Mode 380
13.3.1.3.2.2. Software Address Match Receive Mode 381
13.3.1.4. RS485 Support (for UART_6 and UART_7 Only) 382
13.3.1.4.1. de Assertion and De-assertion Timing 382
13.3.1.4.2. RS485 Modes 382
13.3.1.5. Fractional Baud Rate Support 384
13.3.1.6. IrDA 1.0 SIR Support 385
13.3.2. UART Registers 386
13.3.2.1. Register Map 386
13.3.2.2. Register Descriptions 388
13.3.2.2.1. Receive Buffer Register (RBR) 388
13.3.2.2.2. Transmit Holding Register (THR) 388
13.3.2.2.3. Divisor Latch Low Register (DLL) 389
13.3.2.2.4. Divisor Latch High Register (DLH) 389
13.3.2.2.5. Interrupt Enable Register (IER) 390
13.3.2.2.6. Interrupt Identity Register (IIR) 391
13.3.2.2.7. FIFO Control Register (FCR) 392
13.3.2.2.8. Line Control Register (LCR) 393
13.3.2.2.9. Mode Control Register (MCR) 395
13.3.2.2.10. Line Status Register (LSR) 395
13.3.2.2.11. Scratchpad Register (SCR) 398
13.3.2.2.12. Shadow Receive Buffer Register (SRBRi) 398
13.3.2.2.13. Shadow Transmit Holding Register (STHRi) 398
13.3.2.2.14. FIFO Access Register (FAR) 399
13.3.2.2.15. Transmit FIFO Read Register (TFR) 399
13.3.2.2.16. Receive FIFO Write Register (RFW) 400
13.3.2.2.17. UART Status Register (USR) 400
13.3.2.2.18. Transmit FIFO Level Register (TFL) 401
13.3.2.2.19. Receive FIFO Level Register (RFL) 402
13.3.2.2.20. Software Reset Register (SRR) 402
13.3.2.2.21. Shadow Break Control Register (SBCR) 403
13.3.2.2.22. FIFO Enable Register (SFE) 403
13.3.2.2.23. Shadow RCVR Trigger Register (SRT) 403
13.3.2.2.24. Shadow Tx Empty Trigger Register (STET) 404
13.3.2.2.25. Halt Tx Register (HTx) 404
13.3.2.2.26. DMA Software Acknowledge Register (DMASA) 405
13.3.2.2.27. Transceiver Control Register (TCR) 405

13.3.2.2.28. Driver Output Enable Register (DE_EN)....407
13.3.2.2.29. Receiver Output Enable Register (RE_EN)....407
13.3.2.2.30. Driver Output Enable Timing Register (DET)....407
13.3.2.2.31. TurnAround Timing Register (TAT)....408
13.3.2.2.32. Divisor Latch Fraction Register (DLF)....408
13.3.2.2.33. Receive Address Register (RAR)....409
13.3.2.2.34. Transmit Address Register (TAR)....409
13.3.2.2.35. Line Extended Control Register (LCR_EXT)....410
13.3.2.2.36. Register Timeout Counter Reset Value Register (REG_TIMEOUT_RST)....411
13.4. SPI & QSPI....411
13.4.1. *SPI Functional Description....412
13.4.1.1. *SPI Clock....413
13.4.1.2. Endian Conversion Support....413
13.4.1.3. Transmit and Receive FIFO Buffers....414
13.4.1.4. Transfer Modes....415
13.4.1.5. Dual Data-Rate Support (QSPI only)....416
13.4.1.6. Serial Master Operation Mode (QSPI/SPI Master only)....417
13.4.1.7. Serial-Slave Operation Mode (SPI Slave only)....419
13.4.1.8. Interrupts....420
13.4.2. *SPI Registers....422
13.4.2.1. Register Map....422
13.4.2.2. Register Descriptions....423
13.4.2.2.1. Control Register 0 (CTRLR0)....423
13.4.2.2.2. Control Register 1 (CTRLR1)....427
13.4.2.2.3. SPI Enable Register (SPIENR)....427
13.4.2.2.4. Microwire Control Register (MWCR)....428
13.4.2.2.5. Slave Enable Register (SER)....429
13.4.2.2.6. Baud Rate Select Register (BAUDR)....429
13.4.2.2.7. Transmit FIFO Threshold Level Register (TXFTLR)....430
13.4.2.2.8. Receive FIFO Threshold Level Register (RXFTLR)....430
13.4.2.2.9. Transmit FIFO Level Register (TXFLR)....431
13.4.2.2.10. Receive FIFO Level Register (RXFLR)....431
13.4.2.2.11. Status Register (SR)....431
13.4.2.2.12. Interrupt Mask Register (IMR)....433
13.4.2.2.13. Interrupt Status Register (ISR)....434
13.4.2.2.14. Raw Interrupt Status Register (RISR)....435
13.4.2.2.15. Transmit FIFO Overflow Interrupt Clear Register (TXOICR)....437
13.4.2.2.16. Receive FIFO Overflow Interrupt Clear Register (RXOICR)....437
13.4.2.2.17. Receive FIFO Underflow Interrupt Clear Register (RXUICR)....437
13.4.2.2.18. Multi-Master Interrupt Clear Register (MSTICR)....437
13.4.2.2.19. Interrupt Clear Register (ICR)....438
13.4.2.2.20. DMA Control Register (DMACR)....438
13.4.2.2.21. DMA Transmit Data Level Register (DMATDLR)....439
13.4.2.2.22. DMA Receive Data Level Register (DMARDLR)....439

13.4.2.2.23. Data Register (DRi)....439
13.4.2.2.24. Rx Sample Delay Register (RX_SAMPLE_DLY)....440
13.4.2.2.25. SPI Control Register (SPI_CTRLR0)....440
13.4.2.2.26. Transmit Drive Edge Register (TXD_DRIVE_EDGE)....442
13.4.2.2.27. XIP Command Register (XIP_CMD)....442
13.5. I2C....443
13.5.1. I2C Functional Description....444
13.5.1.1. Addressing Slave Protocol....444
13.5.1.2. Operation Modes....445
13.5.1.2.1. Slave Mode Operation....445
13.5.1.2.2. Master Mode Operation....448
13.5.1.2.3. Disabling I2C Controller....449
13.5.1.2.4. Aborting I2C Transfers....450
13.5.1.3. Fast Mode Plus Operation....450
13.5.1.4. SDA Hold Time....450
13.5.1.4.1. SDA Hold Timings in Receiver....451
13.5.1.4.2. SDA Hold Timings in Transmitter....451
13.5.1.5. Interrupts....451
13.5.2. I2C Registers....453
13.5.2.1. Register Map....453
13.5.2.2. Register Descriptions....455
13.5.2.2.1. I2C Control Register (IC_CON)....455
13.5.2.2.2. I2C Target Address Register (IC_TAR)....458
13.5.2.2.3. I2C Slave Address Register (IC_SAR)....460
13.5.2.2.4. I2C High Speed Master Mode Code Address Register (IC_HS_MADDR)....460
13.5.2.2.5. I2C Rx/Tx Data Buffer and Command Register (IC_DATA_CMD)....461
13.5.2.2.6. Standard Speed I2C Clock SCL High Count Register (IC_SS_SCL_HCNT)....462
13.5.2.2.7. Standard Speed I2C Clock SCL Low Count Register (IC_SS_SCL_LCNT)....462
13.5.2.2.8. Fast Mode or Fast Mode Plus I2C Clock SCL High Count Register (IC_FS_SCL_HCNT)....463
13.5.2.2.9. Fast Mode or Fast Mode Plus I2C Clock SCL Low Count Register (IC_FS_SCL_LCNT)....463
13.5.2.2.10. High Speed I2C Clock SCL High Count Register (IC_HS_SCL_HCNT)....464
13.5.2.2.11. High Speed I2C Clock SCL Low Count Register (IC_HS_SCL_LCNT)....464
13.5.2.2.12. I2C Interrupt Status Register (IC_INTR_STAT)....465
13.5.2.2.13. I2C Interrupt Mask Register (IC_INTR_MASK)....467
13.5.2.2.14. I2C Raw Interrupt Status Register (IC_RAW_INTR_STAT)....469
13.5.2.2.15. I2C Receive FIFO Threshold Register (IC_RX_TL)....473
13.5.2.2.16. I2C Transmit FIFO Threshold Register (IC_TX_TL)....474
13.5.2.2.17. Clear Combined and Individual Interrupt Register (IC_CLR_INTR)....474
13.5.2.2.18. Clear RX_UNDER Interrupt Register (IC_CLR_RX_UNDER)....475
13.5.2.2.19. Clear RX_OVER Interrupt Register (IC_CLR_RX_OVER)....475
13.5.2.2.20. Clear TX_OVER Interrupt Register (IC_CLR_TX_OVER)....475
13.5.2.2.21. Clear RD_REQ Interrupt Register (IC_CLR_RD_REQ)....475

13.5.2.2.22. Clear TX_ABRT Interrupt Register (IC_CLR_TX_ABRT)..... 476
13.5.2.2.23. Clear RX_DONE Interrupt Register (IC_CLR_RX_DONE)..... 476
13.5.2.2.24. Clear ACTIVITY Interrupt Register (IC_CLR_ACTIVITY)..... 476
13.5.2.2.25. Clear STOP_DET Interrupt Register (IC_CLR_STOP_DET)..... 477
13.5.2.2.26. Clear START_DET Interrupt Register (IC_CLR_START_DET)..... 477
13.5.2.2.27. Clear GEN_CALL Interrupt Register (IC_CLR_GEN_CALL)..... 477
13.5.2.2.28. I2C ENABLE Register (IC_ENABLE)..... 478
13.5.2.2.29. I2C STATUS Register (IC_STATUS)..... 479
13.5.2.2.30. I2C Transmit FIFO Level Register (IC_TXFLR)..... 482
13.5.2.2.31. I2C Receive FIFO Level Register (IC_RXFLR)..... 482
13.5.2.2.32. I2C SDA Hold Time Length Register (IC_SDA_HOLD)..... 483
13.5.2.2.33. I2C Transmit Abort Source Register (IC_TX_ABRT_SOURCE)..... 483
13.5.2.2.34. DMA Control Register (IC_DMA_CR) 489
13.5.2.2.35. DMA Transmit Data Level Register (IC_DMA_TDLR)..... 489
13.5.2.2.36. DMA Receive Data Level Register (IC_DMA_RDLR) 490
13.5.2.2.37. I2C SDA Setup Register (IC_SDA_SETUP)..... 490
13.5.2.2.38. I2C ACK General Call Register (IC_ACK_GENERAL_CALL)..... 491
13.5.2.2.39. I2C Enable Status Register (IC_ENABLE_STATUS)..... 491
13.5.2.2.40. I2C SS, FS or FM+ Spike Suppression Limit (IC_FS_SPKLEN)..... 493
13.5.2.2.41. I2C HS Spike Suppression Limit Register (IC_HS_SPKLEN) 494
13.5.2.2.42. I2C SCL Stuck at Low Timeout Register (IC_SCL_STUCK_AT_LOW_TIMEOUT)..... 494
13.5.2.2.43. I2C SDA Stuck at Low Timeout Register (IC_SDA_STUCK_AT_LOW_TIMEOUT)..... 495
13.5.2.2.44. Clear SCL Stuck at Low Detect Interrupt Register (IC_CLR_SCL_STUCK_DET)..... 495
13.5.2.2.45. Register Timeout Counter Reset Value (REG_TIMEOUT_RST)..... 495
13.6. I2S..... 496
13.6.1. I2S Functional Description..... 496
13.6.1.1. I2S Controller Enable..... 497
13.6.1.2. I2S Controller as Transmitter..... 497
13.6.1.2.1. Transmitter Block Enable..... 498
13.6.1.2.2. Transmit Channel Enable..... 498
13.6.1.2.3. Transmit Channel Audio Data Resolution..... 498
13.6.1.2.4. Transmit Channel FIFOs..... 499
13.6.1.2.5. Transmit Channel Interrupts..... 499
13.6.1.2.6. Writing to the Transmit Channel..... 500
13.6.1.3. I2S Controller as Receiver..... 500
13.6.1.3.1. Receiver Block Enable..... 501
13.6.1.3.2. Receive Channel Enable..... 502
13.6.1.3.3. Receive Channel Audio Data Resolution..... 502
13.6.1.3.4. Receive Channel FIFOs..... 502
13.6.1.3.5. Receive Channel Interrupts..... 503
13.6.1.3.6. Reading from the Receive Channel..... 503
13.6.2. I2S Registers..... 503
13.6.2.1. Register Map..... 504
13.6.2.2. Register Descriptions..... 505

13.6.2.2.1. I2S Controller Enable Register (IER)....505
13.6.2.2.2. I2S Receiver Block Enable Register (IRER)....506
13.6.2.2.3. I2S Transmitter Block Enable Register (ITER)....506
13.6.2.2.4. Receiver Block FIFO Reset Register (RXFFR)....506
13.6.2.2.5. Transmitter Block FIFO Reset Register (TXFFR)....507
13.6.2.2.6. Left Receive Buffer Register (LRBR0)....507
13.6.2.2.7. Left Transmit Holding Register (LTHR0)....507
13.6.2.2.8. Right Receive Buffer Register (RRBR0)....508
13.6.2.2.9. Right Transmit Holding Register (RTHR0)....508
13.6.2.2.10. Receive Enable Register (RER0)....509
13.6.2.2.11. Transmit Enable Register (TER0)....509
13.6.2.2.12. Receive Configuration Register (RCR0)....509
13.6.2.2.13. Transmit Configuration Register (TCR0)....510
13.6.2.2.14. Interrupt Status Register (ISR0)....510
13.6.2.2.15. Interrupt Mask Register (IMR0)....511
13.6.2.2.16. Receive Overrun Register (ROR0)....512
13.6.2.2.17. Transmit Overrun Register (TOR0)....512
13.6.2.2.18. Receive FIFO Configuration Register (RFCR0)....513
13.6.2.2.19. Transmit FIFO Configuration Register (TFCR0)....513
13.6.2.2.20. Receive FIFO Flush Register (RFF0)....513
13.6.2.2.21. Transmit FIFO Flush Register (TFF0)....514
13.6.2.2.22. Receiver Block DMA Register (RXDMA)....514
13.6.2.2.23. Reset Receiver Block DMA Register (RRXDMA)....515
13.6.2.2.24. Transmitter Block DMA Register (TXDMA)....516
13.6.2.2.25. Reset Transmitter Block DMA Register (RTXDMA)....516
13.6.2.2.26. DMA Control Register (DMACR)....516
14. USB 2.0....518
14.1. USB 2.0 Controller....518
14.1.1. USB 2.0 Controller Functional Description....519
14.1.1.1. Resets....519
14.1.1.1.1. In Peripheral Mode....520
14.1.1.1.2. In Host Mode....520
14.1.1.2. Modes of Operation....520
14.1.1.3. Suspend/Resume....521
14.1.1.3.1. Control by Inactivity on USB Bus (L0 to L2 State)....522
14.1.1.3.1.1. In Peripheral Mode....522
14.1.1.3.1.2. In Host Mode....522
14.1.1.3.2. Control by LPM Transaction (L0 to L1 State)....523
14.1.1.3.2.1. In Peripheral Mode....523
14.1.1.3.2.2. In Host Mode....525
14.1.1.4. Multiple Devices Support....525
14.1.1.4.1. Allocating Devices to Endpoints....526
14.1.1.4.2. Operation....527
14.1.1.4.3. Bandwidth Issues....528

14.1.1.5. Connect/Disconnect 528
14.1.1.5.1. In Host Mode 528
14.1.1.5.2. In Peripheral Mode 528
14.1.1.6. Programming Scheme 528
14.1.1.6.1. Soft Connect/Disconnect 529
14.1.1.6.2. USB Interrupt Handle 529
14.1.1.7. OTG Session Request 530
14.1.1.7.1. Starting Session 531
14.1.1.7.2. Detecting Activity 531
14.1.1.8. Host Negotiation 532
14.1.1.9. Fundamental DMA Support 532
14.1.1.10. Included DMA Controller 534
14.1.1.10.1. DMA Bus Cycles 534
14.1.1.10.2. Bus Errors 535
14.1.1.10.3. Transferring Packets 535
14.1.1.10.3.1. Individual Packet: Rx Endpoint 535
14.1.1.10.3.2. Individual Packet: Tx Endpoint 536
14.1.1.10.3.3. Multiple Packets: Rx Endpoint 536
14.1.1.10.3.4. Multiple Packets: Tx Endpoint 537
14.1.1.11. VBus Events 538
14.1.1.11.1. Actions as ‘A’ Device 538
14.1.1.11.2. Actions as ‘B’ Device 539
14.1.1.12. Dynamic FIFO Sizing 539
14.1.1.13. Control Transactions (via Endpoint 0) 540
14.1.1.13.1. In Peripheral Mode 540
14.1.1.13.1.1. Zero Data Requests 540
14.1.1.13.1.2. Write Requests 541
14.1.1.13.1.3. Read Requests 542
14.1.1.13.1.4. Endpoint 0 States 542
14.1.1.13.1.5. Endpoint 0 Service Routine 543
14.1.1.13.1.5.1. Idle 545
14.1.1.13.1.5.2. Tx 546
14.1.1.13.1.5.3. Rx 547
14.1.1.13.1.6. Error Handling 548
14.1.1.13.1.7. Additional Actions 549
14.1.1.13.1.7.1. STALL Issued to Control Transfer 549
14.1.1.13.1.7.2. Zero-Length OUT Data Packets in Control Transfers 549
14.1.1.13.2. In Host Mode 549
14.1.1.13.2.1. Setup Phase 550
14.1.1.13.2.2. IN Data Phase 550
14.1.1.13.2.3. OUT Data Phase 551
14.1.1.13.2.4. IN Status Phase (Following Setup Phase or OUT Data Phase) 552
14.1.1.13.2.5. OUT Status Phase (Following IN Data Phase) 552
14.1.1.14. Bulk Transactions 553

14.1.1.14.1. In Peripheral Mode....553
14.1.1.14.1.1. Bulk IN Transactions....553
14.1.1.14.1.1.1. Setup....554
14.1.1.14.1.1.2. Operation....554
14.1.1.14.1.2. Bulk OUT Transactions....555
14.1.1.14.1.2.1. Setup....556
14.1.1.14.1.2.2. Operation....557
14.1.1.14.1.2.3. Error Handling....557
14.1.1.14.2. In Host Mode....557
14.1.1.14.2.1. Bulk IN Transactions....558
14.1.1.14.2.1.1. Setup....558
14.1.1.14.2.1.2. Operation....559
14.1.1.14.2.1.3. Error Handling....560
14.1.1.14.2.2. Bulk OUT Transactions....560
14.1.1.14.2.2.1. Setup....561
14.1.1.14.2.2.2. Operation....562
14.1.1.14.2.2.3. Error Handling....563
14.1.1.14.3. Employing DMA....563
14.1.1.14.3.1. Using DMA with Bulk Tx Endpoints....563
14.1.1.14.3.2. Using DMA with Bulk Rx Endpoints....564
14.1.1.15. Full-Speed/Low-Bandwidth Interrupt Transactions....564
14.1.1.15.1. In Peripheral Mode....564
14.1.1.15.2. In Host Mode....565
14.1.1.16. Full-Speed/Low Bandwidth Isochronous Transactions....565
14.1.1.16.1. In Peripheral Mode....565
14.1.1.16.1.1. IN Transactions....565
14.1.1.16.1.1.1. Setup....566
14.1.1.16.1.1.2. Operation....567
14.1.1.16.1.1.3. Error Handling....567
14.1.1.16.1.2. OUT Transactions....567
14.1.1.16.1.2.1. Setup....568
14.1.1.16.1.2.2. Operation....569
14.1.1.16.1.2.3. Error Handling....569
14.1.1.16.2. In Host Mode....569
14.1.1.16.2.1. IN Transactions....569
14.1.1.16.2.1.1. Setup....570
14.1.1.16.2.1.2. Operation....571
14.1.1.16.2.1.3. Error Handling....571
14.1.1.16.2.2. OUT Transactions....571
14.1.1.16.2.2.1. Setup....572
14.1.1.16.2.2.2. Operation....572
14.1.1.17. High Bandwidth Isochronous/Interrupt Transactions....573
14.1.2. USB 2.0 Controller Registers....574
14.1.2.1. Register Map....574

14.1.2.2. Register Descriptions....578
14.1.2.2.1. Function Address Register (FADDR)....578
14.1.2.2.2. Power Management Register (POWER)....579
14.1.2.2.3. Interrupt Register for Endpoint 0 and Tx Endpoints (INTRTX)....581
14.1.2.2.4. Interrupt Register for Rx Endpoints (INTRRX)....581
14.1.2.2.5. Interrupt Enable Register for INTRTX (INTRTXE)....582
14.1.2.2.6. Interrupt Enable Register for INTRRX (INTRRXE)....583
14.1.2.2.7. USB Common Interrupts Register (INTRUSB)....583
14.1.2.2.8. USB Common Interrupts Enable Register (INTRUSBE)....584
14.1.2.2.9. Frame Number Register (FRAME)....585
14.1.2.2.10. Index Register (INDEX)....585
14.1.2.2.11. USB Test Modes Register (TESTMODE)....586
14.1.2.2.12. Maximum Packet Size for Tx Endpoint #n (TXMAXP)....588
14.1.2.2.13. Endpoint 0 Control and Status Register 0 (CSR0L)....589
14.1.2.2.14. Tx Endpoint #**n** Control and Status Register 0 (TXCSRL)....591
14.1.2.2.15. Endpoint 0 Control and Status Register 1 (CSR0H)....594
14.1.2.2.16. Tx Endpoint #n Control and Status Register 1 (TXCSRH)....596
14.1.2.2.17. Maximum Packet Size for Rx Endpoint #n (RXMAXP)....599
14.1.2.2.18. Rx Endpoint #n Control and Status Register 0 (RXCSRL)....600
14.1.2.2.19. Rx Endpoint #n Control and Status Register 1 (RXCSRH)....603
14.1.2.2.20. Endpoint 0 Data Bytes Counter Register (COUNT0)....606
14.1.2.2.21. Rx Endpoint #n Data Bytes Counter Register (RXCOUNT)....606
14.1.2.2.22. Tx Endpoint #n Settings Register (TXTYPE)....607
14.1.2.2.23. Endpoint 0 Speed Register (TYPE0)....608
14.1.2.2.24. Endpoint 0 NAK Limit Register (NAKLIMIT0)....608
14.1.2.2.25. Tx Endpoint #n NAK Limit/Polling Interval Register (TXINTERVAL)....609
14.1.2.2.26. Rx Endpoint #n Settings Register (RXTYPE)....610
14.1.2.2.27. Rx Endpoint #n NAK Limit/Polling Interval Register (RXINTERVAL)....611
14.1.2.2.28. Configuration Data Register (CONFIGDATA)....612
14.1.2.2.29. Endpoint #**n** FIFO (FIFO**n**)....612
14.1.2.2.30. Device Control Register (DEVCTL)....613
14.1.2.2.31. Tx Endpoint #n FIFO Size (TXFIFOSZ)....614
14.1.2.2.32. Rx Endpoint #n FIFO Size (RXFIFOSZ)....615
14.1.2.2.33. Tx Endpoint #n FIFO Address (TXFIFOAD)....616
14.1.2.2.34. Rx Endpoint #n FIFO Address (RXFIFOAD)....616
14.1.2.2.35. PHY signals Control Register (VCONTROL)....617
14.1.2.2.36. PHY Signals Status Register (VSTATUS)....618
14.1.2.2.37. Information About Numbers of Tx and Rx Endpoints (EPINFO)....619
14.1.2.2.38. Information About RAM (RAMINFO)....619
14.1.2.2.39. Information About Delays (LINKINFO)....620
14.1.2.2.40. Duration of the VBus Pulsing Charge (VPLEN)....620
14.1.2.2.41. Time Buffer Available on High-Speed Transactions (HS_EOF1)....620
14.1.2.2.42. Time Buffer Available on Full-Speed Transactions (FS_EOF1)....621
14.1.2.2.43. Time Buffer Available on Low-Speed Transactions (LS_EOF1)....621

14.1.2.2.44. Tx Endpoint #n Function Address (TXFUNCADDR)....621
14.1.2.2.45. Tx Endpoint #n Hub Address (TXHUBADDR)....622
14.1.2.2.46. Tx Endpoint #n Hub Port (TXHUBPORT)....622
14.1.2.2.47. Rx Endpoint #n Function Address (RXFUNCADDR)....623
14.1.2.2.48. Rx Endpoint #n Hub Address (RXHUBADDR)....623
14.1.2.2.49. Rx Endpoint #n Hub Port (RXHUBPORT)....624
14.1.2.2.50. DMA Channel #n Interrupt Register (DMA_INTR)....624
14.1.2.2.51. DMA Channel #n Transfer Control Register (DMA_CNTL)....625
14.1.2.2.52. DMA Channel #n Address Register (DMA_ADDR)....626
14.1.2.2.53. DMA Channel #n Transfer Count Register (DMA_COUNT)....627
14.1.2.2.54. Number of Requested Packets for Rx Endpoint #n (EPn_RQPKTCOUNT)....627
14.1.2.2.55. Chirp Timeout Timer Register(C_T_UCH)....627
14.1.2.2.56. High Speed Resume Signaling Delay Register (C_T_HSRTN)....628
14.1.2.2.57. High Speed Timeout Increase Register (C_T_HSBT)....628
14.1.2.2.58. LPM Attribute Register (LPM_ATTR)....629
14.1.2.2.59. LPM Control Register (LPM_CNTRL)....630
14.1.2.2.60. LPM Interrupt Enable Register (LPM_INTREN)....632
14.1.2.2.61. LPM Interrupt Register (LPM_INTR)....633
14.1.2.2.62. LPM Function Address Register (LPM_FADDR)....635
14.2. USB 2.0 PHY....635
14.2.1. USB 2.0 PHY Registers....636
14.2.1.1. Register Map....637
14.2.1.2. Register Descriptions....637
14.2.1.2.1. HS Resistor Adjust and Tx Clock Gate Control Register (HS_RES_ADJ_TX_CLK_GATE_CTL)....637
14.2.1.2.2. Super Power Saving and Squelch Control Register (SPWR_SVNG_SQLCH_CTL)....638
14.2.1.2.3. Output Eye Shape and Internal Bias Current Control Register (OUT_EYE_SHAPE_INT_BIAS_CTL)....639
14.2.1.2.4. Tx Clock Phase and PLL Adjust Control Register (TX_CLK_PHASE_PLL_ADJ_CTL)....639
14.2.1.2.5. Parameters Control Register 0 (PARAM_CTL_0)....639
14.2.1.2.6. Parameters Control Register 1 (PARAM_CTL_1)....640
14.2.1.2.7. Parameters Control Register 2 (PARAM_CTL_2)....641
14.2.1.2.8. Reserved Out1 Register 0 (RSVD_OUT1_0)....641
14.2.1.2.9. Reserved Out1 Register 1 (RSVD_OUT1_1)....642
14.2.1.2.10. Reserved Out2 Register 0 (RSVD_OUT2_0)....642
14.2.1.2.11. Reserved Out2 Register 1 (RSVD_OUT2_1)....642
14.2.1.2.12. Reserved Input1 Register 0 (RSVD_IN1_0)....642
14.2.1.2.13. Reserved Input1 Register 1 (RSVD_IN1_1)....642
14.2.1.2.14. Reserved Input2 Register 0 (RSVD_IN2_0)....643
14.2.1.2.15. Reserved Input2 Register 1 (RSVD_IN2_1)....643
15. Core 2 Software JTAG....644
15.1. Software JTAG Register....644
15.1.1. Register Map....644
15.1.2. Register Description....644

15.1.2.1. Software Debug (DEBUG)....644
16. Tracer....646
16.1. Tracer Registers....646
16.1.1. Register Maps....647
16.1.1.1. Trace Select Register Map....647
16.1.1.2. Encoder Block Register Map....647
16.1.1.3. RAM Control Block Register Map....648
16.1.1.4. *Pin Interface Block* (**PIB** Interface) Register Map....648
16.1.2. Register Descriptions....649
16.1.2.1. Trace Select Register....649
16.1.2.1.1. Trace Source Select Register (TRACE_SEL)....649
16.1.2.2. Encoder Block Registers....649
16.1.2.2.1. Control Register (TRTECONTROL)....649
16.1.2.2.2. Encoder Block Implementation Parameters Register (TRTEIMPL)....652
16.1.2.2.3. Trace Instruction Features Register (TRTEINSTFEATURES)....653
16.1.2.2.4. Timestamp Control Register (TRTSCONTROL)....653
16.1.2.2.5. Timestamp Counter Lower Bits (TRTSCOUNTERLOW)....654
16.1.2.2.6. Timestamp Counter Upper Bits (TRTSCOUNTERHIGH)....654
16.1.2.3. RAM Control Block Registers....655
16.1.2.3.1. RAM Memory Control Register (TRRAMCONTROL)....655
16.1.2.3.2. RAM Memory Implementation Parameters Register (TRRAMIMPL)....656
16.1.2.3.3. RAM Memory Start Address Low Bits Register (TRRAMSTARTLOW)....657
16.1.2.3.4. RAM Memory Start Address High Bits Register (TRRAMSTARTHIGH)....657
16.1.2.3.5. RAM Memory End Address Low Bits Register (TRRAMLIMITLOW)....657
16.1.2.3.6. RAM Memory End Address High Bits Register (TRRAMLIMITHIGH)....657
16.1.2.3.7. RAM Memory Write Pointer Address Low Bits Register (TRRAMWPLOW)....658
16.1.2.3.8. RAM Memory Write Pointer Address High Bits Register (TRRAMWPHIGH)....658
16.1.2.3.9. RAM Memory Read Pointer Address Low Bits Register (TRRAMRPLOW)....658
16.1.2.3.10. RAM Memory Read Pointer Address High Bits Register (TRRAMRPHIGH)....658
16.1.2.3.11. RAM Memory Read Data Register (TRRAMDATA)....659
16.1.2.4. PIB Block Registers....659
16.1.2.4.1. PIB Block Control Register (TRPIBCONTROL)....659
17. Bus Matrix....661
17.1. Bus Matrix Registers....661
17.1.1. Register Map....661
17.1.2. Register Descriptions....661
17.1.2.1. Arbitration Priority Master 1 Register (PL1)....662
17.1.2.2. Arbitration Priority Master 2 Register (PL2)....662
17.1.2.3. Early Burst Termination Count Register (EBTCOUNT)....662
17.1.2.4. Early Burst Termination Enable (EBT_EN)....662
17.1.2.5. Early Burst Termination Register (EBT)....663
17.1.2.6. Default Master ID Number Register (DFLT_MASTER)....663

List of Tables

Table 3-1 BE-U1000 Core 0/1 address space....46
Table 3-2 BE-U1000 Core 2 address space....49
Table 4-1 int_local interrupts distribution for Core 0 and Core 1....51
Table 4-2 int_nmi interrupt for Core 1....55
Table 4-3 int_local interrupts distribution for Core 2....55
Table 5-1 Clock Sources....57
Table 5-2 Output Clocks....57
Table 5-3 `refsel` signal values....59
Table 5-4 Influence of the `alarm` and `refsel` signals to the `sys_clk` and `alt_clk` outputs....59
Table 5-5 Recommended PLL settings for the `i_pll_clk` frequency of 25 MHz....61
Table 5-6 Reset Sources block input reset signals....69
Table 5-7 Reset Sources block output reset signals....69
Table 5-8 Memory address region for control registers....72
Table 5-9 CRU register map....72
Table 5-10 Fields for register: `CLKSEL`....74
Table 5-11 Fields for register: `PLLSET`....75
Table 5-12 Fields for register: `CLKCR0`....77
Table 5-13 Fields for register: `PCLK0EN`....79
Table 5-14 Fields for register: `PCLK1EN`....82
Table 5-15 Fields for register: `PCLK2EN`....85
Table 5-16 Fields for register: `SYSCR0`....87
Table 5-17 Fields for register: `SYSCR1`....90
Table 5-18 Fields for register: `SYSCR2`....90
Table 5-19 Fields for register: `PRIOR0`....90
Table 5-20 Fields for register: `PRIOR1`....92
Table 5-21 Fields for register: `IOPUCR0`....93
Table 5-22 Fields for register: `IOPUCR1`....94
Table 5-23 Fields for register: `IOPDCR0`....94
Table 5-24 Fields for register: `IOPDCR1`....95
Table 5-25 Fields for register: `IOAFCR0`....96
Table 5-26 Fields for register: `IOAFCR1`....98
Table 5-27 Fields for register: `IOAFCR2`....100
Table 5-28 Fields for register: `IOAFCR3`....101
Table 5-29 Fields for register: `IOAFCR4`....103
Table 5-30 Fields for register: `IOAFCR5`....105
Table 5-31 Fields for register: `IODSCR0`....107
Table 5-32 Fields for register: `IODSCR1`....108
Table 5-33 Fields for register: `IODSCR2`....109
Table 5-34 Fields for register: `LDOCR`....110
Table 5-35 Fields for register: `USBCR`....111
Table 5-36 Fields for register: `SYSSR0`....114
Table 5-37 Fields for register: `WDTCLRKEY`....114

Table 5-38 Fields for register: INTCR0 115
Table 5-39 Fields for register: CLKCR1 115
Table 5-40 Fields for register: FLASHNVRCR 116
Table 5-41 Fields for register: CORE1CR 117
Table 6-1 HPM events 119
Table 6-2 Memory map of the Core Debug block 123
Table 6-3 CPU flags (DM_CPU_FLAGS) description 123
Table 6-4 CLIC CSRs 125
Table 6-5 Trigger Module CSRs 126
Table 6-6 Core Debug CSRs 126
Table 6-7 Core CSRs 126
Table 6-8 Fields for register: NMITVEC 127
Table 6-9 Fields for register: XCCFG 127
Table 6-10 Fields for register: XICFG 127
Table 6-11 JTAG DTM registers 128
Table 6-12 DM registers (access via DMI) 128
Table 6-13 Memory address region for CLINT registers 129
Table 6-14 CLINT register map 129
Table 6-15 Fields for register: MSIP0 130
Table 6-16 Fields for register: MTIMECMP0 130
Table 6-17 Fields for register: MTIME 131
Table 6-18 Memory address region for Bus Error Unit registers 131
Table 6-19 Bus Error Unit register map 131
Table 6-20 Fields for register: STATUS_CPU 132
Table 6-21 Fields for register: CONTROL_CPU 132
Table 6-22 Fields for register: ADDR_LO_CPU 132
Table 6-23 Fields for register: ADDR_HI_CPU 132
Table 6-24 Memory address region for CLIC registers 133
Table 6-25 CLIC register map 133
Table 6-26 Memory map of the Core Debug block 135
Table 6-27 CPU flags (DM_CPU_FLAGS) description 136
Table 6-28 Trigger Module CSRs 137
Table 6-29 Core Debug CSRs 138
Table 6-30 Core CSRs 138
Table 6-31 Fields for register: XCCFG 138
Table 6-32 JTAG DTM registers 139
Table 6-33 DM registers (access via DMI) 139
Table 6-34 Memory address region for CLINT registers 140
Table 6-35 CLINT register map 140
Table 6-36 Fields for register: MSIP0 141
Table 6-37 Fields for register: MTIMECMP0 141
Table 6-38 Fields for register: MTIME 141
Table 7-1 Memory address region for PIO Block registers 142
Table 7-2 PIO register map 142

Table 7-3 Fields for register: INT....143
Table 7-4 Fields for register: PIO_IN....143
Table 7-5 Fields for register: PIO_EN....143
Table 7-6 Fields for register: PIO_OU....144
Table 8-1 Table of connectivity of DMA Handshake channels....147
Table 8-2 Programming the Alternate Functions....148
Table 8-3 Parameters used in transfer examples....156
Table 8-4 Memory address regions for DMA Controllers....161
Table 8-5 DMA register map....161
Table 8-6 Fields for register: SARx....165
Table 8-7 Fields for register: DARx....165
Table 8-8 Fields for register: LLPx....166
Table 8-9 Fields for register: CTLx....167
Table 8-10 CTLx.SRC_MSIZE and CTLx.DEST_MSIZE decoding....169
Table 8-11 CTLx.SRC_TR_WIDTH and CTLx.DST_TR_WIDTH decoding....170
Table 8-12 CTLx.TT_FC field decoding....170
Table 8-13 Fields for register: CFGx....170
Table 8-14 PROTCTL field to HPROT mapping....174
Table 8-15 Fields for register: RAW_TFR....174
Table 8-16 Fields for register: RAW_BLOCK....175
Table 8-17 Fields for register: RAW_SRC_TRAN....175
Table 8-18 Fields for register: RAW_DST_TRAN....175
Table 8-19 Fields for register: RAW_ERR....176
Table 8-20 Fields for register: STAT_TFR....176
Table 8-21 Fields for register: STAT_BLOCK....177
Table 8-22 Fields for register: STAT_SRC_TRAN....177
Table 8-23 Fields for register: STAT_DST_TRAN....178
Table 8-24 Fields for register: STAT_ERR....178
Table 8-25 Fields for register: MASK_TFR....178
Table 8-26 Fields for register: MASK_BLOCK....179
Table 8-27 Fields for register: MASK_SRC_TRAN....179
Table 8-28 Fields for register: MASK_DST_TRAN....180
Table 8-29 Fields for register: MASK_ERR....180
Table 8-30 Fields for register: CLR_TFR....181
Table 8-31 Fields for register: CLR_BLOCK....181
Table 8-32 Fields for register: CLR_SRC_TRAN....182
Table 8-33 Fields for register: CLR_DST_TRAN....182
Table 8-34 Fields for register: CLR_ERR....182
Table 8-35 Fields for register: STAT....183
Table 8-36 Fields for register: REQ_SRC....184
Table 8-37 Fields for register: REQ_DST....185
Table 8-38 Fields for register: SGL_REQ_SRC....185
Table 8-39 Fields for register: SGL_REQ_DST....186
Table 8-40 Fields for register: LST_SRC....186

Table 8-41 Fields for register: LST_DST 187
Table 8-42 Fields for register: CFG 188
Table 8-43 Fields for register: CH_EN 188
Table 8-44 Fields for register: TEST 189
Table 8-45 Transfer types 190
Table 8-46 Single-block transfer parameters 195
Table 8-47 Multi-block transfer with linked lists for source and destination 196
Table 9-1 CRU registers that affect the operation of the eFLASH Controller 200
Table 9-2 Memory address region for the eFLASH Controller registers 201
Table 9-3 Memory map of the eFLASH Controller registers 201
Table 9-4 Fields for register: EFLASH_CR 202
Table 9-5 Fields for register: EFLASH_ADDR 203
Table 9-6 Fields for register: EFLASH_WDATA 203
Table 9-7 Fields for register: EFLASH_RDATA 203
Table 9-8 Fields for register: EFLASH_SR 204
Table 9-9 Fields for register: EFLASH_TIME0 204
Table 9-10 Fields for register: EFLASH_TIME1 205
Table 9-11 Fields for register: EFLASH_TIME2 205
Table 9-12 Fields for register: EFLASH_TIME3 206
Table 9-13 Fields for register: EFLASH_TIME4 207
Table 9-14 Fields for register: EFLASH_TIME5 207
Table 9-15 Fields for register: EFLASH_TIME6 208
Table 9-16 Time intervals values 209
Table 10-1 Memory address regions for Mailbox registers 214
Table 10-2 Mailbox register map 214
Table 10-3 Fields for register: MBnDATA0 215
Table 10-4 Fields for register: MBnFULL0 215
Table 10-5 Fields for register: MBnEMPTY0 216
Table 10-6 Fields for register: MBnDATA1 216
Table 10-7 Fields for register: MBnFULL1 217
Table 10-8 Fields for register: MBnEMPTY1 217
Table 11-1 ADC pins 219
Table 11-2 ADC channels connection to the VIN pins 219
Table 11-3 Temperature parameters 221
Table 11-4 PWMA event selection 223
Table 11-5 Memory address regions for ADC controller registers 223
Table 11-6 ADC register map 224
Table 11-7 Fields for register: ADC_CR1 224
Table 11-8 Fields for register: ASER 227
Table 11-9 Fields for register: ASTR 228
Table 11-10 Fields for register: ASMPR 229
Table 11-11 Fields for register: ASQR 230
Table 11-12 Fields for register: ADR 231
Table 12-1 ITR to TRGO connection 233

Table 12-2 Timer slave modes..... 237
Table 12-3 Output control bits for complementary OCx and OCxN channels..... 245
Table 12-4 Memory address regions for PWMA registers..... 250
Table 12-5 PWMA register map..... 251
Table 12-6 Fields for register: PWMA_CR1..... 252
Table 12-7 Fields for register: PWMA_CR2..... 254
Table 12-8 Fields for register: PWMA_SMCR..... 257
Table 12-9 Fields for register: PWMA_DIER..... 260
Table 12-10 Fields for register: PWMA_SR..... 262
Table 12-11 Fields for register: PWMA_EGR..... 266
Table 12-12 Fields for register: PWMA_CCMR1..... 269
Table 12-13 PWMA_CCMR1[15:10] bits functions for different CC1S values..... 270
Table 12-14 PWMA_CCMR1[7:2] bits functions for different CC0S values..... 272
Table 12-15 Fields for register: PWMA_CCMR2..... 274
Table 12-16 PWMA_CCMR2[15:10] bits functions for different CC3S values..... 275
Table 12-17 PWMA_CCMR2[7:2] bits functions for different CC2S values..... 276
Table 12-18 Fields for register: PWMA_CCER..... 279
Table 12-19 Fields for register: PWMA_CNT..... 281
Table 12-20 Fields for register: PWMA_PSC..... 281
Table 12-21 Fields for register: PWMA_ARR..... 281
Table 12-22 Fields for register: PWMA_RCR..... 282
Table 12-23 Fields for register: PWMA_CCR0..... 282
Table 12-24 Fields for register: PWMA_CCR1..... 283
Table 12-25 Fields for register: PWMA_CCR2..... 284
Table 12-26 Fields for register: PWMA_CCR3..... 284
Table 12-27 Fields for register: PWMA_BDTR..... 285
Table 12-28 Fields for register: PWMA_DCR..... 287
Table 12-29 Fields for register: PWMA_DMAR..... 288
Table 12-30 Fields for register: PWMA_QESET..... 288
Table 12-31 Fields for register: PWMA_FILTSET..... 290
Table 12-32 Fields for register: PWMA_QEMAX..... 291
Table 12-33 Fields for register: PWMA_INITSET..... 292
Table 12-34 Fields for register: PWMA_QECNT..... 292
Table 12-35 Memory address regions for PWMG registers..... 298
Table 12-36 PWMG register map..... 298
Table 12-37 Fields for register: PWMG_CR1..... 299
Table 12-38 Fields for register: PWMG_IER..... 300
Table 12-39 Fields for register: PWMG_SR..... 300
Table 12-40 Fields for register: PWMG_EGR..... 302
Table 12-41 Fields for register: PWMG_CCMR1..... 302
Table 12-42 PWMG_CCMR1[7:2] bit functions for different CC0S values..... 303
Table 12-43 Fields for register: PWMG_CCER..... 305
Table 12-44 Fields for register: PWMG_CNT..... 306
Table 12-45 Fields for register: PWMG_PSC..... 306

Table 12-46 Fields for register: PWMG_ARR 306
Table 12-47 Fields for register: PWMG_CCR0 307
Table 12-48 Duty cycle, TIMER<N>LOAD_COUNT and TIMER<N>LOAD_COUNT2 relationship 311
Table 12-49 Memory address regions for TIM registers 312
Table 12-50 TIM address range 312
Table 12-51 TIM register map 312
Table 12-52 Fields for register: TIMER<N>LOAD_COUNT 314
Table 12-53 Fields for register: TIMER<N>LOAD_COUNT2 314
Table 12-54 Fields for register: TIMER<N>CURRENT_VALUE 314
Table 12-55 Fields for register: TIMER<N>CONTROL_REG 315
Table 12-56 Fields for register: TIMER<N>EOI 316
Table 12-57 Fields for register: TIMER<N>INT_STATUS 316
Table 12-58 Fields for register: TIMERS_INT_STATUS 316
Table 12-59 Fields for register: TIMERS_EOI 317
Table 12-60 Fields for register: TIMERS_RAW_INT_STATUS 317
Table 12-61 Memory address regions for WDT registers 320
Table 12-62 WDT register map 320
Table 12-63 Fields for register: WDT_CR 321
Table 12-64 Fields for register: WDT_TORR 322
Table 12-65 Fields for register: WDT_CCVR 322
Table 12-66 Fields for register: WDT_CRR 322
Table 12-67 Fields for register: WDT_STAT 323
Table 12-68 Fields for register: WDT_EOI 323
Table 12-69 Fields for register: WDT_COMP_PARAM_5 323
Table 12-70 Fields for register: WDT_COMP_PARAM_4 324
Table 12-71 Fields for register: WDT_COMP_PARAM_3 324
Table 12-72 Fields for register: WDT_COMP_PARAM_2 325
Table 12-73 Fields for register: WDT_COMP_PARAM_1 325
Table 13-1 Memory address regions for GPIO Controller registers 329
Table 13-2 GPIO register map 330
Table 13-3 Fields for register: GPIO_DR 330
Table 13-4 Fields for register: GPIO_DDR 331
Table 13-5 Fields for register: GPIO_INTEN 331
Table 13-6 Fields for register: GPIO_INTMSK 331
Table 13-7 Fields for register: GPIO_INTLVL 332
Table 13-8 Fields for register: GPIO_INTPOLARITY 332
Table 13-9 Fields for register: GPIO_INTSTAT 332
Table 13-10 Fields for register: GPIO_INTSTATRAW 333
Table 13-11 Fields for register: GPIO_DEBOUNCE 333
Table 13-12 Fields for register: GPIO_EOI 333
Table 13-13 Fields for register: GPIO_EXT 334
Table 13-14 Fields for register: GPIO_SYNC 334
Table 13-15 Fields for register: GPIO_BOTHEDGE 334
Table 13-16 CAN FD operation mode transitions 336

Table 13-17 Memory address regions for CAN FD registers....339
Table 13-18 CAN FD register map....340
Table 13-19 Fields for register: SRR....342
Table 13-20 Fields for register: MSR....342
Table 13-21 Fields for register: BRPR....345
Table 13-22 Fields for register: BTR....345
Table 13-23 Fields for register: ECR....346
Table 13-24 Fields for register: ESR....346
Table 13-25 Fields for register: SR....348
Table 13-26 Fields for register: ISR....350
Table 13-27 Fields for register: IER....354
Table 13-28 Fields for register: ICR....357
Table 13-29 Fields for register: TSR....360
Table 13-30 Fields for register: DP_BRPR....360
Table 13-31 Fields for register: DP_BTR....361
Table 13-32 Fields for register: TRR....362
Table 13-33 Fields for register: IETRS....363
Table 13-34 Fields for register: TCR....363
Table 13-35 Fields for register: IETCS....364
Table 13-36 Fields for register: TxE_FSR....364
Table 13-37 Fields for register: TxE_WMR....365
Table 13-38 Fields for register: AFR....366
Table 13-39 Fields for register: FSR....366
Table 13-40 Fields for register: WMR....367
Table 13-41 Fields for register: TBiID....368
Table 13-42 Fields for register: TBiDLC....369
Table 13-43 Fields for register: TBiDWn....370
Table 13-44 Fields for register: AFMRi....371
Table 13-45 Fields for register: AFIRi....372
Table 13-46 Fields for register: TXE_FIFO_TBi_ID....373
Table 13-47 Fields for register: TXE_FIFO_TBi_DLC....374
Table 13-48 Fields for register: RBiID....375
Table 13-49 Fields for register: RBiDLC....376
Table 13-50 Fields for register: RBiDWn....377
Table 13-51 Divisor latch fractional values....385
Table 13-52 Memory address regions for UART Controller registers....386
Table 13-53 UART register map....386
Table 13-54 Fields for register: RBR....388
Table 13-55 Fields for register: THR....388
Table 13-56 Fields for register: DLL....389
Table 13-57 Fields for register: DLH....389
Table 13-58 Fields for register: IER....390
Table 13-59 Fields for register: IIR....391
Table 13-60 Interrupt control functions....392

Table 13-61 Fields for register: FCR....392
Table 13-62 Fields for register: LCR....393
Table 13-63 Fields for register: MCR....395
Table 13-64 Fields for register: LSR....395
Table 13-65 Fields for register: SCR....398
Table 13-66 Fields for register: SRBRi....398
Table 13-67 Fields for register: STHRi....399
Table 13-68 Fields for register: FAR....399
Table 13-69 Fields for register: TFR....399
Table 13-70 Fields for register: RFW....400
Table 13-71 Fields for register: USR....400
Table 13-72 Fields for register: TFL....402
Table 13-73 Fields for register: RFL....402
Table 13-74 Fields for register: SRR....402
Table 13-75 Fields for register: SBCR....403
Table 13-76 Fields for register: SFE....403
Table 13-77 Fields for register: SRT....404
Table 13-78 Fields for register: STET....404
Table 13-79 Fields for register: HTx....405
Table 13-80 Fields for register: DMASA....405
Table 13-81 Fields for register: TCR....405
Table 13-82 Fields for register: DE_EN....407
Table 13-83 Fields for register: RE_EN....407
Table 13-84 Fields for register: DET....407
Table 13-85 Fields for register: TAT....408
Table 13-86 Fields for register: DLF....409
Table 13-87 Fields for register: RAR....409
Table 13-88 Fields for register: TAR....409
Table 13-89 Fields for register: LCR_EXT....410
Table 13-90 Fields for register: REG_TIMEOUT_RST....411
Table 13-91 *SPI feature table....412
Table 13-92 Transmit FIFO Threshold (TFT) decode values....415
Table 13-93 Receive FIFO Threshold (RFT) decode values....415
Table 13-94 Memory address regions for *SPI Controller registers....422
Table 13-95 *SPI register map....422
Table 13-96 Fields for register: CTRLR0....424
Table 13-97 Fields for register: CTRLR1....427
Table 13-98 Fields for register: SPIENR....427
Table 13-99 Fields for register: MWCR....428
Table 13-100 Fields for register: SER....429
Table 13-101 Fields for register: BAUDR....430
Table 13-102 Fields for register: TXFTLR....430
Table 13-103 Fields for register: RXFTLR....431
Table 13-104 Fields for register: TXFLR....431

Table 13-105 Fields for register: RXFLR 431
Table 13-106 Fields for register: SR 432
Table 13-107 Fields for register: IMR 433
Table 13-108 Fields for register: ISR 434
Table 13-109 Fields for register: RISR 436
Table 13-110 Fields for register: TXOICR 437
Table 13-111 Fields for register: RXOICR 437
Table 13-112 Fields for register: RXUICR 437
Table 13-113 Fields for register: MSTICR 438
Table 13-114 Fields for register: ICR 438
Table 13-115 Fields for register: DMACR 438
Table 13-116 Fields for register: DMATDLR 439
Table 13-117 Fields for register: DMARDLR 439
Table 13-118 Fields for register: DRi 440
Table 13-119 Fields for register: RX_SAMPLE_DLY 440
Table 13-120 Fields for register: SPI_CTRLR0 441
Table 13-121 Fields for register: TXD_DRIVE_EDGE 442
Table 13-122 Fields for register: XIP_CMD 443
Table 13-123 I2C definition of bits in the first byte 445
Table 13-124 Maximum values for IC_SDA_RX_HOLD 451
Table 13-125 I2C events monitored by individual interrupts 452
Table 13-126 Memory address regions for I2C registers 453
Table 13-127 I2C register map 453
Table 13-128 Fields for register: IC_CON 455
Table 13-129 Fields for register: IC_TAR 459
Table 13-130 Fields for register: IC_SAR 460
Table 13-131 Fields for register: IC_HS_MADDRR 460
Table 13-132 Fields for register: IC_DATA_CMD 461
Table 13-133 Fields for register: IC_SS_SCL_HCNT 462
Table 13-134 Fields for register: IC_SS_SCL_LCNT 463
Table 13-135 Fields for register: IC_FS_SCL_HCNT 463
Table 13-136 Fields for register: IC_FS_SCL_LCNT 463
Table 13-137 Fields for register: IC_HS_SCL_HCNT 464
Table 13-138 Fields for register: IC_HS_SCL_LCNT 464
Table 13-139 Fields for register: IC_INTR_STAT 465
Table 13-140 Fields for register: IC_INTR_MASK 467
Table 13-141 Fields for register: IC_RAW_INTR_STAT 469
Table 13-142 Fields for register: IC_RX_TL 474
Table 13-143 Fields for register: IC_TX_TL 474
Table 13-144 Fields for register: IC_CLR_INTR 474
Table 13-145 Fields for register: IC_CLR_RX_UNDER 475
Table 13-146 Fields for register: IC_CLR_RX_OVER 475
Table 13-147 Fields for register: IC_CLR_TX_OVER 475
Table 13-148 Fields for register: IC_CLR_RD_REQ 476

Table 13-149 Fields for register: IC_CLR_TX_ABRT ... 476
Table 13-150 Fields for register: IC_CLR_RX_DONE ... 476
Table 13-151 Fields for register: IC_CLR_ACTIVITY ... 477
Table 13-152 Fields for register: IC_CLR_STOP_DET ... 477
Table 13-153 Fields for register: IC_CLR_START_DET ... 477
Table 13-154 Fields for register: IC_CLR_GEN_CALL ... 478
Table 13-155 Fields for register: IC_ENABLE ... 478
Table 13-156 Fields for register: IC_STATUS ... 480
Table 13-157 Fields for register: IC_TXFLR ... 482
Table 13-158 Fields for register: IC_RXFLR ... 483
Table 13-159 Fields for register: IC_SDA_HOLD ... 483
Table 13-160 Fields for register: IC_TX_ABRT_SOURCE ... 484
Table 13-161 Fields for register: IC_DMA_CR ... 489
Table 13-162 Fields for register: IC_DMA_TDLR ... 490
Table 13-163 Fields for register: IC_DMA_RDLR ... 490
Table 13-164 Fields for register: IC_SDA_SETUP ... 491
Table 13-165 Fields for register: IC_ACK_GENERAL_CALL ... 491
Table 13-166 Fields for register: IC_ENABLE_STATUS ... 491
Table 13-167 Fields for register: IC_FS_SPKLEN ... 493
Table 13-168 Fields for register: IC_HS_SPKLEN ... 494
Table 13-169 Fields for register: IC_SCL_STUCK_AT_LOW_TIMEOUT ... 494
Table 13-170 Fields for register: IC_SDA_STUCK_AT_LOW_TIMEOUT ... 495
Table 13-171 Fields for register: IC_CLR_SCL_STUCK_DET ... 495
Table 13-172 Fields for register: REG_TIMEOUT_RST ... 496
Table 13-173 Memory address regions for I2S Controllers ... 503
Table 13-174 I2S register map ... 504
Table 13-175 Fields for register: IER ... 505
Table 13-176 Fields for register: IRER ... 506
Table 13-177 Fields for register: ITER ... 506
Table 13-178 Fields for register: RXFFR ... 506
Table 13-179 Fields for register: TXFFR ... 507
Table 13-180 Fields for register: LRBR0 ... 507
Table 13-181 Fields for register: LTHR0 ... 508
Table 13-182 Fields for register: RRBR0 ... 508
Table 13-183 Fields for register: RTHR0 ... 508
Table 13-184 Fields for register: RER0 ... 509
Table 13-185 Fields for register: TER0 ... 509
Table 13-186 Fields for register: RCR0 ... 509
Table 13-187 Fields for register: TCR0 ... 510
Table 13-188 Fields for register: ISR0 ... 511
Table 13-189 Fields for register: IMR0 ... 511
Table 13-190 Fields for register: ROR0 ... 512
Table 13-191 Fields for register: TOR0 ... 512
Table 13-192 Fields for register: RFCR0 ... 513

Table 13-193 Fields for register: TFCR0 ... 513
Table 13-194 Fields for register: RFF0 ... 514
Table 13-195 Fields for register: TFF0 ... 514
Table 13-196 Fields for register: RXDMA ... 515
Table 13-197 Fields for register: RRXDMA ... 515
Table 13-198 Fields for register: TXDMA ... 516
Table 13-199 Fields for register: RTXDMA ... 516
Table 13-200 Fields for register: DMACR ... 516
Table 14-1 Response to LPM transaction from USB 2.0 Controller ... 523
Table 14-2 EP interrupt associated with RXPKTRDY being set ... 533
Table 14-3 EP interrupt associated with TXPKTRDY being cleared ... 533
Table 14-4 Endpoint #n FIFO parameters registers ... 539
Table 14-5 Values of TXCSRH register ... 554
Table 14-6 Values of RXCSRH register ... 556
Table 14-7 Values of RXCSRH register ... 559
Table 14-8 Values of TXCSRH register ... 561
Table 14-9 Values of TXCSRH register ... 566
Table 14-10 Values of RXCSRH register ... 568
Table 14-11 Values of RXCSRH register ... 571
Table 14-12 Values of TXCSRH register ... 572
Table 14-13 Memory address region for USB registers ... 574
Table 14-14 USB 2.0 Controller register map ... 574
Table 14-15 Fields for register: FADDR ... 579
Table 14-16 Fields for register: POWER ... 579
Table 14-17 Fields for register: INTRTX ... 581
Table 14-18 Fields for register: INTRRX ... 582
Table 14-19 Fields for register: INTRTXE ... 582
Table 14-20 Fields for register: INTRRXE ... 583
Table 14-21 Fields for register: INTRUSB ... 584
Table 14-22 Fields for register: INTRUSBE ... 585
Table 14-23 Fields for register: FRAME ... 585
Table 14-24 Fields for register: INDEX ... 586
Table 14-25 Fields for register: TESTMODE ... 586
Table 14-26 Fields for register: TXMAXP ... 589
Table 14-27 Fields for register: CSR0L (DEVCTL.HOST_MODE = 0x0) ... 589
Table 14-28 Fields for register: CSR0L (DEVCTL.HOST_MODE = 0x1) ... 590
Table 14-29 Fields for register: TXCSRL (DEVCTL.HOST_MODE = 0x0) ... 592
Table 14-30 Fields for register: TXCSRL (DEVCTL.HOST_MODE = 0x1) ... 593
Table 14-31 Fields for register: CSR0H (DEVCTL.HOST_MODE=0x0) ... 595
Table 14-32 Fields for register: CSR0H (DEVCTL>.HOST_MODE=0x1) ... 595
Table 14-33 Fields for register: TXCSRH (DEVCTL.HOST_MODE= 0x0) ... 596
Table 14-34 Fields for register: TXCSRH (DEVCTL.HOST_MODE= 0x1) ... 597
Table 14-35 Fields for register: RXMAXP ... 599
Table 14-36 Fields for register: RXCSRL (DEVCTL.HOST_MODE= 0x0) ... 600

Table 14-37 Fields for register: RXCSRL (DEVCTL.HOST_MODE= 0x1)..... 601
Table 14-38 Fields for register: RXCSRH (DEVCTL.HOST_MODE= 0x0)..... 603
Table 14-39 Fields for register: RXCSRH (DEVCTL.HOST_MODE= 0x1)..... 604
Table 14-40 RXPKTRDY bit clearing..... 606
Table 14-41 Fields for register: COUNT0..... 606
Table 14-42 Fields for register: RXCOUNT..... 607
Table 14-43 Fields for register: TXTYPE..... 607
Table 14-44 Fields for register: TYPE0..... 608
Table 14-45 Fields for register: NAKLIMIT0..... 609
Table 14-46 Fields for register: TXINTERVAL..... 609
Table 14-47 Fields for register: RXTYPE..... 610
Table 14-48 Fields for register: RXINTERVAL..... 611
Table 14-49 Fields for register: CONFIGDATA..... 612
Table 14-50 Fields for register: FIFOn..... 613
Table 14-51 Fields for register: DEVCTL..... 613
Table 14-52 Fields for register: TXFIFOSZ..... 615
Table 14-53 Fields for register: RXFIFOSZ..... 615
Table 14-54 Fields for register: TXFIFOAD..... 616
Table 14-55 Fields for register: RXFIFOAD..... 617
Table 14-56 Fields for register: VCONTROL..... 617
Table 14-57 Fields for register: VSTATUS..... 619
Table 14-58 Fields for register: EPINFO..... 619
Table 14-59 Fields for register: RAMINFO..... 619
Table 14-60 Fields for register: LINKINFO..... 620
Table 14-61 Fields for register: VPLEN..... 620
Table 14-62 Fields for register: HS_EOF1..... 620
Table 14-63 Fields for register: FS_EOF1..... 621
Table 14-64 Fields for register: LS_EOF1..... 621
Table 14-65 Fields for register: TXFUNCADDR..... 622
Table 14-66 Fields for register: TXHUBADDR..... 622
Table 14-67 Fields for register: TXHUBPORT..... 623
Table 14-68 Fields for register: RXFUNCADDR..... 623
Table 14-69 Fields for register: RXHUBADDR..... 624
Table 14-70 Fields for register: RXHUBPORT..... 624
Table 14-71 Fields for register: DMA_INTR..... 625
Table 14-72 Fields for register: DMA_CNTL..... 625
Table 14-73 Fields for register: DMA_ADDR..... 626
Table 14-74 Fields for register: DMA_COUNT..... 627
Table 14-75 Fields for register: EPn_RQPKTCOUNT..... 627
Table 14-76 Fields for register: C_T_UCH..... 628
Table 14-77 Fields for register: C_T_HSRTN..... 628
Table 14-78 Fields for register: C_T_HSBT..... 628
Table 14-79 C_T_HSBT values decoding..... 629
Table 14-80 Fields for register: LPM_ATTR..... 630

Table 14-81 Fields for register: LPM_CNTRL (DEVCTL.HOST_MODE = 0x0)....631
Table 14-82 Fields for register: LPM_CNTRL (DEVCTL.HOST_MODE = 0x1)....632
Table 14-83 Fields for register: LPM_INTREN....633
Table 14-84 Fields for register: LPM_INTR (DEVCTL.HOST_MODE=0x0)....633
Table 14-85 Fields for register: LPM_INTR (DEVCTL.HOST_MODE=0x1)....634
Table 14-86 Fields for register: LPM_FADDR....635
Table 14-87 USB 2.0 PHY register map....637
Table 14-88 Fields for register: HS_RES_ADJ_TX_CLK_GATE_CTL....638
Table 14-89 Fields for register: SPWR_SVNG_SQLCH_CTL....638
Table 14-90 Fields for register: OUT_EYE_SHAPE_INT_BIAS_CTL....639
Table 14-91 Fields for register: TX_CLK_PHASE_PLL_ADJ_CTL....639
Table 14-92 Fields for register: PARAM_CTL_0....639
Table 14-93 Fields for register: PARAM_CTL_1....640
Table 14-94 Fields for register: PARAM_CTL_2....641
Table 14-95 Fields for register: RSVD_OUT1_0....641
Table 14-96 Fields for register: RSVD_OUT1_1....642
Table 14-97 Fields for register: RSVD_OUT2_0....642
Table 14-98 Fields for register: RSVD_OUT2_1....642
Table 14-99 Fields for register: RSVD_IN1_0....642
Table 14-100 Fields for register: RSVD_IN1_1....642
Table 14-101 Fields for register: RSVD_IN2_0....643
Table 14-102 Fields for register: RSVD_IN2_1....643
Table 15-1 Memory address region for software JTAG register....644
Table 15-2 Software JTAG register map....644
Table 15-3 Fields for register: DEBUG....645
Table 16-1 Memory address regions for Tracer registers....647
Table 16-2 Memory address region for Trace Select register....647
Table 16-3 Trace Select register map....647
Table 16-4 Encoder Block register map....648
Table 16-5 RAM Control Block register map....648
Table 16-6 PIB Block register map....649
Table 16-7 Fields for register: TRACE_SEL....649
Table 16-8 Fields for register: TRTECONTROL....649
Table 16-9 Fields for register: TRTEIMPL....652
Table 16-10 Fields for register: TRTEINSTFEATURES....653
Table 16-11 Fields for register: TRTSCONTROL....653
Table 16-12 Fields for register: TRTSCOUNTERLOW....654
Table 16-13 Fields for register: TRTSCOUNTERHIGH....655
Table 16-14 Fields for register: TRRAMCONTROL....655
Table 16-15 Fields for register: TRRAMIMPL....656
Table 16-16 Fields for register: TRRAMSTARTLOW....657
Table 16-17 Fields for register: TRRAMSTARTHIGH....657
Table 16-18 Fields for register: TRRAMLIMITLOW....657
Table 16-19 Fields for register: TRRAMLIMITHIGH....658

Table 16-20 Fields for register: TRRAMWPLOW....658
Table 16-21 Fields for register: TRRAMWPHIGH....658
Table 16-22 Fields for register: TRRAMRPLOW....658
Table 16-23 Fields for register: TRRAMRPHIGH....659
Table 16-24 Fields for register: TRRAMDATA....659
Table 16-25 Fields for register: TRPIBCONTROL....659
Table 17-1 Memory address region for Bus matrix registers....661
Table 17-2 Bus matrix register map....661
Table 17-3 Fields for register: PL1....662
Table 17-4 Fields for register: PL2....662
Table 17-5 Fields for register: EBTCOUNT....662
Table 17-6 Fields for register: EBT_EN....663
Table 17-7 Fields for register: EBT....663
Table 17-8 Fields for register: DFLT_MASTER....663

List of Figures

Figure 2-1 BE-U1000 architecture....45
Figure 3-1 Core 0/1 address space....49
Figure 3-2 Core 2 address space....50
Figure 5-1 CRU block diagram....56
Figure 5-2 Clock Generation block diagram....57
Figure 5-3 Clock Sources block diagram....58
Figure 5-4 `sys_clk` Selection block....59
Figure 5-5 PLL block diagram....60
Figure 5-6 PLL stability tracking....62
Figure 5-7 Dividers block diagram....62
Figure 5-8 Output Clocks Control block diagram....63
Figure 5-9 `cclk` Clock Domain....64
Figure 5-10 `pclk_0` Clock Domain....65
Figure 5-11 `pclk_1` Clock Domain....66
Figure 5-12 `pclk_2` Clock Domain....67
Figure 5-13 `hclk` Clock Domain....67
Figure 5-14 `can_clk` and `can_clk_x2` Clock Domains....68
Figure 5-15 `tclk` Clock Domain....68
Figure 5-16 Reset Sources block diagram....68
Figure 5-17 PA[0] I/O pin multiplexer....70
Figure 5-18 I/O pins functions and control for Port A....70
Figure 5-19 I/O pins functions and control for Port B....71
Figure 5-20 I/O pins functions and control for Port C....71
Figure 6-1 BR-350 Core block diagram....118
Figure 6-2 Debug subsystem....122
Figure 6-3 BM-310 Core block diagram....134
Figure 6-4 Debug subsystem....135
Figure 7-1 PIO Block in the BE-U1000....142
Figure 8-1 DMA Controller block diagram....145
Figure 8-2 Handshaking Interface for DMA_0....151
Figure 8-3 Handshaking Interface for DMA_1....152
Figure 8-4 Software controlled DMA transfers....153
Figure 8-5 SPI receive FIFO....159
Figure 8-6 Flowchart for DMA programming example....194
Figure 9-1 eFLASH Controller block diagram....200
Figure 10-1 Mailbox in BE-U1000 block diagram....213
Figure 11-1 ADC block diagram....218
Figure 12-1 PWMA block diagram....232
Figure 12-2 Control Unit....234
Figure 12-3 External Trigger Control block....235
Figure 12-4 Trigger Selection block....236
Figure 12-5 Slave Mode Controller....236

Figure 12-6 Master Mode Controller..... 237
Figure 12-7 TRGO to ITR connection example..... 238
Figure 12-8 Time-Base Unit..... 238
Figure 12-9 Capture/Compare Channels..... 242
Figure 12-10 Channel 0 Input Stage..... 243
Figure 12-11 Channel 0 Output Stage..... 244
Figure 12-12 Channel 3 Output Stage..... 246
Figure 12-13 Capture/Compare Stage 0..... 246
Figure 12-14 Quadrature Encoder..... 249
Figure 12-15 PWMG block diagram..... 292
Figure 12-16 Time-Base Unit..... 293
Figure 12-17 Capture/Compare Channel..... 295
Figure 12-18 Channel 0 Input Stage..... 295
Figure 12-19 Channel 0 Output Stage..... 296
Figure 12-20 Capture/Compare Stage 0..... 297
Figure 12-21 TIM block diagram..... 307
Figure 12-22 Timer usage flow..... 308
Figure 12-23 WDT_0 block diagram..... 318
Figure 12-24 WDT_1 block diagram..... 318
Figure 13-1 GPIO block diagram 327
Figure 13-2 CAN FD block diagram..... 335
Figure 13-3 UART block diagram..... 377
Figure 13-4 Serial data format..... 378
Figure 13-5 Auto address transmit flow chart..... 380
Figure 13-6 Hardware address match receive mode..... 381
Figure 13-7 *SPI Controller block diagram..... 412
Figure 13-8 Endian conversion for data register and XIP reads..... 414
Figure 13-9 I2C block diagram..... 444
Figure 13-10 7-bit address format..... 444
Figure 13-11 10-bit address format..... 445
Figure 13-12 `IC_SDA_HOLD` register..... 450
Figure 13-13 I2S block diagram..... 496
Figure 13-14 Basic usage flow - I2S Controller as Transmitter..... 497
Figure 13-15 Basic usage flow - I2S Controller as Receiver..... 501
Figure 14-1 USB 2.0 Controller block diagram..... 518
Figure 14-2 Link Power Management diagram..... 522
Figure 14-3 USB Interrupt Service Routine diagram..... 530
Figure 14-4 Endpoint 0 states for peripheral..... 543
Figure 14-5 Flow for Endpoint 0 service routine..... 545
Figure 14-6 Flow for SETUP phase of control transfer..... 546
Figure 14-7 Flow for IN Data phase of control transfer..... 547
Figure 14-8 Flow for OUT Data phase of control transfer..... 548
Figure 14-9 USB 2.0 PHY block diagram..... 636
Figure 16-1 Tracer block diagram..... 646

1. Introduction

This Reference Manual targets application developers. It provides complete information on how to use the memory and the peripherals of the BE-U1000 microcontroller.

For ordering information, mechanical and electrical device characteristics refer to the ***BE-U1000 Datasheet***.

2. Architecture

Simplified BE-U1000 architecture is shown in the figure below.

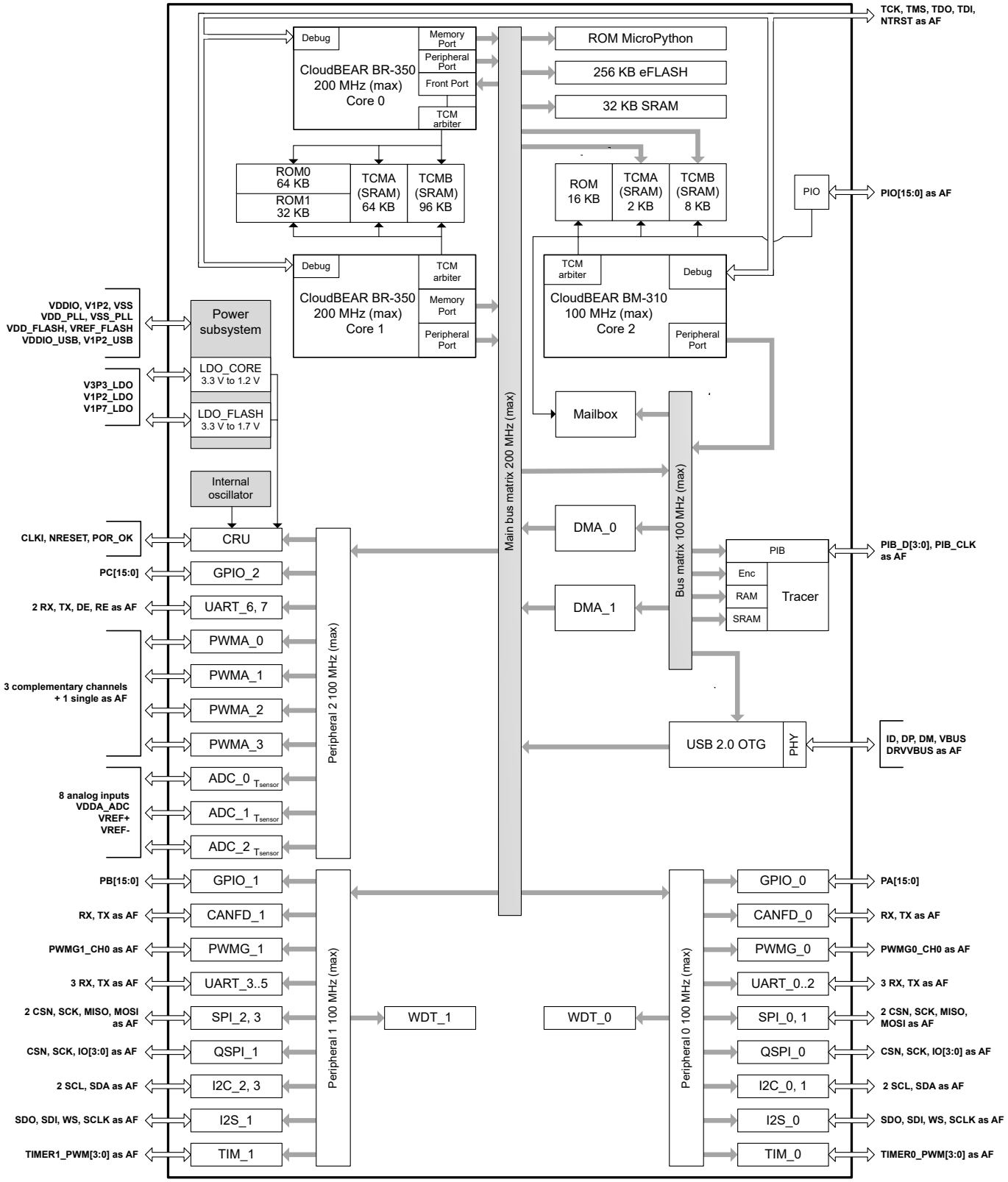

Figure 2-1 BE-U1000 architecture

3. Memory Map

BE-U1000 supports 4 GB (0x0000_0000 - 0xFFFF_FFFF) of addressable memory.

The tables below give the blocks addresses from the Core 0/1 and Core 2 sides.

The **Register Map** column provides the links to the blocks registers.

Table 3-1 BE-U1000 Core 0/1 address space

Region	Block Name	Size	Start Address	End Address	Register Map
Core 0 and Core 1 Resources	Core Debug	4 KB	0x0000_0000	0x0000_0FFF	
	CLINT	64 KB	0x0200_0000	0x0200_FFFF	See CLINT register map
	Bus Error Unit	4 KB	0x0202_0000	0x0202_0FFF	See Bus Error Unit register map
	CLIC	20 KB	0x0D00_0000	0x0D00_4FFF	See CLIC register map
Periphery 0	QSPI_0	4 KB	0x1100_0000	0x1100_0FFF	See *SPI register map
	UART_0	4 KB	0x1100_1000	0x1100_1FFF	See UART register map
	UART_1	4 KB	0x1100_2000	0x1100_2FFF	
	UART_2	4 KB	0x1100_3000	0x1100_3FFF	
	I2C_0	4 KB	0x1100_4000	0x1100_4FFF	See I2C register map
	I2C_1	4 KB	0x1100_5000	0x1100_5FFF	
	I2S_0	4 KB	0x1100_6000	0x1100_6FFF	See I2S register map
	TIM_0	4 KB	0x1100_7000	0x1100_7FFF	See TIM register map
	WDT_0	4 KB	0x1100_8000	0x1100_8FFF	See WDT register map
	GPIO_0	4 KB	0x1100_9000	0x1100_9FFF	See GPIO register map
	PWMG_0	4 KB	0x1100_A000	0x1100_AFFF	See PWMG register map
	SPI_0 (Slave)	4 KB	0x1100_B000	0x1100_BFFF	See *SPI register map
	SPI_1 (Master)	4 KB	0x1100_C000	0x1100_CFFF	
	CANFD_0	32 KB	0x1101_0000	0x1101_7FFF	See CAN FD register map
Periphery 1	QSPI_1	4 KB	0x1300_0000	0x1300_0FFF	See *SPI register map
	UART_3	4 KB	0x1300_1000	0x1300_1FFF	See UART register map
	UART_4	4 KB	0x1300_2000	0x1300_2FFF	
	UART_5	4 KB	0x1300_3000	0x1300_3FFF	
	I2C_2	4 KB	0x1300_4000	0x1300_4FFF	See I2C register map
	I2C_3	4 KB	0x1300_5000	0x1300_5FFF	
	I2S_1	4 KB	0x1300_6000	0x1300_6FFF	See I2S register map

Table 3-1 BE-U1000 Core 0/1 address space (continued)

Region	Block Name	Size	Start Address	End Address	Register Map
	TIM_1	4 KB	0x1300_7000	0x1300_7FFF	See TIM register map
	WDT_1	4 KB	0x1300_8000	0x1300_8FFF	See WDT register map
	GPIO_1	4 KB	0x1300_9000	0x1300_9FFF	See GPIO register map
	PWMG_1	4 KB	0x1300_A000	0x1300_AFFF	See PWMG register map
	SPI_2 (Slave)	4 KB	0x1300_B000	0x1300_BFFF	See *SPI register map
	SPI_3 (Master)	4 KB	0x1300_C000	0x1300_CFFF	
	CANFD_1	32 KB	0x1301_0000	0x1301_7FFF	See CAN FD register map
Periphery 2	CRU	4 KB	0x1400_0000	0x1400_0FFF	See CRU register map
	ADC_0	4 KB	0x1400_1000	0x1400_1FFF	See ADC register map
	ADC_1	4 KB	0x1400_2000	0x1400_2FFF	
	ADC_2	4 KB	0x1400_3000	0x1400_3FFF	
	PWMA_0	4 KB	0x1400_4000	0x1400_4FFF	See PWMA register map
	PWMA_1	4 KB	0x1400_5000	0x1400_5FFF	
	PWMA_2	4 KB	0x1400_6000	0x1400_6FFF	
	PWMA_3	4 KB	0x1400_7000	0x1400_7FFF	
	GPIO_2	4 KB	0x1400_8000	0x1400_8FFF	See GPIO register map
	UART_6	4 KB	0x1400_9000	0x1400_9FFF	See UART register map
	UART_7	4 KB	0x1400_A000	0x1400_AFFF	
High-speed Periphery	USB	1 MB	0x1500_0000	0x150F_FFFF	See USB 2.0 Controller register map and USB 2.0 PHY register map
	DMA_0	4 KB	0x1510_0000	0x1510_0FFF	See DMA register map
	DMA_1	4 KB	0x1510_1000	0x1510_1FFF	
	Mailbox	4 KB	0x1510_3000	0x1510_3FFF	See Mailbox register map
	Tracer encoder control interface	4 KB	0x1510_4000	0x1510_4FFF	See Encoder Block register map
	Tracer RAM control interface	4 KB	0x1510_5000	0x1510_5FFF	See RAM Control Block register map
	Tracer PIB control interface	4 KB	0x1510_6000	0x1510_6FFF	See PIB Block register map
	Tracer SRAM interface	4 KB	0x1510_7000	0x1510_7FFF	

Table 3-1 BE-U1000 Core 0/1 address space (continued)

Region	Block Name	Size	Start Address	End Address	Register Map
	Bus matrix	4 KB	0x1510_8000	0x1510_8FFF	See Bus matrix register map
System Resources	* ROM0/ROM1	64 KB	0x4000_0000	0x4000_FFFF	
	TCMA	64 KB	0x4001_0000	0x4001_FFFF	
	TCMB	96 KB	0x4002_0000	0x4003_7FFF	
	Front Port Core 0	256 KB	0x5000_0000	0x5003_FFFF	
	SRAM	32 KB	0x7000_0000	0x7000_7FFF	
	TCMA Core 2	2 KB	0x7000_F800	0x7000_FFFF	
	TCMB Core 2	8 KB	0x7001_0000	0x7001_1FFF	
	QSPI0 XIP	16 MB	0x8000_0000	0x80FF_FFFF	
	QSPI1 XIP	16 MB	0x9000_0000	0x90FF_FFFF	
	eFLASH Memory	256 KB	0xA000_0000	0xA003_FFFF	
	ROM MicroPython	192 KB	0xA080_0000	0xA082_FFFF	
	eFLASH Controller	4 KB	0xA100_0000	0xA100_0FFF	See eFLASH Controller register map

In the table above an asterisk symbol (*) indicates the following:

- 0x4000_0000 - 0x4000_FFFF 64 KB ROM0 space is only available from Core 0
- 0x4000_0000 - 0x4000_7FFF 32 KB reserved when accessed from Core 1
- 0x4000_8000 - 0x4000_FFFF 32 KB ROM1 space is only available from Core 1

In the table above:

- Accessing the 0x8XXX_XXXX addresses will cause access to 0x8000_0000 - 0x80FF_FFFF area
- Accessing the 0x9XXX_XXXX addresses will cause access to 0x9000_0000 - 0x90FF_FFFF area

The following figure shows the address space distribution of the TCMA and TCMB regions of Core 0/1.

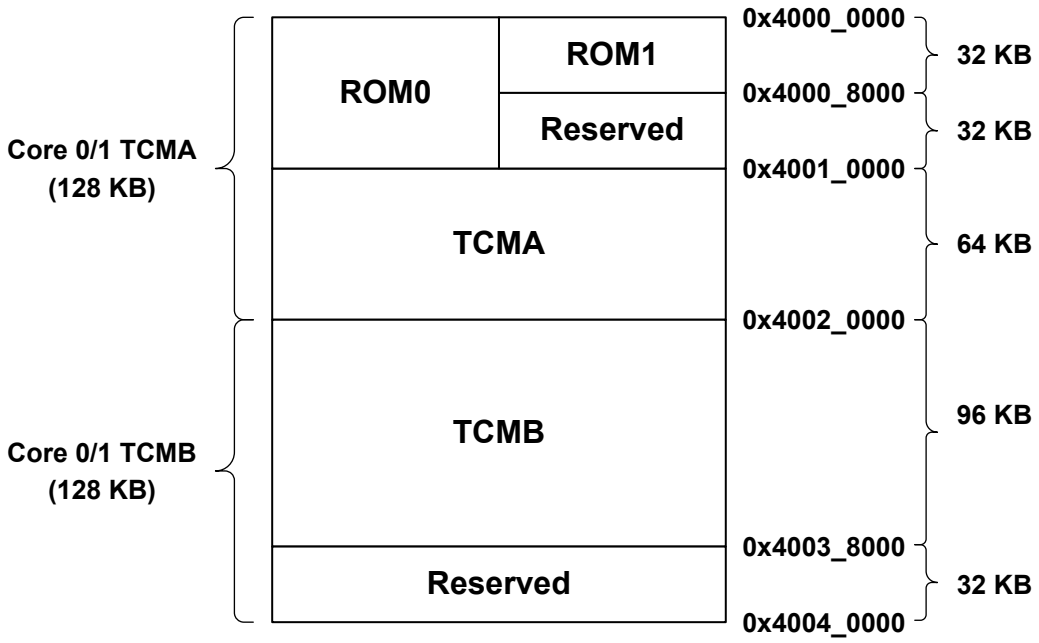

Figure 3-1 Core 0/1 address space

Table 3-2 BE-U1000 Core 2 address space

Region	Block Name	Size	Start Address	End Address	Register Map
Core 2 Resources	Core Debug	4 KB	0x0000_0000	0x0000_0FFF	
	CLINT	64 KB	0x0200_0000	0x0200_FFFF	See CLINT register map
	ROM Core 2	16 KB	0x4000_0000	0x4000_3FFF	
	TCMA Core 2	2 KB	0x4000_7800	0x4000_7FFF	
	TCMB Core 2	8 KB	0x4000_8000	0x4000_9FFF	
	Core 2 PIO, Software Jtag and Trace Select	1 KB	0x4000_A000	0x4000_A3FF	For Core 2 PIO registers, see PIO register map
					For Core 2 Software Jtag register, see Software JTAG register map
					For Core 2 Trace Select register, see Trace Select register map
	Mailbox	1 KB	0x4000_A400	0x4000_A800	See Mailbox register map

The following figure shows the address space distribution of the TCMA and TCMB regions of Core 2.

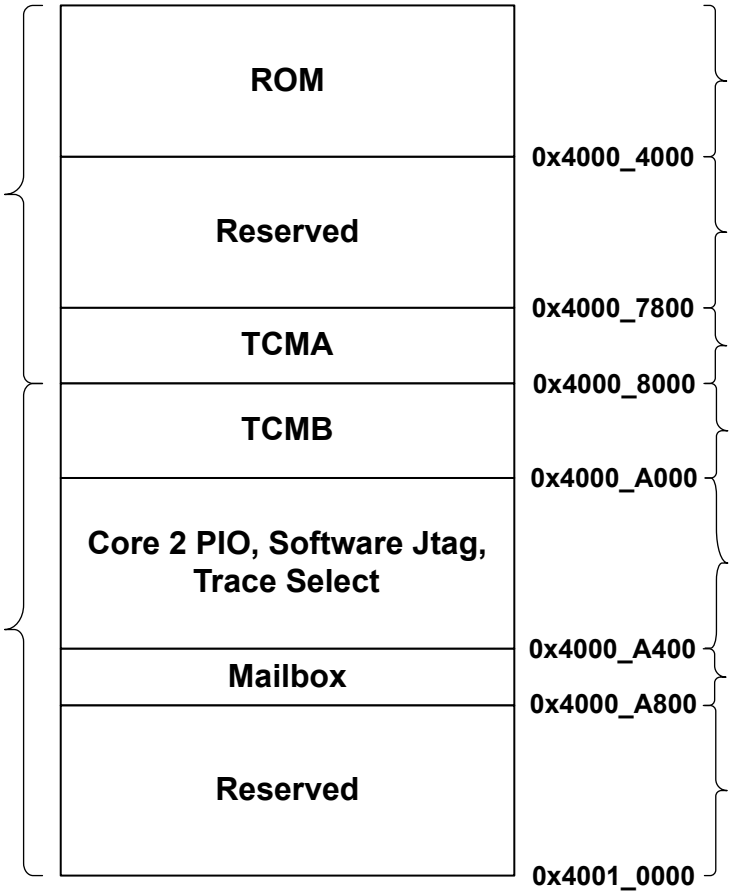

Figure 3-2 Core 2 address space

4. Interrupt List

4.1. Core 0/1 Interrupts

Both Core 0 and Core 1 of the BE-U1000 microcontroller include a *Core-Local Interrupt Controller* (**CLIC**) that conforms to the RISC-V specification:

- Core 0 CLIC supports 64 interrupt lines (excluding 16 interrupts with Interrupt IDs from 0 to 15 that are reserved for legacy interrupts).
- Core 1 CLIC supports 64 interrupt lines (excluding 16 interrupts with Interrupt IDs from 0 to 15 that are reserved for legacy interrupts), that are fully identical to the Core 0 CLIC interrupts.

Both Core 0 and Core 1 can process maskable interrupts (int_local) and non-maskable interrupts (int_nmi). Maskable interrupt processing is carried out by the CLIC and must guarantee saving the BE-U1000 entire context. The exit from the interrupts processing is always carried out to the entry point. Non-maskable interrupts are handled by the command execution subsystem of the corresponding core and require the BE-U1000 rebooting after their processing.

The table below lists the int_local interrupts distribution for Core 0 and Core 1. These interrupts (including interrupts with Interrupt ID from 0 to 15) are processed by the Core 0 and Core 1 CLICs.

Interrupts with Interrupt IDs 71,72, and 73 come directly from the PA, PB, and PC pins, respectively. These interrupts can be used for working in real-time applications. The source of each interrupt is selected via setting the corresponding bits of the `INTCR0` CRU register.

Table 4-1 int_local interrupts distribution for Core 0 and Core 1

Interrupt ID	Source	Description
0	Core	`usip` User software Interrupt
1-2	Reserved	
3	Core	`msip` Machine software interrupt
4	Core	`utip` User timer interrupt
5-6	Reserved	
7	Core	`mtip` Machine timer interrupt
8-11	Reserved	
12	Core	`csip` CLIC software interrupt
13-15	Reserved	
16	QSPI_0	`qspi_0_irq` QSPI_0 combined interrupt
17	UART_0	`uart_0_irq` UART_0 combined interrupt

Table 4-1 int_local interrupts distribution for Core 0 and Core 1 (continued)

Interrupt ID	Source	Description
18	UART_1	`uart_1_irq` UART_1 combined interrupt
19	UART_2	`uart_2_irq` UART_2 combined interrupt
20	I2C_0	`i2c_0_irq` I2C_0 combined interrupt
21	I2C_1	`i2c_1_irq` I2C_1 combined interrupt
22	I2S_0	`i2s_0_irq` I2S_0 combined interrupt
23-26	TIM_0	`tim_0_irq` TIM_0 combined interrupt
27	WDT_0	`wdt_0_irq` WDT_0 Active-High interrupt
28	GPIO_0	`gpio_0_irq` GPIO_0 combined interrupt
29	PWMG_0	`pwmg_0_tim_irq` PWMG_0 interrupt
30	SPI_0	`spi_0_irq` SPI_0 combined interrupt
31	SPI_1	`spi_1_irq` SPI_1 combined interrupt
32	CANFD_0	`canfd_0_irq` CANFD_0 single level-sensitive interrupt
33	QSPI_1	`qspi_1_irq` QSPI_1 combined interrupt
34	UART_3	`uart_3_irq` UART_3 combined interrupt
35	UART_4	`uart_4_irq` UART_4 combined interrupt
36	UART_5	`uart_5_irq` UART_5 combined interrupt
37	I2C_2	`i2c_2_irq` I2C_2 combined interrupt
38	I2C_3	`i2c_3_irq` I2C_3 combined interrupt
39	I2S_1	`i2s_1_irq` I2S_1 combined interrupt
40-43	TIM_1	`tim_1_irq`

Table 4-1 int_local interrupts distribution for Core 0 and Core 1 (continued)

Interrupt ID	Source	Description
		TIM_1 combined interrupt
44	Reserved	
45	GPIO_1	`gpio_1_irq` GPIO_1 combined interrupt
46	PWMG_1	`pwmg_1_tim_irq` PWMG_1 interrupt
47	SPI_2	`spi_2_irq` SPI_2 combined interrupt
48	SPI_3	`spi_3_irq` SPI_3 combined interrupt
49	CANFD_1	`canfd_1_irq` CANFD_1 single level-sensitive interrupt
50	ADC_0	`adc_0_eoc_irq` ADC_0 *End of Conversion* (**EOC**) interrupt
51	ADC_1	`adc_1_eoc_irq` ADC_1 *End of Conversion* (**EOC**) interrupt
52	ADC_2	`adc_2_eoc_irq` ADC_2 *End of Conversion* (**EOC**) interrupt
53	PWMA_0	`pwma_0_tim_irq` PWMA_0 Timer interrupt
54		`pwma_0_qe_irq` PWMA_0 Quadrature Encoder interrupt
55	PWMA_1	`pwma_1_tim_irq` PWMA_1 Timer interrupt
56		`pwma_1_qe_irq` PWMA_1 Quadrature Encoder interrupt
57	PWMA_2	`pwma_2_tim_irq` PWMA_2 Timer interrupt
58		`pwma_2_qe_irq` PWMA_2 Quadrature Encoder interrupt
59	PWMA_3	`pwma_3_tim_irq` PWMA_3 Timer interrupt
60		`pwma_3_qe_irq` PWMA_3 Quadrature Encoder interrupt
61	GPIO_2	`gpio_2_irq` GPIO_2 combined interrupt
62	UART_6	`uart_6_irq` UART_6 combined interrupt

Table 4-1 int_local interrupts distribution for Core 0 and Core 1 (continued)

Interrupt ID	Source	Description
63	UART_7	`uart_7_irq` UART_7 combined interrupt
64	USB	`mc_nint` USB combined interrupt
65	DMA_0	`dma_0_irq` DMA_0 combined interrupt
66	DMA_1	`dma_1_irq` DMA_1 combined interrupt
67	USB DMA	`dma_nint` USB DMA interrupt
68	PIO	`pio_irq` PIO interrupt
69	Mailbox	`mailbox_mb_0_irq` Indicates that the MB0 is fulled by the Core 2
70		`mailbox_mb_1_irq` Indicates that the MB1 is fulled by the Core 2
*71	Port A	`pa_irq` Interrupt from the Port A
*72	Port B	`pb_irq` Interrupt from the Port B
*73	Port C	`pc_irq` Interrupt from the Port C
74	CRU `ref_clk` Monitoring block	`alarm_irq` `alarm` signal from the CRU `ref_clk` Monitoring block
75	Core 0 Bus Error Unit	`core0_bus_err_intr` Core 0 *Bus Error Unit* (**BEU**) interrupt
76	Core 1 Bus Error Unit	`core1_bus_err_intr` Core 1 *Bus Error Unit* (**BEU**) interrupt
77	CRU PLL	`pll_nlock` Indicates that the PLL is unlocked (active high)
78-79	Reserved	

In the table above an asterisk symbol (*) indicates the following:

- Port A pin that acts as an interrupt source is selected via the `INTCR0`.`PADAINTSEL0` bit of the CRU.
- Port B pin that acts as an interrupt source is selected via the `INTCR0`.`PADBINTSEL0` bit of the CRU.
- Port C pin that acts as an interrupt source is selected via the `INTCR0`.`PADCINTSEL0` bit of the CRU.

Interrupts with Interrupt IDs 28 and 45 from the GPIO_0 and GPIO_1 can be generated inside these blocks with or without the glitch filtering option. For more information, see the Debounce

Operation. The filter circuit operates using the `dbnclk` that is obtained from the `sys_clk` divided by 8.

The table below shows the int_nmi interrupt for Core 1. This interrupt is processed by the Core 1 command execution subsystem. The address of the NMI handler is taken from the `NMITVEC` register, and the NMI Interrupt ID can be read from the `MCAUSE` register.

Table 4-2 int_nmi interrupt for Core 1

NMI Interrupt ID	Source	Description
0	WDT_1	`wdt_1_irq` WDT_1 active high interrupt

4.2. Core 2 Interrupts

Core 2 of the BE-U1000 microcontroller has 16 interrupt lines that are processed by the command execution subsystem of the core. The address of the Interrupt handler is taken from the `MTVEC` register, and the Interrupt ID can be read from the `MCAUSE` register.

The table below lists the int_local interrupts distribution for Core 2.

Table 4-3 int_local interrupts distribution for Core 2

Interrupt ID	Source	Description
0	Mailbox	`mailbox_mb_0_irq` Indicates that the MB0 is fulled by the Core 0 and Core 1
1		`mailbox_mb_1_irq` Indicates that the MB1 is fulled by the Core 0 and Core 1
*2	Port A	`pa_irq` Interrupt from the Port A
*3	Port B	`pb_irq` Interrupt from the Port B
*4	Port C	`pc_irq` Interrupt from the Port C
5	USB	`mc_nint` USB combined interrupt
6	USB DMA	`dma_nint` USB DMA interrupt
7-15	Reserved	

In the table above an asterisk symbol (*) indicates the following:

- Port A pin that acts as an interrupt source is selected via the `INTCR0`.`PADAINTSEL2` bit of the CRU.
- Port B pin that acts as an interrupt source is selected via the `INTCR0`.`PADBINTSEL2` bit of the CRU.
- Port C pin that acts as an interrupt source is selected via the `INTCR0`.`PADCINTSEL2` bit of the CRU.

5. Control Registers Unit

The *Control Registers Unit* (**CRU**) is used to control the following system resources of the BE-U1000:

- Clock source selection
- Clock gating and reset
- PLL programming
- Priority of the devices that are connected to the Main bus matrix
- I/O pins configuration
- System status information

The CRU is located in the Periphery 2 region of the Core 0/1 Address Space and designed as a set of software-accessible registers.

5.1. CRU Functional Description

CRU contains the following blocks (see the diagram below):

- Reset Sources
- Clock Generation
- Control Registers

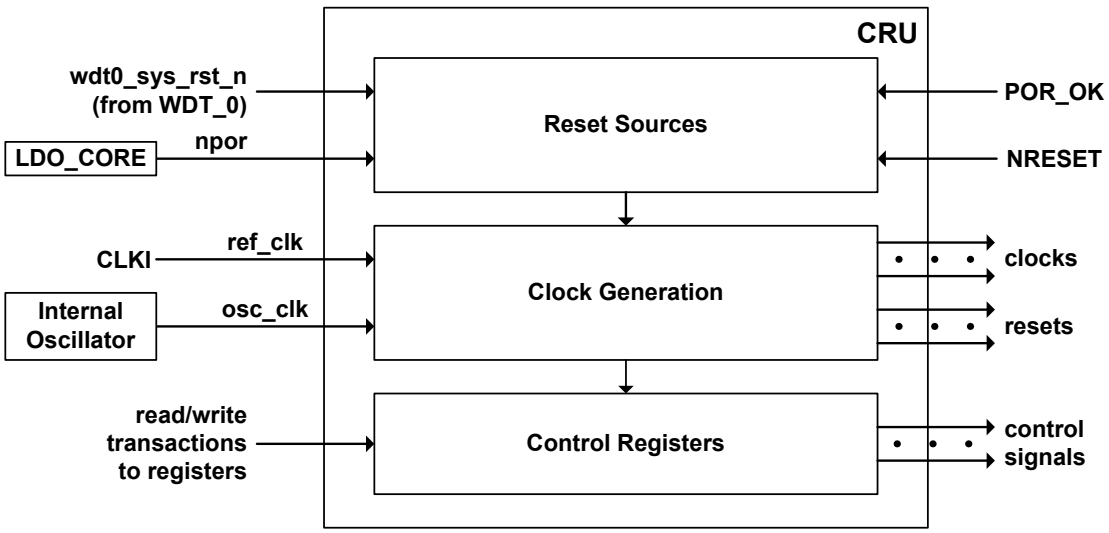

Figure 5-1 CRU block diagram

5.1.1. Clock Generation

This section covers the following topics:

- Clock Management
- Clock Domains

5.1.1.1. Clock Management

The CRU contains Clock Generation block that is used to generate the clocks of the required frequency for the BE-U1000 system domains. The CRU also contains the resync blocks to generate the reset signals of the BE-U1000 system domains, based on the `rst_n` signal from the Reset Sources block.

The Clock Generation block contains the following components (see the block diagram below):

- Clock Sources
- *Phase-Locked Loop* (**PLL**)
- Dividers
- Output Clocks

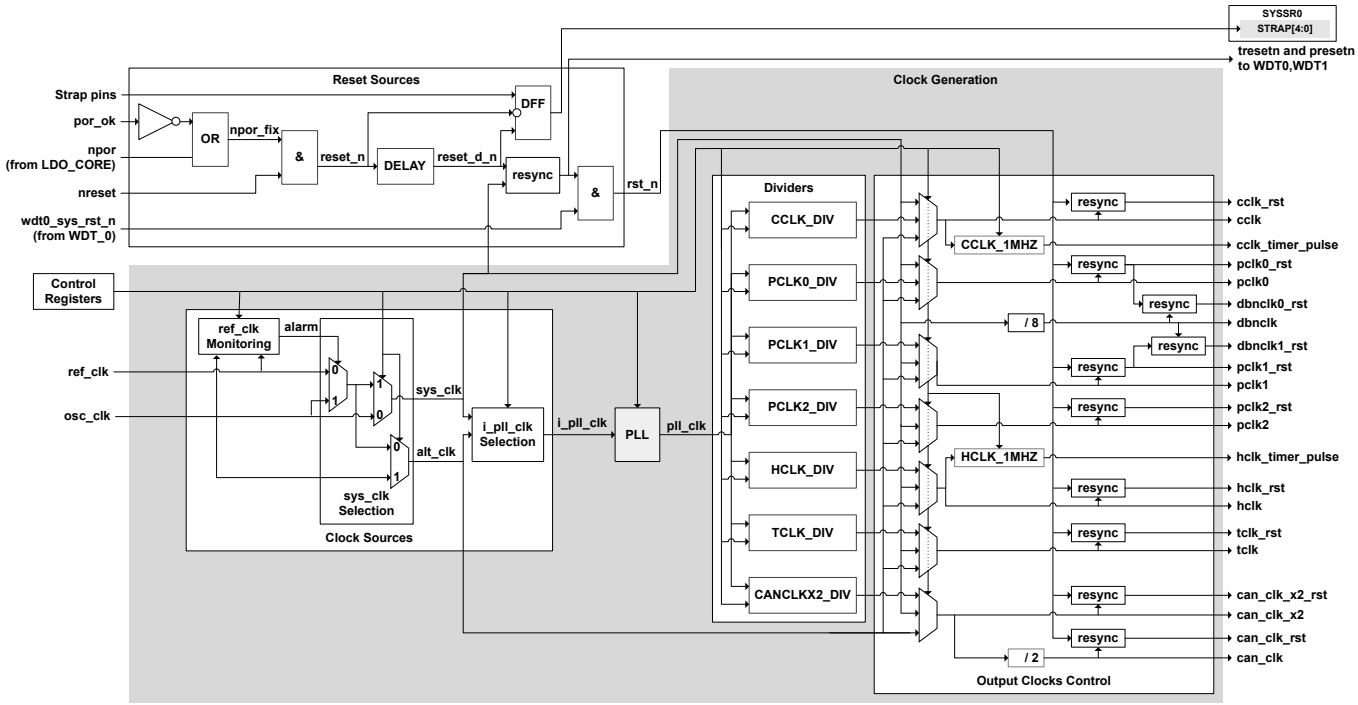

Figure 5-2 Clock Generation block diagram

The clock sources and output clocks are listed in the tables below.

Table 5-1 Clock Sources

Clock Name	Frequency, MHz	Description
ref_clk	25	Clock from the CLKI pin
osc_clk	12.20-32.48	Clock from the Internal Oscillator

Table 5-2 Output Clocks

Clock Name	Maximum Frequency, MHz	Description
*cclk	200	Clock for the cclk Clock Domain
*hclk	100	Clock for the hclk Clock Domain
pclk0	100	Clock for the pclk_0 Clock Domain
pclk1	100	Clock for the pclk_1 Clock Domain
pclk2	100	Clock for the pclk_2 Clock Domain
tclk	100	Clock for the tclk Clock Domain

Table 5-2 Output Clocks (continued)

Clock Name	Maximum Frequency, MHz	Description
`can_clk_x2`	200	Clock for the `can_clk_x2` Clock Domain
`*can_clk`	100	Clock for the `can_clk` Clock Domain
`*dbnclk`	4	Clock signal of the GPIO filtering circuit

In the table above an asterisk symbol (*) indicates the following:

- `cclk` is also used for the `cclk_timer_pulse` signal generation.
- `hclk` is also used for the `hclk_timer_pulse` signal generation.
- `can_clk` is obtained from the `can_clk_x2` divided by 2.
- `dbnclk` is obtained from the `sys_clk` divided by 8.

For information about the BE-U1000 system domains and devices they include, see Clock Domains.

5.1.1.1.1. Clock Sources

Clock Sources block is used to select which of the two input signals (`ref_clk` or `osc_clk`) will be used as `sys_clk`, `alt_clk` and `i_pll_clk` outputs. It also contains the `ref_clk` Monitoring block to track the `ref_clk` presence. The structure of the Clock Sources block is shown in the figure below.

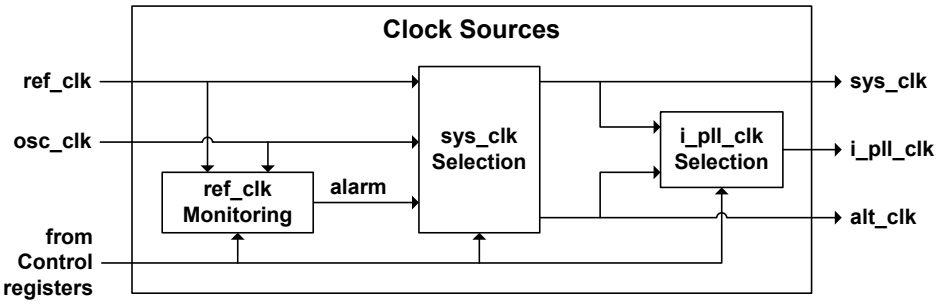

Figure 5-3 Clock Sources block diagram

`sys_clk` Selection

The `sys_clk` Selection block allows you to select the source of the `sys_clk` and `alt_clk` output signals, where the source is the `ref_clk` and `osc_clk` input signals. This operation can be performed while the BE-U1000 is running, as shown in the following figure.

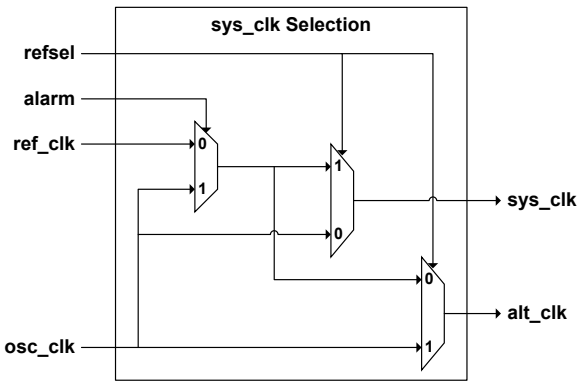

Figure 5-4 `sys_clk` Selection block

As shown in the figure above, the `sys_clk` Selection block consists of several multiplexers that are controlled through the `alarm` and `refsel` signals, where:

- `alarm` signal is generated by the `ref_clk` Monitoring block.

> The `ref_clk` Monitoring block is used to track the `ref_clk` signal presence. To allow the block to work, you must set the `ALARM_EN` bit of the `SYSCR0` register to 0x1. The `alarm` signal is generated when the `SYSCR0`.`ALARM_LVL` value is reached.

- `refsel` signal is the result of the appropriate bits settings of the `SYSSR0` and `CLKCR0` registers as the table below shows.

Table 5-3 `refsel` signal values

`SYSSR0`.STRAP[3] value	`CLKCR0`.BOOTCLKSEL value	`refsel` value
0x0	0x0	1
0x0	0x1	1
0x1	0x0	0
0x1	0x1	1

Depending on the values of the `alarm` signal and `refsel` signal, the `sys_clk` and `alt_clk` outputs will correspond to the `ref_clk` or `osc_clk` signal, as the following table shows.

Table 5-4 Influence of the `alarm` and `refsel` signals to the `sys_clk` and `alt_clk` outputs

`alarm` value	`refsel` value	`sys_clk` corresponds to:	`alt_clk` corresponds to:
0	0	`osc_clk`	`ref_clk`
0	1	`ref_clk`	`osc_clk`
1	0	`osc_clk`	`osc_clk`
1	1	`osc_clk`	`osc_clk`

- When the `alarm` signal is set to 1, the PLL is switched to bypass mode.

- To reset the `alarm` signal, set the `ALARM_EN` bit of the `SYSCR0` register to 0x0. This setting also stops the work of the `ref_clk` Monitoring block.

`i_pll_clk` Selection

`i_pll_clk` Selection block is used to select the source of the `i_pll_clk` signal. Depending on the `I_PLL_CLK_SEL` bit value, the `i_pll_clk` will correspond either to the `sys_clk` signal or to the `alt_clk` signal.

5.1.1.1.2. PLL

The *Phase-Locked Loop* (**PLL**) is designed to divide or multiply the input clock signal using three dividers:

- Reference divider (`NR=PLLSET.CLKR[3:0] + 1`)
- Feedback divider (`NF=PLLSET.CLKF[9:4] + 1`)
- Output divider (`OD=PLLSET.CLKOD[13:10] + 1`)

The PLL also contains loop Bandwidth adjustment (`NB=PLLSET.BWADJ[19:14] + 1`), which must be set equal to the total division in the feedback path.

The PLL accepts a wide range of input frequencies and can produce a wide range of output frequencies.

The PLL block can generate output clock in the following frequency ranges:

- Divided reference frequency range is 10.2 MHz – 1300 MHz (allowed frequency range at the *Phase-Frequency Detector* (**PFD**) input).

  The divided reference frequency (`div_i_pll_clk`) is defined as: `div_i_pll_clk = i_pll_clk/NR`.

- Output frequency range is 16.2 MHz – 1300 MHz.
- Allowed VCO range is 260 MHz – 1300 MHz.

5.1.1.1.2.1. PLL Output Frequency

The following figure shows the PLL block diagram.

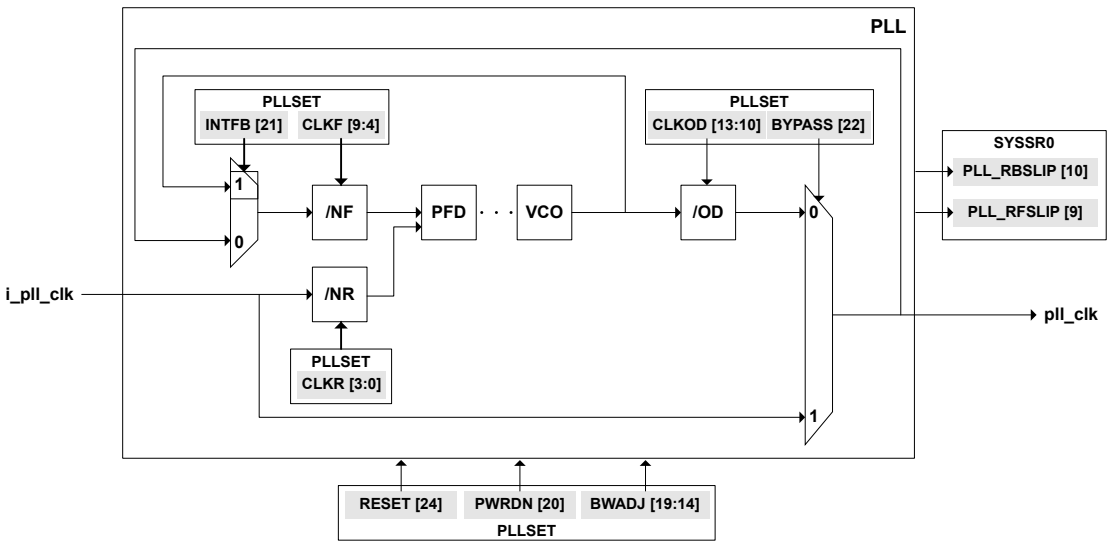

Figure 5-5 PLL block diagram

Locked Mode

The relationship between the PLL output frequency (`pll_clk`) and the reference frequency (`i_pll_clk`) in normal locked operation depends on the divider inputs and `PLLSET.INTFB` bit value.

For `PLLSET.INTFB`=0x1 the output frequency `pll_clk` and *Voltage Controlled Oscillator* (**VCO**) frequency F_{VCO} are related to the reference frequency `i_pll_clk` by:

$$pll\_clk = \frac{i\_pll\_clk*NF}{NR*OD}$$

$$F_{\mathrm{VCO}} = \frac{i\_pll\_clk*NF}{NR}$$

You must also set the `NB` value as follows: `NB`=`NF` if `PLLSET.INTFB`=0x1

The recommended PLL settings for the `i_pll_clk` frequency of 25 MHz are shown in the table below.

Table 5-5 Recommended PLL settings for the `i_pll_clk` frequency of 25 MHz

NR	NF	OD	NB	pll_clk, MHz
1	32	16	32	50
1	52	13	52	100
1	48	8	48	150
1	48	6	48	200

Bypass Mode

The PLL has a bypass mode (`PLLSET`.`BYPASS`=0x1) in which the reference frequency `i_pll_clk` is directly bypassed to the PLL output (`pll_clk` = `i_pll_clk`).

The `PLLSET`.`BYPASS` controls the PLL's output multiplexer that selects either the reference clock or the clock from the PLL's output divider. If the PLL is locked and `BYPASS` is asserted, then the overall power dissipation will be similar to what one would see under functional mode operation.

However, if both power-down and bypass are asserted, then the overall power dissipation will be due mainly to a few buffers toggling in the bypass path. The power dissipation in this case will be very small.

5.1.1.1.2.2. PLL Stability Tracking

There are two signals at the PLL output (`FBSLIP` and `RFSLIP`) that can signal whether the PLL is operating in steady state or not. The `RFSLIP` signal is set to one or more clock cycles of the VCO output frequency divided by `PLLSET`.`CLKF`+1 and it signals that the VCO frequency is too high. The `FBSLIP` signal is set to one or more cycles of `i_pll_clk` divided by `PLLSET`.`CLKR`+1 and it signals that the VCO frequency is too low. By triggering these signals, it is possible to generate a signal indicating that the PLL has exited the steady state.

The PLL stability tracking is shown in the figure below.

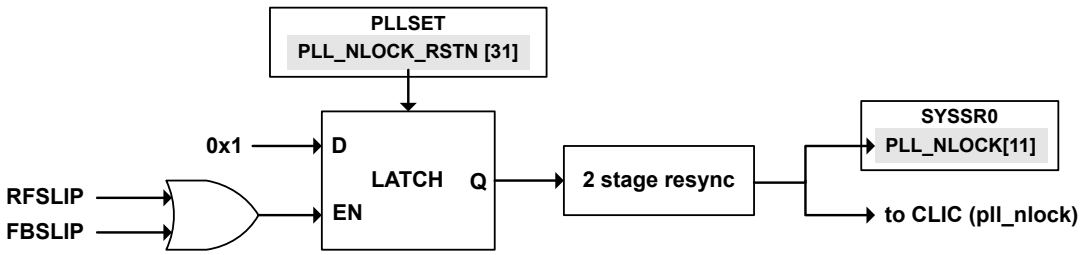

Figure 5-6 PLL stability tracking

The `PLL_NLOCK` flag is enabled to read at the `SYSSR0` and it is also the source of the `pll_nlock` interrupt to Core 0 CLIC and Core 1 CLIC.

5.1.1.1.3. Dividers

The Dividers block is used to create the clock signals of the required frequency, based on the `clk_pll` signal. The block contains seven programmable dividers that are controlled via the corresponding bit fields of the `CLKCR0` and `CLKCR1` registers as shown in the figure below.

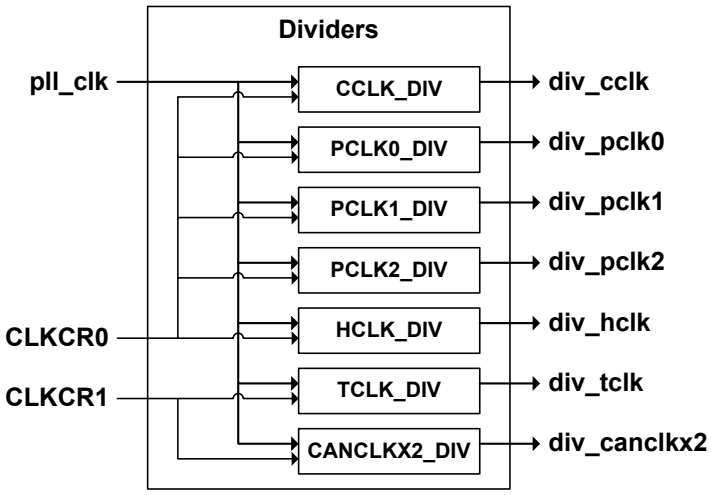

Figure 5-7 Dividers block diagram

5.1.1.1.4. Output Clocks Control

The Output Clocks Control block allows you to select the source of the BE-U1000 system domains signals, where the source is the `ref_clk`, `osc_clk`, and `div_*` input signals. The block consists of several multiplexors that are controlled through the appropriate bit fields of the `CLKSEL` register as the figure below shows.

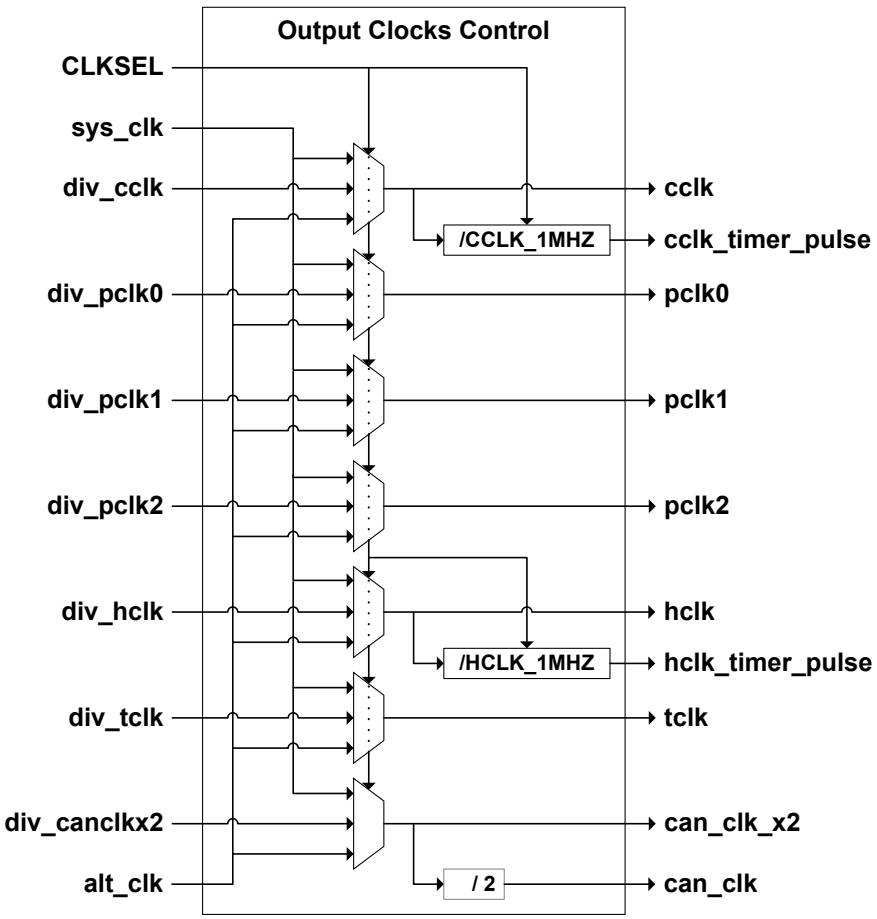

Figure 5-8 Output Clocks Control block diagram

5.1.1.2. Clock Domains

This section covers the following topics:

- `cclk` Clock Domain
- `pclk_0` Clock Domain
- `pclk_1` Clock Domain
- `pclk_2` Clock Domain
- `hclk` Clock Domain
- `can_clk` and `can_clk_x2` Clock Domains
- `tclk` Clock Domain

5.1.1.2.1. `cclk` Clock Domain

The following figure shows the `cclk` clock domain in the BE-U1000 microcontroller.

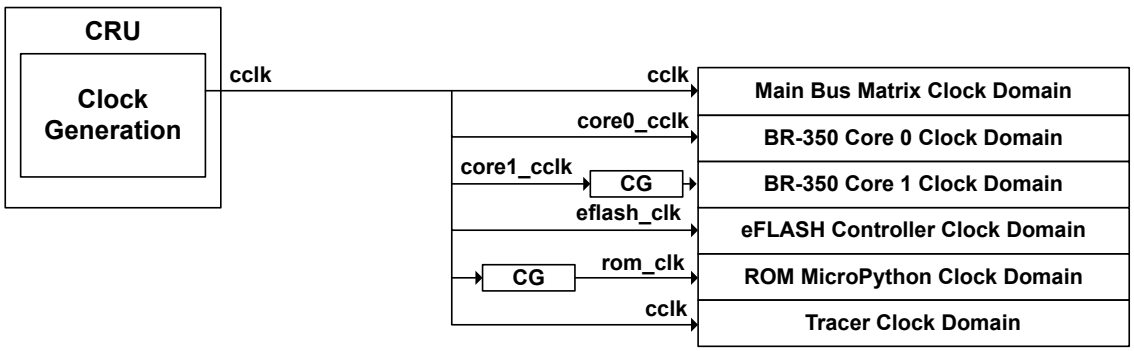

Figure 5-9 `cclk` Clock Domain

As shown in the figure above, the `rom_clk` and `core1_cclk` are gated.

The `rom_clk` signal is obtained from the `cclk` by gating every second pulse.

5.1.1.2.2. `pclk_0` Clock Domain

The following figure shows the `pclk_0` clock domain in the BE-U1000 microcontroller.

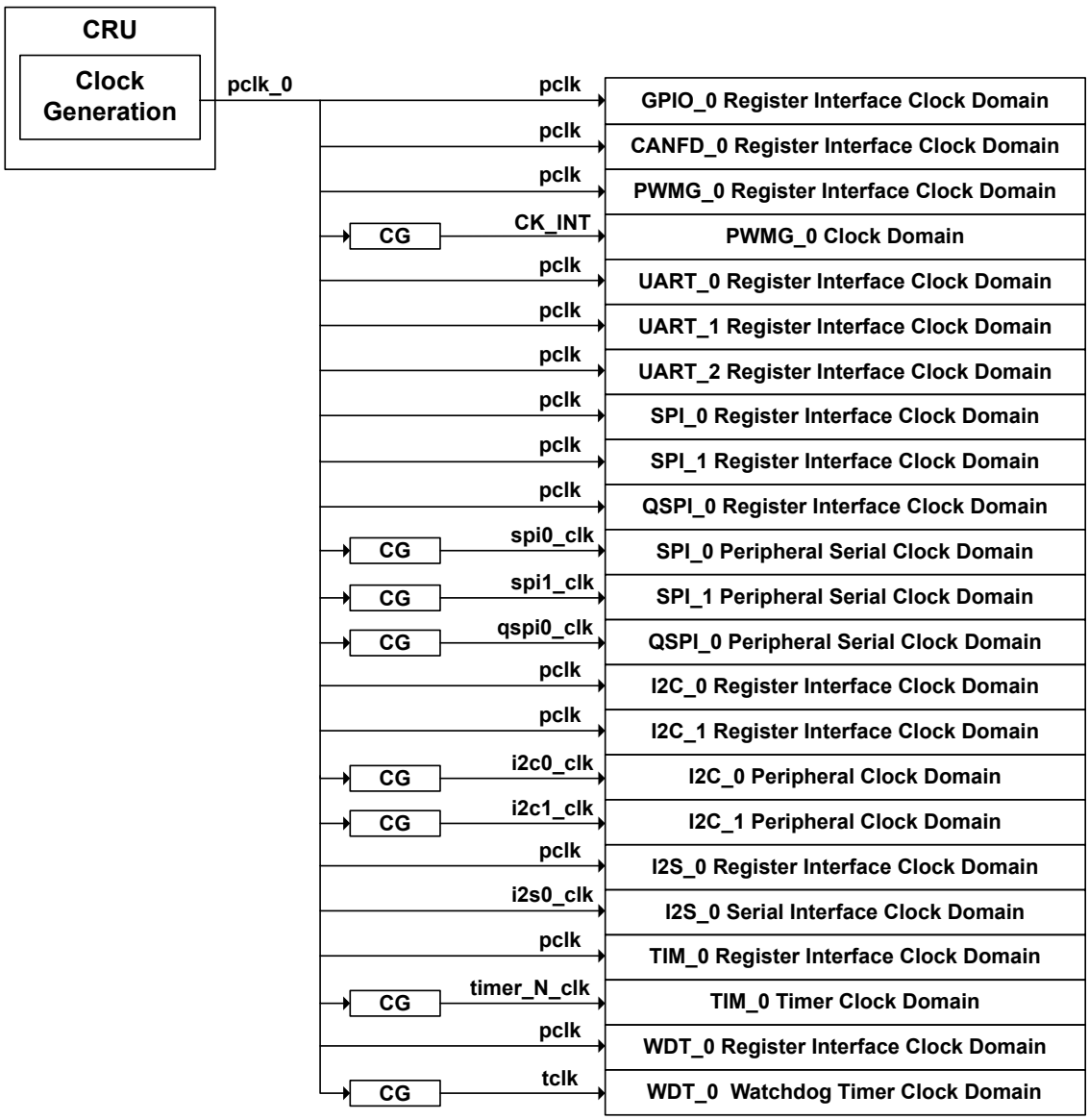

Figure 5-10 `pclk_0` Clock Domain

As shown in the figure above, some clocks of the `pclk_0` clock domain are gated.

5.1.1.2.3. `pclk_1` Clock Domain

The following figure shows the `pclk_1` clock domain in the BE-U1000 microcontroller.

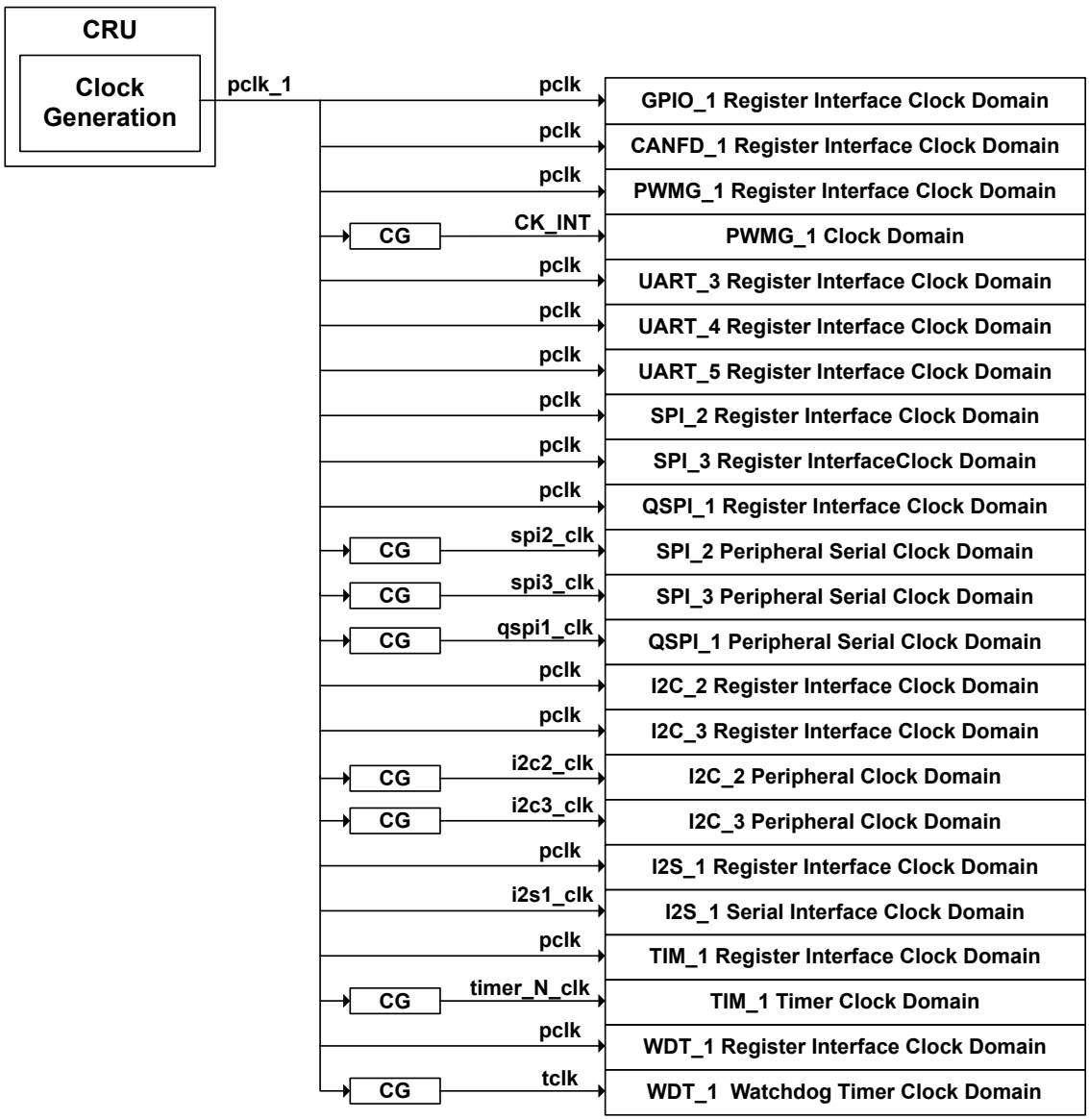

Figure 5-11 `pclk_1` Clock Domain

As shown in the figure above, some clocks of the `pclk_1` clock domain are gated.

5.1.1.2.4. `pclk_2` Clock Domain

The following figure shows the `pclk_2` clock domain in the BE-U1000 microcontroller.

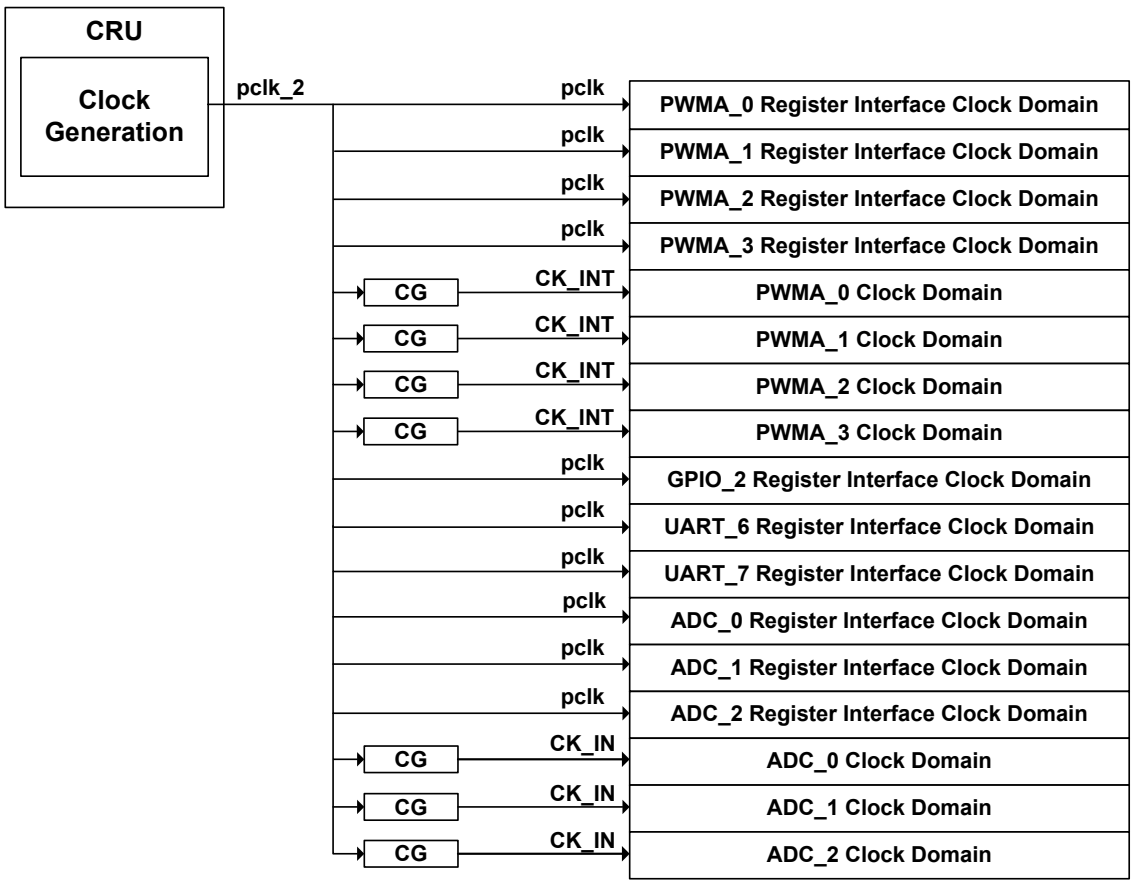

Figure 5-12 `pclk_2` Clock Domain

As shown in the figure above, some clocks of the `pclk_2` clock domain are gated.

5.1.1.2.5. `hclk` Clock Domain

The following figure shows the `hclk` clock domain in the BE-U1000 microcontroller.

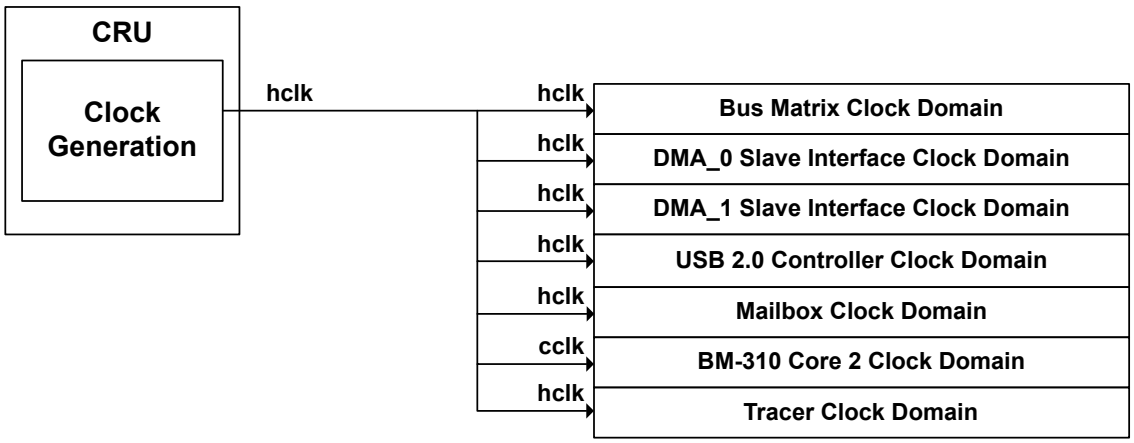

Figure 5-13 `hclk` Clock Domain

5.1.1.2.6. `can_clk` and `can_clk_x2` Clock Domains

The following figure shows the `can_clk` anf `can_clk_x2` clock domains in the BE-U1000 microcontroller.

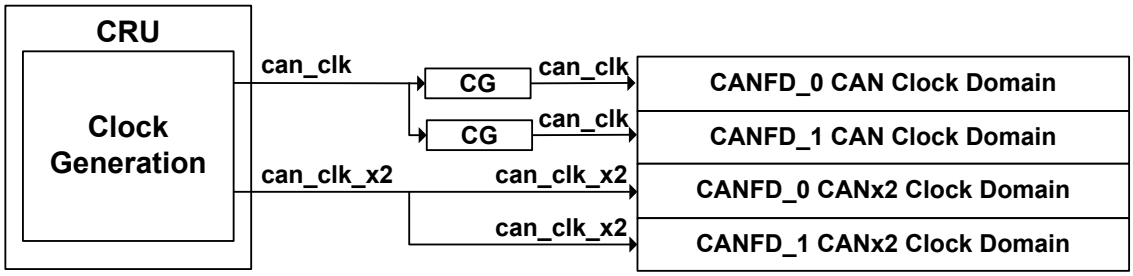

Figure 5-14 `can_clk` and `can_clk_x2` Clock Domains

As shown in the figure above, some clocks of the `can_clk` clock domain are gated.

The `can_clk` is obtained from the divided by 2 `can_clk_x2`.

5.1.1.2.7. `tclk` Clock Domain

The following figure shows the `tclk` clock domain in the BE-U1000 microcontroller.

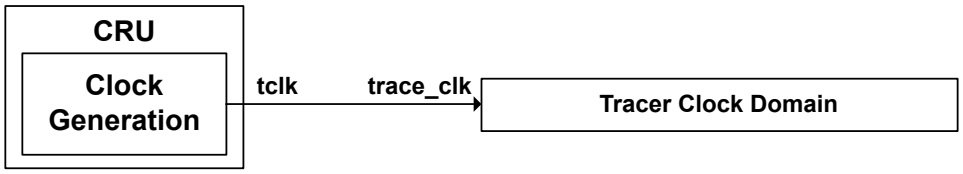

Figure 5-15 `tclk` Clock Domain

5.1.2. Reset Sources

The Reset Sources block provide reset signals for the Clock Generation, WDT_0 and WDT_1.

The Reset Sources block consists of several logical AND elements and resync block, which are used to generate output reset signals from three input reset signals, as shown in the figure below.

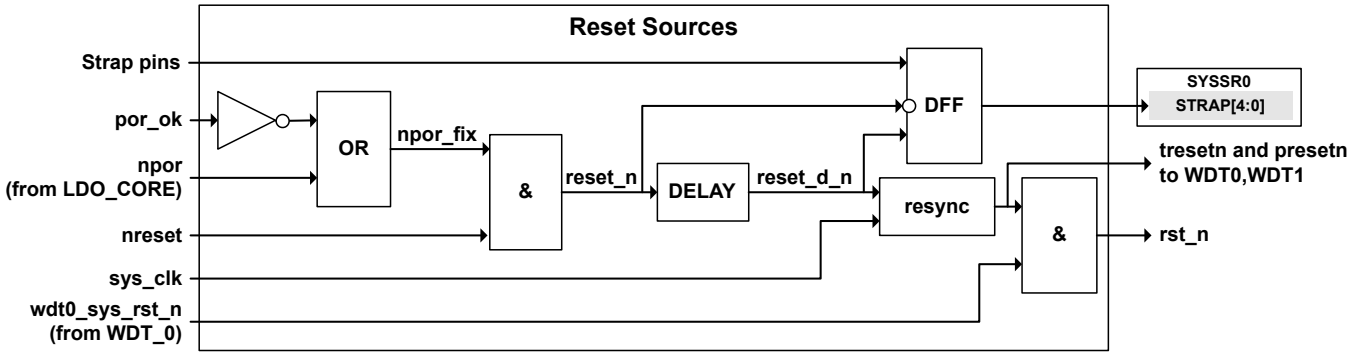

Figure 5-16 Reset Sources block diagram

For more onformation about Strap pins refer to the **3.9.3 Strap pins** section of the ***BE-U1000 Datasheet***

Output reset signals are resynchronized with the `sys_clk`. For more information about the `sys_clk` signal source, see Clock Sources.

The tables below list the input and output reset signals of the Reset Sources block.

Table 5-6 Reset Sources block input reset signals

Signal	Active Level	Description
`npor`	Low	The reset signal from LDO. The signal is active when power supply is applied. The signal can be blocked by an external POR_OK pin. Only their derivative is used inside the chip `npor_fix = npor | !por_ok`.
`*reset_n`	Low	The reset signal for triggers that fix the value of the strap pins.
`reset_d_n`	Low	The `reset_n` signal delayed on the delay line. It is the clock for triggers that fix the value of the strap pins.
`por_ok`	Low	This signal enables the use of the `npor` signal.
`nreset`	Low	The reset signal from the external NRESET pin. For more information, see chapter **5 Pinout and pin description** of the ***BE-U1000 Datasheet***.
`wdt0_sys_rst_n`	Low	The reset signal from the WDT_0.

In the table above an asterisk symbol (*) indicates that the `reset_n` signal is the logical AND of the `npor_fix` and `nreset` signals.

Table 5-7 Reset Sources block output reset signals

Signal	Active Level	Description
`*tresetn`	Low	This signal is used to reset the WDT_0 and WDT_1 counter.
`*presetn`	Low	This signal is used to reset the WDT_0 and WDT_1 register interface.
`*rst_n`	Low	The reset signal for the Clock Generation block. This signal is used to generate the BE-U1000 system domains reset signals. For more information about the clock generation, see Clock Management.

In the table above an asterisk symbol (*) indicates the following:

- `tresetn` signal is the logical AND of the `npor_fix` and `nreset` signals.
- `presetn` signal is the logical AND of the `npor_fix` and `nreset` signals.
- `rst_n` signal is the logical AND of the `npor_fix`, `nreset` and `wdt0_sys_rst_n` signals.

5.1.3. I/O Pins Configuration

The BE-U1000 contains 48 independent and configurable I/O pins that form the following ports (each port has 16 I/O pins):

- Port A (PA[15:0])
- Port B (PB[15:0])
- Port C (PC[15:0])

The following I/O pins options are controllable through the CRU registers:

- Alternate Functions
- Input Enable
- Drive Strength
- Pull-Up and Pull-Down Resistors

5.1.3.1. Alternate Functions

The BE-U1000 I/O pins are connected to onboard peripherals through the multiplexors that allow to select one of six functions for each I/O pin. Each I/O pin multiplexer is controllable through the appropriate bits of the `IOAFCRx` registers, where `x` is the register number. The figure below shows the PA[0] I/O pin multiplexer as an example.

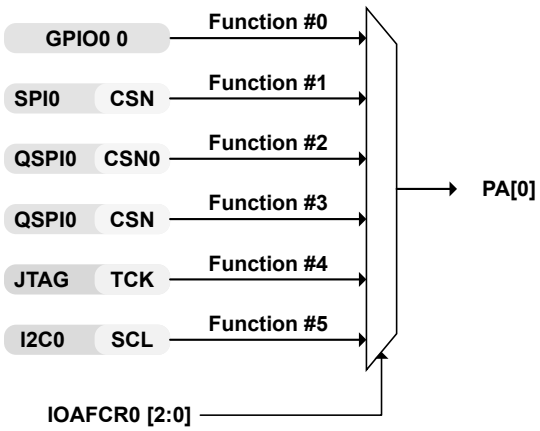

Figure 5-17 PA[0] I/O pin multiplexer

The figures below illustrate the functions that are available for I/O pins of each port (PA, PB и PC) and also the dependence between the settings of the `IOAFCRx` registers and I/O pin functions for each port.

Register field	Func. #0 (0x0)	Func. #1 (0x1)	Func. #2 (0x2)	Func. #3 (0x3)	Func. #4 (0x4)	Func. #5 (0x5)	I/O pin
IOAFCR0[2:0]	GPIO0 0	SPI0_CSN	QSPI0_CSN0	QSPI0_CSN	JTAG_TCK	I2C0_SCL	PA[0]
IOAFCR0[6:4]	GPIO0 1	SPI0_SCK	QSPI0_SCK	QSPI0_SCK	JTAG_TMS	I2C0_SDA	PA[1]
IOAFCR0[10:8]	GPIO0 2	SPI0_MISO	QSPI0_MOSI	QSPI0_IO0	JTAG_TDO	UART0_TX	PA[2]
IOAFCR0[14:12]	GPIO0 3	SPI0_MOSI	QSPI0_MISO	QSPI0_IO1	JTAG_TDI	UART0_RX	PA[3]
IOAFCR0[18:16]	GPIO0 4	I2C0_SCL	QSPI0_CSN1	QSPI0_IO2	JTAG_NTRST	UART1_TX	PA[4]
IOAFCR0[22:20]	GPIO0 5	I2C0_SDA	QSPI0_CSN2	QSPI0_IO3	PWMA0_QA	UART1_RX	PA[5]
IOAFCR0[26:24]	GPIO0 6	UART0_TX	CAN0_TX	I2C0_SCL	PWMA0_QB	PWMG0_CH0	PA[6]
IOAFCR0[30:28]	GPIO0 7	UART0_RX	CAN0_RX	I2C0_SDA	PWMA0_INDX	PWMA0_ETR	PA[7]
IOAFCR1[2:0]	GPIO0 8	SPI1_CSN	I2C0_SCL	UART0_TX	PWMA0_CH0P	PWMA0_CH0P	PA[8]
IOAFCR1[6:4]	GPIO0 9	SPI1_SCK	I2C0_SDA	UART0_RX	PWMA0_CH0N	PWMA0_CH1P	PA[9]
IOAFCR1[10:8]	GPIO0 10	SPI1_MOSI	UART1_TX	I2S0_SDO	PWMA0_CH1P	PWMA0_CH2P	PA[10]
IOAFCR1[14:12]	GPIO0 11	SPI1_MISO	UART1_RX	I2S0_SDI	PWMA0_CH1N	PWMA0_CH3P	PA[11]
IOAFCR1[18:16]	GPIO0 12	I2C1_SCL	UART2_TX	I2S0_WS	PWMA0_CH2P	TIM0_PWM0	PA[12]
IOAFCR1[22:20]	GPIO0 13	I2C1_SDA	UART2_RX	I2S0_SCLK	PWMA0_CH2N	TIM0_PWM1	PA[13]
IOAFCR1[26:24]	GPIO0 14	UART1_TX	I2C1_SCL	CAN0_TX	PWMA0_BRK	TIM0_PWM2	PA[14]
IOAFCR1[30:28]	GPIO0 15	UART1_RX	I2C1_SDA	CAN0_RX	PWMG0_CH0	TIM0_PWM3	PA[15]

Figure 5-18 I/O pins functions and control for Port A

Register field	Func. #0 (0x0)	Func. #1 (0x1)	Func. #2 (0x2)	Func. #3 (0x3)	Func. #4 (0x4)	Func. #5 (0x5)	I/O pin
IOAFCR2[2:0]	GPIO1 0	SPI2_CSN	QSPI1_CSN0	QSPI1_CSN	PIB_D[0]	I2C2_SCL	PB[0]
IOAFCR2[6:4]	GPIO1 1	SPI2_SCK	QSPI1_SCK	QSPI1_SCK	PIB_D[1]	I2C2_SDA	PB[1]
IOAFCR2[10:8]	GPIO1 2	SPI2_MISO	QSPI1_MOSI	QSPI1_IO0	PIB_D[2]	UART3_TX	PB[2]
IOAFCR2[14:12]	GPIO1 3	SPI2_MOSI	QSPI1_MISO	QSPI1_IO1	PIB_D[3]	UART3_RX	PB[3]
IOAFCR2[18:16]	GPIO1 4	I2C2_SCL	QSPI1_CSN1	QSPI1_IO2	PIB_CLK	UART4_TX	PB[4]
IOAFCR2[22:20]	GPIO1 5	I2C2_SDA	QSPI1_CSN2	QSPI1_IO3	PWMA1_QA	UART4_RX	PB[5]
IOAFCR2[26:24]	GPIO1 6	UART3_TX	CAN1_TX	I2C2_SCL	PWMA1_QB	PWMG1_CH0	PB[6]
IOAFCR2[30:28]	GPIO1 7	UART3_RX	CAN1_RX	I2C2_SDA	PWMA1_INDX	PWMA1_ETR	PB[7]
IOAFCR3[2:0]	GPIO1 8	SPI3_CSN	I2C2_SCL	UART3_TX	PWMA1_CH0P	PWMA1_CH0P	PB[8]
IOAFCR3[6:4]	GPIO1 9	SPI3_SCK	I2C2_SDA	UART3_RX	PWMA1_CH0N	PWMA1_CH1P	PB[9]
IOAFCR3[10:8]	GPIO1 10	SPI3_MOSI	UART4_TX	I2S1_SDO	PWMA1_CH1P	PWMA1_CH2P	PB[10]
IOAFCR3[14:12]	GPIO1 11	SPI3_MISO	UART4_RX	I2S1_SDI	PWMA1_CH1N	PWMA1_CH3P	PB[11]
IOAFCR3[18:16]	GPIO1 12	I2C3_SCL	UART5_TX	I2S1_WS	PWMA1_CH2P	TIM1_PWM0	PB[12]
IOAFCR3[22:20]	GPIO1 13	I2C3_SDA	UART5_RX	I2S1_SCLK	PWMA1_CH2N	TIM1_PWM1	PB[13]
IOAFCR3[26:24]	GPIO1 14	UART4_TX	I2C3_SCL	CAN1_TX	PWMA1_BRK	TIM1_PWM2	PB[14]
IOAFCR3[30:28]	GPIO1 15	UART4_RX	I2C3_SDA	CAN1_RX	PWMG1_CH0	TIM1_PWM3	PB[15]

Figure 5-19 I/O pins functions and control for Port B

Register field	Func. #0 (0x0)	Func. #1 (0x1)	Func. #2 (0x2)	Func. #3 (0x3)	Func. #4 (0x4)	Func. #5 (0x5)	I/O pin
IOAFCR4[2:0]	GPIO2 0	PIO [0]	PWMA2_CH0P	JTAG_TCK	PWMA3_INDX	PIB_D[0]	PC[0]
IOAFCR4[6:4]	GPIO2 1	PIO [1]	PWMA2_CH0N	JTAG_TMS	PWMA3_QA	PIB_D[1]	PC[1]
IOAFCR4[10:8]	GPIO2 2	PIO [2]	PWMA2_CH1P	JTAG_TDO	PWMA3_QB	PIB_D[2]	PC[2]
IOAFCR4[14:12]	GPIO2 3	PIO [3]	PWMA2_CH1N	JTAG_TDI	PWMA3_BRK	PIB_D[3]	PC[3]
IOAFCR4[18:16]	GPIO2 4	PIO [4]	PWMA2_CH2P	DRVVBUS	PWMA3_CH3P	PIB_CLK	PC[4]
IOAFCR4[22:20]	GPIO2 5	PIO [5]	PWMA2_CH2N	PWMA2_CH0P	PWMA3_ETR	PWMA3_CH2N	PC[5]
IOAFCR4[26:24]	GPIO2 6	PIO [6]	UART6_TX	PWMA2_CH0N	UART7_DE	PWMA3_CH2P	PC[6]
IOAFCR4[30:28]	GPIO2 7	PIO [7]	UART6_RX	PWMA2_CH1P	UART7_RE	PWMA3_CH1N	PC[7]
IOAFCR5[2:0]	GPIO2 8	PIO [8]	UART6_DE	PWMA2_CH1N	UART7_TX	PWMA3_CH1P	PC[8]
IOAFCR5[6:4]	GPIO2 9	PIO [9]	UART6_RE	PWMA2_CH2P	UART7_RX	PWMA3_CH0N	PC[9]
IOAFCR5[10:8]	GPIO2 10	PIO [10]	PWMA2_ETR	PWMA2_CH2N	PWMA3_CH2N	PWMA3_CH0P	PC[10]
IOAFCR5[14:12]	GPIO2 11	PIO [11]	PWMA2_CH3P	PIB_D[0]	PWMA3_CH2P	JTAG_TCK	PC[11]
IOAFCR5[18:16]	GPIO2 12	PIO [12]	PWMA2_BRK	PIB_D[1]	PWMA3_CH1N	JTAG_TMS	PC[12]
IOAFCR5[22:20]	GPIO2 13	PIO [13]	PWMA2_QA	PIB_D[2]	PWMA3_CH1P	JTAG_TDO	PC[13]
IOAFCR5[26:24]	GPIO2 14	PIO [14]	PWMA2_QB	PIB_D[3]	PWMA3_CH0N	JTAG_TDI	PC[14]
IOAFCR5[30:28]	GPIO2 15	PIO [15]	PWMA2_INDX	PIB_CLK	PWMA3_CH0P	DRVVBUS	PC[15]

Figure 5-20 I/O pins functions and control for Port C

5.1.3.2. Input Enable

Each BE-U1000 I/O pin input can be enabled or disabled manually. You can control the I/O pin input through the corresponding `*_IE` bit of the `IOAFCRx` registers, where `*` is the pin name and `x` is the register number.

5.1.3.3. Drive Strength

BE-U1000 I/O pins have programmable drive strength per individual I/O pin. You can control the drive strength for each I/O pin through the corresponding bits of the `IODSCRx` registers, where `x` is the register number.

5.1.3.4. Pull-Up and Pull-Down Resistors

Each BE-U1000 I/O pin contains Pull-Up resistor and Pull-Down resistor, which are controlled as folows:

- Pull-Up resistors are activated through the corresponding bits of the `IOPUCRx` registers, where `x` is the register number.
- Pull-Down resistors are activated through the corresponding bits of the `IOPDCRx` registers, where `x` is the register number.

5.2. Control Registers

The CRU register region is mapped in the BE-U1000 microcontroller memory map as shown in the table below.

Table 5-8 Memory address region for control registers

Region	Block Name	Size	Start Address	End Address
Periphery 2	CRU	4 KB	0x1400_0000	0x1400_0FFF

 For details, refer to Memory Map chapter.

This section covers the following topics:

- Register Map
- Register Descriptions

5.2.1. Register Map

The following table provides a top-level summary of each register. To view the detailed description of a register, click the register name in the **Description** column.

Table 5-9 CRU register map

Register	Offset	Description
`CLKSEL`	0x00	Clock Select Register
`PLLSET`	0x04	PLL Control Register
`CLKCR0`	0x08	Clock Control Register 0
`PCLK0EN`	0x0C	`pclk0` Clock Gating and Reset Control Register
`PCLK1EN`	0x10	`pclk1` Clock Gating and Reset Control Register
`PCLK2EN`	0x14	`pclk2` Clock Gating and Reset Control Register
`SYSCR0`	0x18	System Control Register 0
`SYSCR1`	0x1C	System Control Register 1
`SYSCR2`	0x20	System Control Register 2
`PRIOR0`	0x24	Main Bus Matrix Priority Control Register 0
`PRIOR1`	0x28	Main Bus Matrix Priority Control Register 1
`IOPUCR0`	0x2C	I/O Pull-Up Control Register 0

Table 5-9 CRU register map (continued)

Register	Offset	Description
IOPUCR1	0x30	I/O Pull-Up Control Register 1
IOPDCR0	0x34	I/O Pull-Down Control Register 0
IOPDCR1	0x38	I/O Pull-Down Control Register 1
IOAFCR0	0x3C	I/O Alternate Function Control Register 0
IOAFCR1	0x40	I/O Alternate Function Control Register 1
IOAFCR2	0x44	I/O Alternate Function Control Register 2
IOAFCR3	0x48	I/O Alternate Function Control Register 3
IOAFCR4	0x4C	I/O Alternate Function Control Register 4
IOAFCR5	0x50	I/O Alternate Function Control Register 5
IODSCR0	0x54	I/O Drive Strength Control Register 0
IODSCR1	0x58	I/O Drive Strength Control Register 1
IODSCR2	0x5C	I/O Drive Strength Control Register 2
LDOCR	0x60	LDO Control Register
USBCR	0x64	USB PHY Control and Status Register
SYSSR0	0x70	System Status Register
WDTCLRKEY	0x74	WDT Clear Key Register
INTCR0	0x80	Interrupt Source Control Register
CLKCR1	0x84	Clock Control Register 1
FLASHNVRCR	0x90	Flash NVR Control Register
CORE1CR	0x94	Core 1 Control Register

5.2.2. Register Descriptions

In the tables below, the **R/W** column specifies how the application and the core can access the register fields.

5.2.2.1. Clock Select Register (CLKSEL)

The CLKSEL register is at offset 0x00.

The following table shows the register bit assignments.

Table 5-10 Fields for register: `CLKSEL`

Bits	Name	R/W	Description
[31:24]	HCLK1MHZ	R/W	`hclk` clock divider value for generating a 1 MHz pulse that is used as `timer_pulse` Core 2 input. The resulting frequency is `hclk` divided by (`HCLK1MHZ` + 1). **Value After Reset:** 0x18
[23:15]	CCLK1MHZ	R/W	`cclk` clock divider value for generating a 1 MHz pulse that is used as `timer_pulse` Core 0, Core 1 input. The resulting frequency is `cclk` divided by (`CCLK1MHZ` + 1). **Value After Reset:** 0x30
[14:13]	CANX2CLKSEL	R/W	This bit field selects the source of the `can_clk` and `can_clk_x2` signals of the `can_clk` and `can_clk_x2` Clock Domains. **Values:** **0x0:** `sys_clk` signal **0x1:** `div_canclkx2` signal **0x2:** `alt_clk` signal **0x3:** Reserved **Value After Reset:** 0x0
[12:11]	TCLK_SEL	R/W	This bit field selects the source of the `tclk` signal of the `tclk` Clock Domain. **Values:** **0x0:** `sys_clk` signal **0x1:** `div_tclk` signal **0x2:** `alt_clk` signal **0x3:** Reserved **Value After Reset:** 0x0
[10:9]	HCLK_SEL	R/W	This bit field selects the source of the `hclk` signal of the `hclk` Clock Domain. **Values:** **0x0:** `sys_clk` signal **0x1:** `div_hclk` signal **0x2:** `alt_clk` signal **0x3:** Reserved **Value After Reset:** 0x0
[8:7]	PCLK2_SEL	R/W	This bit field selects the source of the `pclk_2` signal of the `pclk_2` Clock Domain. **Values:** **0x0:** `sys_clk` signal **0x1:** `div_pclk2` signal **0x2:** `alt_clk` signal **0x3:** Reserved **Value After Reset:** 0x0
[6:5]	PCLK1_SEL	R/W	This bit field selects the source of the `pclk_1` signal of the `pclk_1` Clock Domain. **Values:**

Table 5-10 Fields for register: `CLKSEL` (continued)

Bits	Name	R/W	Description
			0x0: `sys_clk` signal **0x1:** `div_pclk1` signal **0x2:** `alt_clk` signal **0x3:** Reserved **Value After Reset:** 0x0
[4:3]	`PCLK0_SEL`	R/W	This bit field selects the source of the `pclk_0` signal of the `pclk_0` Clock Domain. **Values:** **0x0:** `sys_clk` signal **0x1:** `div_pclk0` signal **0x2:** `alt_clk` signal **0x3:** Reserved **Value After Reset:** 0x0
[2:1]	`CCLK_SEL`	R/W	This bit field selects the source of the `cclk` signal of the `cclk` Clock Domain. **Values:** **0x0:** `sys_clk` signal **0x1:** `div_cclk` signal **0x2:** `alt_clk` signal **0x3:** Reserved **Value After Reset:** 0x0
[0]	`I_PLL_CLK_SEL`	R/W	This bit field selects the source of the `i_pll_clk` signal. **Values:** **0x0:** `sys_clk` signal **0x1:** `alt_clk` signal **Value After Reset:** 0x0

5.2.2.2. PLL Control Register (`PLLSET`)

The `PLLSET` register is at offset 0x04.

The following table shows the register bit assignments.

Table 5-11 Fields for register: `PLLSET`

Bits	Name	R/W	Description
[31]	`PLL_NLOCK_RSTN`	R/W	`PLL_NLOCK` flag reset signal **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x0
[30:25]			Reserved
[24]	`RESET`	R/W	PLL RESET input value **Values:**

Table 5-11 Fields for register: `PLLSET` (continued)

Bits	Name	R/W	Description
			0x0: Reset is inactive **0x1:** Reset is active **Value After Reset:** 0x1
[23]	`TEST`	R/W	PLL TEST input value Reference-to-counters-to-output bypass when high **Value After Reset:** 0x0
[22]	`BYPASS`	R/W	PLL BYPASS input value **Values:** **0x0:** Bypass mode is disabled **0x1:** Bypass mode is enabled **Value After Reset:** 0x1
[21]	`INTFB`	R/W	Selects the feadback signal. PLL INTFB input value **Values:** **0x1:** Internal signal STOP This bit must be always set to 0x1 value. **Value After Reset:** 0x0
[20]	`PWRDN`	R/W	PowerDown mode is enabled. PLL PWRDN input value. **Values:** **0x0:** PowerDown mode is disabled **0x1:** PowerDown mode is enabled **Value After Reset:** 0x0
[19:14]	`BWADJ`	R/W	PLL BWADJ input value: `NB = BWADJ[19:14] + 1`, `BWADJ[14]` is LSB. **Value After Reset:** 0x0
[13:10]	`CLKOD`	R/W	PLL CLKOD input value: `OD = CLKOD[13:10] + 1`, `CLKOD[10]` is LSB. **Value After Reset:** 0x0
[9:4]	`CLKF`	R/W	PLL CLKF input value: `NF = CLKF[9:4] + 1`, `CLKF[4]` is LSB. **Value After Reset:** 0x0
[3:0]	`CLKR`	R/W	PLL CLKR input value: `NR = CLKR[3:0] + 1`, `CLKR[0]` is LSB. **Value After Reset:** 0x0

5.2.2.3. Clock Control Register 0 (`CLKCR0`)

The `CLKCR0` register is at offset 0x08.

The following table shows the register bit assignments.

Table 5-12 Fields for register: `CLKCR0`

Bits	Name	R/W	Description
[31]	`BOOTCLKSEL`	R/W	This bit selects the source of the boot signal. For more information, see Clock Sources. **Value After Reset:** 0x0
[30]	`OSCGENEN`	R/W	This bit enables generation of `osc_clk`. **Values:** **0x0:** Internal oscillator is disabled **0x1:** Internal oscillator is enabled **Value After Reset:** 0x1
[29]	`HCLKDIVEN`	R/W	This bit enables PLL output frequency divider. **Values:** **0x0:** Divider is disabled **0x1:** Divider is enabled **Value After Reset:** 0x1
[28:24]	`HCLKDIV`	R/W	`hclk` domain divisor value. **Values:** **0x0:** `HCLKDIV[0]` = 0 — integer division. The frequency division ratio is equal to `HCLKDIV[4:1]`+1. Valid values `HCLKDIV[4:1]` from 0 to 15. If `HCLKDIV[4:0]` = 0 — bypass **0x1:** `HCLKDIV[0]` = 1 — fractional division. The frequency division ratio is equal to `HCLKDIV[4:1]`+1.5. Valid values `HCLKDIV[4:1]` from 0 to 14. **Value After Reset:** 0x0
[23]	`PCLK2DIVEN`	R/W	This bit enables PLL output frequency divider. **Values:** **0x0:** Divider is disabled **0x1:** Divider is enabled **Value After Reset:** 0x1
[22:18]	`PCLK2DIV`	R/W	`pclk_2` domain divisor value. **Values:** **0x0:** `PCLK2DIV[0]` = 0 — integer division. The frequency division ratio is equal to `PCLK2DIV[4:1]`+1. Valid values `PCLK2DIV[4:1]` from 0 to 15. If `PCLK2DIV[4:0]` = 0 — bypass **0x1:** `PCLK2DIV[0]` = 1 — fractional division. The frequency division ratio is equal to `PCLK2DIV[4:1]`+1.5. Valid values `PCLK2DIV[4:1]` from 0 to 14. **Value After Reset:** 0x0
[17]	`PCLK1DIVEN`	R/W	This bit enables PLL output frequency divider. **Values:** **0x0:** Divider is disabled **0x1:** Divider is enabled **Value After Reset:** 0x1

Table 5-12 Fields for register: `CLKCR0` (continued)

Bits	Name	R/W	Description
[16:12]	`PCLK1DIV`	R/W	`pclk_1` domain divisor value. **Values:** **0x0:** `PCLK1DIV[0]` = 0 — integer division. The frequency division ratio is equal to `PCLK1DIV[4:1]`+1. Valid values `PCLK1DIV[4:1]` from 0 to 15. If `PCLK1DIV[4:0]` = 0 — bypass **0x1:** `PCLK1DIV[0]` = 1 — fractional division. The frequency division ratio is equal to `PCLK1DIV[4:1]`+1.5. Valid values `PCLK1DIV[4:1]` from 0 to 14. **Value After Reset:** 0x0
[11]	`PCLK0DIVEN`	R/W	This bit enables PLL output frequency divider. **Values:** **0x0:** Divider is disabled **0x1:** Divider is enabled **Value After Reset:** 0x1
[10:6]	`PCLK0DIV`	R/W	`pclk_0` domain divisor value. **Values:** **0x0:** `PCLK0DIV[0]` = 0 — integer division. The frequency division ratio is equal to `PCLK0DIV[4:1]`+1. Valid values `PCLK2DIV[4:1]` from 0 to 15. If `PCLK0DIV[4:0]` = 0 — bypass **0x1:** `PCLK0DIV[0]` = 1 — fractional division. The frequency division ratio is equal to `PCLK0DIV[4:1]`+1.5. Valid values `PCLK0DIV[4:1]` from 0 to 14. **Value After Reset:** 0x0
[5]	`CCLKDIVEN`	R/W	This bit enables PLL output frequency divider. **Values:** **0x0:** Divider is disabled **0x1:** Divider is enabled **Value After Reset:** 0x1
[4:0]	`CCLKDIV`	R/W	`cclk` domain divisor value **Values:** **0x0:** `CCLKDIV[0]` = 0 — integer division. The frequency division ratio is equal to `CCLKDIV[4:1]`+1. Valid values `CCLKDIV[4:1]` from 0 to 15. If `CCLKDIV[4:0]` = 0 — bypass **0x1:** `CCLKDIV[0]` = 1 — fractional division. The frequency division ratio is equal to `CCLKDIV[4:1]`+1.5. Valid values `CCLKDIV[4:1]` from 0 to 14. **Value After Reset:** 0x0

5.2.2.4. `pclk0` Clock Gating and Reset Control Register (`PCLK0EN`)

The `PCLK0EN` register is at offset 0x0C.

The following table shows the register bit assignments.

Table 5-13 Fields for register: `PCLK0EN`

Bits	Name	R/W	Description
[31:30]			Reserved
[29]	`CAN0RSTN`	R/W	CANFD_0 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[28]	`PWMG0RSTN`	R/W	PWMG_0 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[27]	`GPIO0RSTN`	R/W	GPIO_0 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[26]	`WDT0RSTN`	R/W	WDT_0 software reset **Values:** **0x0:** Reset is active (when `WDTCLRKEY.WDTCLRKEY`=0xD09C1EA7) **0x1:** Reset is inactive **Value After Reset:** 0x1
[25]	`TIMER0RSTN`	R/W	TIM_0 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[24]	`I2S0RSTN`	R/W	I2S_0 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[23]	`I2C1RSTN`	R/W	I2C_1 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[22]	`I2C0RSTN`	R/W	I2C_0 software reset **Values:**

Table 5-13 Fields for register: `PCLK0EN` (continued)

Bits	Name	R/W	Description
			0x0: Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[21]	`UART2RSTN`	R/W	UART_2 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[20]	`UART1RSTN`	R/W	UART_1 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[19]	`UART0RSTN`	R/W	UART_0 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[18]	`SPI1RSTN`	R/W	SPI_1 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[17]	`SPI0RSTN`	R/W	SPI_0 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[16]	`QSPI0RSTN`	R/W	QSPI_0 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[15:14]			Reserved
[13]	`CAN0CLKEN`	R/W	This bit enables the `can_clk` of the CANFD_0 **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x1

Table 5-13 Fields for register: `PCLK0EN` (continued)

Bits	Name	R/W	Description
[12]	`PWMG0CLKEN`	R/W	This bit enables the `CK_INT` of the PWMG_0 **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x1
[11]			Reserved
[10]	`WDT0CLKEN`	R/W	This bit enables the `tclk` of the WDT_0 **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x1
[9]	`TIMER0CLKEN`	R/W	This bit enables the `timer_N_clk` of the TIM_0 **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x1
[8]			Reserved
[7]	`I2C1CLKEN`	R/W	This bit enables the `i2c1_clk` of the I2C_1 **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x1
[6]	`I2C0CLKEN`	R/W	This bit enables the `i2c0_clk` of the I2C_0 **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x1
[5:3]			Reserved
[2]	`SPI1CLKEN`	R/W	This bit enables the `spi1_clk` of the SPI_1 **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x1
[1]	`SPI0CLKEN`	R/W	This bit enables the `spi0_clk` of the SPI_0 **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x1
[0]	`QSPI0CLKEN`	R/W	This bit enables the `qspi0_clk` of the QSPI_0

Table 5-13 Fields for register: `PCLK0EN` (continued)

Bits	Name	R/W	Description
			Values: **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x1

5.2.2.5. `pclk1` Clock Gating and Reset Control Register (`PCLK1EN`)

The `PCLK1EN` register is at offset 0x10.

The following table shows the register bit assignments.

Table 5-14 Fields for register: `PCLK1EN`

Bits	Name	R/W	Description
[31:30]			Reserved
[29]	`CAN1RSTN`	R/W	CANFD_1 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[28]	`PWMG1RSTN`	R/W	PWMG_1 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[27]	`GPIO1RSTN`	R/W	GPIO_1 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[26]	`WDT1RSTN`	R/W	WDT_1 software reset **Values:** **0x0:** Reset is active (when `WDTCLRKEY.WDTCLRKEY`=0xD09C1EA7) **0x1:** Reset is inactive **Value After Reset:** 0x1
[25]	`TIMER1RSTN`	R/W	TIM_1 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[24]	`I2S1RSTN`	R/W	I2S_1 software reset **Values:**

Table 5-14 Fields for register: `PCLK1EN` (continued)

Bits	Name	R/W	Description
			0x0: Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[23]	`I2C3RSTN`	R/W	I2C_3 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[22]	`I2C2RSTN`	R/W	I2C_2 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[21]	`UART5RSTN`	R/W	UART_5 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[20]	`UART4RSTN`	R/W	UART_4 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[19]	`UART3RSTN`	R/W	UART_3 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[18]	`SPI3RSTN`	R/W	SPI_3 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[17]	`SPI2RSTN`	R/W	SPI_2 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[16]	`QSPI1RSTN`	R/W	QSPI_1 software reset **Values:**

Table 5-14 Fields for register: `PCLK1EN` (continued)

Bits	Name	R/W	Description
			0x0: Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[15:14]			Reserved
[13]	`CAN1CLKEN`	R/W	This bit enables the `can_clk` of the CANFD_1 **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x1
[12]	`PWMG1CLKEN`	R/W	This bit enables the `CK_INT` of the PWMG_1 **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x1
[11]			Reserved
[10]	`WDT1CLKEN`	R/W	This bit enables the `tclk` of the WDT_1 **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x1
[9]	`TIMER1CLKEN`	R/W	This bit enables the `timer_N_clk` of the TIM_1 **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x1
[8]			Reserved
[7]	`I2C3CLKEN`	R/W	This bit enables the `i2c3_clk` of the I2C_3 **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x1
[6]	`I2C2CLKEN`	R/W	This bit enables the `i2c2_clk` of the I2C_2 **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x1
[5:3]			Reserved
[2]	`SPI3CLKEN`	R/W	This bit enables the `spi3_clk` of the SPI_3 **Values:**

Table 5-14 Fields for register: PCLK1EN (continued)

Bits	Name	R/W	Description
			0x0: Disabled **0x1:** Enabled **Value After Reset:** 0x1
[1]	SPI2CLKEN	R/W	This bit enables the spi2_clk of the SPI_2 **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x1
[0]	QSPI1CLKEN	R/W	This bit enables the qspi1_clk of the QSPI_1 **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x1

5.2.2.6. pclk2 Clock Gating and Reset Control Register (PCLK2EN)

The PCLK2EN register is at offset 0x14.

The following table shows the register bit assignments.

Table 5-15 Fields for register: PCLK2EN

Bits	Name	R/W	Description
[31:26]			Reserved
[25]	UART7RSTN	R/W	UART_7 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[24]	UART6RSTN	R/W	UART_6 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[23]	GPIO2RSTN	R/W	GPIO_2 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[22]	PWMA3RSTN	R/W	PWMA_3 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive

Table 5-15 Fields for register: `PCLK2EN` (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x1
[21]	`PWMA2RSTN`	R/W	PWMA_2 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[20]	`PWMA1RSTN`	R/W	PWMA_1 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[19]	`PWMA0RSTN`	R/W	PWMA_0 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[18]	`ADC2RSTN`	R/W	ADC_2 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[17]	`ADC1RSTN`	R/W	ADC_1 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[16]	`ADC0RSTN`	R/W	ADC_0 software reset **Values:** **0x0:** Reset is active **0x1:** Reset is inactive **Value After Reset:** 0x1
[15:14]			Reserved
[6]	`PWMA3CLKEN`	R/W	This bit enables the `CK_INT` of the PWMA_3 **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x1
[5]	`PWMA2CLKEN`	R/W	This bit enables the `CK_INT` of the PWMA_2 **Values:**

Table 5-15 Fields for register: `PCLK2EN` (continued)

Bits	Name	R/W	Description
			0x0: Disabled **0x1:** Enabled **Value After Reset:** 0x1
[4]	`PWMA1CLKEN`	R/W	This bit enables the `CK_INT` of the PWMA_1 **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x1
[3]	`PWMA0CLKEN`	R/W	This bit enables the `CK_INT` of the PWMA_0 **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x1
[2]	`ADC2CLKEN`	R/W	This bit enables the `CK_IN` of the ADC_2 **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x1
[1]	`ADC1CLKEN`	R/W	This bit enables the `CK_IN` of the ADC_1 **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x1
[0]	`ADC0CLKEN`	R/W	This bit enables the `CK_IN` of the ADC_0 **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x1

5.2.2.7. System Control Register 0 (`SYSCR0`)

The `SYSCR0` register is at offset 0x18.

The following table shows the register bit assignments.

Table 5-16 Fields for register: `SYSCR0`

Bits	Name	R/W	Description
[31]	`DMAMODDIS`	R/W	Direct access lock flag. This bit disables the modification of the eFLASH memory using the direct access. **Values:**

Table 5-16 Fields for register: SYSCR0 (continued)

Bits	Name	R/W	Description
			0x0: Means that the controller can modify (program) the eFLASH memory using the direct access. **0x1:** Means that the controller can't modify (program) the eFLASH memory using the direct access. **Value After Reset:** 0x1
[30]	SOFTDBGEN	R/W	This bit enables Core 0 and Core 1 software debugging **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x0
[29:27]			Reserved for internal use
[26]	FLASHNVRDIS	R/W	NVR disable flag. This bit disables the access to the FLASH NVR array. This bit canonly be set to 1 by the software. This bit can only be reset upon power-up or via an external nreset signal. **Values:** **0x0:** Enabled (means that the controller can perform an operation in the NVR array). **0x1:** Disabled (means that it is forbidden for the controller to perform an operation in the NVR array, any request for an operation in the NVR array will be interpreted as a request for an operation in the Main array). **Value After Reset:** 0x0
[25:17]			Reserved
[16]	WDTHITCLR	R/W	WDTHIT flag reset of the SYSSR0 register **Values:** **0x0:** WDTHIT flag is not reset **0x1:** WDTHIT flag is reset **Value After Reset:** 0x0
[15:12]	ALARM_LVL	R/W	The alarm signal is generated when the internal counter of the ref_clk Monitoring block reaches the value written to this bit field. For more information about the alarm, see Clock Sources. **Value After Reset:** 0x0
[11]	ALARM_EN	R/W	This bit enables the ref_clk Monitoring block **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x0
[10]	NVR	R/W	NVR enable flag. This bit enables the access to the FLASH NVR array **Values:** **0x0:** Disabled (operation is performed in the Main array) **0x1:** Enabled (operation is performed in the NVR array)

Table 5-16 Fields for register: `SYSCR0` (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0
[9]	`TRACESEL`	R/W	Selects the core to work with the trace debugging module The selection of the trace source (Core 0 or Core 1) is done via the multiplexor that is controlled via the logical OR of the `TRACE_SEL.ALTTRACESEL` bit and `TRACESEL` bit. **Values:** **0x0:** Core 0 **0x1:** Core 1 **Value After Reset:** 0x0
[8]	`XIP1_EN`	R/W	This bit enables the QSPI_1 XIP mode **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x0
[7]	`XIP0_EN`	R/W	This bit enables the QSPI_0 XIP mode **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x0
[6]	`CORE2CXRSTN`	R/W	This bit resets the logic inside the Core 2, except for the processor core and TCM arbiter. Active low level. **Value After Reset:** 0x0
[5]	`CORE2FPRSTN`	R/W	This bit resets the Core 2 TCM arbiter. Active low level. **Value After Reset:** 0x0
[4]	`CORE2RSTN`	R/W	This bit resets the Core 2 processor core. Active low level. **Value After Reset:** 0x0
[3]	`CORE1CXRSTN`	R/W	This bit resets the logic inside the Core 1, except for the processor core and TCM arbiter. Active low level. **Value After Reset:** 0x0
[2]	`CORE1FPRSTN`	R/W	This bit resets the Core 1 TCM arbiter. Active low level. **Value After Reset:** 0x0
[1]	`CORE1RSTN`	R/W	This bit resets the Core 1 processor core. Active low level. **Value After Reset:** 0x0
[0]	`CORE1CLKEN`	R/W	This bit enables the Core 1 processor core clocking **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x1

5.2.2.8. System Control Register 1 (SYSCR1)

The SYSCR1 register is at offset 0x1C.

The following table shows the register bit assignments.

Table 5-17 Fields for register: SYSCR1

Bits	Name	R/W	Description
[31:0]	CORE1RSTVEC	R/W	The Core 1 command counter start address **Value After Reset:** 0x0

5.2.2.9. System Control Register 2 (SYSCR2)

The SYSCR2 register is at offset 0x20.

The following table shows the register bit assignments.

Table 5-18 Fields for register: SYSCR2

Bits	Name	R/W	Description
[31:0]	CORE2RSTVEC	R/W	The Core 2 command counter start address **Value After Reset:** 0x0

5.2.2.10. Main Bus Matrix Prority Control Register 0 (PRIOR0)

The PRIOR0 register is at offset 0x24.

The following table shows the register bit assignments.

Table 5-19 Fields for register: PRIOR0

Bits	Name	R/W	Description
[31:27]			Reserved
[26:24]	USB_PRIOR	R/W	Priority of the USB **Values:** **0x0:** Lowest priority ... **0x7:** Highest priority **Value After Reset:** 0x0
[23]			Reserved
[22:20]	DMA1_PRIOR	R/W	Priority of the DMA_1 **Values:** **0x0:** Lowest priority ... **0x7:** Highest priority **Value After Reset:** 0x0
[19]			Reserved
[18:16]	DMA0_PRIOR	R/W	Priority of the DMA_0 **Values:**

Table 5-19 Fields for register: `PRIOR0` (continued)

Bits	Name	R/W	Description
			0x0: Lowest priority ... **0x7:** Highest priority **Value After Reset:** 0x0
[15]			Reserved
[14:12]	`CORE1_PP_PRIOR`	R/W	Priority of the Core 1 *Peripheral Port* (**PP**) **Values:** **0x0:** Lowest priority ... **0x7:** Highest priority **Value After Reset:** 0x0
[11]			Reserved
[10:8]	`CORE1_MP_PRIOR`	R/W	Priority of the Core 1 *Memory Port* (**MP**) **Values:** **0x0:** Lowest priority ... **0x7:** Highest priority **Value After Reset:** 0x0
[7]			Reserved
[6:4]	`CORE0_PP_PRIOR`	R/W	Priority of the Core 0 *Peripheral Port* (**PP**) **Values:** **0x0:** Lowest priority ... **0x7:** Highest priority **Value After Reset:** 0x0
[3]			Reserved
[2:0]	`CORE0_MP_PRIOR`	R/W	Priority of the Core 0 *Memory Port* (**MP**) **Values:** **0x0:** Lowest priority ... **0x7:** Highest priority **Value After Reset:** 0x0

5.2.2.11. Main Bus Matrix Prority Control Register 1 (`PRIOR1`)

The `PRIOR1` register is at offset 0x28.

The following table shows the register bit assignments.

Table 5-20 Fields for register: `PRIOR1`

Bits	Name	R/W	Description
[31:27]			Reserved
[26:24]	`EFLASH_PRIOR`	R/W	Priority of the eFLASH Controller **Values:** **0x0:** Lowest priority ... **0x7:** Highest priority **Value After Reset:** 0x0
[23]			Reserved
[22:20]	`SRAM_PRIOR`	R/W	Priority of the SRAM **Values:** **0x0:** Lowest priority ... **0x7:** Highest priority **Value After Reset:** 0x0
[19]			Reserved
[18:16]	`CORE0_FP_PRIOR`	R/W	Priority of the Core 0 *Front Port* (**FP**) **Values:** **0x0:** Lowest priority ... **0x7:** Highest priority **Value After Reset:** 0x0
[15]			Reserved
[14:12]	`BUS_PRIOR`	R/W	Priority of the Bus matrix **Values:** **0x0:** Lowest priority ... **0x7:** Highest priority **Value After Reset:** 0x0
[11]			Reserved
[10:8]	`PERIPH2_PRIOR`	R/W	Priority of the Periphery 2 **Values:** **0x0:** Lowest priority ... **0x7:** Highest priority **Value After Reset:** 0x0
[7]			Reserved
[6:4]	`PERIPH1_PRIOR`	R/W	Priority of the Periphery 1 **Values:**

Table 5-20 Fields for register: `PRIOR1` (continued)

Bits	Name	R/W	Description
			0x0: Lowest priority ... **0x7:** Highest priority **Value After Reset:** 0x0
[3]			Reserved
[2:0]	`PERIPH0_PRIOR`	R/W	Priority of the Periphery 0 **Values:** **0x0:** Lowest priority ... **0x7:** Highest priority **Value After Reset:** 0x0

5.2.2.12. I/O Pull-Up Control Register 0 (`IOPUCR0`)

The `IOPUCR0` register is at offset 0x2C.

The following table shows the register bit assignments.

Table 5-21 Fields for register: `IOPUCR0`

Bits	Name	R/W	Description
[31:16]	`PB_PU_EN`	R/W	This bit field enables Pull-Up resistors for Port B I/O pins. Each bit of the `PB_PU_EN` bit field controls the Pull-Up resistor of the appropriate Port B I/O pin: • `PB_PU_EN [31]`: This bit controls the Pull-Up resistor of the PB[15] I/O pin. • `PB_PU_EN [30]`: This bit controls the Pull-Up resistor of the PB[14] I/O pin. • ... • `PB_PU_EN [16]`: This bit controls the Pull-Up resistor of the PB[0] I/O pin. **Per bit values:** **0x0:** Disable Pull-Up resistor **0x1:** Enable Pull-Up resistor **Value After Reset:** 0x0
[15:0]	`PA_PU_EN`	R/W	This bit field enables Pull-Up resistors for Port A I/O pins. Each bit of the `PA_PU_EN` bit field controls the Pull-Up resistor of the appropriate Port A I/O pin: • `PA_PU_EN [15]`: This bit controls the Pull-Up resistor of the PA[15] I/O pin. • `PA_PU_EN [14]`: This bit controls the Pull-Up resistor of the PA[14] I/O pin. • ... • `PA_PU_EN [0]`: This bit controls the Pull-Up resistor of the PA[0] I/O pin.

Table 5-21 Fields for register: `IOPUCR0` (continued)

Bits	Name	R/W	Description
			Per bit values: **0x0:** Disable Pull-Up resistor **0x1:** Enable Pull-Up resistor **Value After Reset:** 0x0

5.2.2.13. I/O Pull-Up Control Register 1 (`IOPUCR1`)

The `IOPUCR1` register is at offset 0x30.

The following table shows the register bit assignments.

Table 5-22 Fields for register: `IOPUCR1`

Bits	Name	R/W	Description
[31:16]			Reserved
[15:0]	`PC_PU_EN`	R/W	This bit field enables Pull-Up resistors for Port C I/O pins. Each bit of the `PC_PU_EN` bit field controls the Pull-Up resistor of the appropriate Port C I/O pin: • `PC_PU_EN [15]`: This bit controls the Pull-Up resistor of the PC[15] I/O pin. • `PC_PU_EN [14]`: This bit controls the Pull-Up resistor of the PC[14] I/O pin. • ... • `PC_PU_EN [0]`: This bit controls the Pull-Up resistor of the PC[0] I/O pin. **Per bit values:** **0x0:** Disable Pull-Up resistor **0x1:** Enable Pull-Up resistor **Value After Reset:** 0x0

5.2.2.14. I/O Pull-Down Control Register 0 (`IOPDCR0`)

The `IOPDCR0` register is at offset 0x34.

The following table shows the register bit assignments.

Table 5-23 Fields for register: `IOPDCR0`

Bits	Name	R/W	Description
[31:16]	`PB_PD_EN`	R/W	This bit field enables Pull-Down resistors for Port B I/O pins. Each bit of the `PB_PD_EN` bit field controls the Pull-Down resistor of the appropriate Port B I/O pin: • `PB_PD_EN [31]`: This bit controls the Pull-Down resistor of the PB[15] I/O pin. • `PB_PD_EN [30]`: This bit controls the Pull-Down resistor of the PB[14] I/O pin.

Table 5-23 Fields for register: `IOPDCR0` (continued)

Bits	Name	R/W	Description
			• ... • `PB_PD_EN [16]`: This bit controls the Pull-Down resistor of the PB[0] I/O pin. **Per bit values:** **0x0:** Disable Pull-Down resistor **0x1:** Enable Pull-Down resistor **Value After Reset:** 0xFFFF
[15:0]	`PA_PD_EN`	R/W	This bit field enables Pull-Down resistors for Port A I/O pins. Each bit of the `PA_PD_EN` bit field controls the Pull-Down resistor of the appropriate Port A I/O pin: • `PA_PD_EN [15]`: This bit controls the Pull-Down resistor of the PA[15] I/O pin. • `PA_PD_EN [14]`: This bit controls the Pull-Down resistor of the PA[14] I/O pin. • ... • `PA_PD_EN [0]`: This bit controls the Pull-Down resistor of the PA[0] I/O pin. **Per bit values:** **0x0:** Disable Pull-Down resistor **0x1:** Enable Pull-Down resistor **Value After Reset:** 0xFFFF

5.2.2.15. I/O Pull-Down Control Register 1 (`IOPDCR1`)

The `IOPDCR1` register is at offset 0x38.

The following table shows the register bit assignments.

Table 5-24 Fields for register: `IOPDCR1`

Bits	Name	R/W	Description
[31:16]			Reserved
[15:0]	`PC_PD_EN`	R/W	This bit field enables Pull-Down resistors for Port C I/O pins. Each bit of the `PC_PD_EN` bit field controls the Pull-Down resistor of the appropriate Port C I/O pin: • `PC_PD_EN [15]`: This bit controls the Pull-Down resistor of the PC[15] I/O pin. • `PC_PD_EN [14]`: This bit controls the Pull-Down resistor of the PC[14] I/O pin. • ... • `PC_PD_EN [0]`: This bit controls the Pull-Down resistor of the PC[0] I/O pin. **Per bit values:** **0x0:** Disable Pull-Down resistor **0x1:** Enable Pull-Down resistor

Table 5-24 Fields for register: `IOPDCR1` (continued)

Bits	Name	R/W	Description
			Value After Reset: 0xFFFF

5.2.2.16. I/O Alternate Function Control Register 0 (`IOAFCR0`)

The `IOAFCR0` register is at offset 0x3C.

The following table shows the register bit assignments.

Table 5-25 Fields for register: `IOAFCR0`

Bits	Name	R/W	Description
[31]	`PA7_IE`	R/W	This bit enables the PA[7] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[30:28]	`PA7_AF`	R/W	This bit field selects the PA[7] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port A*** figure in Alternate Functions section. **Value After Reset:** 0x0
[27]	`PA6_IE`	R/W	This bit enables the PA[6] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[26:24]	`PA6_AF`	R/W	This bit field selects the PA[6] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port A*** figure in Alternate Functions section. **Value After Reset:** 0x0
[23]	`PA5_IE`	R/W	This bit enables the PA[5] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[22:20]	`PA5_AF`	R/W	This bit field selects the PA[5] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port A*** figure in Alternate Functions section. **Value After Reset:** 0x0
[19]	`PA4_IE`	R/W	This bit enables the PA[4] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1

Table 5-25 Fields for register: `IOAFCR0` (continued)

Bits	Name	R/W	Description
[18:16]	PA4_AF	R/W	This bit field selects the PA[4] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port A*** figure in Alternate Functions section. **Value After Reset:** 0x0
[15]	PA3_IE	R/W	This bit enables the PA[3] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[14:12]	PA3_AF	R/W	This bit field selects the PA[3] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port A*** figure in Alternate Functions section. **Value After Reset:** 0x0
[11]	PA2_IE	R/W	This bit enables the PA[2] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[10:8]	PA2_AF	R/W	This bit field selects the PA[2] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port A*** figure in Alternate Functions section. **Value After Reset:** 0x0
[7]	PA1_IE	R/W	This bit enables the PA[1] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[6:4]	PA1_AF	R/W	This bit field selects the PA[1] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port A*** figure in Alternate Functions section. **Value After Reset:** 0x0
[3]	PA0_IE	R/W	This bit enables the PA[0] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[2:0]	PA0_AF	R/W	This bit field selects the PA[0] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port A*** figure in Alternate Functions section. **Value After Reset:** 0x0

5.2.2.17. I/O Alternate Function Control Register 1 (IOAFCR1)

The IOAFCR1 register is at offset 0x40.

The following table shows the register bit assignments.

Table 5-26 Fields for register: IOAFCR1

Bits	Name	R/W	Description
[31]	PA15_IE	R/W	This bit enables the PA[15] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[30:28]	PA15_AF	R/W	This bit field selects the PA[15] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port A*** figure in Alternate Functions section. **Value After Reset:** 0x0
[27]	PA14_IE	R/W	This bit enables the PA[14] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[26:24]	PA14_AF	R/W	This bit field selects the PA[14] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port A*** figure in Alternate Functions section. **Value After Reset:** 0x0
[23]	PA13_IE	R/W	This bit enables the PA[13] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[22:20]	PA13_AF	R/W	This bit field selects the PA[13] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port A*** figure in Alternate Functions section. **Value After Reset:** 0x0
[19]	PA12_IE	R/W	This bit enables the PA[12] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[18:16]	PA12_AF	R/W	This bit field selects the PA[12] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port A*** figure in Alternate Functions section. **Value After Reset:** 0x0
[15]	PA11_IE	R/W	This bit enables the PA[11] I/O pin input.

Table 5-26 Fields for register: `IOAFCR1` (continued)

Bits	Name	R/W	Description
			Values: **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[14:12]	`PA11_AF`	R/W	This bit field selects the PA[11] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port A*** figure in Alternate Functions section. **Value After Reset:** 0x0
[11]	`PA10_IE`	R/W	This bit enables the PA[10] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[10:8]	`PA10_AF`	R/W	This bit field selects the PA[10] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port A*** figure in Alternate Functions section. **Value After Reset:** 0x0
[7]	`PA9_IE`	R/W	This bit enables the PA[9] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[6:4]	`PA9_AF`	R/W	This bit field selects the PA[9] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port A*** figure in Alternate Functions section. **Value After Reset:** 0x0
[3]	`PA8_IE`	R/W	This bit enables the PA[8] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[2:0]	`PA8_AF`	R/W	This bit field selects the PA[8] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port A*** figure in Alternate Functions section. **Value After Reset:** 0x0

5.2.2.18. I/O Alternate Function Control Register 2 (`IOAFCR2`)

The `IOAFCR2` register is at offset 0x44.

The following table shows the register bit assignments.

Table 5-27 Fields for register: `IOAFCR2`

Bits	Name	R/W	Description
[31]	PB7_IE	R/W	This bit enables the PB[7] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[30:28]	PB7_AF	R/W	This bit field selects the PB[7] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port B*** figure in Alternate Functions section. **Value After Reset:** 0x0
[27]	PB6_IE	R/W	This bit enables the PB[6] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[26:24]	PB6_AF	R/W	This bit field selects the PB[6] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port B*** figure in Alternate Functions section. **Value After Reset:** 0x0
[23]	PB5_IE	R/W	This bit enables the PB[5] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[22:20]	PB5_AF	R/W	This bit field selects the PB[5] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port B*** figure in Alternate Functions section. **Value After Reset:** 0x0
[19]	PB4_IE	R/W	This bit enables the PB[4] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[18:16]	PB4_AF	R/W	This bit field selects the PB[4] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port B*** figure in Alternate Functions section. **Value After Reset:** 0x0
[15]	PB3_IE	R/W	This bit enables the PB[3] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1

Table 5-27 Fields for register: `IOAFCR2` (continued)

Bits	Name	R/W	Description
[14:12]	PB3_AF	R/W	This bit field selects the PB[3] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port B*** figure in Alternate Functions section. **Value After Reset:** 0x0
[11]	PB2_IE	R/W	This bit enables the PB[2] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[10:8]	PB2_AF	R/W	This bit field selects the PB[2] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port B*** figure in Alternate Functions section. **Value After Reset:** 0x0
[7]	PB1_IE	R/W	This bit enables the PB[1] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[6:4]	PB1_AF	R/W	This bit field selects the PB[1] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port B*** figure in Alternate Functions section. **Value After Reset:** 0x0
[3]	PB0_IE	R/W	This bit enables the PB[0] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[2:0]	PB0_AF	R/W	This bit field selects the PB[0] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port B*** figure in Alternate Functions section. **Value After Reset:** 0x0

5.2.2.19. I/O Alternate Function Control Register 3 (`IOAFCR3`)

The `IOAFCR3` register is at offset 0x48.

The following table shows the register bit assignments.

Table 5-28 Fields for register: `IOAFCR3`

Bits	Name	R/W	Description
[31]	PB15_IE	R/W	This bit enables the PB[15] I/O pin input. **Values:**

Table 5-28 Fields for register: `IOAFCR3` (continued)

Bits	Name	R/W	Description
			0x0: Disable **0x1:** Enable **Value After Reset:** 0x1
[30:28]	PB15_AF	R/W	This bit field selects the PB[15] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port B*** figure in Alternate Functions section. **Value After Reset:** 0x0
[27]	PB14_IE	R/W	This bit enables the PB[14] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[26:24]	PB14_AF	R/W	This bit field selects the PB[14] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port B*** figure in Alternate Functions section. **Value After Reset:** 0x0
[23]	PB13_IE	R/W	This bit enables the PB[13] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[22:20]	PB13_AF	R/W	This bit field selects the PB[13] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port B*** figure in Alternate Functions section. **Value After Reset:** 0x0
[19]	PB12_IE	R/W	This bit enables the PB[12] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[18:16]	PB12_AF	R/W	This bit field selects the PB[12] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port B*** figure in Alternate Functions section. **Value After Reset:** 0x0
[15]	PB11_IE	R/W	This bit enables the PB[11] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[14:12]	PB11_AF	R/W	This bit field selects the PB[11] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port B*** figure in Alternate Functions section.

Table 5-28 Fields for register: `IOAFCR3` (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0
[11]	`PB10_IE`	R/W	This bit enables the PB[10] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[10:8]	`PB10_AF`	R/W	This bit field selects the PB[10] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port B*** figure in Alternate Functions section. **Value After Reset:** 0x0
[7]	`PB9_IE`	R/W	This bit enables the PB[9] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[6:4]	`PB9_AF`	R/W	This bit field selects the PB[9] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port B*** figure in Alternate Functions section. **Value After Reset:** 0x0
[3]	`PB8_IE`	R/W	This bit enables the PB[8] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[2:0]	`PB8_AF`	R/W	This bit field selects the PB[8] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port B*** figure in Alternate Functions section. **Value After Reset:** 0x0

5.2.2.20. I/O Alternate Function Control Register 4 (`IOAFCR4`)

The `IOAFCR4` register is at offset 0x4C.

The following table shows the register bit assignments.

Table 5-29 Fields for register: `IOAFCR4`

Bits	Name	R/W	Description
[31]	`PC7_IE`	R/W	This bit enables the PC[7] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1

Table 5-29 Fields for register: `IOAFCR4` (continued)

Bits	Name	R/W	Description
[30:28]	PC7_AF	R/W	This bit field selects the PC[7] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port C*** figure in Alternate Functions section. **Value After Reset:** 0x0
[27]	PC6_IE	R/W	This bit enables the PC[6] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[26:24]	PC6_AF	R/W	This bit field selects the PC[6] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port C*** figure in Alternate Functions section. **Value After Reset:** 0x0
[23]	PC5_IE	R/W	This bit enables the PC[5] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[22:20]	PC5_AF	R/W	This bit field selects the PC[5] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port C*** figure in Alternate Functions section. **Value After Reset:** 0x0
[19]	PC4_IE	R/W	This bit enables the PC[4] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[18:16]	PC4_AF	R/W	This bit field selects the PC[4] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port C*** figure in Alternate Functions section. **Value After Reset:** 0x0
[15]	PC3_IE	R/W	This bit enables the PC[3] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[14:12]	PC3_AF	R/W	This bit field selects the PC[3] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port C*** figure in Alternate Functions section. **Value After Reset:** 0x0
[11]	PC2_IE	R/W	This bit enables the PC[2] I/O pin input. **Values:**

Table 5-29 Fields for register: `IOAFCR4` (continued)

Bits	Name	R/W	Description
			0x0: Disable **0x1:** Enable **Value After Reset:** 0x1
[10:8]	`PC2_AF`	R/W	This bit field selects the PC[2] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port C*** figure in Alternate Functions section. **Value After Reset:** 0x0
[7]	`PC1_IE`	R/W	This bit enables the PC[1] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[6:4]	`PC1_AF`	R/W	This bit field selects the PC[1] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port C*** figure in Alternate Functions section. **Value After Reset:** 0x0
[3]	`PC0_IE`	R/W	This bit enables the PC[0] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[2:0]	`PC0_AF`	R/W	This bit field selects the PC[0] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port C*** figure in Alternate Functions section. **Value After Reset:** 0x0

5.2.2.21. I/O Alternate Function Control Register 5 (`IOAFCR5`)

The `IOAFCR5` register is at offset 0x50.

The following table shows the register bit assignments.

Table 5-30 Fields for register: `IOAFCR5`

Bits	Name	R/W	Description
[31]	`PC15_IE`	R/W	This bit enables the PC[15] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[30:28]	`PC15_AF`	R/W	This bit field selects the PC[15] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port C*** figure in Alternate Functions section. **Value After Reset:** 0x0

Table 5-30 Fields for register: `IOAFCR5` (continued)

Bits	Name	R/W	Description
[27]	PC14_IE	R/W	This bit enables the PC[14] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[26:24]	PC14_AF	R/W	This bit field selects the PC[14] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port C*** figure in Alternate Functions section. **Value After Reset:** 0x0
[23]	PC13_IE	R/W	This bit enables the PC[13] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[22:20]	PC13_AF	R/W	This bit field selects the PC[13] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port C*** figure in Alternate Functions section. **Value After Reset:** 0x0
[19]	PC12_IE	R/W	This bit enables the PC[12] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[18:16]	PC12_AF	R/W	This bit field selects the PC[12] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port C*** figure in Alternate Functions section. **Value After Reset:** 0x0
[15]	PC11_IE	R/W	This bit enables the PC[11] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[14:12]	PC11_AF	R/W	This bit field selects the PC[11] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port C*** figure in Alternate Functions section. **Value After Reset:** 0x0
[11]	PC10_IE	R/W	This bit enables the PC[10] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1

Table 5-30 Fields for register: `IOAFCR5` (continued)

Bits	Name	R/W	Description
[10:8]	PC10_AF	R/W	This bit field selects the PC[10] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port C*** figure in Alternate Functions section. **Value After Reset:** 0x0
[7]	PC9_IE	R/W	This bit enables the PC[9] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[6:4]	PC9_AF	R/W	This bit field selects the PC[9] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port C*** figure in Alternate Functions section. **Value After Reset:** 0x0
[3]	PC8_IE	R/W	This bit enables the PC[8] I/O pin input. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1
[2:0]	PC8_AF	R/W	This bit field selects the PC[8] I/O pin alternate function. For information about bit setting, see ***I/O pins functions and control for Port C*** figure in Alternate Functions section. **Value After Reset:** 0x0

5.2.2.22. I/O Drive Strength Control Register 0 (`IODSCR0`)

The `IODSCR0` register is at offset 0x54.

The following table shows the register bit assignments.

Table 5-31 Fields for register: `IODSCR0`

Bits	Name	R/W	Description
[31:30]	PADS15	R/W	This bit field controls the load function of the PA[15] output driver.
[29:28]	PADS14	R/W	This bit field controls the load function of the PA[14] output driver.
[27:26]	PADS13	R/W	This bit field controls the load function of the PA[13] output driver.
[25:24]	PADS12	R/W	This bit field controls the load function of the PA[12] output driver.
[23:22]	PADS11	R/W	This bit field controls the load function of the PA[11] output driver.
[21:20]	PADS10	R/W	This bit field controls the load function of the PA[10] output driver.
[19:18]	PADS9	R/W	This bit field controls the load function of the PA[9] output driver.
[17:16]	PADS8	R/W	This bit field controls the load function of the PA[8] output driver.

Table 5-31 Fields for register: `IODSCR0` (continued)

Bits	Name	R/W	Description
[15:14]	PADS7	R/W	This bit field controls the load function of the PA[7] output driver.
[13:12]	PADS6	R/W	This bit field controls the load function of the PA[6] output driver.
[11:10]	PADS5	R/W	This bit field controls the load function of the PA[5] output driver.
[9:8]	PADS4	R/W	This bit field controls the load function of the PA[4] output driver.
[7:6]	PADS3	R/W	This bit field controls the load function of the PA[3] output driver.
[5:4]	PADS2	R/W	This bit field controls the load function of the PA[2] output driver.
[3:2]	PADS1	R/W	This bit field controls the load function of the PA[1] output driver.
[1:0]	PADS0	R/W	This bit field controls the load function of the PA[0] output driver. **0x0:** 0 (4.5 mA) **0x1:** 1 (9 mA) **0x2:** 2 (13.5 mA) **0x3:** 3 (maximum) (18 mA) **Value After Reset:** 0x0

5.2.2.23. I/O Drive Strength Control Register 1 (`IODSCR1`)

The `IODSCR1` register is at offset 0x58.

The following table shows the register bit assignments.

Table 5-32 Fields for register: `IODSCR1`

Bits	Name	R/W	Description
[31:30]	PBDS15	R/W	This bit field controls the load function of the PB[15] output driver.
[29:28]	PBDS14	R/W	This bit field controls the load function of the PB[14] output driver.
[27:26]	PBDS13	R/W	This bit field controls the load function of the PB[13] output driver.
[25:24]	PBDS12	R/W	This bit field controls the load function of the PB[12] output driver.
[23:22]	PBDS11	R/W	This bit field controls the load function of the PB[11] output driver.
[21:20]	PBDS10	R/W	This bit field controls the load function of the PB[10] output driver.
[19:18]	PBDS9	R/W	This bit field controls the load function of the PB[9] output driver.
[17:16]	PBDS8	R/W	This bit field controls the load function of the PB[8] output driver.
[15:14]	PBDS7	R/W	This bit field controls the load function of the PB[7] output driver.
[13:12]	PBDS6	R/W	This bit field controls the load function of the PB[6] output driver.
[11:10]	PBDS5	R/W	This bit field controls the load function of the PB[5] output driver.

Table 5-32 Fields for register: IODSCR1 (continued)

Bits	Name	R/W	Description
[9:8]	PBDS4	R/W	This bit field controls the load function of the PB[4] output driver.
[7:6]	PBDS3	R/W	This bit field controls the load function of the PB[3] output driver.
[5:4]	PBDS2	R/W	This bit field controls the load function of the PB[2] output driver.
[3:2]	PBDS1	R/W	This bit field controls the load function of the PB[1] output driver.
[1:0]	PBDS0	R/W	This bit field controls the load function of the PB[0] output driver. **0x0:** 0 (4.5 mA) **0x1:** 1 (9 mA) **0x2:** 2 (13.5 mA) **0x3:** 3 (maximum) (18 mA) **Value After Reset:** 0x0

5.2.2.24. I/O Drive Strength Control Register 2 (IODSCR2)

The IODSCR2 register is at offset 0x5C.

The following table shows the register bit assignments.

Table 5-33 Fields for register: IODSCR2

Bits	Name	R/W	Description
[31:30]	PCDS15	R/W	This bit field controls the load function of the PC[15] output driver.
[29:28]	PCDS14	R/W	This bit field controls the load function of the PC[14] output driver.
[27:26]	PCDS13	R/W	This bit field controls the load function of the PC[13] output driver.
[25:24]	PCDS12	R/W	This bit field controls the load function of the PC[12] output driver.
[23:22]	PCDS11	R/W	This bit field controls the load function of the PC[11] output driver.
[21:20]	PCDS10	R/W	This bit field controls the load function of the PC[10] output driver.
[19:18]	PCDS9	R/W	This bit field controls the load function of the PC[9] output driver.
[17:16]	PCDS8	R/W	This bit field controls the load function of the PC[8] output driver.
[15:14]	PCDS7	R/W	This bit field controls the load function of the PC[7] output driver.
[13:12]	PCDS6	R/W	This bit field controls the load function of the PC[6] output driver.
[11:10]	PCDS5	R/W	This bit field controls the load function of the PC[5] output driver.
[9:8]	PCDS4	R/W	This bit field controls the load function of the PC[4] output driver.
[7:6]	PCDS3	R/W	This bit field controls the load function of the PC[3] output driver.
[5:4]	PCDS2	R/W	This bit field controls the load function of the PC[2] output driver.

Table 5-33 Fields for register: IODSCR2 (continued)

Bits	Name	R/W	Description
[3:2]	PCDS1	R/W	This bit field controls the load function of the PC[1] output driver.
[1:0]	PCDS0	R/W	This bit field controls the load function of the PC[0] output driver. • **0x0:** 0 (4.5 mA) • **0x1:** 1 (9 mA) • **0x2:** 2 (13.5 mA) • **0x3:** 3 (maximum) (18 mA) **Value After Reset:** 0x0

5.2.2.25. LDO Control Register (LDOCR)

The LDOCR register is at offset 0x60.

The following table shows the register bit assignments.

Table 5-34 Fields for register: LDOCR

Bits	Name	R/W	Description
[31:15]			Reserved
[14]	LDOF_PD	R/W	This bit field controls the LDO FLASH mode. **Values:** **0x0:** Normal mode **0x1:** Power Down mode **Value After Reset:** 0x0
[13:11]	LDOF_LDO_PI	R/W	This bit field controls the LDO FLASH protective current. **Values:** **0x0:** 290 mA **0x1:** 260 mA **0x2:** 225 mA **0x3:** 195 mA **0x4:** 410 mA **0x5:** 380 mA **0x6:** 350 mA **0x7:** 320 mA **Value After Reset:** 0x0
[10:8]	LDOF_ADJ	R/W	This bit field adjusts the LDO FLASH output voltage. **Values:** **0x0:** 1.70 V **0x1:** 1.75 V **0x2:** 1.60 V **0x3:** 1.65 V **0x4:** 1.90 V **0x5:** 1.95 V **0x6:** 1.85 V **0x7:** 1.80 V

Table 5-34 Fields for register: LDOCR (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0
[7:6]			Reserved
[5:3]	LDOC_LDO_PI	R/W	This bit field controls the LDO CORE protective current. **Values:** **0x0:** 500 mA **0x1:** 460 mA **0x2:** 420 mA **0x3:** 380 mA **0x4:** 660 mA **0x5:** 620 mA **0x6:** 580 mA **Value After Reset:** 0x0
[2:0]	LDOC_ADJ	R/W	This bit field adjusts the LDO CORE output voltage. **Values:** **0x0:** 1.10 V **0x1:** 1.15 V **0x2:** 1.00 V **0x3:** 1.05 V **0x4:** 1.30 V **0x5:** 1.35 V **0x6:** 1.20 V **0x7:** 1.25 V **Value After Reset:** 0x6

5.2.2.26. USB PHY Control and Status Register (USBCR)

The USBCR register is at offset 0x64.

The following table shows the register bit assignments.

Table 5-35 Fields for register: USBCR

Bits	Name	R/W	Description
[31:28]			Reserved ⚠ Do not change this field. **Value After Reset:** 0x1
[27]	DRVVBUSGPR	R/W	Alternative control of the I/O cell driver control signal when the alternative DRVVBUS function is selected **Value After Reset:** 0x0
[26]	DRVVBUSSRC	R/W	This bit selects the source of the I/O cell driver control signal when the alternative DRVVBUS function is selected. **Values:**

Table 5-35 Fields for register: `USBCR` (continued)

Bits	Name	R/W	Description
			0x0: DRVVBUS output **0x1:** `DRVVBUSGPR` **Value After Reset:** 0x0
[25]	`DRVVBUSPOL`	R/W	This bit selects the polarity of the I/O cell driver control signal when the alternative DRVVBUS function is selected. **Values:** **0x0:** Corresponds to the DRVVBUS output **0x1:** Inversion of the DRVVBUS output **Value After Reset:** 0x0
[24]	`DRVVBUSVAL`	R/W	I/O cell value when the alternative DRVVBUS function is selected **Values:** **0x0:** Logical 0 **0x1:** Logical 1 **Value After Reset:** 0x0
[23:17]			Reserved
[16]	`BIST_STAT`	R	Result of the USB PHY BIST operation **Values:** **0x0:** BIST completed with an error **0x1:** BIST completed without errors **Value After Reset:** 0x0
[15]	`IDPULLUPGPR`	R/W	Alternative control of the USB PHY `id_pullup` signal **Value After Reset:** 0x0
[14]	`IDPULLUPSRC`	R/W	USB PHY ID_PULLUP input source: **Values:** **0x0:** Microcontroller IDPULLUP output **0x1:** `IDPULLUPGPR` **Value After Reset:** 0x0
[13]	`IDPULLUPPOL`	R/W	USB PHY ID_PULLUP input polarity **Values:** **0x0:** Corresponds to microcontroller IDPULLUP output **0x1:** Inversion of the microcontroller IDPULLUP output **Value After Reset:** 0x0
[12:10]			Reserved
[9]	`TX_SE0`	R/W	This bit controls the TX_SE0 input of the USB PHY. This bit enables the SE0 signal: **0x0:** Normal operation **0x1:** SE0 is set on D+/D- **Value After Reset:** 0x0
[8]	`TX_DAT`	R/W	This bit controls the TX_DAT input of the USB PHY. **Value After Reset:** 0x0

Table 5-35 Fields for register: `USBCR` (continued)

Bits	Name	R/W	Description
[7]	`TX_ENABLE_N`	R/W	This bit controls the TX_ENABLE_N input of the USB PHY. **Value After Reset:** 0x0
[6]	`FSLS_SERIALMODE`	R/W	This bit controls the FSLS_SERIALMODE input of the USB PHY. This bit enables serial packet transmission: **0x0:** FS packets are transmitted over the parallel interface **0x1:** FS and LS packets are transmitted over the serial interface **Value After Reset:** 0x0
[5]	`OTG_SUSPENDM`	R/W	This bit controls the OTG_SUSPENDM input of the USB PHY. This bit enables OTG/HOST functions: **0x0:** OTG is disabled **0x1:** OTG is enabled **Value After Reset:** 0x0
[4]	`REFCLK_MODE`	R/W	This bit controls the REFCLK_MODE input of the USB PHY. This bit selects the USP PHY reference clock frequency: **0x0:** 25 MHz **0x1:** 12 MHz ⚠ It is recommended to use the 25 MHz reference clock frequency. **Value After Reset:** 0x0
[3]	`BIST_EN`	R/W	This bit enables the USB PHY BIST operation: **0x0:** BIST is disabled **0x1:** BIST is enabled **Value After Reset:** 0x0
[2]	`PLL_EN`	R/W	This bit enables the USB PHY PLL: **0x0:** PLL is disabled **0x1:** PLL is enabled **Value After Reset:** 0x0
[1]	`RESET`	R/W	This bit controls the RESET input of the USB PHY: **Values:** **0x0:** RESET is inactive **0x1:** RESET is active **Value After Reset:** 0x1
[0]			Reserved (read as 0)

5.2.2.27. System Status Register (`SYSSR0`)

The `SYSSR0` register is at offset 0x70.

The following table shows the register bit assignments.

Table 5-36 Fields for register: SYSSR0

Bits	Name	R/W	Description
[31:13]			Reserved
[12]	NPOR_LDOF	R	NPOR pin of LDO_FLASH block
[11]	PLL_NLOCK	R	The event flag that indicates that the PLL has exited the steady state. **Values:** **0x0:** No event **0x1:** Event **Value After Reset:** 0x0
[10]	PLL_FBSLIP	R	FBSLIP PLL output
[9]	PLL_RFSLIP	R	RFSLIP PLL output
[8]	WDTHIT	R	This bit indicates that wdt0_sys_rst_n is active. **Values:** **0x0:** Inactive **0x1:** Active WDTHIT is reset to 0x0 by writing 1 to the SYSCR0.WDTHITCLR **Value After Reset:** 0x0
[7]	POR_OK	R	This bit indicates POR_OK input status. **Value After Reset:** N/A
[6:5]			Reserved
[4:0]	STRAP	R	The values of triggers that store the state of the strap pins {PC[8], PC[6], PC[4], PB[10], PA[10]}. For more information about the strap pins, see section **3.9.3 Strap Pins** of the ***BE-U1000 Datasheet***. **Value After Reset:** N/A

5.2.2.28. WDT Clear Key Register (WDTCLRKEY)

The WDTCLRKEY register is at offset 0x74.

The following table shows the register bit assignments.

Table 5-37 Fields for register: WDTCLRKEY

Bits	Name	R/W	Description
[31:0]	WDTCLRKEY	R/W	The code word to enable WDT_0 reset by settting 0 to the PCLK0EN.WDT0RSTN and WDT_1 reset by settting 0 to the PCLK1EN.WDT1RSTN. WDT_0 is reset by the PCLK0EN.WDT0RSTN and WDT_1 is reset by the PCLK1EN.WDT1RSTN if the WDTCLRKEY is set to 0xD09C1EA7. **Value After Reset:** 0x0

5.2.2.29. Interrupt Source Control Register (INTCR0)

The INTCR0 register is at offset 0x80.

The following table shows the register bit assignments.

Table 5-38 Fields for register: `INTCR0`

Bits	Name	R/W	Description
[31:28]			Reserved
[27:24]	`PADCINTSEL2`	R/W	This bit field selects the int_local[4] interrupt (`pc_irq`) source of the Core 2. **Value After Reset:** N/A
[23:20]	`PADBINTSEL2`	R/W	This bit field selects the int_local[3] interrupt (`pb_irq`) source of the Core 2. **Value After Reset:** N/A
[19:16]	`PADAINTSEL2`	R/W	This bit field selects the int_local[2] interrupt (`pa_irq`) source of the Core 2. **Value After Reset:** N/A
[15:12]	`PADCINTSEL0`	R/W	This bit field selects the int_local[73] interrupt (`pc_irq`) source of the Core 0 and Core 1. **Value After Reset:** N/A
[11:8]	`PADBINTSEL0`	R/W	This bit field selects the int_local[72] interrupt (`pb_irq`) source of the Core 0 and Core 1. **Value After Reset:** N/A
[7:4]	`PADAINTSEL0`	R/W	This bit field selects the int_local[71] interrupt (`pa_irq`) source of the Core 0 and Core 1. **Value After Reset:** N/A
[3:0]			Reserved

5.2.2.30. Clock Control Register 1 (`CLKCR1`)

The `CLKCR1` register is at offset 0x84.

The following table shows the register bit assignments.

Table 5-39 Fields for register: `CLKCR1`

Bits	Name	R/W	Description
[31:12]			Reserved
[11]	`CANCLKDIVEN`	R/W	Enables the CANCLKX2_DIV divider: **Values:** **0x0:** Divider is disabled **0x1:** Divider is enabled **Value After Reset:** 0x1
[10:6]	`CANCLKDIV`	R/W	CANCLKX2_DIV divisor value. **Values:**

Table 5-39 Fields for register: `CLKCR1` (continued)

Bits	Name	R/W	Description
			0x0: `CANCLKDIV[0]` = 0 — integer division. The frequency division ratio is equal to `CANCLKDIV[4:1]`+1. The valid values of the `CANCLKDIV[4:1]` range from 0 to 15. If `CANCLKDIV[4:0]` = 0 — bypass. **0x1:** `CANCLKDIV[0]` = 1 — fractional division. The frequency division ratio is equal to `CANCLKDIV[4:1]`+1.5. The valid values of the `CANCLKDIV[4:1]` range from 0 to 14. **Value After Reset:** 0x0
[5]	`TCLKDIVEN`	R/W	This bit enables the TCLK_DIV divider: **Values:** **0x0:** Divider is disabled **0x1:** Divider is enabled **Value After Reset:** 0x1
[4:0]	`TCLKDIV`	R/W	TCLK_DIV divisor value. **Values:** **0x0:** `TCLKDIV[0]` = 0 — integer division. The frequency division ratio is equal to `TCLKDIV[4:1]`+1. The valid values of the `TCLKDIV[4:1]` range from 0 to 15. If `TCLKDIV[4:0]` = 0 — bypass. **0x1:** `TCLKDIV[0]` = 1 — fractional division. The frequency division ratio is equal to `TCLKDIV[4:1]`+1.5. The valid values of the `TCLKDIV[4:1]` range from 0 to 14. **Value After Reset:** 0x0

5.2.2.31. Flash NVR Control Register (`FLASHNVRCR`)

The `FLASHNVRCR` register is at offset 0x90.

The following table shows the register bit assignments.

Table 5-40 Fields for register: `FLASHNVRCR`

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	`NVR1DIS`	R/WS1	FLASH NVR 1 lock flag. This bit disables the modification of the 1st block of the NVR array (sectors 0 and 1). **Values:** **0x0:** Modification of the 1st NVR block is enabled **0x1:** Modification of the 1st NVR block is disabled **Value After Reset:** 0x0

5.2.2.32. Core 1 Control Register (`CORE1CR`)

The `CORE1CR` register is at offset 0x94.

The following table shows the register bit assignments.

Table 5-41 Fields for register: `CORE1CR`

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	`CORE1ON`	R/WS0	This bit enables the Core 1. **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x1

6. RISC-V Cores

This chapter covers the following topics:

- BR-350 Core
- BM-310 Core

6.1. BR-350 Core

This section describes the BR-350 Core of the BE-U1000 microcontroller. The BE-U1000 contains two BR-350 Cores (Core 0 and Core 1).

The BR-350 Core is the 32-bit RISC-V Core with 32 integer registers. The BR-350 Core supports two privilege levels: machine (M) and user (U). U-mode provides a mechanism to isolate application processes from each other and from trusted code running in M-mode.

The BR-350 Core also supports processing of the Non-Maskable Interrupts (int_nmi) , Physical Memory Protection mechanism and Hardware Performance Monitor. For more information about the BR-350 Core CSRs, refer to the Core CSRs section.

A simplified block diagram of the BR-350 Core is shown in the following figure.

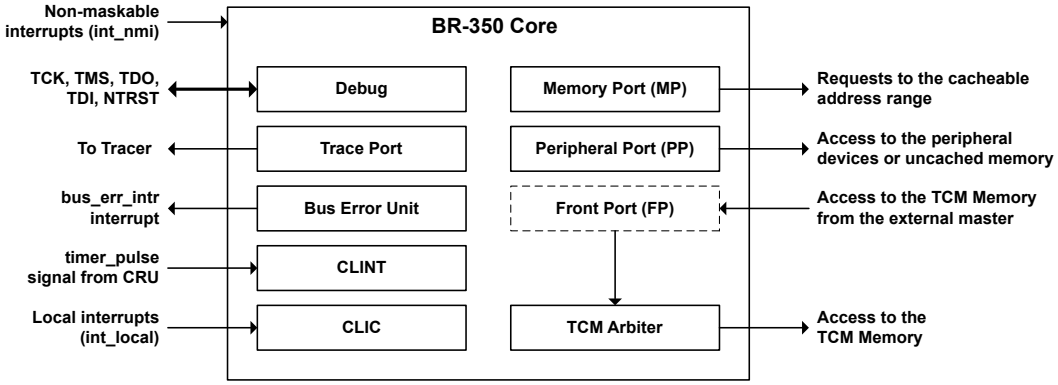

Figure 6-1 BR-350 Core block diagram

As the figure above shows, BR-350 Core includes the following blocks:

- CLIC and CLINT: Used to handle interrupts.
- Memory Port: Used by the BR-350 Core to issue requests to the cacheable address range.
- Peripheral Port: Used by the BR-350 Core to access peripheral devices or uncached memory.
- Trace Port: Used to provide information about executed instructions to the external Tracer.
- Front Port: Used by the external master device to access the TCM Memory via the TCM Arbiter.

  ⚠ Front Port of the BR-350 Core 1 is not used.

- TCM Arbiter: Used to arbitrate access requests to the TCM Memory.
- Bus Error Unit: Used to generate an interrupt (`bus_err_intr`) when a bus error condition is detected.
- Debug: Used to provide external debug access to the core via the JTAG.

6.1.1. BR-350 Core Functional Description

6.1.1.1. Non-Maskable Interrupts (int_nmi)

The *Non-Maskable Interrupts* (**NMIs**) are connected to the core's command execution subsystem. NMIs cannot be disabled programmatically and can be used only for processing the hardware errors (see section **Non-Maskable Interrupts** of the ***RISC-V Instruction Set Manual, Volume II: Privileged Architecture*** for details).

When an NMI is generated, the execution flow is redirected to the address stored in the `NMITVEC` register, regardless of the current state of other CSRs. The hardware performs the following actions:

1. The privilege mode changes to the M-mode.
2. The `MEPC` CSR is written with the virtual address of the interrupted instruction.
3. The active NMI ID is written itto the lower bits of the `MCAUSE` CSR.

`MCAUSE` register format when an NMI occurs complies with the ***Core-Local Interrupt Controller (CLIC) RISC-V Privileged Architecture Extension*** specification.

4. `NMIE` is reset to 0, blocking active interrupts acceptance from `int_nmi`. `NMIE` can be set to 1 only after reset or after execution the `MRET` command.

The exit from an NMI handler using the `MRET` instruction returns the execution flow to the address stored in the `MEPC` register. However, NMIs are fatal because the state of the CSRs may be overwritten (for example, if an NMI occurs during the execution of an M-mode interrupt handler). Therefore, the behavior after exiting an NMI handler is undefined and it is recommended to perform a system reset.

6.1.1.2. Physical Memory Protection

The processor core supports the *Physical Memory Protection* (**PMP**) mechanism. The PMP includes eigh CSRs with the granularity access of 8 bytes. Refer to section **Physical Memory Protection** of the ***RISC-V Instruction Set Manual, Volume II: Privileged Architecture*** for details.

6.1.1.3. Hardware Performance Monitor

The processor core supports the *Hardware Performance Monitor* (**HMP**). Refer to section **Hardware Performance Monitor** of the ***RISC-V Instruction Set Manual, Volume II: Privileged Architecture*** for details.

The processor core implements four event counters. This means that `MHPMEVENT3` - `MHPMEVENT6` and `MHPMCOUNTER3` - `MHPMCOUNTER6` CSRs are implemented.

The following table lists the events supported by the core. Writing an unsupported event sets the `MHPMEVENTx` register to 0 (This corresponds to NO_EVENT).

Table 6-1 HPM events

Identifier	Name	Description
0	NO_EVENT	No event
1	CBRD	Number of executed direct branches
2	CBRI	Number of executed indirect branches
3	CBRC	Number of executed conditional branches

Table 6-1 HPM events (continued)

Identifier	Name	Description
4	CBRR	Number of executed ret instructions
5	MPBRD	Number of mispredicts for direct branches at execute stage
6	MPBRI	Number of mispredicts for indirect branches at execute stage
7	MPBRC	Number of mispredicts for conditional branches at execute stage
8	MPBRR	Number of mispredicts for ret instructions at execute stage
9	PIPEN	Number of cycles when execution pipeline was stalled
10	IFQE	Number of cycles when IFQ was empty
11	SCSUB	Number of cycles when execution pipeline was blocked due to execution of exclusive instruction (SCSU)
12	RAWD	Number of cycles when dependency 'Read-After-Write' was asserted
Frontend unit events		
14	FE.FR101	Number of fetch redirects by main branch predictor
15	FE.FR102	Number of fetch redirects by static branch predictor
16	FE.FJ102	Number of fetches when cell became invalidated in the main branch predictor
17	FE.BICU	Number of cycles when frontend is blocked by ICU
19	FE.BNONE	Number of cycles when fetch request will be repeated after receiving block with type «None»
20	FE.BVR	Number of cycles when fetch request not acked by ICU (void request)
21	FE.BFI	Number of cycles when fetch request is avoided due to fence.i processing
22	FE.IFQF	Number of cycles when fetch request is avoided due to IFQ is full
23	FE.RQ2	Number of non-blocked fetch requests with address aligned to 2 bytes
24	FE.FUSEV0	The number of cycles in which macro-op fusion optimization was applied at the interface between the code fragment swapping subsystem and the command execution subsystem (port 0)
25	FE.FUSEV1	The number of cycles in which macro-op fusion optimization was applied at the interface between the code fragment swapping subsystem and the command execution subsystem (port 1)
26	FE.FUSEV2	The number of cycles in which macro-op fusion optimization was applied at the interface between the code fragment swapping subsystem and the command execution subsystem (port 2)

Table 6-1 HPM events (continued)

Identifier	Name	Description
27	FE.FUSEV3	The number of cycles in which macro-op fusion optimization was applied at the interface between the code fragment swapping subsystem and the command execution subsystem (port 3)
28	FE.VNRDY0	The number of cycles in which the code fragment swap subsystem had a command ready, but the execution subsystem was not ready to accept it (port 0)
29	FE.VNRDY1	The number of cycles in which the code fragment swap subsystem had a command ready, but the execution subsystem was not ready to accept it (port 1)
30	FE.VNRDY2	The number of cycles in which the code fragment swap subsystem had a command ready, but the execution subsystem was not ready to accept it (port 2)
31	FE.VNRDY3	The number of cycles in which the code fragment swap subsystem had a command ready, but the execution subsystem was not ready to accept it (port 3)
32	PIO.L2IO	Number of fetch requests to IO region
Instruction cache unit events		
33	IC.RQS	Total number of fetch requests to cacheable region
34	IC.KILLED	Number of requests killed by frontend subsystem before completion
35	IC.FBKID	Number of fetches blocked due to miss to Instruction cache
36	IC.FBKGEN	Number of fetches blocked due to no vacant entry in ICU Request buffer
37	IC.FBKNONE	Number of fetches blocked due to hazards inside ICU
38	IC.L2RD	Number of L2C read requests
39	IC.L2SR	Number of L2C snoop responses
40	IC.SVPARERR	Number of parity errors in validity flags
41	IC.TPARERR	Number of parity errors in tags
42	IC.DPARERR	Number of parity errors in data

6.1.1.4. Debug

Debug subsystem is compliant with the ***RISC-V External Debug Support Version 0.13***. Supported features are:

- Read/write GPRs via abstract commands (see section **Abstract Commands** in the ***RISC-V External Debug Support Version 0.13***)
- Read/Write 64 bytes program buffer (see section **Program Buffer** in the ***RISC-V External Debug Support Version 0.13***)
- 4 abstract data registers (see section **Abstract Commands** in the ***RISC-V External Debug Support Version 0.13***)

- 4 hardware match triggers (see chapter **Trigger Module** in the ***RISC-V External Debug Support Version 0.13***)
- Debug ROM

The figure below shows the block diagram of the Debug subsystem.

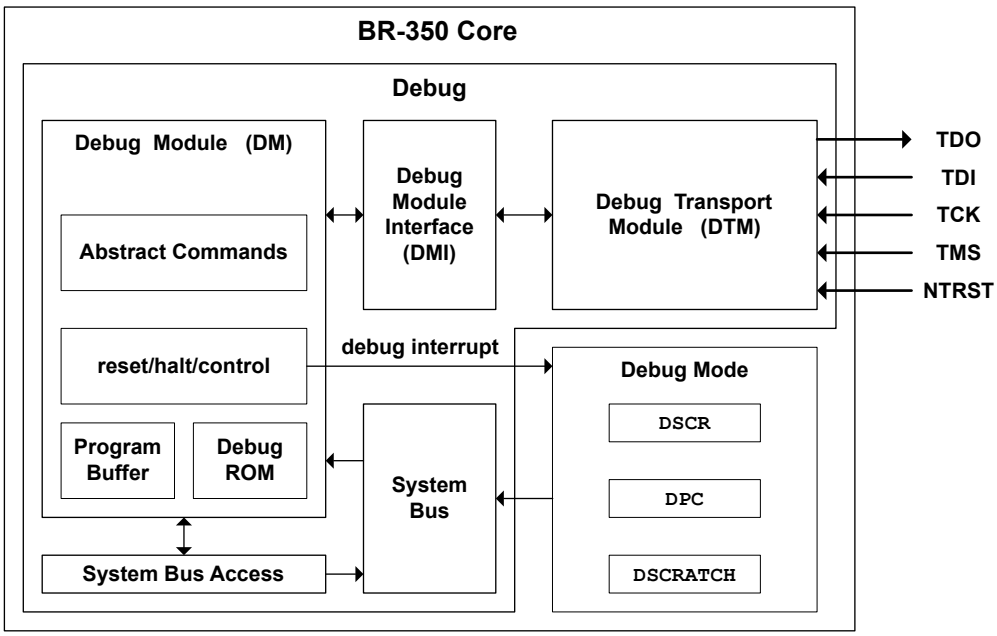

Figure 6-2 Debug subsystem

Each signal ID, shown in the right part of the diagram above, corresponds to the appropriate I/O pin ID. For more information, refer to the ***BE-U1000 Datasheet***.

JTAG can be software emulated by the BM-310 Core. For details, refer to the Core 2 Software JTAG section.

Debug subsystem consists of *Debug Transport Module* (**DTM**) and *Debug Module* (**DM**). DTM connects to DM via *Debug Module Interface* (**DMI**).

DM access via JTAG is done by reading and writing to DTM register named `DMI` (see DTM Registers for details). Refer to the DM Registers (Access via DMI) section for details on the DM registers that are accessed over the DMI bus.

6.1.1.4.1. Core Debug Mode

Debug Mode is a special processor mode used only when a core is halted for external debugging (see also section **4.1 Debug Mode** of the ***RISC-V External Debug Support Version 0.13***). The core also has the debug *Control and Status Registers* (**CSRs**). The core is able to read and write to debug CSRs only when it is operating in Debug Mode. Refer to the Core Debug CSRs section for details on these registers.

The core processor interacts with the *Debug Module* (**DM**) only when in Debug Mode. In this mode, the core reads from and writes to the 4 KB Core Debug block (0000_0000 - 0000_0FFF) and does required actions. The table below shows the memory map of the Core Debug block.

Table 6-2 Memory map of the Core Debug block

Name	Offset	Description	R/W
DM_ZERO	0x0	Safe zero	R/W
DM_HALT_ACK	0x100	Halted ack	W
DM_GO_ACK	0x104	Going ack	W
DM_RESUME_ACK	0x108	Resume ack	W
DM_EXCEPTION	0x10C	Exception	W
DM_WHERETO	0x300	Where to	R
DM_ABST0	0x338	Abstract command 0	R
DM_ABST1	0x33C	Abstract command 1	R
DM_PBUF	0x340 - 0x37C	Program buffer	R/W
DM_ABSDATA	0x380 - 0x38C	Abstract data	R/W
DM_CPU_FLAGS	0x400	CPU flags	R
DM_DROM	0x800 - 0xFFF	Debug ROM	R

When the core switches to the Debug Mode, the control flow continues execution from the beginning of the Debug ROM (address 0x800). The program stored in the Debug ROM does polling of the CPU Flags (address 0x400).

The CPU Flags are described below in the ***CPU flags (DM_CPU_FLAGS) description*** table.

If the `Going` flag was read, it means that abstract command execution request is received and control flow redirects execution to the Abstract command 0 address (0x338). If an exception occurs during the execution, the control flow redirects execution to the exception handler inside the Debug ROM. This handler perform a write operation to the address 0x10C (Exception) of the Debug Module. Then an external debugger can read the Abstract Control and Status register (`ABSTRACTCS`) and determine the fact that an exception occurred during the execution.

If the `Resumereq` flag was read, it means that resume request is received. After resume request is received, the core executes the rest of the Debug ROM program and continues execution from the address stored in the `DPC` CSR.

Table 6-3 CPU flags (`DM_CPU_FLAGS`) description

Bits	Name	R/W	Description
[1]	Resumereq	R	Resumereq flag. When core operates in Debug Mode and reads non-zero value from this field it executes program from the Debug ROM till `dret` instruction and continue execution from the virtual address stored in `DPC` CSR.

Table 6-3 CPU flags (`DM_CPU_FLAGS`) description (continued)

Bits	Name	R/W	Description
[0]	`Going`	R	Going flag. When core operates in Debug Mode and reads non-zero value from this field it jumps to the first abstract command address (0x338).

6.1.1.4.2. Trigger Module

The core implements a Trigger Module to use hardware breakpoints. It provides four match triggers that can halt core execution when a specified condition is met — either on an instruction address match or on a memory access address match. The triggers can be chained, allowing a halt to occur only when multiple match conditions are met. The Trigger Module is controlled through the *Control and Status Registers* (**CSRs**) that are given in the Trigger Module CSRs section. For more information, refer to the **Trigger Module** chapter of the ***RISC-V External Debug Support Version 0.13***.

6.1.1.5. CLINT

According to the RISC-V Privileged ISA Specification, the core must provide support for the machine software interrupts and machine timer interrupts. The *Core Local Interruptor* (**CLINT**) is responsible for generating requests (signals) for these interrupts.

The CLINT includes memory-mapped registers that are described in the CLINT Registers section.

6.1.1.6. Bus Error Unit

The BR-350 Core contains the *Bus Error Unit* (**BEU**). The BEU generates an interrupt (`bus_err_intr`) upon detecting an error in the processor core module (for example, when a write to the I/O region receives an error response from the target slave).

The BEU includes memory-mapped registers that are described in the Bus Error Unit Registers section.

6.1.1.7. CLIC

The BR-350 Core contains the *Core Local Interrupt Controller* (**CLIC**) which is designed to handle local interrupts (int_local). The CLIC implementation corresponds to the ***Core-Local Interrupt Controller (CLIC) RISC-V Privileged Architecture Extension***.

The CLIC includes memory-mapped registers that are described in the CLIC Registers section and *Control and Status Registers* (**CSRs**) that can be acessed via csr instructions and are described in the CLIC CSRs section.

6.1.2. BR-350 Core Registers

This section covers the following topics:

- Control and Status Registers
- DTM Registers
- DM Registers (access via DMI)
- CLINT Registers

- Bus Eror Unit Registers
- CLIC Registers

6.1.2.1. Control and Status Registers

This section describes the *Control and Satus Registers* (**CSRs**) that can be acessed via **csr instructions**. For more information about CSRs, refer to the **Control and Status Registers (CSRs)** chapter of the ***RISC-V Instruction Set Manual, Volume II: Privileged Architecture*** specification.

6.1.2.1.1. CLIC CSRs

6.1.2.1.1.1. CSRs Map

The following table provides a top-level summary of the CLIC *Control and Status Registers* (**CSRs**). Refer to **Chapter 5. CLIC CSRs** of the ***Core-Local Interrupt Controller (CLIC) RISC-V Privileged Architecture Extension*** for details on registers mentioned below.

Table 6-4 CLIC CSRs

Register	Offset	Description
User Mode (U-Mode)		
`UTVT`	0x7	Trap-handler vector table base address
`UNXTI`	0x45	Interrupt handler address and enable modifier
`UINTSTATUS`	0x46	Current interrupt levels
`UINTTHRESH`	0x47	Interrupt-level threshold
`USCRATCHCSW`	0x48	Conditional scratch swap on priv mode change
`USCRATCHCSWL`	0x49	Conditional scratch swap on level change
Machine Mode (M-Mode)		
`MTVT`	0x307	Trap-handler vector table base address
`MNXTI`	0x345	Interrupt handler address and enable modifier
`MINTSTATUS`	0x346	Current interrupt levels
`MINTTHRESH`	0x347	Interrupt-level threshold
`MSCRATCHCSW`	0x348	Conditional scratch swap on priv mode change
`MSCRATCHCSWL`	0x349	Conditional scratch swap on level change

6.1.2.1.2. Trigger Module CSRs

6.1.2.1.2.1. CSRs Map

The following table provides a top-level summary of the Trigger Module *Control and Status Registers* (**CSRs**). Refer to **5.2 Trigger Registers** of the ***RISC-V External Debug Support Version 0.13*** for details on registers mentioned below.

Table 6-5 Trigger Module CSRs

Register	Offset	Description
TSELECT	0x7A0	Trigger Select
MCONTROL	0x7A1	Match Control
TDATA2	0x7A2	Trigger Data 2
TINFO	0x7A4	Trigger Info

6.1.2.1.3. Core Debug CSRs

6.1.2.1.3.1. CSRs Map

The following table provides a top-level summary of the Core Debug *Control and Status Registers* (**CSRs**). Refer to **4.8 Core Debug Registers** of the ***RISC-V External Debug Support Version 0.13*** for details on registers mentioned below.

Processor core is able to read and write to debug CSRs only when it is operating in Debug Mode.

Table 6-6 Core Debug CSRs

Register	Offset	Description
DSCR	0x7B0	Debug Control and Status
DPC	0x7B1	Debug PC
DSCRATCH	0x7B2	Debug Scratch Register

6.1.2.1.4. Core CSRs

6.1.2.1.4.1. CSRs Map

Implementation of the BR-350 Core *Control and Status Registers* (**CSRs**) corresponds to the ***RISC-V Instruction Set Manual, Volume II: Privileged Architecture*** specification. The following table contains only implementation specific CSRs. To view the detailed description of a register, click the register name in the **Description** column.

Table 6-7 Core CSRs

Register	Offset	Description
NMITVEC	0x7C0	Machine NMI Trap Handler Address
XCCFG	0xBC8	Core Configuration
XICFG	0xBD8	Instruction Cache Configuration

6.1.2.1.4.2. CSR Descriptions

6.1.2.1.4.2.1. Machine NMI Trap Handler Address (NMITVEC)

The NMITVEC register is at offset 0x7C0.

Writing to and reading from this register is only possible from the M privileged mode.

The following table shows the register bit assignments.

Table 6-8 Fields for register: NMITVEC

Bits	Name	Access	Description
[31:0]	NMITVEC	WARL	NMI handler address This address must be 4-byte aligned. **Value After Reset:** 0x0

6.1.2.1.4.2.2. Core Configuration (XCCFG)

The XCCFG register is at offset 0xBC8.

This register is only acessible from the M privileged mode.

The following table shows the register bit assignments.

Table 6-9 Fields for register: XCCFG

Bits	Name	R/W	Description
[2]	WBCLR	R/W	When this bit is 1, validities of branch predictor structures will be cleared after *Wait for Interrupt* (**WFI**) instruction commit. **Value After Reset:** 0x0
[1]	FBCLR	R/W	When this bit is 1, validities of branch predictor structures will be cleared after fence.i commit. **Value After Reset:** 0x0
[0]	WFI	R/W	When this bit is 1, WFI instruction will be treated as NOP, i.e. core will not be switched to the WFI low-power state. **Value After Reset:** 0x0

6.1.2.1.4.2.3. Instruction Cache Configuration (XICFG)

The XICFG register is at offset 0xBD8.

This register is only acessible from the M privileged mode.

The following table shows the register bit assignments.

Table 6-10 Fields for register: XICFG

Bits	Name	R/W	Description
[0]	WFIINV	R/W	When this bit is 1, instruction cache will reset validities for all cache lines at exit from the WFI state.

Table 6-10 Fields for register: `XICFG` (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0

6.1.2.2. DTM Registers

This section describes the *Debug Transport Module* (**DTM**) Registers acessible via JTAG.

6.1.2.2.1. Register Map

The following table provides top-level summary of each register. Refer to section **6.1.2 JTAG DTM Registers** of the ***RISC-V External Debug Support Version 0.13*** for details on registers mentioned below.

Table 6-11 JTAG DTM registers

Register	Offset	Description
`BYPASS`	0x0	BYPASS
`IDCODE`	0x1	IDCODE
`DTMCS`	0x10	DTM Control and Status
`DMI`	0x11	Debug Module Interface Access
`BYPASS`	0x12 - 0x1F	BYPASS

6.1.2.3. DM Registers (access via DMI)

The *Debug Module* (**DM**) registers described in this section are accessed over the *Debug Module Interface* (**DMI**) bus.

6.1.2.3.1. Register Map

The following table provides top-level summary of each register. Refer to section **3.12 Debug Module Registers** of the ***RISC-V External Debug Support Version 0.13*** for details on registers mentioned below.

Table 6-12 DM registers (access via DMI)

Register	Offset	Description
`DATAi`	0x4 + (i*0x1)	Abstract Data i (for i = 0; i <= 3)
`DMCONTROL`	0x10	Debug Module Control
`DMSTATUS`	0x11	Debug Module Status
`HARTINFO`	0x12	Hart Info
`ABSTRACTCS`	0x16	Abstract Control and Status
`COMMAND`	0x17	Abstract Command

Table 6-12 DM registers (access via DMI) (continued)

Register	Offset	Description
ABSTRACTAUTO	0x18	Abstract Command Autoexec
PROGBUFi	0x20 + (i*0x1)	Program Buffer i (for i = 0; i <= 15)
SBADDRESS3	0x37	System Bus Address [127:96]
SBCS	0x38	System Bus Access Control and Status
SBADDRESS0	0x39	System Bus Address [31:0]
SBADDRESS1	0x3A	System Bus Address [63:32]
SBADDRESS2	0x3B	System Bus Address [95:64]
SBDATA0	0x3C	System Bus Data [31:0]
SBDATA1	0x3D	System Bus Data [63:32]
SBDATA2	0x3E	System Bus Data [95:64]
SBDATA3	0x3F	System Bus Data [127:96]

6.1.2.4. CLINT Registers

The register region of the *Core Local Interruptor* (**CLINT**) is mapped in the BE-U1000 microcontroller memory map as the following table shows.

Table 6-13 Memory address region for CLINT registers

Region	Block Name	Size	Start Address	End Address
Core 0 and Core 1 Resources	CLINT	64 KB	0x0200_0000	0x0200_FFFF

For details, refer to Memory Map chapter.

6.1.2.4.1. Register Map

The following table provides a top-level summary of each register. To view the detailed description of a register, click the register name in the **Description** column.

Table 6-14 CLINT register map

Register	Offset	Description
MSIP0	0x0	Machine Software Interrupt Pending
MTIMECMP0	0x4000	Machine Timer Compare
MTIME	0xBFF8	Machine Timer Value

Table 6-14 CLINT register map (continued)

Register	Offset	Description
		⚠ Timer value is increased by the `timer_pulse` signal from the Control Registers Unit.

6.1.2.4.2. Register Descriptions

6.1.2.4.2.1. Machine Software Interrupt Pending (`MSIP0`)

The `MSIP0` register is at offset 0x0.

Machine software interrupt pending (`MSIP0`) bit could be set/cleared via corresponding memory-mapped register access. This bit is directly mapped to corresponding core input. Access to this memory-mapped register is only supported using 32-bit operations.

The following table shows the register bit assignments.

Table 6-15 Fields for register: `MSIP0`

Bits	Name	R/W	Description
[31:1]			These bits are wired to zero
[0]	`MSIP0`	R/W	Machine software interrupt pending **Value After Reset:** 0x0

6.1.2.4.2.2. Machine Timer Compare (`MTIMECMP0`)

The `MTIMECMP0` register is at offset 0x4000.

This register contains 64-bit timer compare value. The machine timer interrupt (`mtip`) is asserted if timer value is greater than timer compare value.

The following table shows the register bit assignments.

Table 6-16 Fields for register: `MTIMECMP0`

Bits	Name	R/W	Description
[63:32]	`MTIMECMP0_HI`	R/W	High part of timer compare value **Value After Reset:** 0x0
[31:0]	`MTIMECMP0_LO`	R/W	Low part of timer compare value **Value After Reset:** 0x0

6.1.2.4.2.3. Machine Timer Value (`MTIME`)

The `MTIME` register is at offset 0xBFF8.

This register contains 64-bit timer value. The value is incremented each time by the `timer_pulse` signal from the Control Registers Unit.

The following table shows the register bit assignments.

Table 6-17 Fields for register: `MTIME`

Bits	Name	R/W	Description
[63:32]	`MTIME_HI`	R	High part of timer value **Value After Reset:** 0x0
[31:0]	`MTIME_LO`	R	Low part of timer value **Value After Reset:** 0x0

6.1.2.5. Bus Error Unit Registers

Registers region of the *Bus Error Unit* (**BEU**) mapped in the BE-U1000 microcontroller memory map as the following table shows.

Table 6-18 Memory address region for Bus Error Unit registers

Region	Block Name	Size	Start Address	End Address
Core 0 and Core 1 Resources	Bus Error Unit	4 KB	0x0202_0000	0x0202_0FFF

For details, refer to Memory Map chapter.

6.1.2.5.1. Register Map

The following table provides a top-level summary of each register. To view the detailed description of a register, click the register name in the **Description** column.

The Bus Error Unit registers can only be acessible from the Machine privelege mode (M-mode).

Table 6-19 Bus Error Unit register map

Register	Offset	Description
`STATUS_CPU`	0x0	CPU Status Register
`CONTROL_CPU`	0x8	CPU Interrupt Control Register
`ADDR_LO_CPU`	0x10	CPU Error Address Low Register
`ADDR_HI_CPU`	0x14	CPU Error Address High Register

6.1.2.5.2. Register Descriptions

In the tables below, the **R/W** column specifies how the application and the core can access the register fields.

6.1.2.5.2.1. CPU Status Register (`STATUS_CPU`)

The `STATUS_CPU` register is at offset 0x0.

The following table shows the register bit assignments.

Table 6-20 Fields for register: `STATUS_CPU`

Bits	Name	R/W	Description
[1]	`TYP`	R	Error Type **Values:** **0x0:** An error while recording **0x1:** Reserved **Value After Reset:** 0x0
[0]	`IRQ`	R/W1C	A value of 1 means that an interrupt is set to signal an error in the processor core. Write 0x1 to this bit to reset the setted interrupt. **Value After Reset:** 0x0

6.1.2.5.2.2. CPU Interrupt Control Register (`CONTROL_CPU`)

The `CONTROL_CPU` register is at offset 0x8.

The following table shows the register bit assignments.

Table 6-21 Fields for register: `CONTROL_CPU`

Bits	Name	R/W	Description
[0]	`IRQ_EN`	R/W	Enables the processor core error interrupt. **Value After Reset:** 0x1

6.1.2.5.2.3. CPU Error Address Low Register (`ADDR_LO_CPU`)

The `ADDR_LO_CPU` register is at offset 0x10.

The following table shows the register bit assignments.

Table 6-22 Fields for register: `ADDR_LO_CPU`

Bits	Name	R/W	Description
[31:0]	`ADDR_LO`	R	The lower part of the address where an error occurred while reading/writing (for processor core). This bit field is updated only if the `IRQ` bit of the `STATUS_CPU` register is set to 0x0. **Value After Reset:** N/A

6.1.2.5.2.4. CPU Error Address High Register (`ADDR_HI_CPU`)

The `ADDR_HI_CPU` register is at offset 0x14.

The following table shows the register bit assignments.

Table 6-23 Fields for register: `ADDR_HI_CPU`

Bits	Name	R/W	Description
[31:0]	`ADDR_HI`	R	The higher part of the address where an error occurred while reading/writing (for processor core).

Table 6-23 Fields for register: `ADDR_HI_CPU` (continued)

Bits	Name	R/W	Description
			This bit field is updated only if the `IRQ` bit of the `STATUS_CPU` register is set to 0x0. **Value After Reset:** N/A

6.1.2.6. CLIC Registers

Register region of the *Core Local Interrupt Controller* (**CLIC**) mapped in the BE-U1000 microcontroller memory map as the following table shows.

CLIC also have *Control and Status Registers* (**CSRs**) that are located in a separate address region and can be acessed via csr instructions. Refer to the CLIC CSRs section for details on these CSRs.

Table 6-24 Memory address region for CLIC registers

Region	Block Name	Size	Start Address	End Address
Core 0 and Core 1 Resources	CLIC	20 KB	0x0D00_0000	0x0D00_4FFF

For details, refer to Memory Map chapter.

6.1.2.6.1. Register Map

The following table provides a top-level summary of each register. Refer to **Chapter 4. CLIC Memory-Mapped Registers** of the ***Core-Local Interrupt Controller (CLIC) RISC-V Privileged Architecture Extension*** for details on registers mentioned below.

Table 6-25 CLIC register map

Register	Offset	Description
`CLICCFG`	0x0	CLIC Configuration
`CLICINFO`	0x4	CLIC Information
`CLICINTTRIG[i]`	0x40 + (i*0x4)	CLIC Interrupt Trigger i (for i = 0; i <= 31)
`CLICINTIP[i]`	0x1000 + (i*0x4)	CLIC Interrupt Pending i (for i = 0; i <= 4095)
`CLICINTIE[i]`	0x1001 + (i*0x4)	CLIC Interrupt Enable i (for i = 0; i <= 4095)
`CLICINTATTR[i]`	0x1002 + (i*0x4)	CLIC Interrupt Attribute i (for i = 0; i <= 4095)
`CLICINTCTL[i]`	0x1003 + (i*0x4)	CLIC Interrupt Input Control i (for i = 0; i <= 4095)

6.2. BM-310 Core

This section describes the BM-310 Core of the BE-U1000 microcontroller. The BE-U1000 contains one BM-310 Core (Core 2).

The BM-310 Core is the 32-bit RISC-V Core with 32 integer registers. The BM-310 Core supports two privilege levels: machine (M) and user (U). U-mode provides a mechanism to isolate application processes from each other and from trusted code running in M-mode.

For more information about the BM-310 Core CSRs, refer to the Core CSRs section.

A simplified block diagram of the BM-310 Core is shown in the following figure.

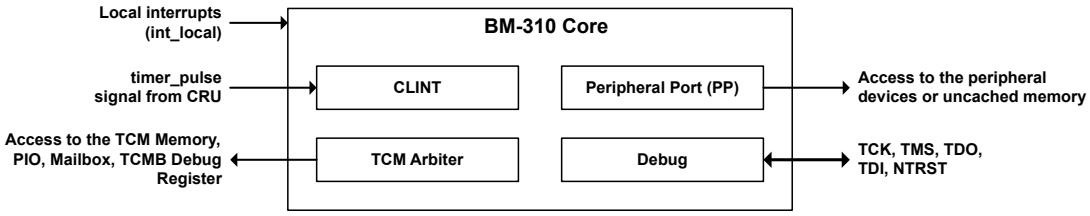

Figure 6-3 BM-310 Core block diagram

As the figure above shows, BM-310 Core includes the following blocks:

- CLINT: Used to handle interrupts.
- Peripheral Port: Used by the BR-350 Core to access peripheral devices or uncached memory.
- TCM Arbiter: Used to arbitrate access requests to the TCM Memory. Additionally, it provides access to the Mailbox, PIO and Software JTAG Register.
- Debug: Used to provide external debug access to the core via the JTAG.

6.2.1. BM-310 Core Functional Description

6.2.1.1. Debug

If the `SYSCR0.SOFTDBGEN` bit is set, debugging is not provided for the BM-310 Core 2.

Debug subsystem is compliant with the ***RISC-V External Debug Support Version 0.13***. Supported features are:

- Read/write GPRs via abstract commands (see section **Abstract Commands** in the ***RISC-V External Debug Support Version 0.13***)
- Read/Write 64 bytes program buffer (see section **Program Buffer** in the ***RISC-V External Debug Support Version 0.13***)
- 4 abstract data registers (see section **Abstract Commands** in the ***RISC-V External Debug Support Version 0.13***)
- 4 hardware match triggers (see chapter **Trigger Module** in the ***RISC-V External Debug Support Version 0.13***)
- Debug ROM

The figure below shows the block diagram of the Debug subsystem.

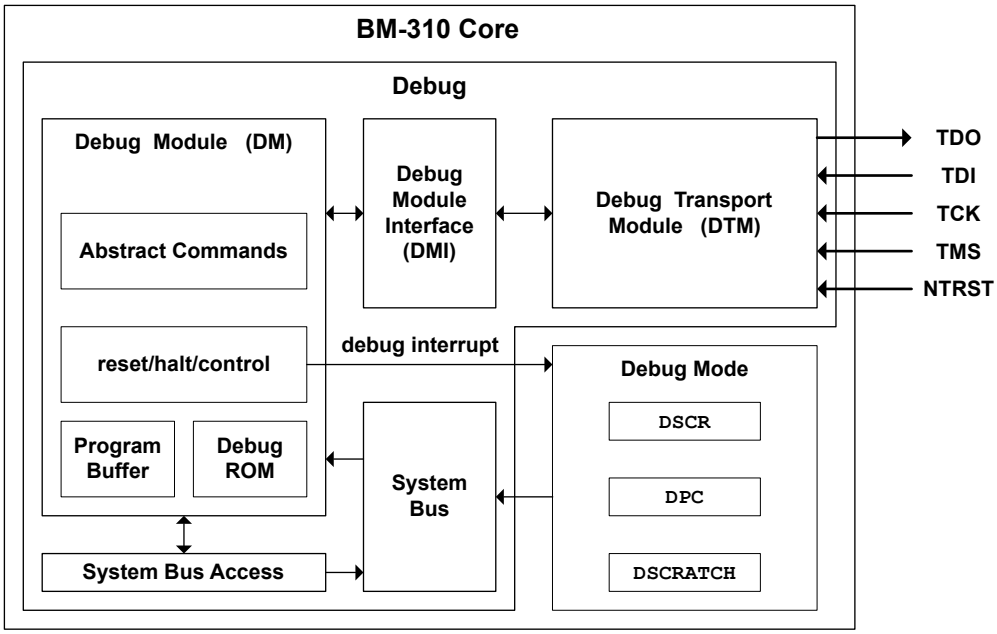

Figure 6-4 Debug subsystem

Each signal ID, shown in the right part of the diagram above, corresponds to the appropriate I/O pin ID. For more information, refer to the ***BE-U1000 Datasheet***.

Debug subsystem consists of *Debug Transport Module* (**DTM**) and *Debug Module* (**DM**). DTM connects to DM via *Debug Module Interface* (**DMI**).

DM access via JTAG is done by reading from and writing to the DTM register named `DMI` (see DTM Registers for details). Refer to the DM Registers (Access via DMI) section for details on the DM registers that are accessed over the DMI bus.

6.2.1.1.1. Core Debug Mode

Debug Mode is a special processor mode used only when a core is halted for external debugging (see also section **4.1 Debug Mode** of the ***RISC-V External Debug Support Version 0.13***). The core also has the debug *Control and Status Registers* (**CSRs**). The core is able to read from and write to debug CSRs only when it operates in Debug Mode. Refer to the Core Debug CSRs section for details on these registers.

Core processor interacts with the *Debug Module* (**DM**) only when in Debug Mode. In this mode, the core reads from and writes to the 4 KB Core Debug block (0000_0000 - 0000_0FFF) and does required actions. The table below shows the memory map of the Core Debug block.

Table 6-26 Memory map of the Core Debug block

Name	Offset	Description	R/W
`DM_ZERO`	0x0	Safe zero	R/W
`DM_HALT_ACK`	0x100	Halted ack	W
`DM_GO_ACK`	0x104	Going ack	W
`DM_RESUME_ACK`	0x108	Resume ack	W
`DM_EXCEPTION`	0x10C	Exception	W

Table 6-26 Memory map of the Core Debug block (continued)

Name	Offset	Description	R/W
DM_WHERETO	0x300	Where to	R
DM_ABST0	0x338	Abstract command 0	R
DM_ABST1	0x33C	Abstract command 1	R
DM_PBUF	0x340 - 0x37C	Program buffer	R/W
DM_ABSDATA	0x380 - 0x38C	Abstract data	R/W
DM_CPU_FLAGS	0x400	CPU flags	R
DM_DROM	0x800 - 0xFFF	Debug ROM	R

When the core switches to the Debug Mode, the control flow continues execution from the beginning of the Debug ROM (address 0x800). The program stored in the Debug ROM does polling of the CPU Flags (address 0x400).

The CPU Flags are described below in the ***CPU flags (DM_CPU_FLAGS) description*** table.

If the `Going` flag was read, it means that abstract command execution request is received and control flow redirects execution to the Abstract command 0 address (0x338). If an exception occurs during the execution, the control flow redirects execution to the exception handler inside the Debug ROM. This handler perform a write operation to the address 0x10C (Exception) of the Debug Module. Then an external debugger can read the Abstract Control and Status register (`ABSTRACTCS`) and determine the fact that an exception occurred during the execution.

If the `Resumereq` flag was read, it means that resume request is received. After resume request is received, the core executes the rest of the Debug ROM program and continues execution from the address stored in the `DPC` CSR.

Table 6-27 CPU flags (`DM_CPU_FLAGS`) description

Bits	Name	R/W	Description
[1]	`Resumereq`	R	Resumereq flag. When core operates in Debug Mode and reads non-zero value from this field it executes program from the Debug ROM till `dret` instruction and continue execution from the virtual address stored in `DPC` CSR.
[0]	`Going`	R	Going flag. When core operates in Debug Mode and reads non-zero value from this field it jumps to the first abstract command address (0x338).

6.2.1.1.2. Trigger Module

The core implements a Trigger Module to use hardware breakpoints. It provides four match triggers that can halt core execution when a specified condition is met — either on an instruction address match or on a memory access address match. The triggers can be chained, allowing a halt to occur only when

multiple match conditions are met. The Trigger Module is controlled through the *Control and Status Registers* (**CSRs**) that are given in the Trigger Module CSRs section. For more information, refer to the **Trigger Module** chapter of the ***RISC-V External Debug Support Version 0.13***.

6.2.1.2. CLINT

According to the RISC-V Privileged ISA Specification, the core must provide support for the machine software interrupts and machine timer interrupts. The *Core Local Interruptor* (**CLINT**) is responsible for generating requests (signals) for these interrupts.

The CLINT includes memory-mapped registers that are described in the CLINT Registers section.

6.2.2. BM-310 Core Registers

This section covers the following topics:

- Control and Status Registers
- DTM Registers
- DM Registers (access via DMI)
- CLINT Registers

6.2.2.1. Control and Status Registers

This section describes the *Control and Satus Registers* (**CSRs**) that can be acessed via **csr instructions**. For more information about CSRs, refer to the **Control and Status Registers (CSRs)** chapter of the ***RISC-V Instruction Set Manual, Volume II: Privileged Architecture*** specification.

6.2.2.1.1. Trigger Module CSRs

6.2.2.1.1.1. CSRs Map

The following table provides a top-level summary of the Trigger Module *Control and Status Registers* (**CSRs**). Refer to **5.2 Trigger Registers** of the ***RISC-V External Debug Support Version 0.13*** for details on registers mentioned below.

Table 6-28 Trigger Module CSRs

Register	Offset	Description
`TSELECT`	0x7A0	Trigger Select
`MCONTROL`	0x7A1	Match Control
`TDATA2`	0x7A2	Trigger Data 2
`TINFO`	0x7A4	Trigger Info

6.2.2.1.2. Core Debug CSRs

6.2.2.1.2.1. CSRs Map

The following table provides a top-level summary of the Core Debug *Control and Status Registers* (**CSRs**). Refer to **4.8 Core Debug Registers** of the ***RISC-V External Debug Support Version 0.13*** for details on registers mentioned below.

Processor core is able to read and write to debug CSRs only when it operates in Debug Mode.

Table 6-29 Core Debug CSRs

Register	Offset	Description
DSCR	0x7B0	Debug Control and Status
DPC	0x7B1	Debug PC
DSCRATCH	0x7B2	Debug Scratch Register

6.2.2.1.3. Core CSRs

6.2.2.1.3.1. CSRs Map

Implementation of the BM-310 Core *Control and Status Registers* (**CSRs**) corresponds to the ***RISC-V Instruction Set Manual, Volume II: Privileged Architecture*** specification. The following table contains only implementation specific CSRs. To view the detailed description of a register, click the register name in the **Description** column.

Table 6-30 Core CSRs

Register	Offset	Description
XCCFG	0xBC8	Core Configuration

6.2.2.1.3.2. CSR Descriptions

6.2.2.1.3.2.1. Core Configuration (XCCFG)

The XCCFG register is at offset 0xBC8.

This register is only acessible from the M privileged mode.

The following table shows the register bit assignments.

Table 6-31 Fields for register: XCCFG

Bits	Name	R/W	Description
[2]	WBCLR	R/W	When this bit is 1, validities of branch predictor structures will be cleared after *Wait for Interrupt* (**WFI**) instruction commit. **Value After Reset:** 0x0
[1]	FBCLR	R/W	When this bit is 1, validities of branch predictor structures will be cleared after fence.i commit. **Value After Reset:** 0x0
[0]	WFI	R/W	When this bit is 1, WFI instruction will be treated as NOP, i.e. core will not be switched to the WFI low-power state. **Value After Reset:** 0x0

6.2.2.2. DTM Registers

This section describes the *Debug Transport Module* (**DTM**) registers acessible via JTAG.

6.2.2.2.1. Register Map

The following table provides a top-level summary of each register. Refer to section **6.1.2 JTAG DTM Registers** of the ***RISC-V External Debug Support Version 0.13*** for details on registers mentioned below.

Table 6-32 JTAG DTM registers

Register	Offset	Description
BYPASS	0x0	BYPASS
IDCODE	0x1	IDCODE
DTMCS	0x10	DTM Control and Status
DMI	0x11	Debug Module Interface Access
BYPASS	0x12 - 0x1F	BYPASS

6.2.2.3. DM Registers (access via DMI)

The *Debug Module* (**DM**) registers described in this section are accessed over the *Debug Module Interface* (**DMI**) bus.

6.2.2.3.1. Register Map

The following table provides a top-level summary of each register. Refer to section **3.12 Debug Module Registers** of the ***RISC-V External Debug Support Version 0.13*** for details on registers mentioned below.

Table 6-33 DM registers (access via DMI)

Register	Offset	Description
DATA**i**	0x4 + (i*0x1)	Abstract Data i (for i = 0; i <= 3)
DMCONTROL	0x10	Debug Module Control
DMSTATUS	0x11	Debug Module Status
HARTINFO	0x12	Hart Info
ABSTRACTCS	0x16	Abstract Control and Status
COMMAND	0x17	Abstract Command
ABSTRACTAUTO	0x18	Abstract Command Autoexec
PROGBUF**i**	0x20 + (i*0x1)	Program Buffer i (for i = 0; i <= 15)
SBADDRESS3	0x37	System Bus Address [127:96]

Table 6-33 DM registers (access via DMI) (continued)

Register	Offset	Description
SBCS	0x38	System Bus Access Control and Status
SBADDRESS0	0x39	System Bus Address [31:0]
SBADDRESS1	0x3A	System Bus Address [63:32]
SBADDRESS2	0x3B	System Bus Address [95:64]
SBDATA0	0x3C	System Bus Data [31:0]
SBDATA1	0x3D	System Bus Data [63:32]
SBDATA2	0x3E	System Bus Data [95:64]
SBDATA3	0x3F	System Bus Data [127:96]

6.2.2.4. CLINT Registers

The register region of the *Core Local Interruptor* (**CLINT**) is mapped in the BE-U1000 microcontroller memory map as the following table shows.

Table 6-34 Memory address region for CLINT registers

Region	Block Name	Size	Start Address	End Address
Core 2 Resources	CLINT	64 KB	0x0200_0000	0x0200_FFFF

For details, refer to Memory Map chapter.

6.2.2.4.1. Register Map

The following table provides a top-level summary of each register. To view the detailed description of a register, click the register name in the **Description** column.

Table 6-35 CLINT register map

Register	Offset	Description
MSIP0	0x0	Machine Software Interrupt Pending
MTIMECMP0	0x4000	Machine Timer Compare
MTIME	0xBFF8	Machine Timer Value  Timer value is increased by the `timer_pulse` signal from the Control Registers Unit.

6.2.2.4.2. Register Descriptions

6.2.2.4.2.1. Machine Software Interrupt Pending (`MSIP0`)

The `MSIP0` register is at offset 0x0.

Machine software interrupt pending (`MSIP0`) bit could be set/cleared via corresponding memory-mapped register access. This bit is directly mapped to corresponding core input. Access to this memory-mapped register is only supported using 32-bit operations.

The following table shows the register bit assignments.

Table 6-36 Fields for register: `MSIP0`

Bits	Name	R/W	Description
[31:1]			These bits are wired to zero
[0]	`MSIP0`	R/W	Machine software interrupt pending **Value After Reset:** 0x0

6.2.2.4.2.2. Machine Timer Compare (`MTIMECMP0`)

The `MTIMECMP0` register is at offset 0x4000.

This register contains 64-bit timer compare value. The machine timer interrupt (`mtip`) is asserted if timer value is greater than timer compare value.

The following table shows the register bit assignments.

Table 6-37 Fields for register: `MTIMECMP0`

Bits	Name	R/W	Description
[63:32]	`MTIMECMP0_HI`	R/W	High part of timer compare value **Value After Reset:** 0x0
[31:0]	`MTIMECMP0_LO`	R/W	Low part of timer compare value **Value After Reset:** 0x0

6.2.2.4.2.3. Machine Timer Value (`MTIME`)

The `MTIME` register is at offset 0xBFF8.

This register contains 64-bit timer value. The value is incremented each time by the `timer_pulse` signal from the Control Registers Unit.

The following table shows the register bit assignments.

Table 6-38 Fields for register: `MTIME`

Bits	Name	R/W	Description
[63:32]	`MTIME_HI`	R	High part of timer value **Value After Reset:** 0x0
[31:0]	`MTIME_LO`	R	Low part of timer value **Value After Reset:** 0x0

7. Core 2 PIO

The BE-U1000 supports user-defined custom interfaces via its *Programmable Input/Output* (**PIO**) feature. This is implemented using the PIO Block and Core 2, allowing the use of interfaces not supported by the microcontroller's existing internal controllers.

The PIO Block includes multiple registers for I/O pin data exchange (see PIO Registers section).

The diagram below illustrates the PIO Block's location and integration within the BE-U1000 system architecture.

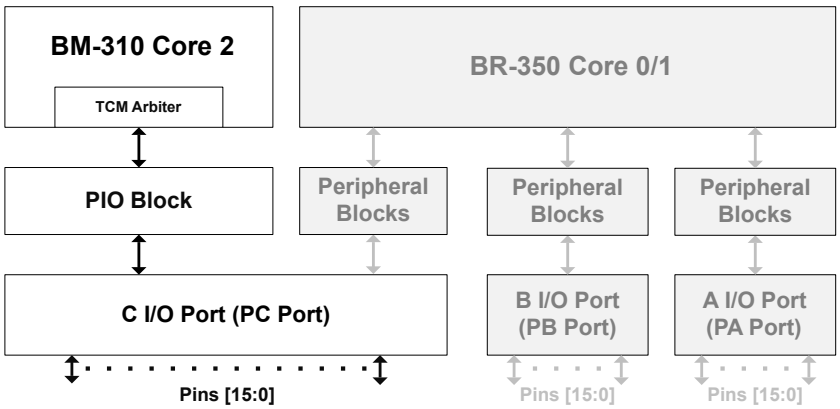

Figure 7-1 PIO Block in the BE-U1000

7.1. PIO Registers

This section covers the following topics:

- Register Map
- Register Descriptions

The following table describes the PIO Block register region.

Table 7-1 Memory address region for PIO Block registers

Block Name	Size	Start Address	End Address	Access
PIO Block Registers	16 B	0x4000_A000	0x4000_A00F	Only Core 2

You can find a list and description of PIO Block registers in the Register Map and Register Descriptions sections below.

7.1.1. Register Map

This section tables contain information about the PIO Block registers memory map.

The following table provides a high-level summary of each register. To view the detailed description of a register, click the register name in the **Description** column.

Table 7-2 PIO register map

Name	Offset	Description	Default Value
`INT`	0x0	Interrupt Register	0x0

Table 7-2 PIO register map (continued)

Name	Offset	Description	Default Value
PIO_IN	0x4	Input Data Register	0x0
PIO_EN	0x8	Enable Register	0x0
PIO_OU	0xC	Output Data Register	0x0

7.1.2. Register Descriptions

7.1.2.1. Interrupt Register (INT)

The INT register is at offset 0x0.

The following table shows the register bit assignments.

Table 7-3 Fields for register: INT

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	INT_CORE2	R/W	Interrupt with Interrupt ID 68 of the Core 0, Core 1 int_local. **Value After Reset:** 0x0

7.1.2.2. Input Data Register (PIO_IN)

The PIO_IN register is at offset 0x4.

The following table shows the register bit assignments.

Table 7-4 Fields for register: PIO_IN

Bits	Name	R/W	Description
[31:16]			Reserved
[15:0]	PIO_IN	R/W	Input data, from the PIO pins. **Value After Reset:** 0x0

7.1.2.3. Enable Register (PIO_EN)

The PIO_EN register is at offset 0x8.

The following table shows the register bit assignments.

Table 7-5 Fields for register: PIO_EN

Bits	Name	R/W	Description
[31:16]			Reserved
[15:0]	PIO_EN	R/W	PIO pins control. Valid values:

Table 7-5 Fields for register: `PIO_EN` (continued)

Bits	Name	R/W	Description
			`PIO_EN[i]` = **1:** `PIO_EN[i]` — configured as output. `PIO_EN[i]` = **0:** `PIO_EN[i]` — configured as input. **Value After Reset:** 0x0

7.1.2.4. Output Data Register (`PIO_OU`)

The `PIO_OU` register is at offset 0xC.

The following table shows the register bit assignments.

Table 7-6 Fields for register: `PIO_OU`

Bits	Name	R/W	Description
[31:16]			Reserved
[15:0]	`PIO_OU`	R/W	The data, that will be transferred to the output. **Value After Reset:** 0x0

8. DMA Controller

This chapter describes the *Direct Memory Access* (**DMA**) Controller of the BE-U1000 microcontroller.

The BE-U1000 integrates two DMA Controllers, both located within the High-speed Periphery block.

A simplified block diagram of the DMA Controller is shown in the following figure.

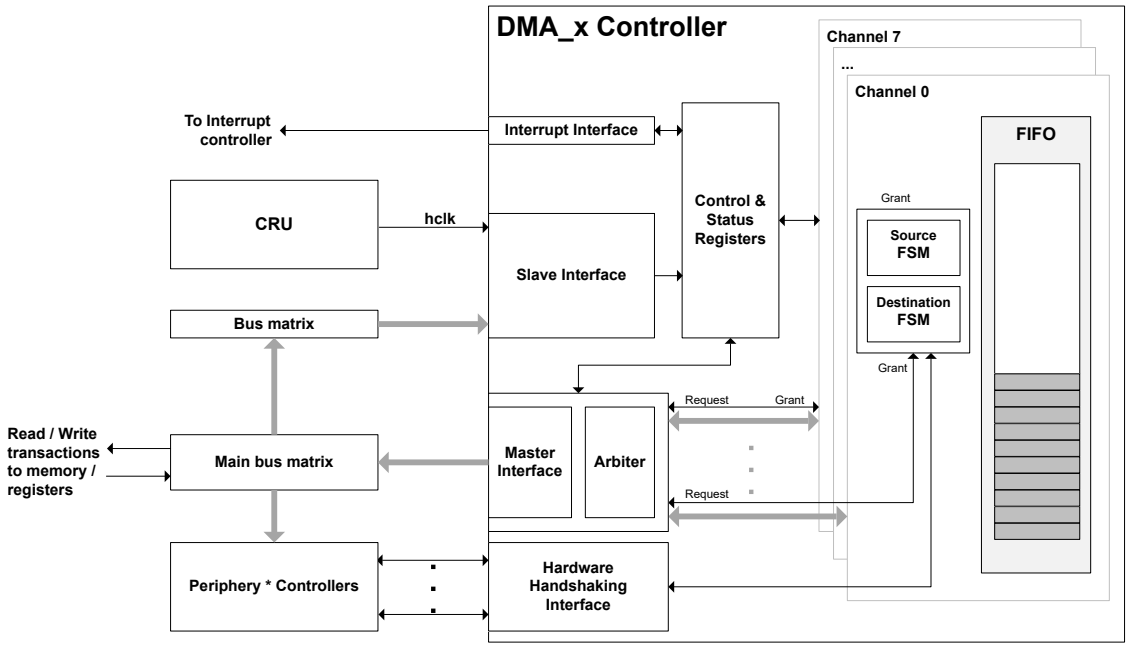

Figure 8-1 DMA Controller block diagram

- x indicates 0 for DMA_0 and 1 for DMA_1
- Asterisk symbol (*) indicates the number of an appropriate Periphery

The DMA Controller provides the following features:

- General
  - Slave Interface is used to program the DMA Controller
  - 8 Channels, one per source-destination pair
  - One Master Interface
  - Transfers
    - Support for memory-to-memory, memory-to-peripheral, peripheral-to-memory, and peripheral-to-peripheral DMA transfers
    - DMA Controller to or from Periphery * through the Peripheral * bridge
- Address Generation
  - Programmable source and destination addresses
  - Multi-block transfers (for channels 2 and 3 only) achieved through:
    - Linked Lists (block chaining)
    - Auto-reloading of channel registers
  - Independent source and destination selection of multi-block transfer type
- Channel Buffering

- Single FIFO per channel for source and destination
- FIFO depth of 16 bytes (for channels 0,1,4,5,6, and 7)
- FIFO depth of 32 bytes (for channels 2 and 3)

- Channel Control
  - Programmable source and destination for each channel
  - Programmable transfer type for each channel (memory-to-memory, memory-to-peripheral, peripheral-to-memory, and peripheral-to-peripheral)
  - Programmable maximum value of burst transaction size for each channel
  - Programmable enable and disable of DMA channel
  - Programmable maximum block size in source transfer width per channel
  - Option to include or exclude logic to enable multi-block DMA transfers on channels 2 and 3 only
- Transfer Initiation
  - 16 hardware handshaking interfaces, controlled via CRU when selecting the Alternate Functions through a multiplexer
  - 2 software handshaking interfaces
  - Handshaking interface supports single or burst DMA transactions
  - Enabling and disabling individual DMA handshaking interfaces

8.1. DMA Functional Description

8.1.1. Handshaking Interface

Handshaking interfaces are used at the transaction level to control the flow of single or burst transactions.

The peripheral uses the handshaking interface to indicate to the DMA Controller that it is ready to transfer or accept data.

A non-memory peripheral can request a DMA transfer through the DMA Controller using one of two types of handshaking interfaces:

- Hardware
- Software

Throughout the remainder of this document, references to both source and destination hardware handshaking interfaces assume an active-high interface (mode set by the default values — 0 — of `CFGx.SRC_HS_POL` and `CFGx.DST_HS_POL` bits in the `CFGx` register). When active-low handshaking interfaces are used, then the active level and edge are reversed from that of an active-high interface.

Software selects between the hardware or software handshaking interface on a per-channel basis. Software handshaking is accomplished through memory-mapped registers, while hardware handshaking is accomplished using a dedicated handshaking interface.

Source and destination peripherals can independently select the handshaking interface type; that is, hardware or software handshaking. For more information, see the description of `CFGx.HS_SEL_SRC` and `CFGx.HS_SEL_DST` bits in the `CFGx` register.

The following table describes DMA Handshake channels and peripheral blocks for DMA Handshake channels respectively.

Table 8-1 Table of connectivity of DMA Handshake channels

DMA HS channel	Peripheral Block 0 (high priority)		Peripheral Block 1 (low priority)	
DMA_0				
0	I2C_0 TX	* Use all I2C_0 I/O pins (I2C0_SCL, I2C0_SDA)	UART_6 TX	* Use al UART_6 I/O pins (UART6_TX, UART6_RX, UART6_DE, UART6_RE)
1	I2C_0 RX		UART_6 RX	
2	I2C_1 TX	* Use all I2C_1 I/O pins (I2C1_SCL, I2C1_SDA)	UART_2 TX	* Use all UART_2 I/O pins (UART2_TX, UART2_RX)
3	I2C_1 RX		UART_2 RX	
4	SPI_1 TX	* Use all SPI_1 I/O pins (SPI1_CSN, SPI1_SCK, SPI1_MOSI, SPI1_MISO)	I2S_0 TX	* Use all I2S_0 I/O pins (I2S0_SDO, I2S0_SDI, I2S0_WS, I2S0_SCLK)
5	SPI_1 RX		I2S_0 RX	
6	QSPI_0 TX	* Use all QSP0_1 I/O pins (QSPI0_CSN0, QSPI0_SCK, QSPI0_MOSI, QSPI0_MISO, QSPI0_CSN1, QSPI0_CSN2 or QSPI0_CSN, QSPI0_SCK, QSPI0_IO0, QSPI0_IO1, QSPI0_IO2, QSPI0_IO3)	SPI_0 TX	* Use all SPI_0 I/O pins (SPI0_CSN, SPI0_SCK, SPI0_MOSI, SPI0_MISO)
7	QSPI_0 RX		SPI_0 RX	
8	UART_0 TX		–	
9	UART_0 RX		–	
10	UART_1 TX		–	
11	UART_1 RX		–	
12	PWMA_0 TX		–	
13	ADC_0 RX		–	
14	PWMA_2 TX		–	
15	ADC_1 RX		–	
DMA_1				
0	I2C_2 TX	* Use all I2C_2 I/O pins (I2C2_SCL, I2C2_SDA)	UART_7 TX	* Use all UART_7 I/O pins (UART7_TX, UART7_RX, UART7_DE, UART7_RE)
1	I2C_2 RX		UART_7 RX	
2	I2C_3 TX	* Use all I2C_3 I/O pins (I2C3_SCL, I2C3_SDA)	UART_5 TX	* Use all UART_5 I/O pins (UART5_TX, UART5_RX)
3	I2C_3 RX		UART_5 RX	
4	SPI_3 TX	* Use allSPI_3 I/O pins (SPI3_CSN, SPI3_SCK, SPI3_MOSI, SPI3_MISO)	I2S_1 TX	* Use all I2S_1 I/O pins (I2S1_SDO, I2S1_SDI, I2S1_WS, I2S1_SCLK)
5	SPI_3 RX		I2S_1 RX	
6	QSPI_1 TX	* Use all QSPI_1 I/O pins (QSPI1_CSN0, QSPI1_SCK, QSPI1_MOSI, QSPI1_MISO, QSPI1_CSN1, QSPI1_CSN2 or	SPI_2 TX	* Use all SPI_2 I/O pins (SPI2_CSN, SPI2_SCK, SPI2_MOSI, SPI2_MISO)
7	QSPI_1 RX		SPI_2 RX	

Table 8-1 Table of connectivity of DMA Handshake channels (continued)

DMA HS channel	Peripheral Block 0 (high priority)		Peripheral Block 1 (low priority)	
		QSPI1_CSN, QSPI1_SCK, QSPI1_IO0, QSPI1_IO1, QSPI1_IO2, QSPI1_IO3)		
8	UART_3 TX		–	
9	UART_3 RX		–	
10	UART_4 TX		–	
11	UART_4 RX		–	
12	PWMA_1 TX		–	
13	ADC_2 RX		–	
14	PWMA_3 TX		–	
15	–		–	

* As shown in the table above, some DMA handshake channels are shared between pairs of peripheral blocks. Using these shared channels involves the following considerations:

- To use a shared DMA handshake channel with a specific peripheral, all I/O pins associated with that peripheral must be programmed to the correct alternate function. *Example:* To use the DMA handshake channel for UART_6, Port C I/O pins PC[6] through PC[9] must be configured to Function #2 via the appropriate `IOAFCRx` CRU registers. If any of these I/O pins is programmed to a different alternate function, the DMA handshake channel cannot be used for UART_6. It is not possible to configure the same DMA handshake channel for two different peripherals simultaneously. For instance, DMA handshake channel 0 (Tx) cannot be configured to work with I2C_0 while DMA handshake channel 1 (Rx) is configured to work with UART_6 at the same time, because both channels belong to the same shared pair.
- When both peripherals in a shared pair are used simultaneously, Peripheral Block 0 has higher priority than Peripheral Block 1. *Example:* If both I2C_1 and UART_2 are used together (with all I/O pins correctly programmed for their respective alternate functions), I2C_1 takes priority over UART_2. The priority order for each block is shown in the table above.

For DMA Handshake channels connected to a single peripheral block, there is no requirement to program all I/O pins of that block to their alternate functions. You may use only the pins needed for your specific application. *Example:* It is possible to configure PA[6] to Alternate Function #1 (for UART_0 with DMA handshake channel 8, Tx) while PA[7] remains configured to Alternate Function #0 (GPIO_0).

The following table shows the devices for which use of DMA HS channels requires programming all I/O pins associated with that device to the corresponding Alternate Functions.

Table 8-2 Programming the Alternate Functions

Peripheral Block	I/O pins	Registers settings
DMA_0		
I2C_0	I2C0_SCL	`IOAFCR0[2:0]` = 0x5 && `IOAFCR0[6:4]` = 0x5 || `IOAFCR0[20:18]` = 0x1 && `IOAFCR0[22:20]` = 0x1 || `IOAFCR0[26:24]` = 0x3 && `IOAFCR0[30:28]` = 0x3 || `IOAFCR1[2:0]` = 0x2 && `IOAFCR1[6:4]` = 0x2
	I2C0_SDA	

Table 8-2 Programming the Alternate Functions (continued)

Peripheral Block	I/O pins	Registers settings
I2C_1	I2C1_SCL	IOAFCR1[18:16] = 0x1 && IOAFCR1[22:20] = 0x1 || IOAFCR1[26:24] = 0x2 && IOAFCR1[30:28] = 0x2
	I2C1_SDA	
SPI_1	SPI1_CSN	IOAFCR1[2:0] = 0x1 && IOAFCR1[6:4] = 0x1 && IOAFCR1[10:8] = 0x1 && IOAFCR1[14:12] = 0x1
	SPI1_SCK	
	SPI1_MOSI	
	SPI1_MISO	
QSPI_0	QSPI0_CSN0	IOAFCR0[2:0] = 0x2 && IOAFCR0[6:4] = 0x2 && IOAFCR0[10:8] = 0x2 && IOAFCR0[14:12] = 0x2 && IOAFCR0[20:18] = 0x2 && IOAFCR0[22:20] = 0x2 || IOAFCR0[2:0] = 0x3 && IOAFCR0[6:4] = 0x3 && IOAFCR0[10:8] = 0x3 IOAFCR0[14:12] = 0x3 && IOAFCR0[20:18] = 0x3 && IOAFCR0[22:20] = 0x3
	QSPI0_SCK	
	QSPI0_MOSI	
	QSPI0_MISO	
	QSPI0_CSN1	
	QSPI0_CSN2	
	QSPI0_CSN	
	QSPI0_SCK	
	QSPI0_IO0	
	QSPI0_IO1	
	QSPI0_IO2	
	QSPI0_IO3	
UART_6	UART6_TX	IOAFCR4[26:24] = 0x2 && IOAFCR4[30:28] = 0x2 && IOAFCR5[2:0] = 0x2 && IOAFCR5[6:4] = 0x2
	UART6_RX	
	UART6_DE	
	UART6_RE	
UART_2	UART2_TX	IOAFCR1[18:16] = 0x2 && IOAFCR1[22:20] = 0x2
	UART2_RX	
I2S_0	I2S0_SDO	IOAFCR1[10:8] = 0x3 && IOAFCR1[14:12] = 0x3 && IOAFCR1[18:16] = 0x3 && IOAFCR1[22:20] = 0x3
	I2S0_SDI	
	I2S0_WS	
	I2S0_SCLK	
SPI_0	SPI0_CSN	IOAFCR0[2:0] = 0x1 && IOAFCR0[6:4] = 0x1 && IOAFCR0[10:8] = 0x1 && IOAFCR0[14:12] = 0x1
	SPI0_SCK	
	SPI0_MOSI	
	SPI0_MISO	

Table 8-2 Programming the Alternate Functions (continued)

Peripheral Block	I/O pins	Registers settings
DMA_1		
I2C_2	I2C2_SCL	IOAFCR2[2:0] = 0x5 && IOAFCR2[6:4] = 0x5 || IOAFCR2[20:18] = 0x1 && IOAFCR2[22:20] = 0x1 || IOAFCR2[26:24] = 0x3 && IOAFCR2[30:28] = 0x3 || IOAFCR3[2:0] = 0x2 && IOAFCR3[6:4] = 0x2
	I2C2_SDA	
I2C_3	I2C3_SCL	IOAFCR3[18:16] = 0x1 && IOAFCR3[22:20] = 0x1 || IOAFCR3[26:24] = 0x2 && IOAFCR3[30:28] = 0x2
	I2C3_SDA	
SPI_3	SPI3_CSN	IOAFCR3[2:0] = 0x1 && IOAFCR3[6:4] = 0x1 && IOAFCR3[10:8] = 0x1 && IOAFCR3[14:12] = 0x1
	SPI3_SCK	
	SPI3_MOSI	
	SPI3_MISO	
QSPI_1	QSPI1_CSN0	IOAFCR2[2:0] = 0x2 && IOAFCR2[6:4] = 0x2 && IOAFCR2[10:8] = 0x2 && IOAFCR2[14:12] = 0x2 && IOAFCR2[20:18] = 0x2 IOAFCR2[22:20] = 0x2 || IOAFCR2[2:0] = 0x3 && IOAFCR2[6:4] = 0x3 && IOAFCR2[10:8] = 0x3 && IOAFCR2[14:12] = 0x3 && IOAFCR2[20:18] = 0x3 && IOAFCR2[22:20] = 0x3
	QSPI1_SCK	
	QSPI1_MOSI	
	QSPI1_MISO	
	QSPI1_CSN1	
	QSPI1_CSN2	
	QSPI1_CSN	
	QSPI1_SCK	
	QSPI1_IO0	
	QSPI1_IO1	
	QSPI1_IO2	
	QSPI1_IO3	
UART_7	UART7_DE	IOAFCR4[26:24] = 0x4 && IOAFCR4[30:28] = 0x4 && IOAFCR5[2:0] = 0x4 && IOAFCR5[6:4] = 0x4
	UART7_RE	
	UART7_TX	
	UART7_RX	
UART_5	UART5_TX	IOAFCR3[18:16] = 0x2 && IOAFCR3[22:20] = 0x2
	UART5_RX	
I2S_1	I2S1_SDO	IOAFCR3[10:8] = 0x3 && IOAFCR3[14:12] = 0x3 && IOAFCR3[18:16] = 0x3 && IOAFCR3[22:20] = 0x3
	I2S1_SDI	
	I2S1_WS	
	I2S1_SCLK	

Table 8-2 Programming the Alternate Functions (continued)

Peripheral Block	I/O pins	Registers settings
SPI_2	SPI2_CSN	`IOAFCR2[2:0]` = 0x1 && `IOAFCR2[6:4]` = 0x1 && `IOAFCR2[10:8]` = 0x1 && `IOAFCR2[14:12]` = 0x1
	SPI2_SCK	
	SPI2_MOSI	
	SPI2_MISO	

The following figures show handshaking channel connectivity for DMA_0 and DMA_1 (note that Tx and Rx channels are shown as single lines).

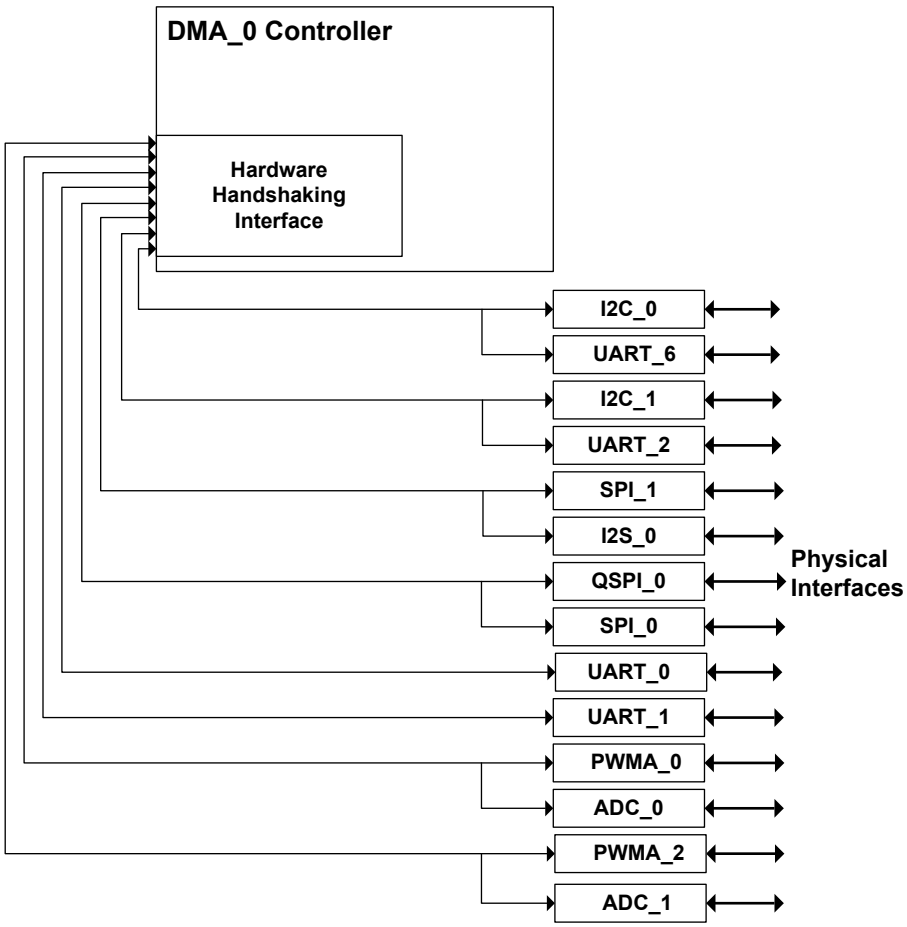

Figure 8-2 Handshaking Interface for DMA_0

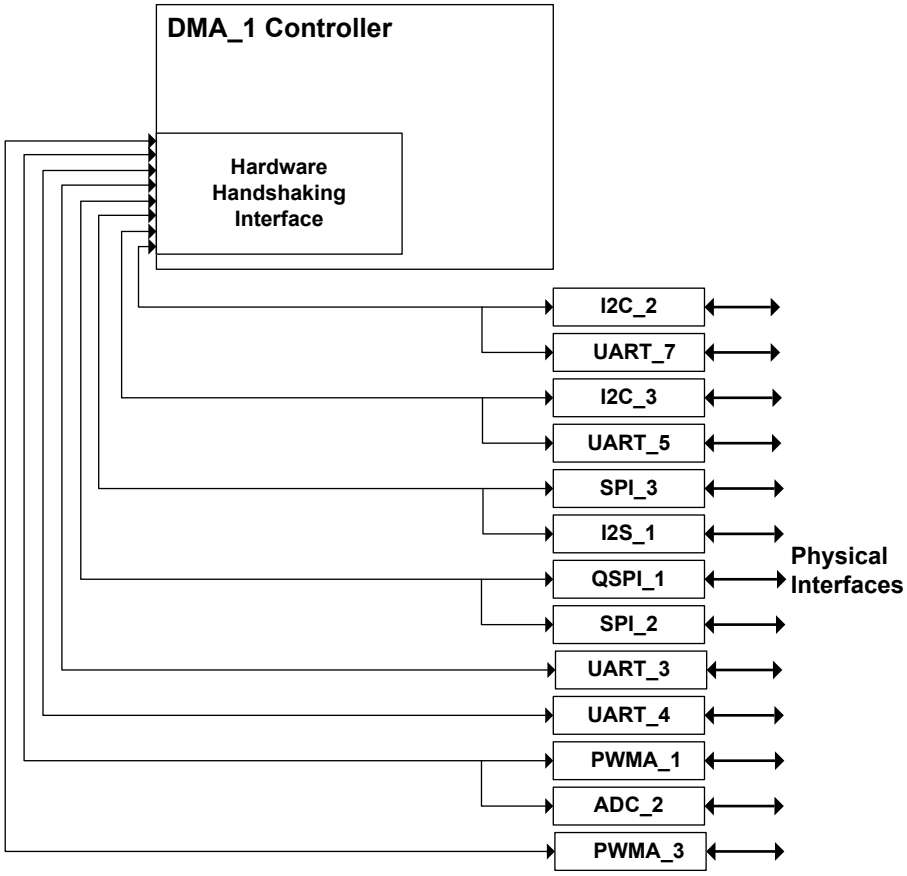

Figure 8-3 Handshaking Interface for DMA_1

8.1.2. Basic Interface Definitions

In this chapter and the following equations, references to `CTLx.SRC_MSIZE`, `CTLx.DEST_MSIZE`, `CTLx.SRC_TR_WIDTH`, and `CTLx.DST_TR_WIDTH` refer to the decoded values of the parameters; for example, `CTLx.SRC_MSIZE` = 3'b001 decodes to 4, and `CTLx.SRC_TR_WIDTH` = 3'b010 decodes to 32 bits.

The following definitions are used in this chapter:

- Source single transaction size in bytes

  `src_single_size_bytes` = `CTLx.SRC_TR_WIDTH/8`
- Source burst transaction size in bytes

  `src_burst_size_bytes` = `CTLx.SRC_MSIZE * src_single_size_bytes`
- Destination single transaction size in bytes

  `dst_single_size_bytes` = `CTLx.DST_TR_WIDTH/8`
- Destination burst transaction size in bytes

  `st_burst_size_bytes` = `CTLx.DEST_MSIZE * dst_single_size_bytes`
- Block size in bytes

  `blk_size_bytes_dma` = `CTLx.BLOCK_TS * src_single_size_bytes`

8.1.3. Memory Peripherals

The alternative to not having a transaction-level handshaking interface is to allow the DMA Controller to attempt transfers to the peripheral once the channel is enabled. If the peripheral slave cannot accept these transfers, it inserts wait states onto the bus until it is ready; it is not recommended that more than 16 wait states be inserted onto the bus. By using the handshaking interface, the peripheral can signal to the DMA Controller that it is ready to transmit or receive data, and then the DMA Controller can access the peripheral without the peripheral inserting wait states onto the bus.

The `CTLx.SRC_MSIZE` and `CTLx.DEST_MSIZE` are properties valid only for peripherals with a handshaking interface; they cannot be used for defining the burst length for memory peripherals. When the peripherals are memory, the DMA Controller uses DMA transfers to move blocks; thus the `CTLx.SRC_MSIZE` and `CTLx.DEST_MSIZE` values are not used for memory peripherals. The `SRC_MSIZE / DEST_MSIZE` limitations are used to accommodate devices that have limited resources, such as a FIFO. Memory does not normally have limitations similar to the FIFOs.

Therefore:

- Length of burst transfers to memory is always equal to the number of data items available in a channel FIFO or data items required to complete the block transfer, whichever is smaller.
- Length of burst transfers from memory is always equal to the space available in a channel FIFO or number of data items required to complete the block transfer, whichever is smaller.

8.1.4. Software Handshaking

When the slave peripheral requires the DMA Controller to perform a DMA transaction, it communicates this request by sending an interrupt to the CPU or interrupt controller. The interrupt service routine then uses the software registers (see Software Handshaking Registers) to initiate and control a DMA transaction. This group of software registers is used to implement the software handshaking interface.

The `HS_SEL_SRC/HS_SEL_DST` bit in the `CFGx` channel configuration register must be set to enable software handshaking.

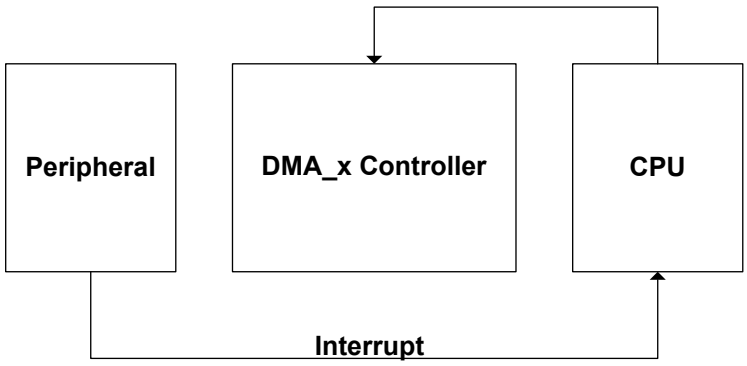

Figure 8-4 Software controlled DMA transfers

Program and enable channel through Channel Registers.

After interrupt, initiate and control DMA transaction between peripherals DMA Controller through software handshaking registers.

The software handshaking registers are:

- `REQ_SRC` – source software transaction request
- `REQ_DST` – destination software transaction request
- `SGL_REQ_SRC` – single source transaction request
- `SGL_REQ_DST` – single destination transaction request
- `LST_SRC` – last source transaction request
- `LST_DST` – last destination transaction request

For details on how the software handshaking flow works, see Software Handshaking Operation.

8.1.5. Handshaking Interface Operation

The DMA Controller tries to efficiently transfer the data using as little of the bus bandwidth as possible. The DMA Controller can also lock the arbitration for the master bus interface so that a channel is permanently granted the master bus interface.

Before describing the handshaking interface operation, the following sections define the terms Single Transaction Region and Early-Terminated Burst Transaction.

8.1.5.1. Single Transaction Region

There are cases where a DMA block transfer cannot complete using only burst transactions. Typically this occurs when the block size is not a multiple of the burst transaction length. In these cases, the block transfer uses burst transactions up to the point where the amount of data left to complete the block is less than the amount of data in a burst transaction. At this point, the DMA Controller samples the “single” status flag and completes the block transfer using single transactions.The peripheral asserts a "single" status flag to indicate to the DMA Controller that there is enough data or space to complete a single transaction from or to the source/destination peripheral.

For hardware handshaking, the "single" status flag is a signal on the hardware handshaking interface. For software handshaking, the single status flag is one of the software handshaking interface registers; (see Software Handshaking Operation).

The Single Transaction Region is the time interval where the DMA Controller uses single transactions to complete the block transfer; burst transactions are exclusively used outside this region.

Burst transactions can also be used in this region; for more information, refer to Early-Terminated Burst Transaction.

The Single Transaction Region applies to a peripheral only:

- The source peripheral enters the Single Transaction Region when the number of bytes left to complete in the source block transfer is less than `src_burst_size_bytes`. If:

  `blk_size_bytes / src_burst_size_bytes = integer`

  then the source never enters this region, and the source block uses only burst transactions.

  The destination peripheral enters the Single Transaction Region when the number of bytes left to complete in the destination block transfer is less than `dst_burst_size_bytes`. If: `blk_size_bytes / dst_burst_size_bytes = integer` then the destination never enters this region, and the destination block uses only burst transactions.

  

  The above conditions cause a peripheral to enter the Single Transaction Region. When the peripheral is outside the Single Transaction Region, then the DMA Controller responds to only burst transaction requests. Whether the peripheral knows that it is in the Single Transaction

Region or not, it must always generate burst requests outside the Single Transaction Region, or the DMA block transfer stalls. Once in the Single Transaction Region, the DMA Controller can complete the block transfer using single transactions.

8.1.5.2. Early-Terminated Burst Transaction

When a source or destination peripheral is in the Single Transaction Region, a burst transaction can still be requested. However, `src_burst_size_bytes` (`dst_burst_size_bytes`) is greater than the number of bytes left to complete in the source/destination block transfer at the time that the burst transaction is triggered. In this case, the burst transaction is started and "early-terminated" at block completion without transferring the programmed amount of data – that is, `src_burst_size_bytes` or `dst_burst_size_bytes` – but only the amount required to complete the block transfer.

If the DMA Controller detects a burst request when in the Single Transaction Region, the DMA Controller only issues SINGLE type transfers to complete the burst.

8.1.5.3. Software Handshaking Operation

Last transaction registers – `LST_SRC` and `LST_DST` – are not used, and the values in these registers are ignored.

Peripheral Not In Single Transaction Region

Writing a 1 to the `REQ_SRC[x]` (`REQ_DST[x]`) register is always interpreted as a burst transaction request, where x is the channel number. However, in order for a burst transaction request to start, software must write a 1 to the `SGL_REQ_SRC[x]` (`SGL_REQ_DST[x]`) register.

You can write a 1 to the `SGL_REQ_SRC[x]` (`SGL_REQ_DST[x]`) and `REQ_SRC[x]` (`REQ_DST[x]`) registers in any order, but both registers must be asserted in order to initiate a burst transaction. Upon completion of the burst transaction, the hardware clears the `SGL_REQ_SRC[x]` (`SGL_REQ_DST[x]`) and `REQ_SRC[x]` (`REQ_DST[x]`) registers.

Peripheral In Single Transaction Region

Writing a 1 to the `SGL_REQ_SRC` initiates a single transaction. Upon completion of the single transaction, both the `SGL_REQ_SRC` (`SGL_REQ_DST`) and `REQ_SRC` bits are cleared by hardware. Therefore, writing a 1 to the `REQ_SRC(REQ_DST)` is ignored while a single transaction has been initiated, and the requested burst transaction is not serviced.

Again, writing a 1 to the `REQ_SRC(REQ_DST)` register is always a burst transaction request. However, in order for a burst transaction request to start, the corresponding channel bit in the d must be asserted. Therefore, to ensure that a burst transaction is serviced in this region, you must write a 1 to the `REQ_SRC (REQ_DST)` before writing a 1 to the `SGL_REQ_SRC (SGL_REQ_DST)` register. If the programming order is reversed, a single transaction is started instead of a burst transaction. The hardware clears both the `REQ_SRC (REQ_DST)` and the `SGL_REQ_SRC (SGL_REQ_DST)` registers after the burst transaction request completes. When a burst transaction is initiated in the Single Transaction Region, then the block completes using an Early-Terminated Burst Transaction.

Software can poll the relevant channel bit in the `SGL_REQ_SRC(SGL_REQ_DST)` and `REQ_SRC(REQ_DST)` registers. When both are 0, then either the requested burst or single transaction has completed.

The transaction-complete interrupts are triggered when both single and burst transactions are complete. The same transaction-complete interrupt is used for both single and burst transactions.

8.1.5.4. Single Transactions

The source peripheral can hardcode `dma_single` to an inactive level (hardware handshaking), or software will never need to initiate single transactions from the source (software handshaking). This can happen if either of the following is true:

- Block size is a multiple of the burst transaction length:

  `blk_size_bytes_dma / src_burst_size_bytes = integer`

- Block size is not a multiple of the burst transaction length, but the peripheral can dynamically adjust the watermark level that triggers a burst request in order to enable block completion.

The destination peripheral can hardcode `dma_single` to an inactive level (hardware handshaking), or software will never need to initiate single transactions to the destination (software handshaking). This can happen when any of the following are true:

- Block size is a multiple of the burst transaction length:

  `block_size_bytes_dma / dst_burst_size_bytes = integer`

- The destination peripheral can dynamically adjust the watermark level upwards so that a burst request is triggered in order to enable a destination block completion.
- It is guaranteed that data at some point will be extracted from the destination FIFO in the Single transaction region in order to trigger a burst transaction.

The destination peripheral requires data to be extracted in order to empty the destination FIFO below a watermark level for triggering a destination burst request. If it is guaranteed that data at some point will be extracted from the destination FIFO in the Single Transaction Region, then in this case, the destination block completes with an Early-Terminated Burst Transaction.

If none of the above is true, then a series of burst transactions followed by single transactions is needed to complete the source/destination block transfer.

8.1.6. Setting up Transfers

Transfers are set up by programming fields of the `CTLx` and `CFGx` registers for that channel. A peripheral requests a transaction through the handshaking interface to the DMA Controller (see Handshaking Interface).

The following table lists the parameters that are investigated in the following examples. The effects of these parameters on the flow of the block transfer are highlighted. In addition to the software parameters, it includes the channel FIFO depth, which is 16.

The FIFO depth for channels 2 and 3 is 32.

For definitions of the register parameters, refer to Register and Field Descriptions.

Table 8-3 Parameters used in transfer examples

Parameter	Description
FIFO Depth	Channel x FIFO depth in bytes — 16 Channel 2,3 FIFO depth in bytes — 32
`CTLx.TT_FC`	Transfer type and flow control

Table 8-3 Parameters used in transfer examples (continued)

Parameter	Description
CTLx.BLOCK_TS	Block transfer size
CTLx.SRC_TR_WIDTH	Source transfer width
CTLx.DST_TR_WIDTH	Destination transfer width
CTLx.SRC_MSIZE	Source burst transaction length
CTLx.DEST_MSIZE	Destination burst transaction length
CFGx.FIFO_MODE	FIFO mode select
CFGx.FCMODE	Flow-control mode

8.1.7. Peripheral Burst Transaction Requests

For a source FIFO, an active edge is triggered when the source FIFO exceeds some watermark level. For a destination FIFO, an active edge is triggered when the destination FIFO drops below some watermark level.

This section investigates the optimal settings of these watermark levels on the source and destination peripherals and their relationship to, respectively:

- Source transaction length, CTLx.SRC_MSIZE
- Destination transaction length, CTLx.DEST_MSIZE

For demonstration purposes, a receive *SPI Controller is used as a source peripheral, and a transmit *SPI Controller is used as a destination peripheral.

Throughout this section, *SPI Controller-related parameters are prefixed with *SPI. DMA Controller-related parameters are prefixed with DMA.

8.1.7.1. Transmit Watermark Level and Transmit FIFO Underflow

During *SPI Controller serial transfers, *SPI Controller transmit FIFO requests are made to the DMA Controller whenever the number of entries in the *SPI Controller transmit FIFO is less than or equal to the *SPI Controller Transmit Data Level Register (DMATDLR) value. This is known as the watermark level. The DMA Controller responds by writing a burst of data to the *SPI Controller transmit FIFO buffer, of length CTLx.DEST_MSIZE.

Data should be fetched from the DMA Controller often enough for the *SPI Controller transmit FIFO to continuously perform serial transfers; that is, when the *SPI Controller transmit FIFO begins to empty, another burst transaction request should be triggered. Otherwise the *SPI Controller transmit FIFO runs out of data (underflow). To prevent this condition, the user must set the watermark level correctly.

8.1.7.2. Choosing the Transmit Watermark Level

Consider an example with the following assumption:

```
CTLx.DEST_MSIZE = SPI*_TX_FIFO_DEPTH - DMATDLR
```

`SPI*_TX_FIFO_DEPTH` is the *SPI Controller transmit FIFO depth. SPI* `DMATDLR` controls the level at which a DMA Controller destination burst request is made by the *SPI Controller transmit logic. It is equal to the watermark level; that is, a destination burst request is generated (active-edge of `dma_req` triggered) when the number of valid data entries in the *SPI Controller transmit FIFO is equal to or below this field value.

In this situation, the number of data items to be transferred in a DMA Controller burst is equal to the empty space in the *SPI Controller transmit FIFO.

8.1.7.3. Selecting `CTLx.DEST_MSIZE` and Transmit FIFO Overflow

Programming `CTLx.DEST_MSIZE` to a value greater than the watermark level that triggers the DMA Controller request may cause overflow when there is not enough space in the *SPI Controller transmit FIFO to service the destination burst request. Therefore, the following equation must be adhered to in order to avoid overflow:

```
CTLx.DEST_MSIZE <= SPI*_TX_FIFO_DEPTH - DMATDLR
```

For optimal operation, `CTLx.DEST_MSIZE` should be set at the FIFO level that triggers a transmit DMA Controller request; that is:

```
CTLx.DEST_MSIZE = SPI*_TX_FIFO_DEPTH - DMATDLR
```

Adhering to the equation above reduces the number of DMA Controller bursts needed for a block transfer, and this in turn improves bus utilization.

The *SPI Controller transmit FIFO is not full at the end of a DMA Controller burst transaction if the *SPI has successfully transmitted one data item or more on the *SPI serial transmit line before the end of the burst transaction.

8.1.7.4. Receive Watermark Level and Receive FIFO Overflow

During *SPI Controller serial transfers, *SPI Controller receive FIFO requests are made to the DMA Controller whenever the number of entries in the *SPI Controller Receive FIFO is at or above the DMA Controller Receive Data Level Register; that is, `DMARDLR+1`. This is known as the watermark level. The DMA Controller responds by reading a burst of data from the receive FIFO buffer of length `CTLx.SRC_MSIZE`.

`DMARDLR` controls the level at which a source burst request is made by the receive logic. When the number of valid data entries in the *SPI Controller receive FIFO is equal to or greater than the watermark level (`DMARDL+1`), a source burst request is generated (active-edge of `dma_req` triggered).

Data should be fetched by the DMA Controller often enough for the *SPI Controller receive FIFO to accept serial transfers continuously; that is, when the *SPI Controller receive FIFO begins to fill, another burst transaction request should be triggered. Otherwise, the *SPI Controller Receive FIFO fills with data (overflow). To prevent this condition, the user must correctly set the watermark level.

8.1.7.5. Choosing the Receive Watermark level

Similar to choosing the transmit watermark level described earlier, the receive watermark level, `DMARDLR+1`, should be set to minimize the probability of overflow. It is a trade-off between the number of burst transactions required per block versus the probability of an overflow occurring.

8.1.7.6. Selecting `CTLx.SRC_MSIZE` and Receive FIFO Underflow

As can be seen in the figure below, programming a source burst transaction length greater than the watermark level may cause underflow when there is not enough data to service the source burst request. Therefore, the equation below must be adhered to in order to avoid underflow.

If the number of data items in the *SPI Controller receives FIFO is equal to the source burst length at the time the source burst request is made – `CTLx.SRC_MSIZE` – the *SPI Controller receive FIFO may be emptied, but not underflowed, at the completion of the source burst transaction. For optimal operation, `CTLx.SRC_MSIZE` should be set at the watermark level; that is:

```
CTLx.SRC_MSIZE = DMARDLR + 1
```

Adhering to this equation reduces the number of burst transactions in a block transfer, and this in turn can improve bus utilization.

The *SPI Controller receive FIFO is not empty at the end of the source burst transaction if the *SPI has successfully received one data item or more on the *SPI serial receive line before the end of the burst, as illustrated in the figure below.

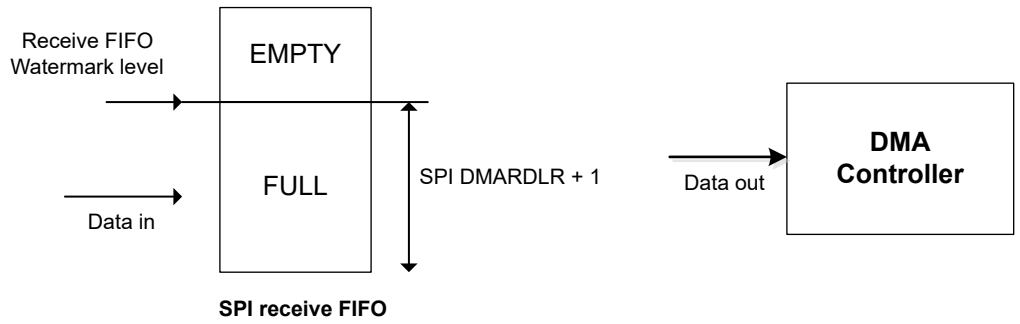

Figure 8-5 SPI receive FIFO

8.1.8. Generating Requests for the Master Bus Interface

Each channel has a source state machine and destination state machine running in parallel. These state machines generate the request inputs to the arbiter, which arbitrates for the master bus interface (one arbiter per master bus interface).

When the source/destination state machine is granted control of the master bus interface, and when the master bus interface is granted control of the external bus, then transfers between the peripheral and the DMA Controller (on behalf of the granted state machine) can take place.

Transfers from the source peripheral or to the destination peripheral cannot proceed until the channel FIFO is ready. For burst transaction requests and for transfers involving memory peripherals, the criterion for “FIFO readiness” is controlled by the `FIFO_MODE` field of the `CFGx` register.

- The definition of FIFO readiness is the same for:
  - Single transactions
  - Burst transactions, where `CFGx.FIFO_MODE = 0`
  - Transfers involving memory peripherals, where `CFGx.FIFO_MODE = 0`

The channel FIFO is deemed ready when the space/data available is sufficient to complete a single transfer of the specified transfer width. FIFO readiness for source transfers occurs when the channel FIFO contains enough room to accept at least a single transfer of `CTLx.SRC_TR_WIDTH` width. FIFO

readiness for destination transfers occurs when the channel FIFO contains data to form at least a single transfer of `CTLx.DST_TR_WIDTH` width.

An exception to FIFO readiness for destination transfers occurs in “FIFO flush mode.” In this mode, FIFO readiness for destination transfers occurs when the channel FIFO contains data to form at least a single transfer of `CTLx.SRC_TR_WIDTH` width (and not `CTLx.DST_TR_WIDTH` width, as is the normal case). For an explanation of FIFO flush mode.

When `CFGx.FIFO_MODE = 1`, then the criteria for FIFO readiness for burst transaction requests and transfers involving memory peripherals are as follows:

- A FIFO is ready for a source burst transfer when the FIFO is less than half empty.
- A FIFO is ready for a destination burst transfer when the FIFO is greater than or equal to half full.

Exceptions to this “readiness” occur. During these exceptions, a value of `CTLx.FIFO_MODE = 0` is assumed. The following are the exceptions:

- Near the end of a burst transaction or block transfer – The channel source state machine does not wait for the channel FIFO to be less than half empty if the number of source data items left to complete the source burst transaction or source block transfer is less than FIFO depth — 128. Similarly, the channel destination state machine does not wait for the channel FIFO to be greater than or equal to half full, if the number of destination data items left to complete the destination burst transaction or destination block transfer is less than FIFO depth — 128.
- In FIFO flush mode – For an explanation of FIFO flush mode.
- When a channel is suspended – The destination state machine does not wait for the FIFO to become half empty to flush the FIFO, regardless of the value of the `FIFO_MODE` field.
- After receipt of a split/retry response from a source or destination – After an master receives a split/retry response, it must re-issue the transfer that received the split/retry before attempting any other transfer. Therefore, a transfer is re-issued to the same address that returned the split/retry, regardless of `FIFO_MODE`, when the DMA Controller is next granted the bus. This is repeated until an OKAY response is received on the `hresp` bus.

When the source/destination peripheral is not memory, the source/destination state machine waits for a single/burst transaction request. Upon receipt of a transaction request and only if the channel FIFO is “ready” for source/destination transfers, a request for the master bus interface is made by the source/destination state machine.

There is one exception to this, which occurs when `CFGx.FCMODE = 1` (data pre-fetching is disabled). Then the source state machine does not generate a request for the master bus interface (even if the FIFO is “ready” for source transfers and has received a source transaction request) until the destination requests new data.

When the source/destination peripheral is memory, the source/destination state machine must wait until the channel FIFO is “ready.” A request is then made for the master bus interface. There is no handshaking mechanism employed between a memory peripheral and the DMA Controller.

8.2. DMA Registers

The BE-U1000 integrates two DMA Controllers, both located within the High-speed Periphery block.

The following table describes address regions for the DMA Controllers.

Table 8-4 Memory address regions for DMA Controllers

Block Name	Size	Start Address	End Address
DMA_0	4 KB	0x1510_0000	0x1510_0FFF
DMA_1	4 KB	0x1510_1000	0x1510_1FFF

8.2.1. Register Map

This section contains information about the register memory map.

The following table provides a high-level summary of each register. To view the detailed description of a register, click the register name in the **Description** column.

The table below contains 32-bit and 64-bit registers, but the access to registers is implemented via the 32-bit wide bus. It means that if it is needed to perform an access to the upper 32 bits of the 64-bit registers (bits [63:32]), you should add 0x4 to the register base address. For example, `CTL0[63:32]` bits are available at the address: {DMA base address + 0x18 (`CTL0` base address) + 0x4}.

Table 8-5 DMA register map

Name	Address Offset	Bits	Memory Access	Description
Channel 0 Registers				
SAR0	0x0	32	R/W	Source Address for Channel 0
DAR0	0x8	32	R/W	Destination Address Register for Channel 0
CTL0	0x18	64	R/W	Control Register for Channel 0
CFG0	0x40	64	R/W	Configuration Register for Channel 0
Channel 1 Registers				
SAR1	0x58	32	R/W	Source Address for Channel 1
DAR1	0x60	32	R/W	Destination Address Register for Channel 1
CTL1	0x70	64	R/W	Control Register for Channel 1
CFG1	0x98	64	R/W	Configuration Register for Channel 1
Channel 2 Registers				
SAR2	0xB0	32	R/W	Source Address for Channel 2
DAR2	0xB8	32	R/W	Destination Address Register for Channel 2
LLP2	0xC0	32	R/W	Linked List Pointer Register for Channel 2
CTL2	0xC8	64	R/W	Control Register for Channel 2

Table 8-5 DMA register map (continued)

Name	Address Offset	Bits	Memory Access	Description
CFG2	0xF0	64	R/W	Configuration Register for Channel 2
Channel 3 Registers				
SAR3	0x108	32	R/W	Source Address for Channel 3
DAR3	0x110	32	R/W	Destination Address Register for Channel 3
LLP3	0x118	32	R/W	Linked List Pointer Register for Channel 3
CTL3	0x120	64	R/W	Control Register for Channel 3
CFG3	0x148	64	R/W	Configuration Register for Channel 3
Channel 4 Registers				
SAR4	0x160	32	R/W	Source Address for Channel 4
DAR4	0x168	32	R/W	Destination Address Register for Channel 4
CTL4	0x178	64	R/W	Control Register for Channel 4
CFG4	0x1A0	64	R/W	Configuration Register for Channel 4
Channel 5 Registers				
SAR5	0x1B8	32	R/W	Source Address for Channel 5
DAR5	0x1C0	32	R/W	Destination Address Register for Channel 5
CTL5	0x1D0	64	R/W	Control Register for Channel 5
CFG5	0x1F8	64	R/W	Configuration Register for Channel 5
Channel 6 Registers				
SAR6	0x210	32	R/W	Source Address for Channel 6
DAR6	0x218	32	R/W	Destination Address Register for Channel 6
CTL6	0x228	64	R/W	Control Register for Channel 6
CFG6	0x250	64	R/W	Configuration Register for Channel 6
Channel 7 Registers				
SAR7	0x268	32	R/W	Source Address for Channel 7
DAR7	0x270	32	R/W	Destination Address Register for Channel 7

Table 8-5 DMA register map (continued)

Name	Address Offset	Bits	Memory Access	Description
CTL7	0x280	64	R/W	Control Register for Channel 7
CFG7	0x2A8	64	R/W	Configuration Register for Channel 7
Interrupt Registers				
RAW_TFR	0x2C0	32	R	Raw Status for IntTfr Interrupt
RAW_BLOCK	0x2C8	32	R	Raw Status for IntBlock Interrupt
RAW_SRC_TRAN	0x2D0	32	R	Raw Status for IntSrcTran Interrupt
RAW_DST_TRAN	0x2D8	32	R	Raw Status for IntDstTran Interrupt
RAW_ERR	0x2E0	32	R	Raw Status for IntErr Interrupt
STAT_TFR	0x2E8	32	R	Status for IntTfr Interrupt
STAT_BLOCK	0x2F0	32	R	Status for IntBlock Interrupt
STAT_SRC_TRAN	0x2F8	32	R	Status for IntSrcTran Interrupt
STAT_DST_TRAN	0x300	32	R	Status for IntDstTran Interrupt
STAT_ERR	0x308	32	R	Status for IntErr Interrupt
MASK_TFR	0x310	32	R/W	Mask for IntTfr Interrupt
MASK_BLOCK	0x318	32	R/W	Mask for IntBlock Interrupt
MASK_SRC_TRAN	0x320	32	R/W	Mask for IntSrcTran Interrupt
MASK_DST_TRAN	0x328	32	R/W	Mask for IntDstTran Interrupt
MASK_ERR	0x330	32	R/W	Mask for IntErr Interrupt
CLR_TFR	0x338	32	W	Clear for IntTfr Interrupt
CLR_BLOCK	0x340	32	W	Clear for IntBlock Interrupt
CLR_SRC_TRAN	0x348	32	W	Clear for IntSrcTran Interrupt
CLR_DST_TRAN	0x350	32	W	Clear for IntDstTran Interrupt
CLR_ERR	0x358	32	W	Clear for IntErr Interrupt
STAT	0x360	32	W	Status for each interrupt type
Software Handshaking Registers				

Table 8-5 DMA register map (continued)

Name	Address Offset	Bits	Memory Access	Description
REQ_SRC	0x368	32	R/W	Source Software Transaction Request Register
REQ_DST	0x370	32	R/W	Destination Software Transaction Request Register
SGL_REQ_SRC	0x378	32	R/W	Source Single Transaction Request Register
SGL_REQ_DST	0x380	32	R/W	Destination Single Transaction Request register
LST_SRC	0x388	32	R/W	Source Last Transaction Request register
LST_DST	0x390	32	R/W	Destination Last Transaction Request register
Miscellaneous Registers				
CFG	0x398	32	R/W	DMA Configuration Register
CH_EN	0x3A0	32	R/W	DMA Channel Enable Register
TEST	0x3B0	32	R/W	DMA Test Register

8.2.2. Register Descriptions

The following subsections describe the data fields of the DMA registers.

8.2.2.1. Channel Registers

Each channel is controlled by five registers: SARx, DARx, LLPx, CTLx, and CFGx. These registers must be programmed before the channel is enabled. Writing to the SARx, DARx, LLPx, or CTLx registers while the channel is enabled is prohibited and triggers an Illegal Register Access error.

LLP channel registers exist only on channels 2 and 3.

8.2.2.1.1. Source Address Register (SARx) for Channel x

The SARx register is at offset: for x=0 to 7: 0x000+**x***0x058.

The starting source address is programmed by software before the DMA channel is enabled, or by an LLI update before the start of the DMA transfer. While the DMA transfer is in progress, this register is updated to reflect the source address of the current transfer.

You must program the SAR address to be aligned to CTLx.SRC_TR_WIDTH.

The following table shows the register bit assignments.

Table 8-6 Fields for register: SARx

Bits	Name	R/W	Description
[31:0]	SAR	R/W	Current Source Address of DMA transfer Updated after each source transfer. The SINC field in the CTLx register determines whether the address increments, decrements, or is left unchanged on every source transfer throughout the block transfer. **Value After Reset**: 0x0

8.2.2.1.2. Destination Address Register (DARx) for Channel x

The DARx register is at offset: for x=0 to 7: 0x8+**x***0x058.

The starting destination address is programmed by software before the DMA channel is enabled, or by an LLI update before the start of the DMA transfer. While the DMA transfer is in progress, this register is updated to reflect the destination address of the current transfer.

You must program the DAR to be aligned to CTLx.DST_TR_WIDTH.

For information on how the DARx is updated at the start of each DMA block for multi-block transfers, refer to the transfer mode table.

The following table shows the register bit assignments.

Table 8-7 Fields for register: DARx

Bits	Name	R/W	Description
[31:0]	DAR	R/W	Current destination address of DMA transfer Updated after each destination transfer. The DINC field in the CTLx register determines whether the address increments, decrements, or is left unchanged on every destination transfer throughout the block transfer. **Value After Reset**: 0x0

8.2.2.1.3. Hardware Realignment of SAR/DAR Registers

In a particular circumstance, during contiguous multi-block DMA transfers, the destination address can become misaligned between the end of one block and the start of the next block. When this situation occurs, DMA Controller re-aligns the destination address before the start of the next block.

Consider the following example. If the block length is 9, the source transfer width is 16 (halfword), and the destination transfer width is 32 (word) — the destination is programmed for contiguous block transfers — then the destination performs four word transfers followed by a halfword transfer to complete the block transfer to the destination. At the end of the destination block transfer, the address is aligned to a 16-bit transfer as the last transfer is halfword. This is misaligned to the programmed transfer size of 32 bits for the destination. However, for contiguous destination multi-block transfers, DMA Controller re-aligns the DAR address to the nearest 32-bit address (next 32-bit address upwards if address control is incrementing or next address downwards if address control is decrementing).The destination address is automatically realigned by the DMA Controller in the following DMA transfer setup scenario:

Contiguous multi-block transfers on destination side, AND

DST_TR_WIDTH > SRC_TR_WIDTH, and

`BLOCK_TS * SRC_TR_WIDTH)/DST_TR_WIDTH != integer`

where `SRC_TR_WIDTH`, `DST_TR_WIDTH` is byte width of transfer

8.2.2.1.4. Linked List Pointer Register (`LLPx`) for Channel x

This register is valid only for channel 2 and 3.

The `LLPx` register is at offset 0xC0 and 0x118 for channel 2 and for channel 3 respectively.

The `LLPx` register has two functions:

- The logical result of the equation `LLPx.LOC != 0` is used to set up the type of DMA transfer — single or multi-block. Transfer mode table shows how the method of updating the channel registers is a function of `LLPx.LOC != 0`. If `LLPx.LOC` is set to 0x0, then transfers using linked lists are not enabled. This register must be programmed prior to enabling the channel in order to set up the transfer type.
- `LLPx.LOC != 0` contains the pointer to the next LLI for block chaining using linked lists; refer to Block Chaining Using Linked Lists. The `LLPx` register can also point to the address where write-back of the control and source/destination status information occur after block completion.

You need to program this register to point to the first *Linked List Item* (**LLI**) in memory prior to enabling the channel if block chaining is enabled – rows 6 to 10 of transfer mode table.

The DMA Controller returns an ERROR response to an illegal register access, which includes accessing registers that have been removed during DMA Controller configuration.

The following table shows the register bit assignments.

Table 8-8 Fields for register: `LLPx`

Bits	Name	R/W	Description
[31:2]	`LOC`	R/W	Starting Address In Memory of next LLI if block chaining is enabled. Note that the two LSBs of the starting address are not stored because the address is assumed to be aligned to a 32-bit boundary. LLI accesses are always 32-bit accesses (`Hsize = 2`) aligned to 32-bit boundaries and cannot be changed or programmed to anything other than 32-bit. **Value After Reset**: 0x0
[1:0]			Reserved

8.2.2.1.5. Control Register (`CTLx`) for Channel x

The `CTLx` register is at offset: for x=0 to 7: 0x018+**x***0x058.

This register contains fields that control the DMA transfer.

The `CTLx` register is part of the block descriptor (LLI) when block chaining is enabled. It can be varied on a block-by-block basis within a DMA transfer when block chaining is enabled. For information about the behaviour of this register between blocks, refer to Multi-Block Transfers.

If status write-back is enabled, the upper word of the control register, `CTLx[63:32]`, is written to the control register location of the LLI in system memory at the end of the block transfer.

You need to program this register prior to enabling the channel.

The following table shows the register bit assignments.

Table 8-9 Fields for register: `CTLx`

Bits	Name	R/W	Description
[63:45]			Reserved
[44]	`DONE`	R/W	Done bit If status write-back is enabled, the upper word of the control register, `CTLx[63:32]`, is written to the control register location of the *Linked List Item* (**LLI**) in system memory at the end of the block transfer with the done bit set. Software can poll the LLI `DONE` bit to see when a block transfer is complete. The LLI `DONE` bit should be cleared when the linked lists are set up in memory prior to enabling the channel. **Value After Reset**: 0x0
[43:40]			Reserved
[39:32]	`BLOCK_TS`	R/W	Block Transfer Size The user writes this field before the channel is enabled in order to indicate the block size. The number programmed into `BLOCK_TS` indicates the total number of single transactions to perform for every block transfer; a single transaction is mapped to a single beat. **Width**: The width of the single transaction is determined by `SRC_TR_WIDTH`. Once the transfer starts, the read-back value is the total number of data items already read from the source peripheral. **Value After Reset**: 0x2
[31:29]			Reserved
[28]	`LLP_SRC_EN`	R/W	This bit field is available only for channel 2 and 3. Block chaining is enabled on the source side only if the `LLP_SRC_EN` field is high and `LLPx.LOC` is non-zero. For more information, see Block Chaining using Linked Lists. **Values:** **0x0** (`LLP_SRC_DISABLE`): Block chaining using Linked List is disabled on the Source side **0x1** (`LLP_SRC_ENABLE`): Block chaining using Linked List is enabled on the Source side **Value After Reset:** 0x0
[27]	`LLP_DST_EN`	R/W	This bit field is available only for channel 2 and 3. Block chaining is enabled on the destination side only if `LLP_DST_EN` field is high and `LLPx.LOC` is non-zero. For more information, see Block Chaining using Linked Lists. **Values:**

Table 8-9 Fields for register: `CTLx` (continued)

Bits	Name	R/W	Description
			0x0 (`LLP_DST_DISABLE`): Block chaining using Linked List is disabled on the Destination side **0x1** (`LLP_DST_ENABLE`): Block chaining using Linked List is enabled on the Destination side **Value After Reset:** 0x0
[26:22]			Reserved
[21:20]	`TT_FC`	R/W	**Transfer Type and Flow Control.** Flow control is assigned to the DMA controller. **Values:** **0x0** (`TT_FC_0`): Transfer type is Memory to Memory **0x1** (`TT_FC_1`): Transfer type is Memory to Peripheral **0x2** (`TT_FC_2`): Transfer type is Peripheral to Memory **0x3** (`TT_FC_3`): Transfer type is Peripheral to Peripheral **Value After Reset:** 0x3
[19:17]			Reserved
[16:14]	`SRC_MSIZE`	R/W	Source Burst Transaction Length Number of data items, each of width `SRC_TR_WIDTH`, to be read from the source every time a source burst transaction request is made from either the corresponding hardware or software handshaking interface. See the decoding for this field. **Value After Reset**: 0x1
[13:11]	`DEST_MSIZE`	R/W	Destination Burst Transaction Length Number of data items, each of width `DST_TR_WIDTH`, to be written to the destination every time a destination burst transaction request is made from either the corresponding hardware or software handshaking interface. See the decoding for this field. **Value After Reset**: 0x1
[10:9]	`SINC`	R/W	Source Address Increment Indicates whether to increment or decrement the source address on every source transfer. If the device is fetching data from a source peripheral FIFO with a fixed address, then set this field to “No change”. **Values:** **0x0** (SINC_0): Increments the source address **0x1** (SINC_1): Decrements the source address **0x2** (SINC_2): No change in the source address **0x3** (SINC_3): No change in the source address **Value After Reset**: 0x0
[8:7]	`DINC`	R/W	Destination Address Increment

Table 8-9 Fields for register: `CTLx` (continued)

Bits	Name	R/W	Description
			Indicates whether to increment or decrement the destination address on every destination transfer. If your device is writing data to a destination peripheral FIFO with a fixed address, then set this field to "No change". **Values:** **0x0** (DINC_0): Increments the destination address **0x1** (DINC_1): Decrements the destination address **0x2** (DINC_2): No change in the destination address **0x3** (DINC_3): No change in the destination address **Value After Reset**: 0x0
[6:4]	`SRC_TR_WIDTH`	R/W	Source Transfer Width (See the decoding for this field) Mapped to bus `hsize`. For a non-memory peripheral, typically the peripheral (source) FIFO width. **Value After Reset**: 0x0
[3:1]	`DST_TR_WIDTH`	R/W	Destination Transfer Width See the decoding for this field. Mapped to bus `hsize`. For a non-memory peripheral, typically `rgw` peripheral (destination) FIFO width. **Value After Reset**:0x0
[0]	`INT_EN`	R/W	Interrupt Enable Bit If set, then all interrupt-generating sources are enabled. Functions as a global mask bit for all interrupts for the channel; raw interrupt registers still assert if `INT_EN = 0`. **Value After Reset**:0x1

The following table shows the decoding of the **`CTLx.SRC_MSIZE / CTLx.DEST_MSIZE`** bits.

Table 8-10 `CTLx.SRC_MSIZE` and `CTLx.DEST_MSIZE` decoding

`CTLx.SRC_MSIZE` / `CTLx.DEST_MSIZE`	Number of data items to be transferred (of width `CTLx.SRC_TR_WIDTH` or `CTLx.DST_TR_WIDTH`)
000	1
001	4
010	8 (Can be set only for channels 2 and 3)
011	Using these values will result in erroneous behavior
100	
101	
110	
111	

The following table shows the decoding of the `CTLx.SRC_TR_WIDTH / CTLx.DST_TR_WIDTH` bits.

Table 8-11 `CTLx.SRC_TR_WIDTH` and `CTLx.DST_TR_WIDTH` decoding

`CTLx.SRC_TR_WIDTH / CTLx.DST_TR_WIDTH`	Size (bits)
000	8
001	16
010	32
011	Do not use these values
100	
101 / 111	

The following table shows the decoding of the `CTLx.TT_FC` bits.

Table 8-12 `CTLx.TT_FC` field decoding

`CTLx.TT_FC` Field	Transfer Type
000	Memory to Memory
001	Memory to Peripheral
010	Peripheral to Memory
011	Peripheral to Peripheral

8.2.2.1.6. Configuration Register (`CFGx`) for Channel x

The `CFGx` register is at offset: for x=0 to 7: 0x040+**x***0x058.

This register contains fields that configure the DMA transfer. The channel configuration register remains fixed for all blocks of a multi-block transfer.

You must program this register prior to enabling the channel.

The following table shows the register bit assignments.

Table 8-13 Fields for register: `CFGx`

Bits	Name	R/W	Description
[63:47]			Reserved
[46:43]	`DEST_PER`	R/W	Destination hardware interface This bit field assigns a hardware handshaking interface (0 - 15) to the destination of channel `x` if the `HS_SEL_DST` field is 0; otherwise, this field is ignored. The channel can then communicate with the destination peripheral connected to that interface through the assigned hardware handshaking interface.  For correct DMA operation, only one peripheral (source or destination) should be assigned to the same handshaking interface. **Value After Reset**: 0x0

Table 8-13 Fields for register: CFGx (continued)

Bits	Name	R/W	Description
[42:39]	SRC_PER	R/W	Source Hardware Interface This bit field assigns a hardware handshaking interface (0 - 15) to the source of channel x if the HS_SEL_SRC field is 0; otherwise, this field is ignored. The channel can then communicate with the source peripheral connected to that interface through the assigned hardware handshaking interface. **Value After Reset**: 0x0
[38:37]			Reserved
[36:34]	PROTCTL	R/W	Protection Control Protection Control bits are used to drive the HPROT[3:1] bus. The default value of HPROT indicates a non-cached, non-buffered, privileged data access. The Value After Reset is used to indicate such an access. HPROT[0] is tied high because all transfers are data accesses, as there are no opcode fetches. There is a one-to-one mapping of these register bits to the HPROT[3:1] master interface signals. The table below shows the mapping of bits in this field to the HPROT[3:1] bus. **Value After Reset**: 0x1
[33]	FIFO_MODE	R/W	FIFO Mode Select This bit determines how much space or data needs to be available in the FIFO before a burst transaction request is serviced. **Values:** **0x0:** Space/data available for single transfer of the specified transfer width. **0x1:** Data available is greater than or equal to half the FIFO depth for destination transfers and space available is greater than half the FIFO depth for source transfers. The exceptions are at the end of a burst transaction request or at the end of a block transfer. **Value After Reset**: 0x0
[32]	FCMODE	R/W	Flow Control Mode When the destination peripheral is configured as the flow controller, this bit determines how source transaction requests are serviced. This bit must always be set to 0x0 because the DMA is always the flow controller. **Values**: **0x0:** Source transaction requests are serviced when they occur. Data pre-fetching is enabled. **Value After Reset**: 0x0
[31]	RELOAD_DST	R/W	This bit field is available only for channels 2 and 3.

Table 8-13 Fields for register: CFGx (continued)

Bits	Name	R/W	Description
			Automatic Destination Reload. The DARx register can be automatically reloaded from its initial value at the end of every block for multi-block transfers. A new block transfer is then initiated. **Values:** **0x0** (DISABLE): Destination reload disabled **0x1** (ENABLE): Destination reload enabled **Value After Reset:** 0x0
[30]	RELOAD_SRC	R/W	This bit field is available only for channels 2 and 3. **Automatic Source Reload.** The SARx register can be automatically reloaded from its initial value at the end of every block for multi-block transfers. A new block transfer is then initiated. **Values:** **0x0** (DISABLE): Source reload disabled **0x1** (ENABLE): Source reload enabled **Value After Reset:** 0x0
[29:20]			Reserved
[19]	SRC_HS_POL	R/W	Source Handshaking Interface Polarity **Values:** **0x0:** Active high **0x1:** Active low **Value After Reset**: 0x0
[18]	DST_HS_PL	R/W	Destination Handshaking Interface Polarity **Values:** **0x0:** Active high **0x1:** Active low **Value After Reset**: 0x0
[17:12]			Reserved
[11]	HS_SEL_SRC	R/W	Source Software or Hardware Handshaking Select This register field selects which of the handshaking interfaces – hardware or software – is active for source requests on this channel. **Values:** **0x0:** Hardware handshaking interface. Software-initiated transaction requests are ignored. **0x1:** Software handshaking interface. Hardware-initiated transaction requests are ignored. If the source peripheral is memory, then this bit is ignored. **Value After Reset**: 0x1
[10]	HS_SEL_DST	R/W	Destination Software or Hardware Handshaking Select

Table 8-13 Fields for register: `CFGx` (continued)

Bits	Name	R/W	Description
			This register selects which of the handshaking interfaces — hardware or software — is active for destination requests on this channel. **Values:** **0x0:** Hardware handshaking interface. Software-initiated transaction requests are ignored. **0x1:** Software handshaking interface. Hardware-initiated transaction requests are ignored. If the destination peripheral is memory, then this bit is ignored. **Value After Reset**: 0x1
[9]	`FIFO_EMPTY`	R	Channel FIFO status Indicates if there is data left in the channel FIFO. Can be used in conjunction with `CH_SUSP` to cleanly disable a channel. **Values:** **0x0** (NOT_EMPTY): Channel FIFO is not empty **0x1** (EMPTY): Channel FIFO is empty **Value After Reset**: 0x1
[8]	`CH_SUSP`	R/W	Channel Suspend Suspends all DMA data transfers from the source until this bit is cleared. There is no guarantee that the current transaction will complete. Can also be used in conjunction with `FIFO_EMPTY` to cleanly disable a channel without losing any data. **Values:** **0x0** (NOT_SUSPENDED): DMA transfer from the source is not suspended **0x1** (SUSPENDED): Suspend DMA transfer from the source **Value After Reset**: 0x0
[7:5]	`CH_PRIOR`	R/W	Channel priority A priority of 7 is the highest priority, and 0 is the lowest. This field must be programmed within the following range: 0:7 A programmed value outside this range will cause erroneous behavior. **Values:** **0x0** (`CH_PRIOR_0`): Channel priority is 0 **0x1** (`CH_PRIOR_1`): Channel priority is 1 **0x2** (`CH_PRIOR_2`): Channel priority is 2 **0x3** (`CH_PRIOR_3`): Channel priority is 3 **0x4** (`CH_PRIOR_4`): Channel priority is 4 **0x5** (`CH_PRIOR_5`): Channel priority is 5 **0x6** (`CH_PRIOR_6`): Channel priority is 6 **0x7** (`CH_PRIOR_7`): Channel priority is 7 **Value After Reset**: Channel Number (x)

Table 8-13 Fields for register: `CFGx` (continued)

Bits	Name	R/W	Description
[4:0]			Reserved

Table 8-14 `PROTCTL` field to `HPROT` mapping

1'b1	HPROT[0]
CFGx.PROTCTL[1]	HPROT[1]
CFGx.PROTCTL[2]	HPROT[2]
CFGx.PROTCTL[3]	HPROT[3]

8.2.2.2. Interrupt Registers

8.2.2.2.1. Raw Status for `IntTfr` Interrupt (`RAW_TFR`)

The `RAW_TFR` register is at offset 0x2C0.

Interrupt events are stored in this Raw Interrupt Status register before masking. This register has a bit allocated per channel; for example, `RAW_TFR[2]` is the Channel 2 raw transfer complete interrupt. Each bit in this register is cleared by writing a 1 to the corresponding location in the `CLR_TFR` register.

Write access is available to this register or software testing purposes only. Under normal operation, writes to this register are not recommended.

The following table shows the register bit assignments.

Table 8-15 Fields for register: `RAW_TFR`

Bits	Name	R/W	Description
[31:8]			Reserved
[7:0]	RAW	R/W	Raw Status for `IntTfr` Interrupt **Values:** **0x0** (INACTIVE): Inactive Raw Interrupt Status **0x1** (ACTIVE): Active Raw Interrupt Status **Value After Reset** : 0x0

8.2.2.2.2. Raw Status for `IntBlock` Interrupt (`RAW_BLOCK`)

The `RAW_BLOCK` register is at offset 0x2C8.

Interrupt events are stored in this Raw Interrupt Status register before masking. This register has a bit allocated per channel; for example, `RAW_BLOCK[2]` is the Channel 2 raw block complete interrupt. Each bit in this register is cleared by writing a 1 to the corresponding location in the `CLR_BLOCK` register.

The following table shows the register bit assignments.

Table 8-16 Fields for register: `RAW_BLOCK`

Bits	Name	R/W	Description
[31:8]			Reserved
[7:0]	`RAW`	R/W	Raw Status for `IntBlock` Interrupt **Values:** **0x0** (INACTIVE): Inactive Raw Interrupt Status **0x1** (ACTIVE): Active Raw Interrupt Status **Value After Reset:** 0x0

8.2.2.2.3. Raw Status for `IntSrcTran` Interrupt (`RAW_SRC_TRAN`)

The `RAW_SRC_TRAN` register is at offset 0x2D0.

Interrupt events are stored in this Raw Interrupt Status register before masking. This register has a bit allocated per channel; for example, `RAW_SRC_TRAN[2]` is the Channel 2 raw source transaction complete interrupt. Each bit in this register is cleared by writing a 1 to the corresponding location in the `CLR_SRC_TRAN` register.

The following table shows the register bit assignments.

Table 8-17 Fields for register: `RAW_SRC_TRAN`

Bits	Name	R/W	Description
[31:8]			Reserved
[7:0]	`RAW`	R/W	Raw Status for `IntSrcTran` Interrupt **Values:** **0x0** (INACTIVE): Inactive Raw Interrupt Status **0x1** (ACTIVE): Active Raw Interrupt Status **Value After Reset**: 0x0

8.2.2.2.4. Raw Status for `IntDstTran` Interrupt (`RAW_DST_TRAN`)

The `RAW_DST_TRAN` register is at offset 0x2D8.

Interrupt events are stored in this Raw Interrupt Status register before masking. This register has a bit allocated per channel; for example, `RAW_DST_TRAN[2]` is the Channel 2 raw destination transaction complete interrupt. Each bit in this register is cleared by writing a 1 to the corresponding location in the `CLR_DST_TRAN` register.

The following table shows the register bit assignments.

Table 8-18 Fields for register: `RAW_DST_TRAN`

Bits	Name	R/W	Description
[31:8]			Reserved
[7:0]	`RAW`	R/W	Raw Status for `IntDstTran` Interrupt **Values:**

Table 8-18 Fields for register: RAW_DST_TRAN (continued)

Bits	Name	R/W	Description
			0x0 (INACTIVE): Inactive Raw Interrupt Status **0x1** (ACTIVE): Active Raw Interrupt Status **Value After Reset** : 0x0

8.2.2.2.5. Raw Status for IntErr Interrupt (RAW_ERR)

The RAW_ERR register is at offset 0x2E0.

Interrupt events are stored in this Raw Interrupt Status register before masking. This register has a bit allocated per channel; for example, RAW_ERR[2] is the Channel 2 raw error interrupt. Each bit in this register is cleared by writing a 1 to the corresponding location in the CLR_ERR register.

Write access is available to this register or software testing purposes only. Under normal operation, writes to this register are not recommended.

The following table shows the register bit assignments.

Table 8-19 Fields for register: RAW_ERR

Bits	Name	R/W	Description
[31:8]			Reserved
[7:0]	RAW	R/W	Raw Status for IntDstTran Interrupt **Values:** **0x0** (INACTIVE): Inactive Raw Interrupt Status **0x1** (ACTIVE): Active Raw Interrupt Status **Value After Reset**: 0x0

8.2.2.2.6. Status for IntTfr Interrupt (STAT_TFR)

The STAT_TFR register is at offset 0x2E8.

Channel DMA Transfer complete interrupt event from all channels is stored in this Interrupt Status register after masking. This register has a bit allocated per channel; for example, STAT_TFR[2] is the Channel 2 source DMA transfer complete interrupt. The contents of this register are used to generate the interrupt signals (int bus) leaving the DMA Controller.

The following table shows the register bit assignments.

Table 8-20 Fields for register: STAT_TFR

Bits	Name	R/W	Description
[31:8]			Reserved
[7:0]	STATUS	R	Status for IntTfr Interrupt **Values:** **0x0** (INACTIVE): Inactive Interrupt Status **0x1** (ACTIVE): Active Interrupt Status **Value After Reset:** 0x0

8.2.2.2.7. Status for `IntBlock` Interrupt (`STAT_BLOCK`)

The `STAT_BLOCK` register is at offset 0x2F0.

Channel Block complete interrupt event from all channels is stored in this Interrupt Status register after masking. This register has a bit allocated per channel; for example, `STAT_BLOCK[2]` is the Channel 2 block complete interrupt. The contents of this register are used to generate the interrupt signals (`int` bus) leaving the DMA Controller.

The following table shows the register bit assignments.

Table 8-21 Fields for register: `STAT_BLOCK`

Bits	Name	R/W	Description
[31:8]			Reserved
[7:0]	`STATUS`	R	Status for `IntBlock` Interrupt **Values:** **0x0** (INACTIVE): Inactive Interrupt Status **0x1** (ACTIVE): Active Interrupt Status **Value After Reset:** 0x0

8.2.2.2.8. Status for `IntSrcTran` Interrupt (`STAT_SRC_TRAN`)

The `STAT_SRC_TRAN` register is at offset 0x2F8.

Channel Source Transaction complete interrupt event from all channels is stored in this Interrupt Status register after masking. This register has a bit allocated per channel; for example, `STAT_SRC_TRAN[2]` is the Channel 2 source transaction complete interrupt. The contents of this register are used to generate the interrupt signals (`int` bus) leaving the DMA.

The following table shows the register bit assignments.

Table 8-22 Fields for register: `STAT_SRC_TRAN`

Bits	Name	R/W	Description
[31:8]			Reserved
[7:0]	`STATUS`	R	Status for `IntSrcTran` Interrupt **Values:** **0x0** (INACTIVE): Inactive Interrupt Status **0x1** (ACTIVE): Active Interrupt Status **Value After Reset:** 0x0

8.2.2.2.9. Status for `IntDstTran` Interrupt (`STAT_DST_TRAN`)

The `STAT_DST_TRAN` register is at offset 0x300.

Channel destination transaction complete interrupt event from all channels is stored in this Interrupt Status register after masking. This register has a bit allocated per channel; for example, `STAT_DST_TRAN[2]` is the Channel 2 status destination transaction complete interrupt. The contents of this register are used to generate the interrupt signals (`int` bus) leaving the DMA Controller.

The following table shows the register bit assignments.

Table 8-23 Fields for register: `STAT_DST_TRAN`

Bits	Name	R/W	Description
[31:8]			Reserved
[7:0]	`STATUS`	R	Status for `IntDstTran` Interrupt **Values:** **0x0** (INACTIVE): Inactive Interrupt Status **0x1** (ACTIVE): Active Interrupt Status **Value After Reset:** 0x0

8.2.2.2.10. Status for `IntErr` Interrupt (`STAT_ERR`)

The `STAT_ERR` register is at offset 0x308.

Channel Error interrupt event from all channels is stored in this Interrupt Status register after masking. This register has a bit allocated per channel; for example, `STAT_ERR[2]` is the Channel 2 status Error interrupt. The contents of this register are used to generate the interrupt signals (`int` bus) leaving the DMA Controller.

The following table shows the register bit assignments.

Table 8-24 Fields for register: `STAT_ERR`

Bits	Name	R/W	Description
[31:8]			Reserved
[7:0]	`STATUS`	R	Status for `IntErr` Interrupt **Values:** **0x0** (INACTIVE): Inactive Interrupt Status **0x1** (ACTIVE): Active Interrupt Status **Value After Reset:** 0x0

8.2.2.2.11. Mask for `IntTfr` Interrupt (`MASK_TFR`)

The `MASK_TFR` register is at offset 0x310.

The following table shows the register bit assignments.

Table 8-25 Fields for register: `MASK_TFR`

Bits	Name	R/W	Description
[31:16]			Reserved
[15:8]	`INT_MASK_WE`	W	Interrupt Mask Write Enable **Values:** **0x0** (DISABLED): Interrupt mask write disable **0x1** (ENABLED): Interrupt mask write enable **Value After Reset:** 0x0
[7:0]	`INT_MASK`	R/W	Mask for `IntTfr` Interrupt **Values:**

Table 8-25 Fields for register: `MASK_TFR` (continued)

Bits	Name	R/W	Description
			0x0 (MASK): Mask the interrupts **0x1** (UNMASK): Unmask the interrupts **Value After Reset:** 0x0

8.2.2.2.12. Mask for `IntBlock` Interrupt (`MASK_BLOCK`)

The `MASK_BLOCK` register is at offset 0x318.

The following table shows the register bit assignments.

Table 8-26 Fields for register: `MASK_BLOCK`

Bits	Name	R/W	Description
[31:16]			Reserved
[15:8]	`INT_MASK_WE`	W	Interrupt Mask Write Enable **Values:** **0x0** (DISABLED): Interrupt mask write disable **0x1** (ENABLED): Interrupt mask write enable **Value After Reset:** 0x0
[7:0]	`INT_MASK`	R/W	Mask for `IntBlock` Interrupt **Values:** **0x0** (MASK): Mask the interrupts **0x1** (UNMASK): Unmask the interrupts **Value After Reset:** 0x0

8.2.2.2.13. Mask for `IntSrcTran` Interrupt (`MASK_SRC_TRAN`)

The `MASK_SRC_TRAN` register is at offset 0x320.

The following table shows the register bit assignments.

Table 8-27 Fields for register: `MASK_SRC_TRAN`

Bits	Name	R/W	Description
[31:16]			Reserved
[15:8]	`INT_MASK_WE`	W	Interrupt Mask Write Enable **Values:** **0x0** (DISABLED): Interrupt mask write disable **0x1** (ENABLED): Interrupt mask write enable **Value After Reset:** 0x0
[7:0]	`INT_MASK`	R/W	Mask for `IntSrcTran` Interrupt **Values:**

Table 8-27 Fields for register: `MASK_SRC_TRAN` (continued)

Bits	Name	R/W	Description
			0x0 (MASK): Mask the interrupts **0x1** (UNMASK): Unmask the interrupts **Value After Reset:** 0x0

8.2.2.2.14. Mask for `IntDstTran` Interrupt (`MASK_DST_TRAN`)

The `MASK_DST_TRAN` register is at offset 0x328.

The following table shows the register bit assignments.

Table 8-28 Fields for register: `MASK_DST_TRAN`

Bits	Name	R/W	Description
[31:16]			Reserved
[15:8]	`INT_MASK_WE`	W	Interrupt Mask Write Enable **Values:** **0x0** (DISABLED): Interrupt mask write disable **0x1** (ENABLED): Interrupt mask write enable **Value After Reset:** 0x0
[7:0]	`INT_MASK`	R/W	Mask for `IntDstTran` Interrupt **Values:** **0x0** (MASK): Mask the interrupts **0x1** (UNMASK): Unmask the interrupts **Value After Reset:** 0x0

8.2.2.2.15. Mask for `IntErr` Interrupt (`MASK_ERR`)

The `MASK_ERR` register is at offset 0x330.

The following table shows the register bit assignments.

Table 8-29 Fields for register: `MASK_ERR`

Bits	Name	R/W	Description
[31:16]			Reserved
[15:8]	`INT_MASK_WE`	W	Interrupt Mask Write Enable **Values:** **0x0** (DISABLED): Interrupt mask write disable **0x1** (ENABLED): Interrupt mask write enable **Value After Reset:** 0x0
[7:0]	`INT_MASK`	R/W	Mask for `IntErr` Interrupt **Values:**

Table 8-29 Fields for register: `MASK_ERR` (continued)

Bits	Name	R/W	Description
			0x0 (MASK): Mask the interrupts **0x1** (UNMASK): Unmask the interrupts **Value After Reset:** 0x0

8.2.2.2.16. Clear for `IntTfr` Interrupt (`CLR_TFR`)

The `CLR_TFR` register is at offset 0x338.

The following table shows the register bit assignments.

Table 8-30 Fields for register: `CLR_TFR`

Bits	Name	R/W	Description
[31:8]			Reserved
[7:0]	`CLEAR`	W	Clear for `IntTfr` Interrupt **Values:** **0x0** (NOT_CLEAR): No effect **0x1** (CLEAR): Clears interrupts **Value After Reset:** 0x0

8.2.2.2.17. Clear for `IntBlock` Interrupt (`CLR_BLOCK`)

The `CLR_BLOCK` register is at offset 0x340.

Each bit in the `RAW_BLOCK` and `STAT_BLOCK` is cleared on the same cycle by writing a 1 to the corresponding location in the this registers. Each bit is allocated per channel; for example, `CLR_BLOCK[2]` is the clear bit for the Channel 2 block done interrupt. Writing a 0 has no effect. This registers are not readable.

The following table shows the register bit assignments.

Table 8-31 Fields for register: `CLR_BLOCK`

Bits	Name	R/W	Description
[31:8]			Reserved
[7:0]	`CLEAR`	W	Clear for `IntBlock` Interrupt **Value After Reset** : 0x0

8.2.2.2.18. Clear for `IntSrcTran` Interrupt (`CLR_SRC_TRAN`)

The `CLR_SRC_TRAN` register is at offset 0x348.

Each bit in the `RAW_SRC_TRAN` and `STAT_SRC_TRAN` is cleared on the same cycle by writing a 1 to the corresponding location in the this registers. Each bit is allocated per channel; for example, `CLR_SRC_TRAN[2]` is the clear bit for the Channel 2 source transaction done interrupt. Writing a 0 has no effect. These registers are not readable.

The following table shows the register bit assignments.

Table 8-32 Fields for register: `CLR_SRC_TRAN`

Bits	Name	R/W	Description
[31:8]			Reserved
[7:0]	`CLEAR`	W	Clear for `IntSrcTran` Interrupt **Values:** **0x0** (NOT_CLEAR): No effect **0x1** (CLEAR): Clears interrupts **Value After Reset:** 0x0

8.2.2.2.19. Clear for `IntDstTran` Interrupt (`CLR_DST_TRAN`)

The `CLR_DST_TRAN` register is at offset 0x350.

Each bit in the `RAW_DST_TRAN` and `STAT_DST_TRAN` is cleared on the same cycle by writing a 1 to the corresponding location in the this registers. Each bit is allocated per channel; for example, `CLR_DST_TRAN[2]` is the clear bit for the Channel 2 destination transaction done interrupt. Writing a 0 has no effect. These registers are not readable.

The following table shows the register bit assignments.

Table 8-33 Fields for register: `CLR_DST_TRAN`

Bits	Name	R/W	Description
[31:8]			Reserved
[7:0]	`CLEAR`	W	Clear for `IntDstTran` Interrupt **Values:** **0x0** (NOT_CLEAR): No effect **0x1** (CLEAR): Clears interrupts **Value After Reset:** 0x0

8.2.2.2.20. Clear for `IntErr` Interrupt (`CLR_ERR`)

The `CLR_ERR` register is at offset 0x358.

Each bit in the `RAW_ERR` and `STAT_ERR` is cleared on the same cycle by writing a 1 to the corresponding location in the this registers. Each bit is allocated per channel; for example, `CLR_ERR[2]` is the clear bit for the Channel 2 error interrupt. Writing a 0 has no effect. This registers are not readable.

The following table shows the register bit assignments.

Table 8-34 Fields for register: `CLR_ERR`

Bits	Name	R/W	Description
[31:8]			Reserved
[7:0]	`CLEAR`	W	Clear for `IntErr` Interrupt **Values:** **0x0** (NOT_CLEAR): No effect **0x1** (CLEAR): Clears interrupts

Table 8-34 Fields for register: CLR_ERR (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0

8.2.2.2.21. Status for Each Interrupt Type (STAT)

The STAT register is at offset 0x360.

he contents of each of the five Status registers STAT_TFR, STAT_BLOCK, STAT_SRC_TRAN, STAT_DST_TRAN, STAT_ERR is ORed to produce a single bit for each interrupt type in the Combined Status register (STAT).

This register is read-only.

The following table shows the register bit assignments.

Table 8-35 Fields for register: STAT

Bits	Name	R/W	Description
[31:5]			Reserved
[4]	ERR	R	OR of the contents of STAT_ERR **Values:** **0x0** (INACTIVE): OR of the contents of STAT_ERR register is 0 **0x1** (ACTIVE): OR of the contents of STAT_ERR register is 1 **Value After Reset:** 0x0
[3]	DSTT	R	OR of the contents of STAT_DST_TRAN **Values:** **0x0** (INACTIVE): OR of the contents of STAT_DST_TRAN register is 0 **0x1** (ACTIVE): OR of the contents of STAT_DST_TRAN register is 1 **Value After Reset:** 0x0
[2]	SRCT	R	OR of the contents of STAT_SRC_TRAN **Values:** **0x0** (INACTIVE): OR of the contents of STAT_SRC_TRAN register is 0 **0x1** (ACTIVE): OR of the contents of STAT_SRC_TRAN register is 1 **Value After Reset:** 0x0
[1]	BLOCK	R	OR of the contents of STAT_BLOCK register **Values:** **0x0** (INACTIVE): OR of the contents of STAT_BLOCK register is 0 **0x1** (ACTIVE): OR of the contents of STAT_BLOCK register is 1 **Value After Reset:** 0x0
[0]	TFR	R	OR of the contents of STAT_TFR register **Values:** **0x0** (INACTIVE): OR of the contents of STAT_TFR register is 0 **0x1** (ACTIVE): OR of the contents of STAT_TFR register is 1

Table 8-35 Fields for register: STAT (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0

8.2.2.3. Software Handshaking Registers

The registers that comprise the software handshaking registers allow software to initiate single or burst transaction requests in the same way that handshaking interface signals do in hardware.

8.2.2.3.1. Source Software Transaction Request Register (REQ_SRC)

The REQ_SRC register is at offset 0x368.

A bit is assigned for each channel in this register. REQ_SRC[n] is ignored when software handshaking is not enabled for the source of channel n.

A channel SRC_REQ bit is written only if the corresponding channel write enable bit in the SRC_REQ_WE field is asserted on the same write transfer, and if the channel is enabled in the CH_EN register. For example, writing hex 0101 writes a 1 into REQ_SRC[0], while REQ_SRC[7:1] remains unchanged. Writing hex 00xx leaves REQ_SRC[7:0] unchanged. This allows software to set a bit in the REQ_SRC register without performing a read-modified write operation.

The following table shows the register bit assignments.

Table 8-36 Fields for register: REQ_SRC

Bits	Name	R/W	Description
[31:16]			Reserved
[15:8]	SRC_REQ_WE	W	Source Software Transaction Request write enable **Values:** **0x0** (DISABLED): Source request write disable **0x1** (ENABLED): Source request write enable **Value After Reset:** 0x0
[7:0]	SRC_REQ	R/W	Source Software Transaction Request **Values:** **0x0** (INACTIVE): Source request is not active **0x1** (ACTIVE): Source request is active **Value After Reset:** 0x0

8.2.2.3.2. Destination Software Transaction Request Register (REQ_DST)

The REQ_DST register is at offset 0x370.

A bit is assigned for each channel in this register. REQ_DST[n] is ignored when software handshaking is not enabled for the source of channel n.

A channel DST_REQ bit is written only if the corresponding channel write enable bit in the DST_REQ_WE field is asserted on the same write transfer, and if the channel is enabled in the CH_EN register.

The following table shows the register bit assignments.

Table 8-37 Fields for register: `REQ_DST`

Bits	Name	R/W	Description
[31:16]			Reserved
[15:8]	`DST_REQ_WE`	W	Destination Software Transaction Request write enable **Values:** **0x0** (DISABLED): Destination request write disable **0x1** (ENABLED): Destination request write enable **Value After Reset:** 0x0
[7:0]	`DST_REQ`	R/W	Destination Software Transaction Request **Values:** **0x0** (INACTIVE): Destination request is not active **0x1** (ACTIVE): Destination request is active **Value After Reset:** 0x0

8.2.2.3.3. Source Single Transaction Request Register (`SGL_REQ_SRC`)

The `SGL_REQ_SRC` register is at offset 0x378.

A bit is assigned for each channel in this register. `SGL_REQ_SRC[n]` is ignored when software handshaking is not enabled for the source of channel n.

A channel `SRC_SGLREQ` bit is written only if the corresponding channel write enable bit in the `SRC_SGLREQ_WE` field is asserted on the same write transfer, and if the channel is enabled in the `CH_EN` register.
The following table shows the register bit assignments.

Table 8-38 Fields for register: `SGL_REQ_SRC`

Bits	Name	R/W	Description
[63:16]			Reserved
[15:8]	`SRC_SGLREQ_WE`	W	Source Single Transaction Request write enable **Values:** **0x0** (DISABLED): Single write disable **0x1** (ENABLED): Single write enable **Value After Reset:** 0x0
[7:0]	`SRC_SGLREQ`	R/W	Source Single Transaction Request **Values:** **0x0** (INACTIVE): Source request is not active **0x1** (ACTIVE): Source request is active **Value After Reset:** 0x0

8.2.2.3.4. Destination Single Transaction Request Register (`SGL_REQ_DST`)

The `SGL_REQ_DST` register is at offset 0x380.

A bit is assigned for each channel in this register. SGL_REQ_DST[n] is ignored when software handshaking is not enabled for the source of channel n.

A channel DST_SGLREQ bit is written only if the corresponding channel write enable bit in the DST_SGLREQ_WE field is asserted on the same write transfer, and if the channel is enabled in the CH_EN register.

The following table shows the register bit assignments.

Table 8-39 Fields for register: SGL_REQ_DST

Bits	Name	R/W	Description
[31:16]			Reserved
[15:8]	DST_SGLREQ_WE	W	Destination Single Transaction Request write enable **Values:** **0x0** (DISABLED): Destination write disable **0x1** (ENABLED): Destination write enable **Value After Reset:** 0x0
[7:0]	DST_SGLREQ	R/W	Destination Single Transaction Request **Values:** **0x0** (INACTIVE): Destination Single or burst request is not active **0x1** (ACTIVE): Destination Single or burst request is active **Value After Reset:** 0x0

8.2.2.3.5. Source Last Transaction Request Register (LST_SRC)

The LST_SRC register is at offset 0x388.

A bit is assigned for each channel in this register. LST_SRC[n] is ignored when software handshaking is not enabled for the source of channel n.

A channel LSTSRC bit is written only if the corresponding channel write enable bit in the LSTSRC_WE field is asserted on the same write transfer, and if the channel is enabled in the CH_EN register.
The following table shows the register bit assignments.

Table 8-40 Fields for register: LST_SRC

Bits	Name	R/W	Description
[31:16]			Reserved
[15:8]	LSTSRC_WE	W	Source Last Transaction Request write enable **Values:** **0x0** (DISABLED): Source last transaction request write disable **0x1** (ENABLED): Source last transaction request write enable **Value After Reset:** 0x0
[7:0]	LSTSRC	R/W	Source Last Transaction Request register **Values:**

Table 8-40 Fields for register: LST_SRC (continued)

Bits	Name	R/W	Description
			0x0 (NOT_LAST): Not last transaction in current block **0x1** (LAST): Last transaction in current block **Value After Reset:** 0x0

8.2.2.3.6. Destination Last Transaction Request Register (LST_DST)

The LST_DST register is at offset 0x390.

A bit is assigned for each channel in this register. LST_DST[n] is ignored when software handshaking is not enabled for the destination of channel n.

A channel LSTDST bit is written only if the corresponding channel write enable bit in the LSTDST_WE field is asserted on the same write transfer, and if the channel is enabled in the CH_EN register.

The following table shows the register bit assignments.

Table 8-41 Fields for register: LST_DST

Bits	Name	R/W	Description
[31:16]			Reserved
[15:8]	LSTDST_WE	W	Source Last Transaction Request write enable **Values:** **0x0** (DISABLED): Destination last transaction request write disable **0x1** (ENABLED): Destination last transaction request write enable **Value After Reset:** 0x0
[7:0]	LSTDST	R/W	Destination Last Transaction Request **Values:** **0x0** (NOT_LAST): Not last transaction in current block **0x1** (LAST): Last transaction in current block **Value After Reset:** 0x0

8.2.2.4. Miscellaneous DMA Controller Registers

8.2.2.4.1. DMA Configuration Register (CFG)

The CFG register is at offset 0x398.

This register is used to enable the DMA Controller, which must be done before any channel activity can begin.

If the global channel enable bit is cleared while any channel is still active, then CFG.DMA_EN still returns 1 to indicate that there are channels still active until hardware has terminated all activity on all channels, at which point the CFG.DMA_EN bit returns 0. For more information, refer to Abnormal Transfer Termination.

The following table shows the register bit assignments.

Table 8-42 Fields for register: `CFG`

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	`DMA_EN`	R/W	DMA Controller Enable bit Valid values: **0x0:** DMA Controller Disabled **0x1:** DMA Controller Enabled **Reset value**: 0x0

8.2.2.4.2. DMA Channel Enable Register (`CH_EN`)

The `CH_EN` register is at offset 0x3A0.

This is the DMA Controller Channel Enable Register. If software needs to set up a new channel, then it can read this register in order to find out which channels are currently inactive; it can then enable an inactive channel with the required priority.

All bits of this register are cleared to 0 when the global DMA Controller channel enable bit, `CFG[0]`, is 0. When the global channel enable bit is 0, then a write to the `CH_EN` register is ignored and a read will always read back 0.

The channel enable bit, `CH_EN.CH_EN`, is written only if the corresponding channel write enable bit, `CH_EN.CH_EN_WE`, is asserted on the same write transfer. For example, writing hex 01x1 writes a 1 into `CH_EN[0]`, while `CH_EN[7:1]` remains unchanged. Writing hex `00xx` leaves `CH_EN[7:0]` unchanged. Note that a read-modified write is not required.

For information on software disabling a channel by writing 0 to `CH_EN.CH_EN`, refer to Disabling a Channel Prior to Transfer Completion.

The following table shows the register bit assignments.

Table 8-43 Fields for register: `CH_EN`

Bits	Name	R/W	Description
[31:16]			Reserved
[15:8]	`CH_EN_WE`	W	Channel enable register **Value After Reset** : 0x0
[7:0]	`CH_EN`	R/W	**Channel Enable** Valid values: **0x0:** Disable the Channel **0x1:** Enable the Channel The `CH_EN` bit is automatically cleared by hardware to disable the channel after the last transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Value After Reset** : 0x0

8.2.2.4.3. DMA Test Register (`TEST`)

The `TEST` register is at offset 0x3B0.

This register is used to put the Slave Interface into test mode, during which the readback value of the writable registers match the value written, assuming the DMA Controller configuration has not optimized the same registers. In normal operation, the readback value of some registers is a function of the DMA Controller state and does not match the value written.

The following table shows the register bit assignments.

Table 8-44 Fields for register: TEST

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	TEST_SLV_IF	R/W	Puts the Slave Interface into test mode. In this mode, the readback value of the writable registers always matches the value written. This bit does not allow writing to read-only registers. Valid values: **0x0** (NORMAL_MODE): Puts the Slave Interface into Normal mode **0x1** (TEST_MODE): Puts the Slave Interface into Test mode. In this mode, the readback value of the writable registers always matches the values written. **Value After Reset**: 0x0

8.3. Programming the DMA

8.3.1. Register Access

All registers are aligned to a 64-bit boundary and are 64 bits wide. In general, the upper 32 bits of a register are reserved. A write to reserved bits within the register is ignored. A read from reserved bits in the register reads back 0.

An attempt to read from write-only registers or write into read-only registers may lead to unpredictable results.

8.3.2. DMA Controller Transfer Types

A DMA transfer may consist of single or multi-block transfers. On successive blocks of a multi-block transfer, the SARx/DARx register in the DMA Controller is reprogrammed using either of the following methods:

- Block chaining using linked lists
- Auto-reloading
- Contiguous address between blocks

On successive blocks of a multi-block transfer, the CTLx register in the DMA Controller is reprogrammed using either of the following methods:

- Block chaining using linked lists
- Auto-reloading

When block chaining, using Linked Lists is the multi-block method of choice. On successive blocks, the LLPx register in the DMA Controller is reprogrammed using block chaining with linked lists. A

block descriptor consists of six registers: SARx, DARx, LLPx, CTLx. The first four registers, along with the CFGx register, are used by the DMA Controller to set up and describe the block transfer.

The term *Linked List Item* (**LLI**) and block descriptor are synonymous.

8.3.2.1. Multi-Block Transfers

Multi-block transfers - in which the source and destination are swapped during the transfer - are not supported. In a multi-block transfer, the direction must not change for the duration of the transfer.

8.3.2.1.1. Block Chaining Using Linked Lists

In this case, the DMA Controller reprograms the channel registers prior to the start of each block by fetching the block descriptor for that block from system memory. This is known as an LLI update.

DMA Controller block chaining uses a Linked List Pointer register (LLPx) that stores the address in memory of the next linked list item. Each LLI contains the corresponding block descriptors:

1. SARx
2. DARx
3. LLPx
4. CTLx

To set up block chaining, you program a sequence of Linked Lists in memory.

LLI accesses are always 32-bit accesses (Hsize = 2) aligned to 32-bit boundaries and cannot be changed or programmed to anything other than 32-bit, even if the Master Interface of the LLI supports more than a 32-bit data width.

The SARx, DARx, LLPx, and CTLx registers are fetched from system memory on an LLI update. The updated contents of the CTLx register is written back to memory on block completion.

Rows 6 through 10 of the table below show the required values of LLPx, CTLx, and CFGx for multi-block DMA transfers using block chaining.

For rows 6 through 10 of the table below, the LLI.CTLx, LLI.LLPx, LLI.SARx, and LLI.DARx register locations of the LLI are always affected at the start of every block transfer. The LLI.LLPx and LLI.CTLx locations are always used to reprogram the DMA Controller LLPx and CTLx registers. However, depending on the row number in the table below, the LLI.SARx/LLI.DARx address may or may not be used to reprogram the DMA Controller SARx/DARx registers.

Table 8-45 Transfer types

Transfer Type	LLPx[31:2] = 0	CTLx [28]	CFGx [30]	CTLx [27]	CFGx [31]	CTLx, LLPx Update Method	SARx Update Method	DARx Update Method	Write Back
1. Single-block or last transfer of multi-block.	Yes	0	0	0	0	None, user reprograms	None (single)	None (single)	No
2. Auto-reload multi-block transfer with contiguous SAR	Yes	0	0	0	1	CTLx, LLPx are reloaded from initial values.	Conti-guous	Auto-Reload	No

Table 8-45 Transfer types (continued)

Transfer Type	LLPx[31:2] = 0	CTLx [28]	CFGx [30]	CTLx [27]	CFGx [31]	CTLx, LLPx Update Method	SARx Update Method	DARx Update Method	Write Back
3. Auto-reload multi-block transfer with contiguous DAR.	Yes	0	1	0	0	CTLx, LLPx are reloaded from initial values.	Auto-Reload	Conti-guous	No
4. Auto-reload multi-block transfer	Yes	0	1	0	1	CTLx, LLPx are reloaded from initial values.	Auto-Reload	Auto-Reload	No
5. Single-block or last transfer of multi-block.	No	0	0	0	0	None, user reprograms	None (single)	None (single)	Yes
6. Linked list multi-block transfer with contiguous SAR	No	0	0	1	0	CTLx, LLPx loaded from next LLI.	Conti-guous	Linked List	Yes
7. Linked list multi-block transfer with auto-reload SAR	No	0	1	1	0	CTLx, LLPx loaded from next LLI.	Auto-Reload	Linked List	Yes
8. Linked list multi-block transfer with contiguous DAR	No	1	0	0	0	CTLx, LLPx loaded from next LLI.	Linked List	Conti-guous	Yes
9. Linked list multi-block transfer with auto-reload DAR	No	1	0	0	1	CTLx, LLPx loaded from next LLI.	Linked List	Auto-Reload	Yes
10. Linked list multi-block transfer	No	1	0	1	0	CTLx, LLPx loaded from next LLI.	Linked List	Linked List	Yes

Throughout this document, there are descriptions about fetching the LLI.CTLx register from the location pointed to by the LLPx register. This exact location is the LLI base address (stored in LLPx register) plus the fixed offset.

Referring to the table above, if the Write Back column entry is “Yes”, then the CTLx[63:32] register is always written to system memory (to LLI.CTLx[63:32]) at the end of every block transfer.

8.3.2.1.2. Auto-Reloading of Channel Registers

During auto-reloading, the channel registers are reloaded with their initial values at the completion of each block and the new values used for the new block. Depending on the row number in the table above some or all of the SARx, DARx, and CTLx channel registers are reloaded from their initial value at the start of a block transfer.

8.3.2.1.3. Contiguous Address Between Blocks

In this case, the address between successive blocks is selected as a continuation from the end of the previous block.

Enabling the source or destination address to be contiguous between blocks is a function of the CTLx.LLP_SRC_EN, CFGx.RELOAD_SRC, CTLx.LLP_DST_EN, and CTLx.RELOAD_DST registers (see the table above).

You cannot select both SARx and DARx updates to be contiguous. If you want this functionality, you should increase the size of the Block Transfer (CTLx.BLOCK_TS), or if this is at the maximum value, use Row 10 of the above table and set up the LLI.SARx address of the block descriptor to be equal to the end SARx address of the previous block. Similarly, set up the LLI.DARx address of the block descriptor to be equal to the end DARx address of the previous block. For more information, refer to Multi-Block Transfer with Linked List for Source and Linked List for Destination.

8.3.2.1.4. Suspension of Transfers Between Blocks

At the end of every block transfer, an end-of-block interrupt is asserted if:

1. Interrupts are enabled, CTLx.INT_EN = 1, and
2. The channel block interrupt is unmasked, MASK_BLOCK[n] = 1, where n is the channel number.

The block-complete interrupt is generated at the completion of the block transfer to the destination.

For rows 6, 8, and 10 of the table above, the DMA transfer does not stall between block transfers. For example, at the end-of-block N, the DMA Controller automatically proceeds to block N + 1.

For rows 2, 3, 4, 7, and 9 of the table above (SARx and/or DARx auto-reloaded between block transfers), the DMA transfer automatically stalls after the end-of-block interrupt is asserted, if the end-of-block interrupt is enabled and unmasked.

The DMA Controller does not proceed to the next block transfer until a write to the CLR_BLOCK[n] block interrupt clear register, done by software to clear the channel block-complete interrupt, is detected by hardware.

For rows 2, 3, 4, 7, and 9 of the table above (SARx and/or DARx auto-reloaded between block transfers), the DMA transfer does not stall if either:

- Interrupts are disabled, CTLx.INT_EN = 0, or
- The channel block interrupt is masked, MASK_BLOCK[n]= 0, where n is the channel number.

Channel suspension between blocks is used to ensure that the end-of-block *Interrupt Service Routine* (**ISR**) of the next-to-last block is serviced before the start of the final block commences. This ensures that the ISR has cleared the CFGx.RELOAD_SRC and/or CFGx.RELOAD_DST bits before completion of the final block. The reload bits CFGx.RELOAD_SRC and/or CFGx.RELOAD_DST should be cleared in the end-of-block ISR for the next-to-last block transfer.

8.3.2.2. Ending Multi-Block Transfers

All multi-block transfers must end as shown in either Row 1 or Row 5 of the table above. At the end of every block transfer, the DMA Controller samples the row number, and if the DMA Controller is in the Row 1 or Row 5 state, then the previous block transferred was the last block and the DMA transfer is terminated.

Row 1 and Row 5 are used for single-block transfers or terminating multi-block transfers. Transfers initiated in rows 2, 3 or 4 can only end in row 1; similarly, transfers initiated in rows 6 through 10 can only end in row 5. Ending in the Row 5 state enables status fetch and write-back for the last block. Ending in the Row 1 state disables status fetch and write-back for the last block.

For rows 2, 3, and 4 of the table above, (`LLPx.LOC = 0` and `CFGx.RELOAD_SRC` and/or `CFGx.RELOAD_DST` is set), multi-block DMA transfers continue until both the `CFGx.RELOAD_SRC` and `CFGx.RELOAD_DST` registers are cleared by software. They should be programmed to 0 in the end-of-block interrupt service routine that services the next-to-last block transfer; this puts the DMA Controller into the Row 1 state.

For rows 6, 8, and 10 of the table above (both `CFGx.RELOAD_SRC` and `CFGx.RELOAD_DST` cleared), the user must set up the last block descriptor in memory so that both LLI.`CTLx.LLP_SRC_EN` and LLI.`CTLx.LLP_DST_EN` are 0.

The sampling of the `LLPx.LOC` bit takes place exclusively at the beginning of the transfer when the channel is enabled. This determines whether writeback is enabled throughout the complete transfer, and changing the value of this bit in subsequent blocks on the same transfer does not have any effect.

The only allowed transitions between the rows of the table above are from any row into Row 1 or Row 5. As already stated, a transition into row 1 or row 5 is used to terminate the DMA transfer; all other transitions between rows are not allowed. Software must ensure that illegal transitions between rows do not occur between blocks of a multi-block transfer. For example, if block N is in row 10, then the only allowed rows for block N +1 are rows are rows 10 or 5.

8.3.3. Programming the Linked List Multi-Block Transfer

The following diagram shows an overview of programming the DMA Controller for linked list multi-block transfer.

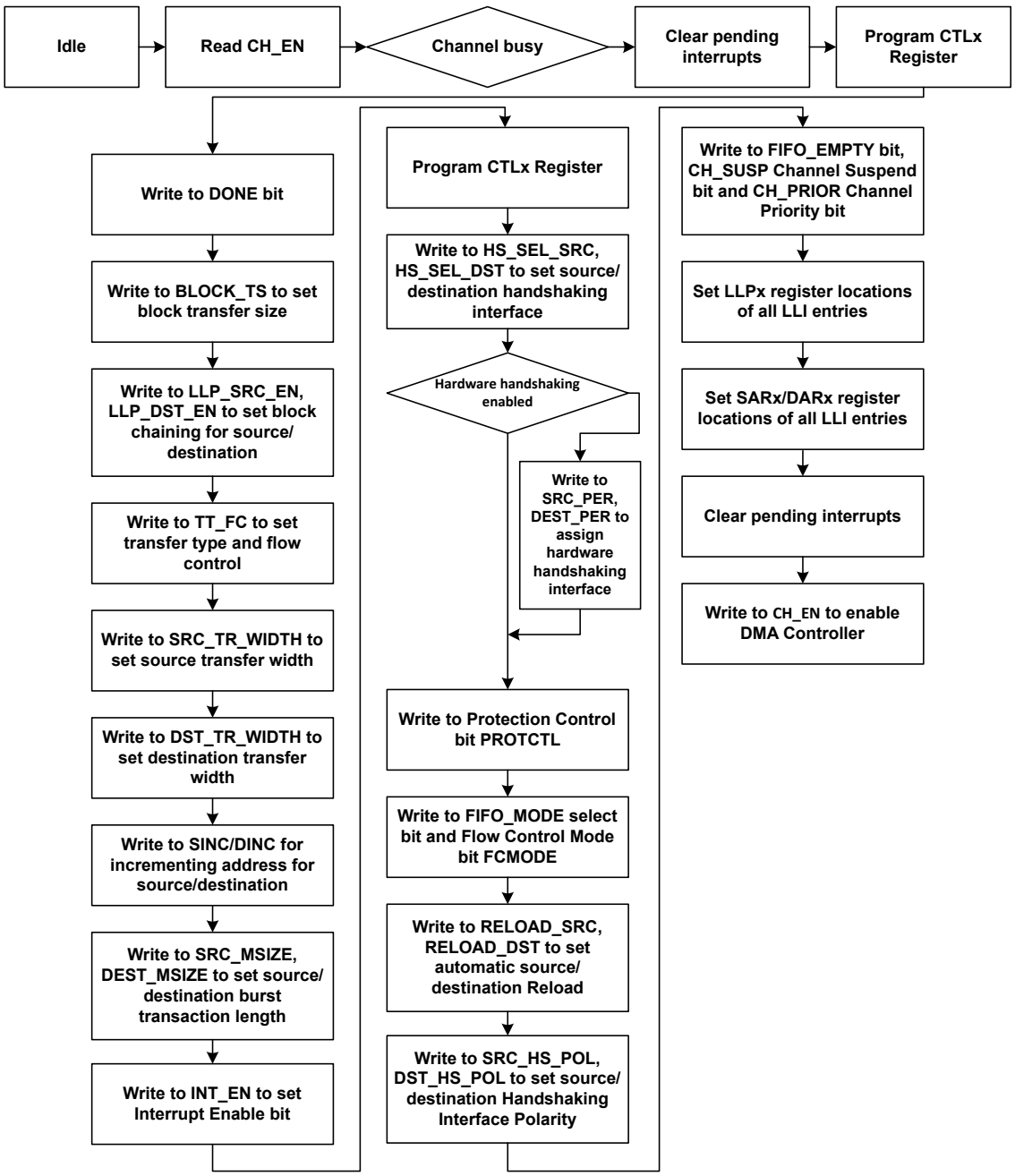

Figure 8-6 Flowchart for DMA programming example

8.3.4. Programming a Channel

Three registers – `LLPx`, `CTLx`, and `CFGx` – need to be programmed to determine whether single- or multi-block transfers occur, and which type of multi-block transfer is used. The different transfer types are shown in the table above.

The “Update Method” columns indicate where the values of `SARx`, `DARx`, `CTLx`, and `LLPx` are obtained for the next block transfer when multi-block DMA Controller transfers are enabled.

In the table above, all other combinations of `LLPx.LOC = 0`, `CTLx.LLP_SRC_EN`, `CFGx.RELOAD_SRC`, `CTLx.LLP_DST_EN`, and `CFGx.RELOAD_DST` are illegal, and will cause indeterminate or erroneous behaviour.

8.3.4.1. Programming Examples

8.3.4.1.1. Single-Block Transfer

To perform a single-block transfer, follow the sequence of steps below, using values from the Transfer types table, corresponding to the single-block transfer mode.

Table 8-46 Single-block transfer parameters

Transfer Type	LLPx[31:2] = 0	CTLx [28]	CFGx [30]	CTLx [27]	CFGx [31]	CTLx, LLPx Update Method	SARx Update Method	DARx Update Method	Write Back
1. Single-block or last transfer of multi-block.	Yes	0	0	0	0	None, user reprograms	None (single)	None (single)	No

1. Read the Channel Enable register to choose a free (disabled) channel; refer to CH_EN.
2. Clear any pending interrupts on the channel from the previous DMA transfer by writing to the Interrupt Clear registers: CLR_TFR, CLR_BLOCK, CLR_SRC_TRAN, CLR_DST_TRAN, and CLR_ERR. Reading the Interrupt Raw Status and Interrupt Status registers confirms that all interrupts have been cleared.
3. Program the following channel registers:
    a. Write the starting source address in the SARx register for channel x.
    b. Write the starting destination address in the DARx register for channel.
    c. Program CTLx and CFGx according to the table above. Program the LLPx register with 0.
    d. Write the control information for the DMA transfer in the CTLx register for channel x. For example, in the register, you can program the following:
        i. Set up the transfer type (memory or non-memory peripheral for source and destination) and flow control device by programming the TT_FC of the CTLx register. (See the decoding for this field.)
        ii. Set up the transfer characteristics, such as:
            - Transfer width for the source in the SRC_TR_WIDTH field. (See the decoding for this field.)
            - Transfer width for the destination in the DST_TR_WIDTH field. (See the decoding for this field.)
            - Incrementing/decrementing or fixed address for the source in the SINC field.
            - Incrementing/decrementing or fixed address for the destination in the DINC field.
    e. Write the channel configuration information into the CFGx register for channel x.
        i. Designate the handshaking interface type (hardware or software) for the source and destination peripherals; this is not required for memory.

        This step requires programming the HS_SEL_SRC / HS_SEL_DST bits, respectively. Writing a 0 activates the hardware handshaking interface to handle source/

destination requests. Writing a 1 activates the software handshaking interface to handle source and destination requests.

ii. If the hardware handshaking interface is activated for the source or destination peripheral, assign a handshaking interface to the source and destination peripheral; this requires programming the `SRC_PER` and `DEST_PER` bits, respectively.

4. Ensure that bit 0 of the `CFG` register is enabled before writing to `CH_EN`.
5. Source and destination request single and burst DMA transactions in order to transfer the block of data (assuming non-memory peripherals). The DMA Controller acknowledges at the completion of every transaction (burst and single) in the block and carries out the block transfer.
6. Once the transfer completes, hardware sets the interrupts and disables the channel. At this time, you can respond to either the Block Complete or Transfer Complete interrupts, or poll for the transfer complete raw interrupt status register (`RAW_TFR[n]`, n = channel number) until it is set by hardware, in order to detect when the transfer is complete. Note that if this polling is used, the software must ensure that the transfer complete interrupt is cleared by writing to the Interrupt Clear register, `CLR_TFR[n]`, before the channel is enabled.

8.3.4.1.2. Multi-Block Transfer with Linked List for Source and Linked List for Destination

To perform multi-block transfer with Linked Lists for source and destination follow the steps below using corresponding values from the transfer mode table.

Table 8-47 Multi-block transfer with linked lists for source and destination

Transfer Type	LLPx[31:2] = 0	CTLx [28]	CFGx [30]	CTLx [27]	CFGx [31]	CTLx, LLPx Update Method	SARx Update Method	DARx Update Method	Write Back
Linked list multi-block transfer	No	1	0	1	0	CTLx, LLPx loaded from next Linked List item.	Linked List	Linked List	Yes

1. Read the Channel Enable register (see `CH_EN`) to choose a free (disabled) channel.
2. Set up the chain of Linked List Items (otherwise known as block descriptors) in memory. Write the control information in the LLI.`CTLx` register location of the block descriptor for each LLI in memory for channel x. For example, in the register, you can program the following:
   a. Set up the transfer type (memory or non-memory peripheral for source and destination) and flow control device by programming the `TT_FC` of the `CTLx` register. See the decoding for this field.
   b. Set up the transfer characteristics, such as:
      i. Transfer width for the source in the `SRC_TR_WIDTH` field. (See the decoding for this field.)
      ii. Transfer width for the destination in the `DST_TR_WIDTH` field. (See the decoding for this field.)
      iii. Incrementing/decrementing or fixed address for the source in the `SINC` field.
      iv. Incrementing/decrementing or fixed address for the destination in the `DINC` field.
3. Write the channel configuration information into the `CFGx` register for channel x.

a. Designate the handshaking interface type (hardware or software) for the source and destination peripherals; this is not required for memory.

   This step requires programming the HS_SEL_SRC / HS_SEL_DST bits, respectively. Writing a 0 activates the hardware handshaking interface to handle source/destination requests for the specific channel. Writing a 1 activates the software handshaking interface to handle source/destination requests.

b. If the hardware handshaking interface is activated for the source or destination peripheral, assign the handshaking interface to the source and destination peripheral. This requires programming the SRC_PER and DEST_PER bits, respectively.

4. Make sure that the LLI.CTLx register locations of all LLI entries in memory (except the last) are set as shown in the table above. The LLI.CTLx register of the last Linked List Item must be set according to the Single-block Transfer.
5. Make sure that the LLI.LLPx register locations of all LLI entries in memory (except the last) are non-zero and point to the base address of the next Linked List Item.
6. Make sure that the LLI.SARx/LLI.DARx register locations of all LLI entries in memory point to the start source/destination block address preceding that LLI fetch.
7. Ensure that the LLI.CTLx.DONE field of the LLI.CTLx register locations of all LLI entries in memory is cleared.
8. Clear any pending interrupts on the channel from the previous DMA transfer by writing to the Interrupt Clear registers: CLR_TFR, CLR_BLOCK, CLR_SRC_TRAN, CLR_DST_TRAN, and CLR_ERR. Reading the Interrupt Raw Status and Interrupt Status registers confirms that all interrupts have been cleared.
9. Program the CTLx and CFGx registers according to the table above.
10. Program the LLPx register with LLP(0), the pointer to the first linked list item.
11. Finally, enable the channel by writing a 1 to the CH_EN.CH_EN bit; the transfer is performed.
12. The DMA Controller fetches the first LLI from the location pointed to by LLPx(0).

The LLI.SARx, LLI.DARx, LLI.LLPx, and LLI.CTLx registers are fetched. The DMA Controller automatically reprograms the SARx, DARx, LLPx, and CTLx channel registers from the LLPx(0).

13. Source and destination request single and burst DMA transactions to transfer the block of data (assuming non-memory peripheral). The DMA Controller acknowledges at the completion of every transaction (burst and single) in the block and carries out the block transfer.
14. The CTLx[63:32] register is written out to system memory. For conditions under which the CTLx[63:32] register is written out to system memory, refer to the Write Back column of the table above. Only the second word of the CTLx register is written out – CTLx[63:32] – because only the CTLx.BLOCK_TS and CTLx.DONE fields have been updated by the DMA Controller hardware. Additionally, the CTLx.DONE bit is asserted to indicate block completion. Therefore, software can poll the LLI.CTLx.DONE bit of the CTLx register in the LLI to ascertain when a block transfer has completed.

Do not poll the CTLx.DONE bit in the DMA Controller memory map; instead, poll the LLI.CTLx.DONE bit in the LLI for that block. If the polled LLI.CTLx.DONE bit is asserted, then this block transfer has completed. This LLI.CTLx.DONE bit was cleared at the start of the transfer (Step 7).

15. The DMA Controller does not wait for the block interrupt to be cleared, but continues fetching the next LLI from the memory location pointed to by the current LLPx register and automatically reprograms the SARx, DARx, LLPx, and CTLx channel registers. The DMA transfer continues until

the DMA Controller determines that the CTLx and LLPx registers at the end of a block transfer match the ones described in Single Block Transfer (as discussed earlier). The DMA Controller then knows that the previously transferred block was the last block in the DMA transfer.

If the user needs to execute a DMA transfer where the source and destination address are contiguous, but where the amount of data to be transferred is greater than the maximum block size CTLx.BLOCK_TS, then this can be achieved using the type of multi-block transfer.

8.3.5. Disabling a Channel Prior to Transfer Completion

Under normal operation, software enables a channel by writing a 1 to the channel enable register, CH_EN.CH_EN, and hardware disables a channel on transfer completion by clearing the CH_EN.CH_EN register bit.

The recommended way for software to disable a channel without losing data is to use the CH_SUSP bit in conjunction with the FIFO_EMPTY bit in the Channel Configuration Register (CFGx).

1. If software wishes to disable a channel prior to the DMA transfer completion, then it can set the CFGx.CH_SUSP bit to tell the DMA Controller to halt all transfers from the source peripheral. Therefore, the channel FIFO receives no new data.
2. Software can now poll the CFGx.FIFO_EMPTY bit until it indicates that the channel FIFO is empty.
3. The CH_EN.CH_EN bit can then be cleared by software once the channel FIFO is empty. When CTLx.SRC_TR_WIDTH < CTLx.DST_TR_WIDTH and the CFGx.CH_SUSP bit is high, the CFGx.FIFO_EMPTY is asserted once the contents of the FIFO do not permit a single word of CTLx.DST_TR_WIDTH to be formed. However, there may still be data in the channel FIFO, but not enough to form a single transfer of CTLx.DST_TR_WIDTH. In this scenario, once the channel is disabled, the remaining data in the channel FIFO is not transferred to the destination peripheral.

It is permissible to remove the channel from the suspension state by writing a 0 to the CFGx.CH_SUSP register. The DMA transfer completes in the normal manner.

If a channel is disabled by software, an active single or burst transaction is not guaranteed to receive an acknowledgement.

8.3.5.1. Abnormal Transfer Termination

A DMA transfer may be terminated abruptly by software by clearing the channel enable bit, CH_EN.CH_EN. You must not assume that the channel is disabled immediately after the CH_EN. The CH_EN bit is cleared over the Slave Interface. Consider this as a request to disable the channel. You must poll CH_EN.CH_EN and confirm that the channel is disabled by reading back 0. A case where the channel is not disabled after a channel disable request is where either the source or destination has received a split or retry response. The DMA Controller must keep re-attempting the transfer to the system HADDR that originally received the split or retry response until an OKAY response is returned.

Software may terminate all channels abruptly by clearing the global enable bit in the DMA Controller Configuration Register (CFG[0]). Again, you must not assume that all channels are disabled immediately after the CFG[0] is cleared over the Slave Interface. Consider this as a request to disable all channels. You must poll CH_EN and confirm that all channels are disabled by reading back 0.

If the channel enable bit is cleared while there is data in the channel FIFO, this data is not sent to the destination peripheral and is not present when the channel is re-enabled. For read-sensitive source peripherals, such as a source FIFO, this data is therefore lost. When the source is not a read-sensitive

device (such as memory), disabling a channel without waiting for the channel FIFO to empty may be acceptable, since the data is available from the source peripheral upon request and is not lost.

If a channel is disabled by software, an active single or burst transaction is not guaranteed to receive an acknowledgement.

8.3.6. Defined-length Burst Support on DMA Controller

By default, the DMA Controller support incremental (INCR) bursts only. To achieve better performance, defined length bursts, such as INCR4, INCR8 and INCR16 are required.

9. eFLASH Controller

The eFLASH Controller is the controller of the eFLASH memory. A simplified block diagram of the controller is shown in the following figure.

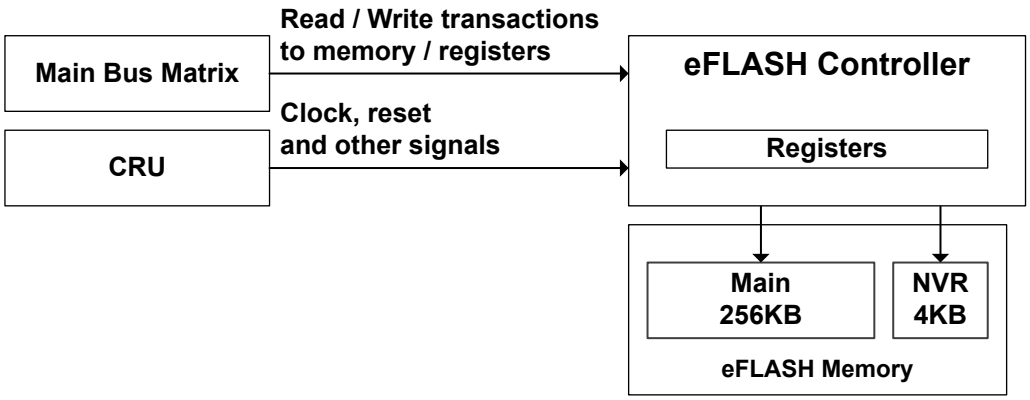

Figure 9-1 eFLASH Controller block diagram

As shown in the figure above, the eFLASH Memory consists of the Main array (256 KB) and NVR array (4 KB).

The eFLASH Memory can be acessed in the following ways:

- Registers access: access to the eFLASH Memory occurs through the eFLASH Controller registers
- Direct access: acces to the eFLASH Memory through the Core 0/1 Address Space

The table below shows the CRU registers that affect the operation of the eFLASH Controller.

Table 9-1 CRU registers that affect the operation of the eFLASH Controller

Description	Controlled via
NVR enable flag. Enables the access to the eFLASH memory NVR array 0: disabled (operation is performed in the Main array) 1: enabled (operation is performed in the NVR array)	`SYSCR0.NVR`
NVR disable flag. Disables the access to the eFLASH memory NVR array. 0: enabled (means that the controller can perform an operation in the NVR array) 1: disabled (means that it is forbidden for the controller to perform an operation in the NVR array, any request for an operation in the NVR array will be interpreted as a request for an operation in the Main array)	`SYSCR0.FLASHNVRDIS`
NVR 1 lock flag. Disables the modification of the 1st block of the NVR array. 0: modification of the 1st NVR block is enabled 1: modification of the 1st NVR block is disabled If the NVR 1 lock flag is set to 1 and the CHIP_ERASE operation is performed in the NVR array, then only the 2nd block of the NVR array will be erased (it means that the BLOCK_ERASE operation will be performed instead of the CHIP_ERASE operation)	`FLASHNVRCR.NVR1DIS`

Table 9-1 CRU registers that affect the operation of the eFLASH Controller (continued)

Description	Controlled via
Direct access lock flag. Disables the modification of the eFLASH memory using the direct access. 0: means that the controller can modify (program) the eFLASH memory using the direct access 1: means that the controller can't modify (program) the eFLASH memory using the direct access	`SYSCR0.DAMODDIS`

Pay attention that NVR 1 is forbidden to modify because it contains the information (indentification, configuration and reference information) which is recorded during the eFLASH memory factory tasting. NVR 1 modification will lead to the unpredictable behavior of the eFLASH memory.

9.1. eFLASH Controller Registers

The register region of the eFLASH Controller is mapped in the BE-U1000 microcontroller memory map as the following table shows.

Table 9-2 Memory address region for the eFLASH Controller registers

Region	Block Name	Size	Start Address	End Address
System Resources	eFLASH Controller	4 KB	0xA100_0000	0xA100_0FFF

For details, refer to Memory Map chapter.

This chapter covers the following topics:

- Register Map
- Register Descriptions

9.1.1. Register Map

The following table provides a top-level summary of each register. To view the detailed description of a register, click the register name in the **Description** column.

Table 9-3 Memory map of the eFLASH Controller registers

Register	Offset	Description	Default Value
`EFLASH_CR`	0x0	eFLASH Controller Control Register	0x0
`EFLASH_ADDR`	0x4	eFLASH Controller Address Register	0x0
`EFLASH_WDATA`	0x8	eFLASH Controller Write Data Register	0x0
`EFLASH_RDATA`	0xC	eFLASH Controller Read Data Register	0x0
`EFLASH_SR`	0x10	eFLASH Controller Status Register	0x1
`EFLASH_TIME0`	0x14	eFLASH Controller Time Intervals Register 0	0x1061A801*

Table 9-3 Memory map of the eFLASH Controller registers (continued)

Register	Offset	Description	Default Value
EFLASH_TIME1	0x18	eFLASH Controller Time Intervals Register 1	0x9C407D1*
EFLASH_TIME2	0x1C	eFLASH Controller Time Intervals Register 2	0xC80C805*
EFLASH_TIME3	0x20	eFLASH Controller Time Intervals Register 3	0x7D0107D*
EFLASH_TIME4	0x24	eFLASH Controller Time Intervals Register 4	0xC30D401*
EFLASH_TIME5	0x28	eFLASH Controller Time Intervals Register 5	0x2827100*
EFLASH_TIME6	0x2C	eFLASH Controller Time Intervals Register 6	0xFA*
	0x38 - 0x40	Reserved. Do not modify.	

*The default value for the EFLASH_TIMEx registers may differ from those listed in the table above due to their changes by the Boot ROM.

9.1.2. Register Descriptions

In the tables below, the **R/W** column of a register description specifies how the application and the core can access the register fields.

9.1.2.1. eFLASH Controller Control Register (EFLASH_CR)

The EFLASH_CR register is at offset 0x0.

The following table shows the register bit assignments.

Table 9-4 Fields for register: EFLASH_CR

Bits	Name	R/W	Description
[31:6]			Reserved (return 0 when read)
[5]			Reserved for internal use
[4]	NVR	R/W	The operation specified in the OP_CODE field will be performed in the NVR array if this bit is set to 0x1. **Value After Reset:** 0x0.
[3:1]	OP_CODE	R/W	Operation Code. **Values:** **0x0:** (CFG) Memory confguration **0x1:** (WRITE) Writes a 32-bit word to the eFLASH memory **0x2:** (READ) Reads a 32-bit word from the eFLASH memory **0x3:** (CHIP_ERASE) Chip erase **0x4:** (BLOCK_ERASE) Block erase **0x5:** (SECTOR_ERASE) Sector erase **0x6:** (ENTER_DPD) Enters the *Deep Power Down* (**DPD**) Mode **0x7:** (EXIT_DPD) Exit DPD Mode

Table 9-4 Fields for register: `EFLASH_CR` (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0.
[0]	`RUN`	R/W	Write 0x1 to this bit to execute the command from the `OP_CODE` field. This bit is automatically set to 0x0 after the command execution begins. **Value After Reset:** 0x0.

9.1.2.2. eFLASH Controller Address Register (`EFLASH_ADDR`)

The `EFLASH_ADDR` register is at offset 0x4.

The following table shows the register bit assignments.

Table 9-5 Fields for register: `EFLASH_ADDR`

Bits	Name	R/W	Description
[31:16]			Reserved (return 0 when read)
[15:0]	`ADDR`	R/W	This bit filed contains the address where the WRITE, READ, BLOCK_ERASE and SECTOR_ERASE commands will be executed. After executing the command, the address will be automatically incremented by 1. When performing an operation in the NVR array, the address is specified by the 10 least significant bits of this bit field. **Value After Reset:** 0x0

9.1.2.3. eFLASH Controller Write Data Register (`EFLASH_WDATA`)

The `EFLASH_WDATA` register is at offset 0x8.

The following table shows the register bit assignments.

Table 9-6 Fields for register: `EFLASH_WDATA`

Bits	Name	R/W	Description
[31:0]	`DATA`	R/W	This bit field contains data that will be written to memory when executing the WRITE command. **Value After Reset:** 0x0

9.1.2.4. eFLASH Controller Read Data Register (`EFLASH_RDATA`)

The `EFLASH_RDATA` register is at offset 0xC.

The following table shows the register bit assignments.

Table 9-7 Fields for register: `EFLASH_RDATA`

Bits	Name	R/W	Description
[31:0]	`DATA`	R	This bit field contains data that will be read from memory when executing the READ command. **Value After Reset:** 0x0.

9.1.2.5. eFLASH Controller Status Register (EFLASH_SR)

The EFLASH_SR register is at offset 0x10.

The following table shows the register bit assignments.

Table 9-8 Fields for register: EFLASH_SR

Bits	Name	R/W	Description
[31:1]			Reserved (return 0 when read)
[0]	BUSY	R	This flag is set if the operation is not completed. **Value After Reset:** 0x1

9.1.2.6. eFLASH Controller Time Intervals Register 0 (EFLASH_TIME0)

The EFLASH_TIME0 register is at offset 0x14.

Please note that the **Values After Reset**, mentioned for the bit fields in the table below, may differ from the actual values due to changes made by the Boot ROM.

For more information about setting the time intervals, refer to the Time Intervals Configuration section. Default values of the EFLASH_TIME0 bit fields correspond to the coefficients for the 50 MHz frequency.

The following table shows the register bit assignments.

Table 9-9 Fields for register: EFLASH_TIME0

Bits	Name	R/W	Description
[31:27]	AA_T	R/W	Configures the coefficient for the t_{AA} and t_{RC}. The default value of this bit field corresponds to the 25 ns time interval at the 50 MHz frequency. **Value After Reset:** 0x2
[26:4]	SCE_T	R/W	Configures the coefficient for the t_{SCE}. The default value of this bit field corresponds to the 8ms time interval at the 50 MHz frequency. **Value After Reset:** 0x61A80
[3:0]	DS_T	R/W	Configures the coefficient for the t_{DS}, t_{APS}, t_{APH}, t_{DH}, t_{PGH}, t_{PREPGH}, t_{CFS}, t_{CFH}, t_{CONFEN}. The default value of this bit field corresponds to the 15 ns time interval at the 50 MHz frequency. **Value After Reset:** 0x1

9.1.2.7. eFLASH Controller Time Intervals Register 1 (EFLASH_TIME1)

The EFLASH_TIME1 register is at offset 0x18.

Please note that the **Values After Reset**, mentioned for the bit fields in the table below, may differ from the actual values due to changes made by the Boot ROM.

For more information about setting the time intervals, refer to the Time Intervals Configuration section. Default values of the `EFLASH_TIME1` bit fields correspond to the coefficients for the 50 MHz frequency.

The following table shows the register bit assignments.

Table 9-10 Fields for register: `EFLASH_TIME1`

Bits	Name	R/W	Description
[31:16]	`RCV_T`	R/W	Configures the coefficient for the t_{RCV} (ERASE). The default value of this bit field corresponds to the 50 us time interval at the 50 MHz frequency. **Value After Reset:** 0x9C4
[15:2]	`RHR_T`	R/W	Configures the coefficient for the t_{RHR}, t_{NVS} (CHIP ERASE), t_{DPDH}, t_{CFL}. The default value of this bit field corresponds to the 10 us time interval at the 50 MHz frequency. **Value After Reset:** 0x1F4
[1:0]	`AS_T`	R/W	Configures the coefficient for the t_{AS}, t_{CS}, t_{AH}. The default value of this bit field corresponds to the 2 ns time interval at the 50 MHz frequency. **Value After Reset:** 0x1

9.1.2.8. eFLASH Controller Time Intervals Register 2 (`EFLASH_TIME2`)

The `EFLASH_TIME2` register is at offset 0x1C.

Please note that the **Values After Reset**, mentioned for the bit fields in the table below, may differ from the actual values due to changes made by the Boot ROM.

For more information about setting the time intervals, refer to the Time Intervals Configuration section. Default values of the `EFLASH_TIME2` bit fields correspond to the coefficients for the 50 MHz frequency.

The following table shows the register bit assignments.

Table 9-11 Fields for register: `EFLASH_TIME2`

Bits	Name	R/W	Description
[31:20]	`NVS_T`	R/W	Configures the coefficient for the t_{NVS} (PROGRAM, SECTOR ERASE, BLOCK ERASE). The default value of this bit field corresponds to the 4 us time interval at the 50 MHz frequency. **Value After Reset:** 0xC8
[19:7]	`PROG_T`	R/W	Configures the coefficient for the t_{PROG}.

Table 9-11 Fields for register: `EFLASH_TIME2` (continued)

Bits	Name	R/W	Description
			The default value of this bit field corresponds to the 8 us time interval at the 50 MHz frequency. **Value After Reset:** 0x190
[6:0]	`RW_T`	R/W	Configures the coefficient for the t_{RW}. The default value of this bit field corresponds to the 100 ns time interval at the 50 MHz frequency. **Value After Reset:** 0x5

9.1.2.9. eFLASH Controller Time Intervals Register 3 (`EFLASH_TIME3`)

The `EFLASH_TIME3` register is at offset 0x20.

Please note that the **Values After Reset**, mentioned for the bit fields in the table below, may differ from the actual values due to changes made by the Boot ROM.

For more information about setting the time intervals, refer to the Time Intervals Configuration section. Default values of the `EFLASH_TIME3` bit fields correspond to the coefficients for the 50 MHz frequency.

The following table shows the register bit assignments.

Table 9-12 Fields for register: `EFLASH_TIME3`

Bits	Name	R/W	Description
[31:17]	`PGS_T`	R/W	Configures the coefficient for the t_{PGS}. The default value of this bit field corresponds to the 20 us time interval at the 50 MHz frequency. **Value After Reset:** 0x3E8
[16:12]	`CBD_T`	R/W	Configures the coefficient for the t_{CBD}. The default value of this bit field corresponds to the 20 ns time interval at the 50 MHz frequency. **Value After Reset:** 0x1
[11:0]	`PREPROG_T`	R/W	Configures the coefficient for the $t_{PREPROG}$. The default value of this bit field corresponds to the 2.5 us time interval at the 50 MHz frequency. **Value After Reset:** 0x7D

9.1.2.10. eFLASH Controller Time Intervals Register 4 (`EFLASH_TIME4`)

The `EFLASH_TIME4` register is at offset 0x24.

Please note that the **Values After Reset**, mentioned for the bit fields in the table below, may differ from the actual values due to changes made by the Boot ROM.

For more information about setting the time intervals, refer to the Time Intervals Configuration section. Default values of the EFLASH_TIME4 bit fields correspond to the coefficients for the 50 MHz frequency.

The following table shows the register bit assignments.

Table 9-13 Fields for register: EFLASH_TIME4

Bits	Name	R/W	Description
[31:26]	TEST_T	R/W	Configures the coefficient for the t_{TEST}. The default value of this bit field corresponds to the 50 ns time interval at the 50 MHz frequency. **Value After Reset:** 0x3
[25:4]	SBE_T	R/W	Configures the coefficient for the t_{SBE}, t_{HV_pg}, t_{HV_prepg}. The default value of this bit field corresponds to the 4 ms time interval at the 50 MHz frequency. **Value After Reset:** 0x30D40
[3:0]	RS_T	R/W	Configures the coefficient for the t_{RS} (RECALL), t_{RR} (RECALL). The default value of this bit field corresponds to the 10 ns time interval at the 50 MHz frequency. **Value After Reset:** 0x1

9.1.2.11. eFLASH Controller Time Intervals Register 5 (EFLASH_TIME5)

The EFLASH_TIME5 register is at offset 0x28.

Please note that the **Values After Reset**, mentioned for the bit fields in the table below, may differ from the actual values due to changes made by the Boot ROM.

For more information about setting the time intervals, refer to the Time Intervals Configuration section. Default values of the EFLASH_TIME5 bit fields correspond to the coefficients for the 50 MHz frequency.

The following table shows the register bit assignments.

Table 9-14 Fields for register: EFLASH_TIME5

Bits	Name	R/W	Description
[31:30]			Reserved (return 0 when read)
[29:22]	RCR_T	R/W	Configures the coefficient for the t_{RCR}, t_{AAR}. The default value of this bit field corresponds to the 200 ns time interval at the 50 MHz frequency. **Value After Reset:** 0xA
[21:0]	ERASE_T	R/W	Configures the coefficient for the t_{ERASE}.

Table 9-14 Fields for register: `EFLASH_TIME5` (continued)

Bits	Name	R/W	Description
			The default value of this bit field corresponds to the 3.2 ms time interval at the 50 MHz frequency. **Value After Reset:** 0x27100

9.1.2.12. eFLASH Controller Time Intervals Register 6 (`EFLASH_TIME6`)

The `EFLASH_TIME6` register is at offset 0x2C.

Please note that the **Values After Reset**, mentioned for the bit fields in the table below, may differ from the actual values due to changes made by the Boot ROM.

For more information about setting the time intervals, refer to the Time Intervals Configuration section. Default values of the `EFLASH_TIME6` bit fields correspond to the coefficients for the 50 MHz frequency.

The following table shows the register bit assignments.

Table 9-15 Fields for register: `EFLASH_TIME6`

Bits	Name	R/W	Description
[31:13]			Reserved (return 0 when read)
[12:0]	`PREPGS_T`	R/W	Configures the coefficient for the t_{PREPGS}, t_{RCV} (PROGRAM). The default value of this bit field corresponds to the 5 us time interval at the 50 MHz frequency. **Value After Reset:** 0xFA

9.2. Programming the eFLASH Controller

This section covers the following topics:

- Time Intervals Configuration
- Memory Configuration Operation (CFG)
- Memory Write Operation (WRITE)
- Memory Read Operation (READ)
- Chip Erase Operation (CHIP_ERASE)
- Block Erase Operation (BLOCK_ERASE)
- Sector Erase Operation (SECTOR_ERASE)
- Entering the DPD Mode Operation (ENTER_DPD)
- Exiting the DPD Mode Operation (EXIT_DPD)
- Direst Access to the eFLASH Memory

9.2.1. Time Intervals Configuration

Before starting to work with the eFLASH Controller and eFLASH Memory, you can also configure the time intervals. The time intervals are adjusted using coefficients that are written to the bits of the

EFLASH_TIMEx registers, where x is the register number. Note that the default values of these registers correspond to the coefficients for the 50 MHz frequency.

The time intervals coefficients are calculated as follows:

$$X = \left[\frac{t_x}{T_f} \right]$$

Where:

- X – time interval coefficient
- t_x – time interval
- T_f – frequency period

For example, to set the coefficient for the t_{DS} time interval (t_{DS}=15 ns) at the 100 MHz frequency (T_f=10 ns) you should write 0x2 to the EFLASH_TIME0.DS_T (calculated as follows: [t_{DS}/T_f]=[15/10]=[1.5]=2).

Coefficients of the time intervals should be adjusted so that the time intervals correspond to the values listed in the table below.

Table 9-16 Time intervals values

Time interval	Value
t_{AA}, t_{RC}	25 ns
t_{SCE}	8 ms
t_{DS}, t_{APS}, t_{APH}, t_{DH}, t_{PGH}, t_{PREPGH}, t_{CFS}, t_{CFH}, t_{CONFEN}	15 ns
t_{RCV} (ERASE)	50 µs
t_{RHR}, t_{NVS} (CHIP ERASE), t_{DPDH}, t_{CFL}	10 µs
t_{AS}, t_{CS}, t_{AH}	2 ns
t_{NVS} (PROGRAM, SECTOR ERASE, BLOCK ERASE)	4 µs
t_{PROG}	8 µs
t_{RW}	100 ns
t_{PGS}	20 µs
t_{CBD}	20 ns
$t_{PREPROG}$	2.5 µs
t_{TEST}	50 ns
t_{SBE}, t_{HV_pg}, t_{HV_prepg}	4 ms
t_{RS} (RECALL), t_{RR} (RECALL)	10 ns
t_{RCR}, t_{AAR}	200 ns
t_{ERASE}	3.2 ms
t_{PREPGS}, t_{RCV} (PROGRAM)	5 µs

9.2.2. Memory Configuration Operation (CFG)

Execute the following steps to perform the eFLASH memory configuration process:

1. Write 0x1 to the RUN bit of the EFLASH_CR register
2. Wait until the EFLASH_SR.BUSY will be set to 0x0

9.2.3. Memory Write Operation (WRITE)

The Memory Write Operation writes one 32-bit word in the eFLASH memory per command. Data for WRITE operation is taken from the EFLASH_WDATA register and recorded at the address that specified in the ADDR bit field of the EFLASH_ADDR register.

Execute the following steps to perform the eFLASH memory WRITE operation:

1. Write address to the ADDR bit field of the EFLASH_ADDR register
2. Write data to the EFLASH_WDATA register
3. Set EFLASH_CR.OP_CODE = 0x1 and EFLASH_CR.RUN = 0x1 (you should also set the EFLASH_CR.NVR = 0x1 if you need to write to the NVR array)
4. Wait until the EFLASH_SR.BUSY will be set to 0x0
5. To perform one more write:

   If the next address for write = EFLASH_ADDR.ADDR + 1 – repeat steps **2**-**5**, else repeat steps **1**-**5**.

9.2.4. Memory Read Operation (READ)

The Memory Read Operation reads one 32-bit word from the eFLASH memory per command. The address for the READ operation is specified in the ADDR bit field of the EFLASH_ADDR register and the read data is written to the EFLASH_RDATA register.

Execute the following steps to perform the eFLASH memory READ operation:

1. Write address to the ADDR bit field of the EFLASH_ADDR register
2. Set EFLASH_CR.OP_CODE = 0x2 and EFLASH_CR.RUN = 0x1 (you should also set the EFLASH_CR.NVR = 0x1 if you need to read from the NVR array)
3. Wait until the EFLASH_SR.BUSY will be set to 0x0
4. Read data from the EFLASH_RDATA register
5. To perform one more read:

   If the next address for read = EFLASH_ADDR.ADDR + 1 – repeat steps **2**-**5**, else repeat steps **1**-**5**.

9.2.5. Chip Erase Operation (CHIP_ERASE)

Perform the following steps to execute the CHIP_ERASE operation:

1. Set EFLASH_CR.OP_CODE = 0x3 and EFLASH_CR.RUN = 0x1 (you should also set the EFLASH_CR.NVR = 0x1 if you need to erase the NVR array)
2. Wait until the EFLASH_SR.BUSY will be set to 0x0

9.2.6. Block Erase Operation (BLOCK_ERASE)

- Main array contains 32 blocks. Main array block address is specified by the 5 most significant bits of the address (EFLASH_ADDR.ADDR[15:11]).
- NVR array contains 2 blocks. NVR array block address is specified by the 9th address bit (EFLASH_ADDR.ADDR[9])

Perform the following steps to execute the BLOCK_ERASE operation:

1. Write block address to the ADDR bit field of the EFLASH_ADDR register
2. Set EFLASH_CR.OP_CODE = 0x4 and EFLASH_CR.RUN = 0x1 (you should also set the EFLASH_CR.NVR = 0x1 if you need to erase the NVR block)
3. Wait until the EFLASH_SR.BUSY will be set to 0x0

9.2.7. Sector Erase Operation (SECTOR_ERASE)

- Main array contains 32 blocks, each of which has 8 sectors. Main array sector address is specified by the 8 most significant bits of the address (EFLASH_ADDR.ADDR[15:8]).
- NVR array contains 2 blocks, each of which has 2 sectors. NVR array sector address is specified by the 9th and 8th address bits (EFLASH_ADDR.ADDR[9:8])

Perform the following steps to execute the SECTOR_ERASE operation:

1. Write sector address to the ADDR bit field of the EFLASH_ADDR register
2. Set EFLASH_CR.OP_CODE = 0x5 and EFLASH_CR.RUN = 0x1 (you should also set the EFLASH_CR.NVR = 0x1 if you need to erase the NVR sector)
3. Wait until the EFLASH_SR.BUSY will be set to 0x0

9.2.8. Entering the DPD Mode Operation (ENTER_DPD)

Perform the following steps to put the eFLASH memory into the *Deep Power Down* (**DPD**) mode:

1. Set EFLASH_CR.OP_CODE = 0x6 and EFLASH_CR.RUN = 0x1
2. Wait until the EFLASH_SR.BUSY will be set to 0x0

9.2.9. Exiting the DPD Mode Operation (EXIT_DPD)

Perform the following steps to bring the eFLASH memory out of the *Deep Power Down* (**DPD**) mode:

1. Set EFLASH_CR.OP_CODE = 0x7 and EFLASH_CR.RUN = 0x1
2. Wait until the EFLASH_SR.BUSY will be set to 0x0

9.2.10. Direst Access to the eFLASH Memory

To enable the direct access to the eFLASH memory, set the SYSCR0.DAMODDIS to 0. eFLASH memory read/write operations via the direct access are performed at the address {eFLASH memory + {ADDR,2'b00}}, where eFLASH memory is the eFLASH memory base address (0xA000_0000) and ADDR is the address of the eFLASH memory cell.

- For read/write operations in the NVR array, the SYSCR0.NVR should be preset to 1. Note that for operations in the NVR array, the address is specified by the 10 least significant bits of the ADDR.
- If the SYSCR0.FLASHNVRDIS is set to 1, then the operation in the NVR array is not possible and it will be performed in the Main array.
- If the FLASHNVRCR.NVR1DIS is set to 1, then any operation that modifies the first block of the NVR array will not be performed, the exception is the CHIP_ERASE operation (only second block of the NVR array will be erased)

If the 6 most significant bits of the ADDR are not equal to 1 and 10 least significant bits of the ADDR are equal to 1, then when forming the next address (for INCR burst transaction with size of 2), the address corresponding to the 0th word, 0th sector of the NVR array will be formed. For example:

- Current address: ADDR = 101010_1111111111 (NVR ADDR = 1111111111)
- Next address: ADDR = 101011_0000000000 (NVR ADDR = 0000000000)

10. Mailbox

The Mailbox enables message exchange between Core 0/1 and Core 2. Each core can write data messages to this shared memory space and read messages from the other cores. The access address for the Mailbox depends on the direction of access — whether it is accessed from the Core 0/1 side or from the Core 2 side.

The Mailbox is intended only for message exchange between Core 0/1 and Core 2.

The Mailbox consists of two identical modules, **MB0** and **MB1**, each with its own set of registers.

See the Mailbox Registers section for detailed register descriptions.

The diagram below provides an overview of the Mailbox within the BE-U1000 microcontroller.

Due to architectural features, the Core 2 has two equivalent ways to access the Mailbox: via the TCM Arbiter and via the Peripheral Port. These two access paths use different offsets for the same Mailbox registers. For details, refer to Mailbox Registers section.

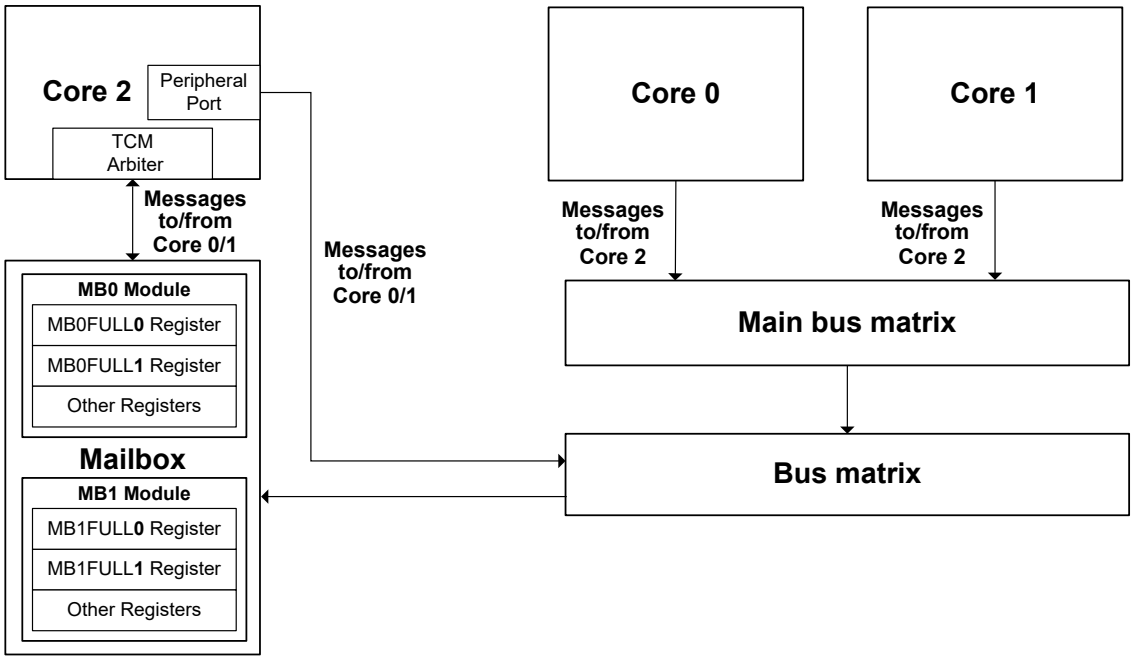

Figure 10-1 Mailbox in BE-U1000 block diagram

10.1. Mailbox Registers

This section covers the following topics:

- Register Map
- Register Descriptions

The following table describes the BE-U1000 microcontroller memory address regions for Mailbox registers.

Table 10-1 Memory address regions for Mailbox registers

Device	Size	Start Address	End Address
The Mailbox registers, accessed from the Core 0...2 side*	4 KB	0x1510_3000	0x 1510_3FFF
The Mailbox registers, accessed from the Core 2 side	1 KB	0x4000_A400	0x4000_A800

*Due to architectural features, the Core 2 has two equivalent ways to access the Mailbox: via the TCM Arbiter and via the Peripheral Port. These two access paths use different offsets for the same Mailbox registers.

Therefore, the same Mailbox registers for the Core 2 located at the two start offsets (as shown in the table above): 0x1510_3000 and 0x4000_A400.

The list and description of the Mailbox registers are provided in the Register Map and Register Descriptions sections below.

10.1.1. Register Map

This section tables contain information about the Mailbox registers memory map.

Since the Mailbox consists of two identical modules, **MB0** and **MB1**, the table below shows memory mapping for registers of both modules.

The following table provides a high-level summary of each register. To view the detailed description of the register, click the register name in the **Description** column.

Table 10-2 Mailbox register map

Name	Offset	Description	Default Value
MB0 Module of Mailbox Registers			
MB0DATA0	0x0	Messages Exchange Register 0	0x0
MB0FULL0	0x4	Message Ready Flag Register 0	0x0
MB0EMPTY0	0x8	Message Not Ready Flag Register 0	0x1
MB0DATA1	0x10	Messages Exchange Register 1	0x0
MB0FULL1	0x14	Message Ready Flag Register 1	0x0
MB0EMPTY1	0x18	Message Not Ready Flag Register 1	0x1
MB1 Module of Mailbox Registers			
MB1DATA0	0x20	Messages Exchange Register 0	0x0
MB1FULL0	0x24	Message Ready Flag Register 0	0x0
MB1EMPTY0	0x28	Message Not Ready Flag Register 0	0x1

Table 10-2 Mailbox register map (continued)

Name	Offset	Description	Default Value
MB1DATA1	0x30	Messages Exchange Register 1	0x0
MB1FULL1	0x34	Message Ready Flag Register 1	0x0
MB1EMPTY1	0x38	Message Not Ready Flag Register 1	0x1

10.1.2. Register Descriptions

10.1.2.1. Messages Exchange Register 0 (MBnDATA0)

The MBnDATA0 register offset bits are the following:

n = 0 (MB0DATA0): 0x0

n = 1 (MB1DATA0): 0x20

The following table shows the register bit assignments.

Table 10-3 Fields for register: MBnDATA0

Bits	Name	Access		Description
		Core 0/1	Core 2	
[31:0]	MB_N_DATA0	R/W	R	Message data. **Value After Reset:** 0x0

10.1.2.2. Message Ready Flag Register 0 (MBnFULL0)

The MBnFULL0 register offset bits are the following:

n = 0 (MB0FULL0): 0x4

n = 1 (MB1FULL0): 0x24

The following table shows the register bit assignments.

Table 10-4 Fields for register: MBnFULL0

Bits	Name	Access		Description
		Core 0/1	Core 2	
[0]	MB_N_FULL0	R/W	R	This bit acts as a readiness flag for the message in the MBnDATA0 register. • This bit is set to 0x1 when the MBnDATA0 register is written. • This bit is set to 0x0 when the MBnDATA0 register is read from the Core 2 side. This bit can be set to any value by writing from the Core 0/1 side. **Value After Reset:** 0x0

10.1.2.3. Message Not Ready Flag Register 0 (MBnEMPTY0)

The MBnEMPTY0 register offset bits are the following:

n = 0 (MB0EMPTY0): 0x8

n = 1 (MB1EMPTY0): 0x28

The following table shows the register bit assignments.

Table 10-5 Fields for register: MBnEMPTY0

Bits	Name	Access		Description
		Core 0/1	Core 2	
[0]	MB_N_EMPTY0	R	R	This bit acts as message not ready flag in the MBnDATA0 register. • This bit is set to 0x0 when the MBnDATA0 register is written. • This bit is set to 0x1 when the MBnDATA0 register is read from the Core 2 side. **Value After Reset:** 0x1

10.1.2.4. Messages Exchange Register 1 (MBnDATA1)

The MBnDATA1 register offset bits are the following:

n = 0 (MB0DATA1): 0x10

n = 1 (MB1DATA1): 0x30

The following table shows the register bit assignments.

Table 10-6 Fields for register: MBnDATA1

Bits	Name	Access		Description
		Core 0/1	Core 2	
[31:0]	MB_N_DATA1	R	R/W	Message data. **Value After Reset:** 0x0

10.1.2.5. Message Ready Flag Register 1 (MBnFULL1)

The MBnFULL1 register offset bits are the following:

n = 0 (MB0FULL1): 0x14

n = 1 (MB1FULL1): 0x34

This register also acts as an interrupts source for Core 0/1.

The following table shows the register bit assignments.

Table 10-7 Fields for register: `MBnFULL1`

Bits	Name	Access		Description
		Core 0/1	Core 2	
[0]	`MB_N_FULL1`	R	R/W	This bit acts as a readiness flag for the message in the `MBnDATA1` register. • This bit is set to 0x1 when the `MBnDATA1` register is written. • This bit is set to 0x0 when the `MBnDATA1` register is read from the Core 0/1 side. This bit can be set to any value by writing from the Core 2 side. **Value After Reset:** 0x0

10.1.2.6. Message Not Ready Flag Register 1 (`MBnEMPTY1`)

The `MBnEMPTY1` register offset bits are the following:

n = 0 (`MB0EMPTY1`): 0x18

n = 1 (`MB1EMPTY1`): 0x38

The following table shows the register bit assignments.

Table 10-8 Fields for register: `MBnEMPTY1`

Bits	Name	Access		Description
		Core 0/1	Core 2	
[0]	`MB_N_EMPTY1`	R	R	This bit acts as message not ready flag in the `MBnDATA1` register. • This bit is set to 0x0 when the `MBnDATA1` register is written. • This bit is set to 0x1 when the `MBnDATA1` register is read from the Core 0/1 side. **Value After Reset:** 0x1

11. ADC

This section describes the *Analog-to-Digital Converter* (**ADC**) of the BE-U1000 microcontroller. The BE-U1000 contains three ADCs.

A simplified block diagram of the ADC is shown in the following figure.

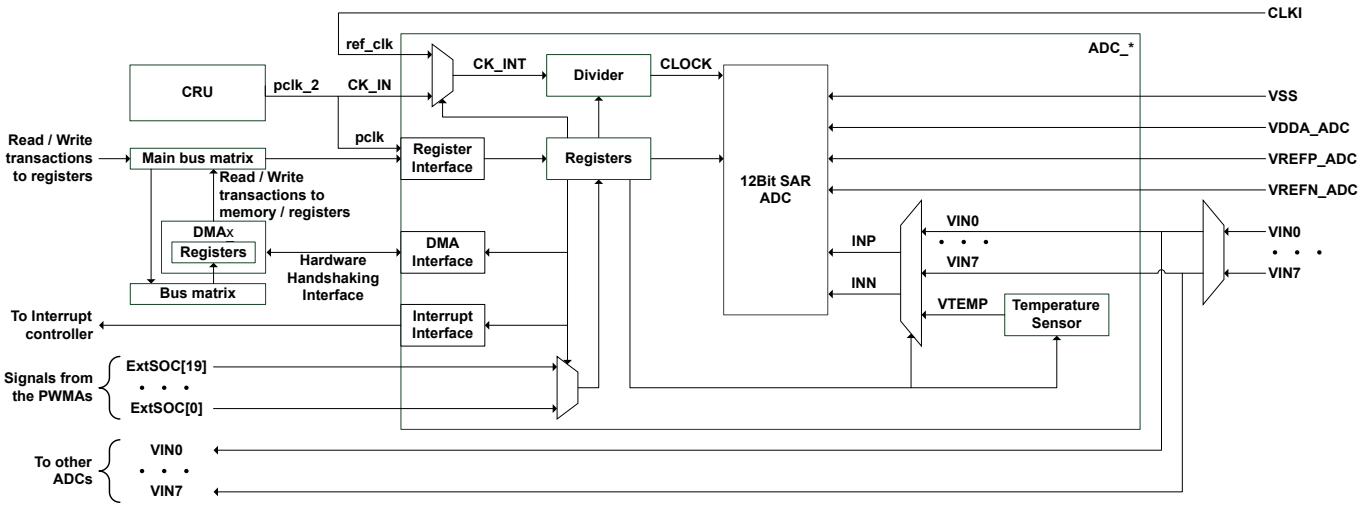

Figure 11-1 ADC block diagram

In the figure above, an asterisk symbol (*) indicates the ADC number and `x` indicates the DMA number.

Each signal ID, shown in the right part of the diagram above, corresponds to the appropriate pin ID. For more information, refer to the ***BE-U1000 Datasheet*** and ADC Pins section.

The ADC controller supports the following features:

- 12-bit resolution
- 8 single ended or 4 differential channels
- Programmable prescaler for the ADC clock (`CLOCK`) frequency
- Possibility to select source of the `CK_INT` signal: from CLKI pin or from CRU
- Possibility to select the channel for conversion and programmable sample time for each channel
- 4 operation modes:
  - Single mode
  - Scan mode
  - Discontinuous mode
  - Temperature sensor mode
- Autonomous work in the scan mode with the programmable delay before the each measurement
- Embedded temperature sensor
- *Start of Conversion* (**SOC**) by software request, PWMA event and internal divider
- *End of Conversion* (**EOC**) interrupt generation

11.1. ADC Functional Description

This section covers the following topics:

- ADC Pins
- Operation Modes
- Conversion on PWMA Event
- Direct Clocking

11.1.1. ADC Pins

This section gives the ADC pins description and `VIN` pins connection with the ADC channels. For more information about the ADC pins, refer to the ***BE-U1000 Datasheet***.

Table 11-1 ADC pins

Name	Description
`VREFP_ADC`	Input, positive reference voltage
`VREFN_ADC`	Input, negative reference voltage
`VDDA_ADC`	Input, analog power supply
`VSS`	Input, analog ground
`VIN[7:0]`	Analog input channels

The tables below describe the connection of the `VIN` pins to the ADC channels.

The signal is sampled in the range from `VREFP_ADC` to `VREFN_ADC`.

Both `INP` and `INN` ADC input signals can be used only if the input type is set to differential (`ASER.SELVI_HD_LS` = 0x1). If the input type is single-ended (`ASER.SELVI_HD_LS` = 0x0), only the `INP` signal is used.

Table 11-2 ADC channels connection to the `VIN` pins

ADC Channel	Single-ended input type	Differential input type	
		`VIN` connection to `INP`	`VIN` connection to `INN`
Channel 0	`VIN0`	`VIN0`	`VIN1`
Channel 1	`VIN1`	`VIN1`	`VIN0`
Channel 2	`VIN2`	`VIN2`	`VIN3`
Channel 3	`VIN3`	`VIN3`	`VIN2`
Channel 4	`VIN4`	`VIN4`	`VIN5`
Channel 5	`VIN5`	`VIN5`	`VIN4`
Channel 6	`VIN6`	`VIN6`	`VIN7`
Channel 7	`VIN7`	`VIN7`	`VIN6`

11.1.2. Operation Modes

Before starting the measurement in each of the ADC operation modes, it is necessary to set the `ADC_CR1.ADON` bit and configure the ADC clock (`CLOCK`) via setting the `ADC_CR1.CKD` bit.

This section covers the following topics:

- Single Mode
- Scan Mode
- Discontinuous Mode
- Temperature Sensor Mode

11.1.2.1. Single Mode

Single mode is used if the SCANEN and DISCEN bits of the ADC_CR1 register are set to 0x0.

The number of the input to be converted is specified in the SQ0 bit field of the ASQR register.

In single mode the conversion is started by setting the SWSTART bit of the ADC_CR1 register or by an external event (see Conversion on External Trigger for details). Start of conversion can be monitored through the STRT bit of the ASTR register. When conversion is started, after at least 25 ADC clock cycles, the EOC bit of the ASTR register is set to 0x1, it means that the conversion is completed. The conversion result is stored in the ADR register.

11.1.2.2. Scan Mode

Scan mode is used only if the DISCEN bit of the ADC_CR1 register is set to 0x0.

In this mode, the specified sequence of the ADC analog inputs is measured. Scan mode can be selected by setting the SCANEN bit in the ADC_CR1 register.

Scan mode conversion sequence is controlled through the L (set the sequence length) and SQn (defines the channel number for the conversion n in the sequence) bits of the ASQR register.

For example, you can define the sequence with the length of five where the measuring of the inputs will be performed in the following order: channel 1, channel 2, channel 1, channel 3, channel 7. This can be done by setting the L and SQn bits of the ASQR register as follows:

- ASQR.L = 0x4
- ASQR.SQ0 = 0x1
- ASQR.SQ1 = 0x2
- ASQR.SQ2 = 0x1
- ASQR.SQ3 = 0x3
- ASQR.SQ4 = 0x7

It is also possible to specify the delay before the each next measurement via the SMPn bits of the ASMPR register. Each ASMPR.SMPn bit field corresponds to a channel specified in the ASQR.SQn bit field, where n is the conversion number. For example, the delay for a channel specified in the ASQR.SQ0 bit field is configured through the ASMPR.SMP0 bit field.

Full conversion time for each measurement of the sequence is calculated as follows: conversion time = conversion sample start delay + 25 ADC clock cycles + 5 register interface clock cycles. Conversion sample start delay is set via the ASMPR.SMPn bit field, where n is the conversion number.

After the entire sequence has been measured, scan mode will be stopped. Scan mode can be restarted by setting the SWSTART bit of the ADC_CR1 register or by an external event (see Conversion on External Trigger for details).

Scan mode will be restarted automatically if the CONT bit of the ADC_CR1 register is set.

11.1.2.3. Discontinuous Mode

This mode is enabled by setting the DISCEN bit in the ADC_CR1 register. It can be used to convert a short sequence of x conversions (x <= 8) which is a part of the full sequence of conversions selected in the ASQR register. The value of x is specified by writing to the DISCNUM bits in the ADC_CR1 register.

For example, if the ASQR.L = 0x7 (full sequence length is set to eight) and ADC_CR1.DISCNUM = 0x1 (discontinuous mode will be stopped after short sequence of two conversions), the sequence will be performed as follows:

Start of the each short sequence is performed via setting the SWSTART bit of the ADC_CR1 register or by an external event (see Conversion on External Trigger for details).

- First short sequence: conversion of channels specified in the SQ0 and SQ1 fields of the ASQR register
- Second short sequence: conversion of channels specified in the SQ2 and SQ3 fields of the ASQR register
- Third short sequence: conversion of channels specified in the SQ4 and SQ5 fields of the ASQR register
- Fourth short sequence: conversion of channels specified in the SQ6 and SQ7 fields of the ASQR register

11.1.2.4. Temperature Sensor Mode

This mode is enabled by setting the TEMPSENS bit of the ADC_CR1 register. When TEMPSENS bit is set, the ADC starts working only with a built-in temperature sensor without usage the VIN pins. In other words, to measure the temperature, you should set the TEMPSENS bit and start the measurement in single mode. If you enable other ADC operation modes while the TEMPSENS bit is set, then, regardless of any settings, the ADC will receive data only from the temperature sensor.

The following sections describe the temperature measurement algorithms.

- Measurement Without Trimming or Calibration
- Measurement With Trimming and Calibration

11.1.2.4.1. Measurement Without Trimming or Calibration

This section describes the temperature measurement without trimming or calibration.

The table below specifies the temperature parameters that are used for measuring the temperature.

Table 11-3 Temperature parameters

Symbol	Parameter
$D_{initial}$	542.7
C_e	0.4854
D_{ref}	1496

When VREFP_ADC is steady and no variation:

1. Set ADC_CR1.TEMPSENS = 0x1, ASER.SELVI_HD_LS = 0x0, ASER.SELDO_HS_LU = 0x0, and ASER.TS_MODE = 0x1 or 0x2 or 0x3.
2. Set ADON and SWSTART bits of the ADC_CR1 register.
3. Set ASER.CAL_SEL = 0x1, ASER.ADJ_TD_GA[8] = 0x0 and waiting for the third *End of Conversion* (**EOC**) high pulse, then record ADR.DATA as D0.
4. After record D0, set ASER.CAL_SEL = 0x0, ASER.ADJ_TD_GA[8] = 0x0 and waiting for the third EOC high pulse, then record ADR.DATA as D1.
5. Calculate the current temperature (T_C) as follows:

$$T_C = (D0 - D1 - D_{initial}) * C_e$$

VREFP_ADC is the BE-U1000 AI pin, refer to the ***BE-U1000 Datasheet*** for details.

When VREFP_ADC variation with VDDA_ADC:

1. Set ADC_CR1.TEMPSENS = 0x1, ASER.SELVI_HD_LS = 0x0, ASER.SELDO_HS_LU = 0x0, and ASER.TS_MODE = 0x1 or 0x2 or 0x3.
2. Set ADON and SWSTART bits of the ADC_CR1 register.
3. Set ASER.CAL_SEL = 0x1, ASER.ADJ_TD_GA[8] = 0x0 and waiting for the third *End of Conversion* (**EOC**) high pulse, then record ADR.DATA as D0.
4. After record D0, set ASER.CAL_SEL = 0x0, ASER.ADJ_TD_GA[8] = 0x0 and waiting for the third EOC high pulse, then record ADR.DATA as D1.
5. After record D1, set ASER.CAL_SEL = 0xX, ASER.ADJ_TD_GA[8] = 0x1 and waiting for the third EOC high pulse, then record ADR.DATA as D2.
6. Calculate the current temperature (T_C) as follows:

$$T_C = ((D0 - D1) * (D0 + 2048 - D2) / D_{ref} - D_{initial}) * C_e$$

VREFP_ADC and VDDA_ADC are the BE-U1000 AI pins, refer to the ***BE-U1000 Datasheet*** for details.

11.1.2.4.2. Measurement With Trimming and Calibration

This section describes the temperature measurement with trimming and calibration.

Pay attention, the best accuracy approximation is obtained at the T_L = 0 °C and T_H = 50 °C

Before using, the offset and slope must be trimming at first. The applications of trimming method and step are as below:

1. Configure ADC to temperature detectiong mode.
2. Record the ADR.DATA at T_L (0 °C) as D_{TL}.
3. Record the ADR.DATA at T_H (50 °C) as D_{TH}.

In the practical application, use following equation to get the finial result (here D_{TEST} is the data recorded at the T_{TEST} temperature):

$$T_{TEST} = (D_{TEST} - D_{TL}) * (T_H - T_L) / (D_{TH} - D_{TL}).$$

11.1.3. Conversion on PWMA Event

Conversion can be triggered by an PWMA event if the ADC_CR1.EXTRIG bit is set. Each ADC contains the PWMA event that can be used as an external event source. The PWMA event source is selected via setting the ADC_CR1.EXTSEL bits as shown in the table below.

Table 11-4 PWMA event selection

Register settings	ADC_0	ADC_1	ADC_2
ADC_CR1.EXTSEL = 0x13	PWMA_0 OC0 signal	PWMA_1 OC0 signal	PWMA_2 OC0 signal
ADC_CR1.EXTSEL = 0x12	PWMA_0 OC1 signal	PWMA_1 OC1 signal	PWMA_2 OC1 signal
ADC_CR1.EXTSEL = 0x11	PWMA_0 OC2 signal	PWMA_1 OC2 signal	PWMA_2 OC2 signal
ADC_CR1.EXTSEL = 0x10	PWMA_0 OC3 signal	PWMA_1 OC3 signal	PWMA_2 OC3 signal
ADC_CR1.EXTSEL = 0xF	PWMA_0 TRGO signal	PWMA_1 TRGO signal	PWMA_2 TRGO signal
ADC_CR1.EXTSEL = 0xE	PWMA_0 pwma_0_tim_irq signal	PWMA_1 pwma_1_tim_irq signal	PWMA_2 pwma_2_tim_irq signal
ADC_CR1.EXTSEL = 0xD	PWMA_1 OC0 signal	PWMA_2 OC0 signal	PWMA_3 OC0 signal
ADC_CR1.EXTSEL = 0xC	PWMA_1 OC1 signal	PWMA_2 OC1 signal	PWMA_3 OC1 signal
ADC_CR1.EXTSEL = 0xB	PWMA_1 OC2 signal	PWMA_2 OC2 signal	PWMA_3 OC2 signal
ADC_CR1.EXTSEL = 0xA	PWMA_1 OC3 signal	PWMA_2 OC3 signal	PWMA_3 OC3 signal
ADC_CR1.EXTSEL = 0x9	PWMA_1 TRGO signal	PWMA_2 TRGO signal	PWMA_3 TRGO signal
ADC_CR1.EXTSEL = 0x8	PWMA_1 pwma_1_tim_irq signal	PWMA_2 pwma_2_tim_irq signal	PWMA_3 pwma_3_tim_irq signal
ADC_CR1.EXTSEL = 0x7	PWMA_2 OC0 signal	PWMA_3 OC0 signal	PWMA_0 OC0 signal
ADC_CR1.EXTSEL = 0x6	PWMA_2 OC1 signal	PWMA_3 OC1 signal	PWMA_0 OC1 signal
ADC_CR1.EXTSEL = 0x5	PWMA_2 OC2 signal	PWMA_3 OC2 signal	PWMA_0 OC2 signal
ADC_CR1.EXTSEL = 0x4	PWMA_2 OC3 signal	PWMA_3 OC3 signal	PWMA_0 OC3 signal
ADC_CR1.EXTSEL = 0x3	PWMA_2 TRGO signal	PWMA_3 TRGO signal	PWMA_0 TRGO signal
ADC_CR1.EXTSEL = 0x2	PWMA_2 pwma_2_tim_irq signal	PWMA_3 pwma_3_tim_irq signal	PWMA_0 pwma_0_tim_irq signal
ADC_CR1.EXTSEL = 0x1	PWMA_3 TRGO signal	PWMA_0 TRGO signal	PWMA_1 TRGO signal
ADC_CR1.EXTSEL = 0x0	PWMA_3 OC0 signal	PWMA_0 OC0 signal	PWMA_1 OC0 signal

11.1.4. Direct Clocking

ADC can be clocked directly from the CLKI pin. To select the CLKI pin as a source of the CK_INT signal, set DIRECT_CLOCK bit of the ADC_CR1 register to 0x1.

11.2. ADC Registers

Register regions of the ADC controllers are mapped in the BE-U1000 microcontroller Memory Map as the following table shows.

Table 11-5 Memory address regions for ADC controller registers

Region	Block Name	Size	Start Address	End Address
Periphery 2	ADC_0	4 KB	0x1400_1000	0x1400_1FFF

Table 11-5 Memory address regions for ADC controller registers (continued)

Region	Block Name	Size	Start Address	End Address
	ADC_1	4 KB	0x1400_2000	0x1400_2FFF
	ADC_2	4 KB	0x1400_3000	0x1400_3FFF

 For details, refer to Memory Map chapter.

This section covers the following topics:

- Register Map
- Register Descriptions

11.2.1. Register Map

The following table provides a top-level summary of each register. To view the detailed description of a register, click the register name in the **Description** column.

Table 11-6 ADC register map

Register	Offset	Description	Default Value
ADC_CR1	0x00	ADC Control Register 1	0x0
ASER	0x04	ADC Set Register	0x346
ASTR	0x08	ADC Status Register	0x0
ASMPR	0x0C	ADC Sample Time Register	0x0
ASQR	0x10	ADC Sequence Register	0x0
ADR	0x14	ADC Data Register	0x0

11.2.2. Register Descriptions

In the tables below, the **R/W** column specifies how the application and the core can access the register fields.

11.2.2.1. ADC Control Register 1 (ADC_CR1)

The ADC_CR1 register is at offset 0x0.

The following table shows the register bit assignments.

Table 11-7 Fields for register: ADC_CR1

Bits	Name	R/W	Description
[31:23]			Reserved (return 0 when read)
[22]	DIRECT_CLOCK	R/W	Selects the ADC clocking from the CLKI pin **Values:** **0x0:** pclk_2 **0x1:** CLKI

Table 11-7 Fields for register: `ADC_CR1` (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0
[21]	`TEMPSENS`	R/W	Temperature Sensor Enables the Temperature Sensor Mode **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x0
[20]	`CLRINTRPT`	W	Interrupt clear Write 0x1 to this bit to reset the setted interrupt. Writing 0x0 has no effect. **Value After Reset:** 0x0
[19:16]	`CKD`	R/W	Clock division This bit-field indicates the division ratio between the input clock (`CK_INT`) and ADC clock (`CLOCK`). This bit field can be set to 0x0 only in Single Mode. When setting this bit field, note that the maximum ADC clock (`CLOCK`) frequency is 25 MHz. **Values:** **0x0:** $t_{CLOCK}=t_{CK\_INT}$ **0x1:** $t_{CLOCK}=2*t_{CK\_INT}$ **0x2:** $t_{CLOCK}=3*t_{CK\_INT}$ **0x3:** $t_{CLOCK}=4*t_{CK\_INT}$ **0x4:** $t_{CLOCK}=5*t_{CK\_INT}$ **0x5:** $t_{CLOCK}=6*t_{CK\_INT}$ **0x6:** $t_{CLOCK}=7*t_{CK\_INT}$ **0x7:** $t_{CLOCK}=8*t_{CK\_INT}$ **0x8:** $t_{CLOCK}=9*t_{CK\_INT}$ **0x9:** $t_{CLOCK}=10*t_{CK\_INT}$ **0xA:** $t_{CLOCK}=11*t_{CK\_INT}$ **0xB:** $t_{CLOCK}=12*t_{CK\_INT}$ **0xC:** $t_{CLOCK}=13*t_{CK\_INT}$ **0xD:** $t_{CLOCK}=14*t_{CK\_INT}$ **0xE:** $t_{CLOCK}=15*t_{CK\_INT}$ **0xF:** $t_{CLOCK}=16*t_{CK\_INT}$ **Value After Reset:** 0x0
[15]	`SWSTART`	R0/W	Start of Conversion This bit is used by software to start the conversion. Setting this bit to 0x1 starts the conversion. **Value After Reset:** 0x0
[14]	`EXTRIG`	R/W	PWMA event conversion mode Enables the PWMA event to be used to start the conversion. **Values:**

Table 11-7 Fields for register: `ADC_CR1` (continued)

Bits	Name	R/W	Description
			0x0: PWMA event conversion mode disabled **0x1:** PWMA event conversion mode enabled **Value After Reset:** 0x0
[13:9]	`EXTSEL`	R/W	External event select Selects the PWMA event source. For details on the possible field values, refer to the PWMA Event Selection table in the Conversion on PWMA Event section. **Value After Reset:** 0x0
[8:6]	`DISCNUM`	R/W	Discontinuous mode short sequence length This bit field define the number of the length of short sequence to be converted in Discontinuous Mode minus 1 **Value After Reset:** 0x0 (means 1 sample)
[5]	`DMAEN`	R/W	DMA transfer enable **Values:** **0x0**: DMA transfer is disabled **0x1**: DMA transfer is enabled **Value After Reset:** 0x0
[4]	`IEEOC`	R/W	Interrupt enable EOC Enables the interrupt on the EOC signal edge **Values:** **0x0**: Disabled **0x1**: Enabled **Value After Reset:** 0x0
[3]	`DISCEN`	R/W	Discontinuous mode Enables the Discontinuous Mode **Values:** **0x0**: Disabled **0x1**: Enabled If both `SCANEN` and `DISCEN` bits are set to 0x1, discontinuous mode will be used **Value After Reset:** 0x0
[2]	`SCANEN`	R/W	Scan mode Enables the Scan Mode **Values:** **0x0**: Disabled **0x1**: Enabled If both `SCANEN` and `DISCEN` bits are set to 0x1, discontinuous mode will be used **Value After Reset:** 0x0

Table 11-7 Fields for register: ADC_CR1 (continued)

Bits	Name	R/W	Description
[1]	CONT	R/W	Continuous conversion **Values:** **0x0**: Disabled **0x1**: Enabled **Value After Reset:** 0x0
[0]	ADON	R/W	ADC on/off **Values:** **0x0**: ADC Off **0x1**: ADC On **Value After Reset:** 0x0

11.2.2.2. ADC Set Register (ASER)

The ASER register is at offset 0x4.

The following table shows the register bit assignments.

Table 11-8 Fields for register: ASER

Bits	Name	R/W	Description
[31:19]			Reserved (return 0 when read)
[18]	CAL_SEL	R/W	Select the temperature relative signal for ADC. **Values:** **0:** Measurement **1:** Zero calibration **Value After Reset:** 0x0
[17:12]			Reserved (return 0 when read)
[11:8]	ADJ_TD_GA	R/W	Temperature sensor gain adjustment signal. **Values:** **Xx0x**: Disable temperature sensor offset adjustment **Xx1x**: Enable temperature sensor offset adjustment **Value After Reset:** 0x3
[7:4]	ADJ_TD_OS	R/W	Temperature sensor offset adjustment signal. **Values:** **000x**: For CLOCK period >160 ns **001x**: For CLOCK period 80 ns to 160 ns **010x**: For CLOCK period 40 ns to 80 ns **Value After Reset:** 0x4
[3]	SELDO_HS_LU	R/W	Output data type selection **Values:** **0:** D[11:0] is unsigned data **1:** D[11:0] is signed data

Table 11-8 Fields for register: ASER (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0
[2:1]	TS_MODE	R/W	Filler mode of temperature sensor. **Values:** **00**: By pass internal low pass filter **01**: 64 times average **10**: 128 times average **11**: 256 times average **Value After Reset:** 0x3
[0]	SELVI_HD_LS	R/W	Input type selection. **Values:** **0**: Single ended **1**: Differential **Value After Reset:** 0x0

11.2.2.3. ADC Status Register (ASTR)

The ASTR register is at offset 0x8.

The following table shows the register bit assignments.

Table 11-9 Fields for register: ASTR

Bits	Name	R/W	Description
[31:2]			Reserved (return 0 when read)
[1]	STRT	R/WC	*Start of conversion* (**SOC**) complete Any write clears this register field. **Values:** **0x0**: Conversion not started **0x1**: Conversion started **Value After Reset:** 0x0
[0]	EOC	R/WC	*End of Conversion* (**EOC**) complete Any write clears this register field. **Values:** **0x0**: Conversion not completed **0x1**: Conversion completed **Value After Reset:** 0x0

11.2.2.4. ADC Sample Time Register (ASMPR)

The ASMPR register is at offset 0xC.

ASMPR register specify the channels sample time as the number of the ADC clock periods. Each ASMPR.SMPx bit field corresponds to a channel specified in the ASQR.SQx bit field, where x is the

conversion number. The possible values are calculated from the ADC input capacity (30 pF) and input resistance (300 ohm). Different number of samples can be used for the external sources with the different capacities and resistances.

The following table shows the register bit assignments.

Table 11-10 Fields for register: `ASMPR`

Bits	Name	R/W	Description
[31:24]			Reserved
[23:21]	`SMP7`	R/W	7th conversion sample start delay selection The possible values of this bit are the same as the `SMP0` bit. **Value After Reset:** 0x0
[20:18]	`SMP6`	R/W	6th conversion sample start delay selection The possible values of this bit are the same as the `SMP0` bit. **Value After Reset:** 0x0
[17:15]	`SMP5`	R/W	5th conversion sample start delay selection The possible values of this bit are the same as the `SMP0` bit. **Value After Reset:** 0x0
[14:12]	`SMP4`	R/W	4th conversion sample start delay selection The possible values of this bit are the same as the `SMP0` bit. **Value After Reset:** 0x0
[11:9]	`SMP3`	R/W	3rd conversion sample start delay selection The possible values of this bit are the same as the `SMP0` bit. **Value After Reset:** 0x0
[8:6]	`SMP2`	R/W	2nd conversion sample start delay selection The possible values of this bit are the same as the `SMP0` bit. **Value After Reset:** 0x0
[5:3]	`SMP1`	R/W	1st conversion sample delay selection The possible values of this bit are the same as the `SMP0` bit. **Value After Reset:** 0x0
[2:0]	`SMP0`	R/W	0 conversion sample start delay selection **Values:** **0x0**: 0 ADC clock periods (0* t_{CLOCK}) **0x1**: 25 ADC clock periods (25* t_{CLOCK}) **0x2**: 50 ADC clock periods (50* t_{CLOCK}) **0x3**: 75 ADC clock periods (75* t_{CLOCK}) **0x4**: 125 ADC clock periods (125* t_{CLOCK})

Table 11-10 Fields for register: ASMPR (continued)

Bits	Name	R/W	Description
			0x5: 175 ADC clock periods (175* t_{CLOCK}) **0x6**: 225 ADC clock periods (225* t_{CLOCK}) **0x7**: 275 ADC clock periods (275* t_{CLOCK}) **Value After Reset:** 0x0

11.2.2.5. ADC Sequence Register (ASQR)

The ASQR register is at offset 0x10.

The following table shows the register bit assignments.

Table 11-11 Fields for register: ASQR

Bits	Name	R/W	Description
[31:27]			Reserved
[26:24]	L	R/W	Full sequence length Defines the total number of the conversions in the conversion sequence minus 1. **Value After Reset:** 0x0 (means 1 conversion)
[23:21]	SQ7	R/W	7th conversion in sequence Defines the channel number (0..7), that will be assigned as the 7th in the conversion sequence. **Value After Reset:** 0x0
[20:18]	SQ6	R/W	6th conversion in sequence Defines the channel number (0..7), that will be assigned as the 6th in the conversion sequence. **Value After Reset:** 0x0
[17:15]	SQ5	R/W	5th conversion in sequence Defines the channel number (0..7), that will be assigned as the 5th in the conversion sequence. **Value After Reset:** 0x0
[14:12]	SQ4	R/W	4th conversion in sequence Defines the channel number (0..7), that will be assigned as the 4th in the conversion sequence. **Value After Reset:** 0x0
[11:9]	SQ3	R/W	3rd conversion in sequence Defines the channel number (0..7), that will be assigned as the 3th in the conversion sequence. **Value After Reset:** 0x0
[8:6]	SQ2	R/W	2nd conversion in sequence Defines the channel number (0..7), that will be assigned as the 2th in the conversion sequence. **Value After Reset:** 0x0
[5:3]	SQ1	R/W	1st conversion in sequence

Table 11-11 Fields for register: ASQR (continued)

Bits	Name	R/W	Description
			Defines the channel number (0..7), that will be assigned as the 1th in the conversion sequence. **Value After Reset:** 0x0
[2:0]	SQ0	R/W	0 conversion in sequence Defines the channel number (0..7), that will be assigned as the 0th in the conversion sequence. **Value After Reset:** 0x0

11.2.2.6. ADC Data Register (ADR)

The ADR register is at offset 0x14.

The following table shows the register bit assignments.

Table 11-12 Fields for register: ADR

Bits	Name	R/W	Description
[31:12]			Reserved
[11:0]	DATA	R	Conversion result **Value After Reset:** 0x0

12. Timers

This chapter covers the following topics:

- PWMA
- PWMG
- TIM
- WDT

12.1. PWMA

This section describes the advanced-control timer (below referred to as PWMA) of the BE-U1000 microcontroller. The BE-U1000 contains four PWMAs.

A simplified block diagram of the PWMA is shown in the following figure.

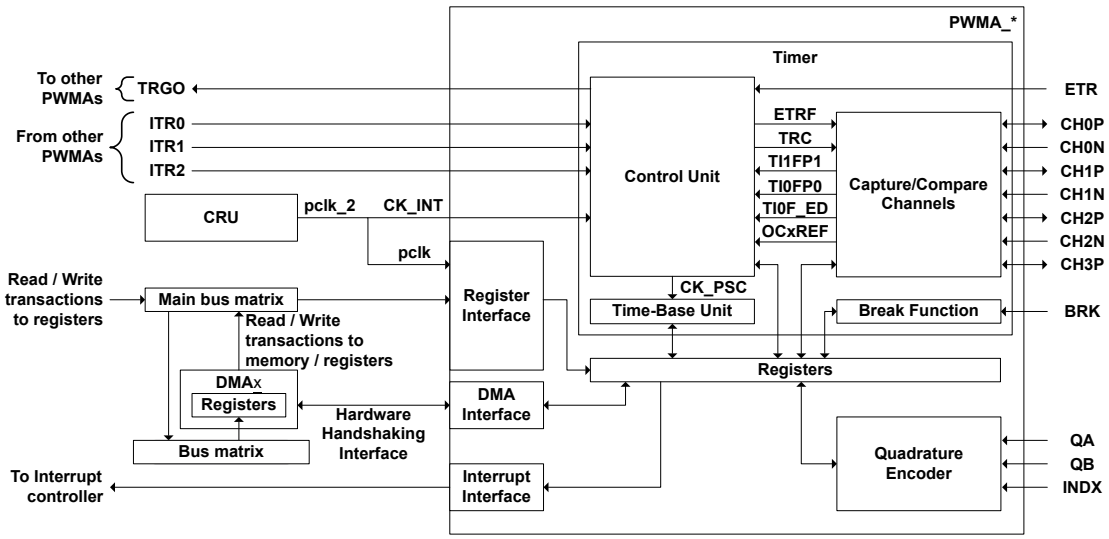

Figure 12-1 PWMA block diagram

In the figure above, an asterisk symbol (*) indicates the PWMA number, and `x` indicates the DMA number.

The Quadrature Encoder block is completely independent of the Timer block. The description and programming recommendations for the Quadrature Encoder are given in Quadrature Encoder section.

Each signal ID, shown in the right part of the diagram above, corresponds to the appropriate I/O pin ID. For more information, refer to the ***BE-U1000 Datasheet***.

The PWMA features:

- 4 independent channels: 3 complimentary channels and one single channel
- Programmable prescaler for the counter clock (`CK_CNT`) frequency
- Synchronization circuit to control the timer with external signals and to interconnect several timers together
- Programmable dead-time for the complementary channels
- Break function to put the timer's output signals in reset state or in a known state
- Embedded Quadrature Encoder for measuring shaft rotation angle

- Repetition counter to update the timer registers only after a given number of cycles of the counter.
- Interrupt/DMA request generation:
  - Update event (counter overflow/underflow, counter initialization)
  - Trigger event (counter start, stop, initialization or count by internal/external trigger)
  - Input capture
  - Output compare
  - Break input

12.1.1. PWMA Functional Description

This section covers the following topics:

- Inter-Block Trigger Connections
- Control Unit
- Time-base Unit
- Counter Modes
- Capture/Compare Channels
- XOR Function
- Dead-time Insertion
- Break Function
- Shadow Registers
- Quadrature Encoder

12.1.1.1. Inter-Block Trigger Connections

Each PWMA Timer block is interconnected to the Timers of the other PWMAs via the `TRGO` output signal and the `ITR` input buses. The table below shows the connection of the `TRGO_x` signals to the `ITR` inputs (`TRGO_x` is the `TRGO` signal from the PWMA_x, where `x` is the PWMA number).

For information about `TRGO` generation, see Master Mode Controller section.

Table 12-1 `ITR` to `TRGO` connection

`ITR` signal	`TRGO` signal
PWMA_0	
ITR2	TRGO_1
ITR1	TRGO_2
ITR0	TRGO_3
PWMA_1	
ITR2	TRGO_0
ITR1	TRGO_2
ITR0	TRGO_3

Table 12-1 ITR to TRGO connection (continued)

ITR signal	TRGO signal
PWMA_2	
ITR2	TRGO_0
ITR1	TRGO_1
ITR0	TRGO_3
PWMA_3	
ITR2	TRGO_0
ITR1	TRGO_1
ITR0	TRGO_2

12.1.1.2. Control Unit

The Control Unit consists of the following blocks, as the figure below shows:

- External Trigger Control
- Trigger Selection
- Slave Mode Controller
- Master Mode Controller

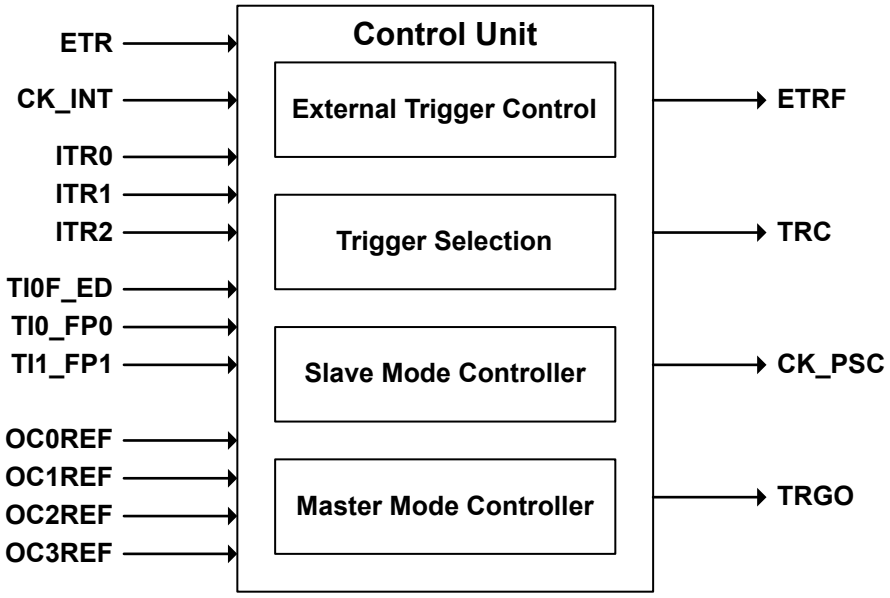

Figure 12-2 Control Unit

12.1.1.2.1. External Trigger Control

The External Trigger Control block is used for the ETR signal conversion, where ETR is the signal from the appropriate BE-U1000 I/O pin. The result of the ETR signal conversion is the ETRF signal at the block output. The following figure shows the diagram of the External Trigger Control block.

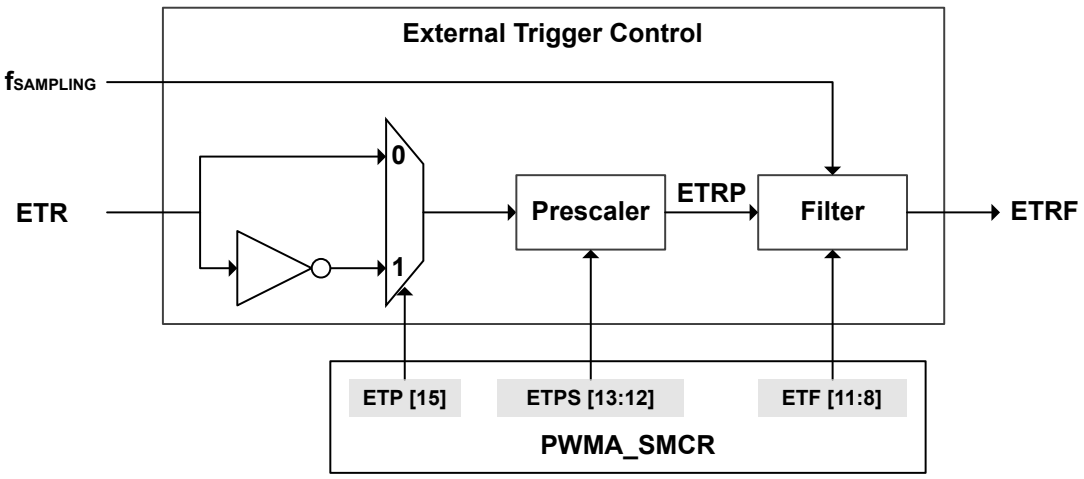

Figure 12-3 External Trigger Control block

As the figure above shows, the `ETR` signal conversion is controlled via the `ETP`, `ETPS` and `ETF` bits of the `PWMA_SMCR` register, where:

- `PWMA_SMCR.ETP` bit is used to select the active level of the `ETR` input signal.
- `PWMA_SMCR.ETPS` bit controls the prescaler that is used to divide the `ETR` input signal to produce the `ETRP` signal.
- `PWMA_SMCR.ETF` is used to select the sampling frequency ($f_{SAMPLING}$) of the `ETRP` signal.

`ETRP` frequency must be at most 1/4 of Timer clock (`CK_INT`) frequency. It means that the maximum `ETR` frequency can be two times higher than the Timer clock frequency if the `ETPS` bit is set to 0x3 (`ETR` signal divided by 8).

12.1.1.2.2. Trigger Selection

Trigger Selection block is used to select the source of the trigger input (`TRGI`) signal. This block is also generates the `TRC` signal that can be used in the Capture/Compare Channel Input Stage as the source of the `ICx` signal, where `x` is the channel number. As shown in the figure below, `TRGI` source is selected by setting the `PWMA_SMCR.TS` bit.

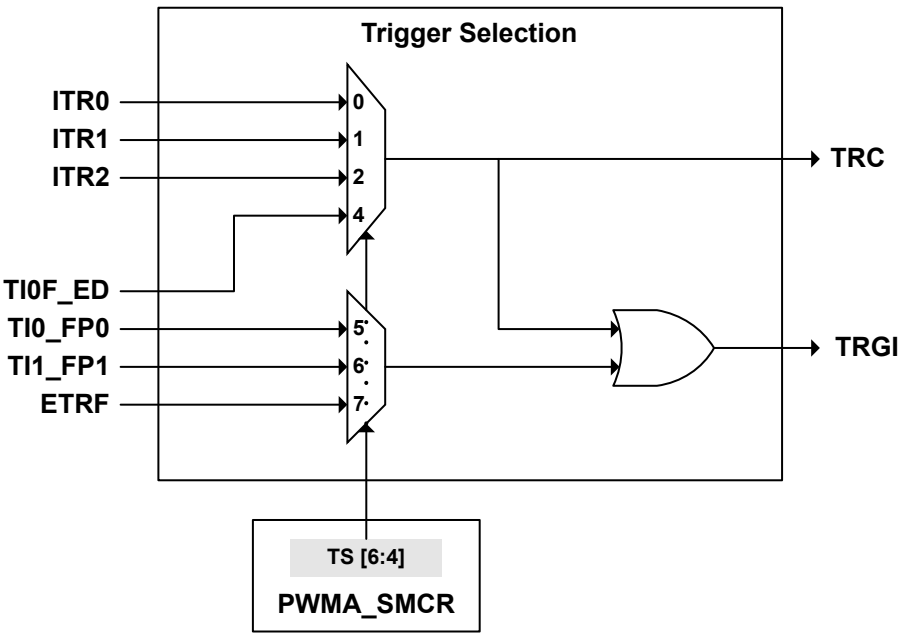

Figure 12-4 Trigger Selection block

The TRC signal can be used only if the PWMA_SMCR.TS bit field is set to 0x0, 0x1, 0x2 or 0x4.

As shown in the figure above, the following signals can be selected as the TRGI source:

- Internal trigger signals (ITR0, ITR1 and ITR2). The source of the ITRx signals is the TRGO signals from the other PWMAs. For more information, see Internal Trigger x Connection section.
- TI0F_ED, TI0_FP0 signals from the Channel 0 Input Stage and the TI1_FP1 signal from the Channel 1 Input Stage. For more information, see Input Stage section.
- ETRF signal from the External Trigger Control block.

12.1.1.2.3. Slave Mode Controller

The Slave Mode Controller is used to select the CK_PSC signal source and configure the Timer in slave mode.

As the figure below shows, the following signals can be used as the CK_PSC signal source:

- TRGI signal form the Trigger Selection block
- ETRF signal from the External Trigger Control block
- Timer clock (CK_INT) signal from the CRU

CK_PSC signal source and the Timer operation in slave mode are controlled via the SMS and ECE bits of the PWMA_SMCR register.

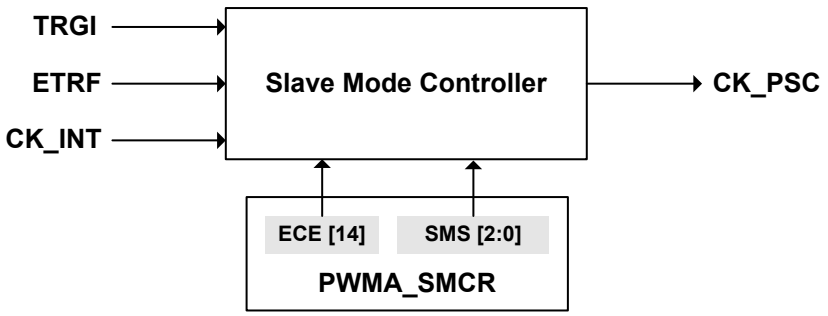

Figure 12-5 Slave Mode Controller

The following table describes the relationships between the settings of the SMS and ECE bits of the PWMA_SMCR register and the Timer slave modes.

Table 12-2 Timer slave modes

PWMA_SMCR		Slave Mode	CK_PSC signal source
SMS	ECE		
0x0	0x0	Internal Clock Mode	CK_INT
	0x1	External Clock Mode 2	ETRF
0x4	0x0	Reset Mode The counter is controlled through the TRGI signal.	CK_INT
	0x1	External Clock Mode 2 + Reset Mode The counter is controlled through the TRGI signal.	ETRF
0x5	0x0	Gated Mode The counter is controlled through the TRGI signal.	CK_INT
	0x1	External Clock Mode 2 + Gated Mode The counter is controlled through the TRGI signal.	ETRF
0x6	0x0	Trigger Mode The counter is controlled through the TRGI signal.	CK_INT
	0x1	External Clock Mode 2 + Trigger Mode The counter is controlled through the TRGI signal.	ETRF
0x7	0x0	External Clock Mode 1	TRGI
	0x1	External Clock Mode 2	ETRF

Refer to the SMS and ECE bits of the PWMA_SMCR register for a detailed description of each slave mode.

12.1.1.2.4. Master Mode Controller

The Master Mode Controller is used to generate the TRGO signal. The TRGO signal generation is controlled through the MMS bit of the PWMA_CR2 register. A simplified diagram of the Master Mode Controller is shown in the figure below.

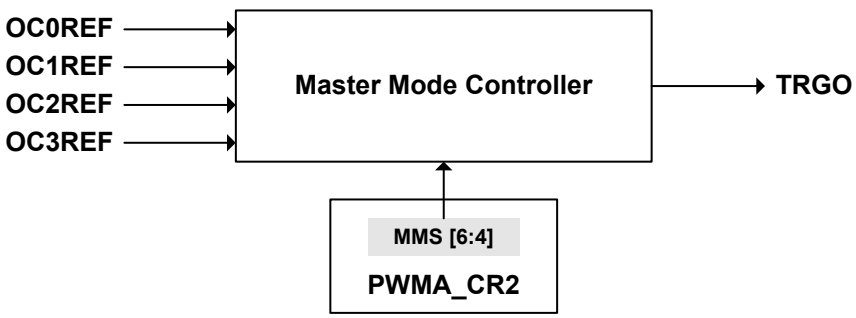

Figure 12-6 Master Mode Controller

The TRGO signal is used to control the Timers of the other PWMAs. The TRGO signal from the Timer is connected to the corresponding ITR input of the other Timer (for more information, see Internal

Trigger x Connection section). The figure below shows an example of the PWMA_0 `TRGO` signal usage (shown as the `TRGO_0`) as the `ITR2` signal of the PWMA_1.

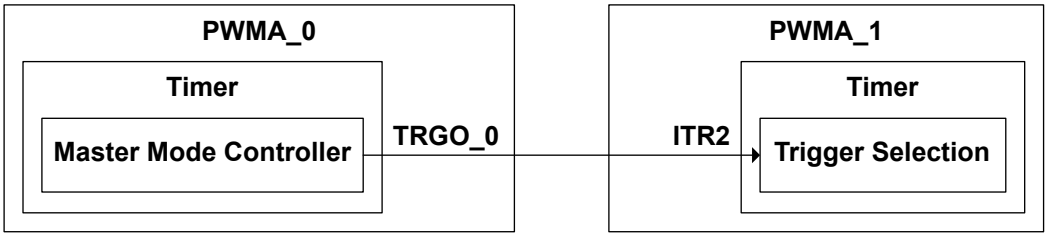

Figure 12-7 TRGO to ITR connection example

12.1.1.3. Time-Base Unit

The Time-Base Unit consists of the following blocks, as the figure below shows:

- Counter
- Prescaler
- Repetition Counter

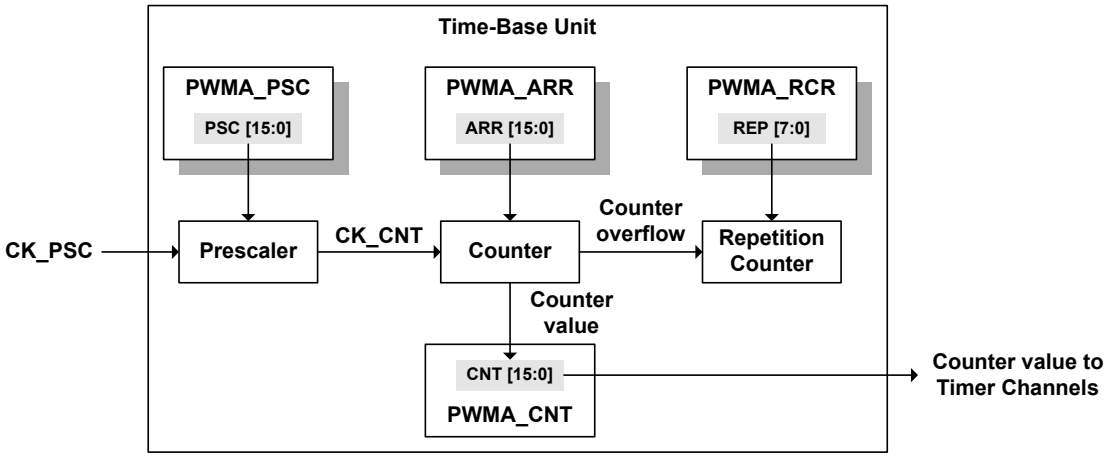

Figure 12-8 Time-Base Unit

As shown in the figure above, the blocks of the Time-Base Unit can be controlled through the following registers:

- PWMA Counter (`PWMA_CNT`)
- PWMA Prescaler (`PWMA_PSC`)
- PWMA Auto-reload Register (`PWMA_ARR`)
- PWMA Repetition Counter Register (`PWMA_RCR`)

These registers can be written or read by software even when the counter is running.

In the figure above, some registers are shown as rectangles with the gray background. This means that the register has a software inaccessible shadow register. For more information, refer to Shadow Registers section.

12.1.1.3.1. Counter

The main block of the Timer is a 16-bit counter with its related auto-reload register (PWMA_ARR). The counter can count up, down or both up and down. The counter clock (CK_CNT) can be divided by a Prescaler.

The Counter is enabled only when the CEN bit in the PWMA_CR1 register is set. Note that the Counter starts counting the one clock cycle after setting the CEN bit in the PWMA_CR1 register.

The auto-reload register (PWMA_ARR) is preloaded. Writing to or reading from the auto-reload register accesses the preload register. The content of the preload register is transferred into the shadow register permanently or at each *Update Event* (**UEV**), depending on the auto-reload preload enable bit (ARPE) in PWMA_CR1 register. The Update Event is sent when the counter reaches the overflow (or underflow when downcounting) and if the UDIS bit equals 0 in the PWMA_CR1 register. It can also be generated by software by setting the UG bit in the PWMA_EGR register. For information about the UEV generation and PWMA_ARR usage, see Counter Modes section.

12.1.1.3.2. Prescaler

The Prescaler can divide the counter clock frequency by any factor between 1 and 65536. It is based on a 16-bit counter controlled through a 16-bit PWMA_PSC register. It can be changed on the fly as this control register is buffered. The new prescaler ratio is taken into account at the next *Update Event* (**UEV**).

- The UEV can be disabled by software by setting the UDIS bit in the PWMA_CR1 register.
- The UEV can be generated by software by setting the UG bit in the PWMA_EGR register.

12.1.1.3.3. Repetition Counter

The *Update Event* (**UEV**) is generated only when the Repetition Counter has reached zero. Therefore, using the PWMA_RCR register, you can reduce the frequency of the UEV generation. This means that data are transferred from the preload registers to the shadow registers (PWMA_ARR, PWMA_PSC, PWMA_CCRx) every N+1 counter overflows or underflows, where N is the value in the PWMA_RCR register. This can be useful when generating PWM signals.

In center-aligned mode, for odd values of PWMA_RCR, the Update Event occurs on either the overflow or the underflow. This depends on whether the PWMA_RCR register was written before or after the counter was started. If the PWMA_RCR was written before starting the counter, the UEV occurs on the overflow. If the PWMA_RCR was written after the counter was started, the UEV occurs on the underflow. For example, with PWMA_RCR.REP = 0x3, the UEV is generated on each 4th overflow or underflow event, depending on when PWMA_RCR was written.

The repetition counter is decremented:

- At each counter overflow in upcounting mode
- At each counter underflow in downcounting mode
- At each counter overflow and at each counter underflow in center-aligned mode

The repetition counter is an auto-reload type; the repetition rate is maintained as defined by the PWMA_RCR register value. When the Update Event is generated by software (by setting the UG bit in PWMA_EGR register) or by hardware, it occurs immediately whatever the value of the Repetition Counter is and the Repetition Counter is reloaded with the content of the PWMA_RCR register.

12.1.1.4. Counter Modes

The timer Counter can operate in the following modes depending on the settings of the CMS and DIR bits of the PWMA_CR1 register:

- Upcounting Mode (PWMA_CR1.CMS = 0x0 and PWMA_CR1.DIR = 0x0)
- Downcounting Mode (PWMA_CR1.CMS = 0x0 and PWMA_CR1.DIR = 0x1)
- Center-aligned Mode (PWMA_CR1.CMS != 0x0)

If PWMA_CR1.CMS != 0x0, then PWMA_CR1.DIR is a read only bit that gives the current direction of the counter.

12.1.1.4.1. Upcounting Mode

In upcounting mode, the Counter counts from 0 to the auto-reload value (content of the PWMA_ARR register), then restarts from 0 and generates a counter overflow event.

If the Repetition Counter is used, an *Update Event* (**UEV**) is generated after upcounting is repeated for the number of times programmed in the Repetition Counter Register plus one (PWMA_RCR.REP+1). Otherwise, the UEV is generated at each counter overflow.

Setting the UG bit in the PWMA_EGR register (by software or by hardware in the Timer slave mode) also generates the EUV.

When the UEV is generated, the Counter restarts from 0, as well as the Prescaler counter (but the prescale rate remains unchanged).

The UEV can be disabled by software by setting the UDIS bit in the PWMA_CR1 register. This prevents the shadow registers from being updated while new values are written to the preload registers. No Update Event will occur until the UDIS bit is written to 0. However, the counter continues counting up based on the current auto-reload value.

When an UEV occurs, all the registers are updated and the update flag (UIF bit in PWMA_SR register) is set:

- The Repetition Counter is reloaded with the content of PWMA_RCR register.
- The auto-reload shadow register is updated with the preload value (PWMA_ARR).
- The Prescaler buffer is reloaded with the preload value (content of the PWMA_PSC register).

Depending on the Timer settings, the occurence of an UEV may lead to the interrupt generation or DMA request sending.

12.1.1.4.2. Downcounting Mode

In downcounting mode, the counter counts from the auto-reload value (content of the PWMA_ARR register) down to 0, then restarts from the auto-reload value and generates a counter underflow event.

If the Repetition Counter is used, an *Update Event* (**UEV**) is generated after downcounting is repeated for the number of times programmed in the Repetition Counter Register plus one (PWMA_RCR.REP+1). Otherwise, the UEV is generated at each counter underflow.

Setting the UG bit in the PWMA_EGR register (by software or by hardware in the Timer slave mode) also generates the EUV.

When the UEV is generated, the Counter restarts from the current auto-reload value, whereas the Prescaler counter restarts from 0 (but the prescale rate remains unchanged).

The UEV can be disabled by software by setting the UDIS bit in PWMA_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no Update Event occurs until UDIS bit has been written to 0. However, the counter continues counting down based on the current auto-reload value.

When an Update Event occurs, all the registers are updated and the update flag (UIF bit in PWMA_SR register) is set

- The Repetition Counter is reloaded with the content of PWMA_RCR register.
- The Prescaler buffer is reloaded with the preload value (content of the PWMA_PSC register).
- The auto-reload active register is updated with the preload value (content of the PWMA_ARR register). Note that the auto-reload is updated before the counter is reloaded.

Depending on the Timer settings, the occurrence of an Update Event can lead to an interrupt generation or a DMA request sending.

12.1.1.4.3. Center-aligned Mode (Up/Down Counting)

In center-aligned mode, the counter counts from 0 to the auto-reload value minus 1 (PWMA_ARR.ARR-1), generates a counter overflow event, then counts from the auto-reload value down to 1 and generates a counter underflow event. Then it restarts counting from 0. This counter mode can be useful when generating a PWM signals.

If the Repetition Counter is used, an *Update Event* (**UEV**) is generated when the total number of downcounting and upcounting is repeated for the number of times programmed in the Repetition Counter Register plus one (PWMA_RCR.REP+1). Otherwise, the EUV is generated at each counter overflow and at each counter underflow.

Center-aligned mode is active when the CMS bits in PWMA_CR1 register are not equal to 0x0. The Output compare interrupt flag of channels configured in output is set when:

- The counter counts down (center-aligned mode 1, PWMA_CR1.CMS = 0x1).
- The counter counts up (center-aligned mode 2, PWMA_CR1.CMS = 0x2).
- The counter counts up and down (center-aligned mode 3, PWMA_CR1.CMS = 0x3).

In this mode, the DIR direction bit in the PWMA_CR1 register cannot be written. It is updated by hardware and gives the current direction of the counter.

The UEV can be generated at each counter overflow and at each counter underflow or by setting the UG bit in the PWMA_EGR register (by software or by hardware in the Timer slave mode).

If the UG bit of the PWMA_EGR register is set, the Counter restarts counting from 0, as well as the Prescaler counter.

If the UEV is generated at counter overflow, the counter restarts from the current auto-reload value. If the UEV is generated at counter underflow, the counter restarts from 0. The counter of the prescaler restarts from 0 (but the prescale rate doesn't change).

The UEV can be disabled by software by setting the UDIS bit in the PWMA_CR1 register. This prevents the shadow registers from being updated while new values are written to the preload registers. No Update Event will occur until UDIS bit is written to 0. However, the counter continues counting up and down, based on the current auto-reload value.

When an UEV occurs, all the registers are updated and the update flag (UIF bit in PWMA_SR register) is set

- The Repetition Counter is reloaded with the content of PWMA_RCR register.
- The Prescaler buffer is reloaded with the preload value (content of the PWMA_PSC register).
- The auto-reload active register is updated with the preload value (content of the PWMA_ARR register). Note that if the update source is a counter overflow, the auto-reload is updated before the counter is reloaded.

12.1.1.5. Capture/Compare Channels

Capture/Compare Channels are used for input signals capture and output signals generation. Each Capture/Compare Channel has one input and one or two outputs (main output and complimentary output) that are connected to the BE-U1000 I/O pins.

A Capture/Compare Channel combines two functional blocks, one used when the channel operates in input mode (to capture an input signal) and the other used in output mode (to generate an output signal). Each Capture/Compare Channel is built around a Capture/Compare Stage that includes a PWMA Capture/Compare Register (with a shadow register), an Input Stage for capture operations (with a digital filter, a multiplexors and the Prescaler), and an Output Stage (with output control).

Since a channel can't work in input and output mode at the same time, the input and main output of the channel are physically combined into one CHxP I/O pin, where x corresponds to the channel number.

A simplified block diagram of the Timer Capture/Compare Channels is shown in the figure below.

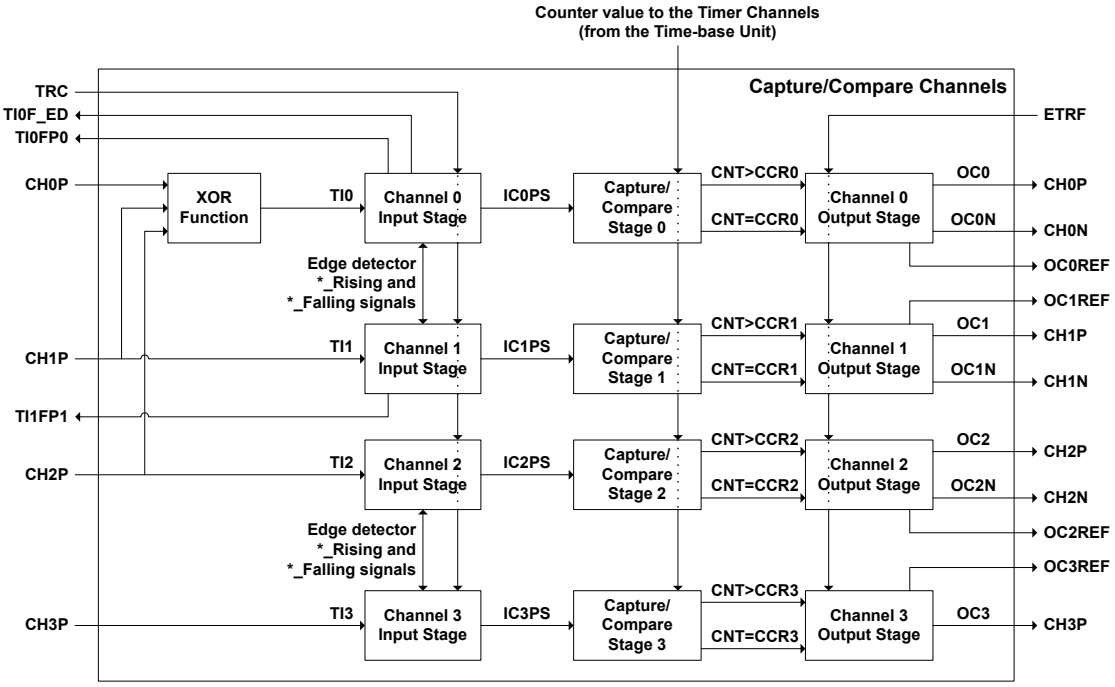

Figure 12-9 Capture/Compare Channels

As shown in the figure above, the channel 0 input (TI0) can be connected to either the CH0P I/O pin or to the logic element that performs a XOR operation on signals from the CH0P, CH1P and CH2P I/O pins. Refer to XOR Function section for details.

12.1.1.5.1. Input Stage

In this section, the operation of the Input Stage will be described using the Channel 0 Input Stage as an example. A block diagram of the Channel 0 Input Stage is shown in the figure below.

The main difference in the Channel 0 Input Stage is the presence of the TI0F_ED signal, which is used to detect both rising and falling edge of the TI0 signal and represents as XOR of the TI0F_Rising and TI0F_Falling signals. Other Input Stages do not contain signals similar to TI0F_ED.

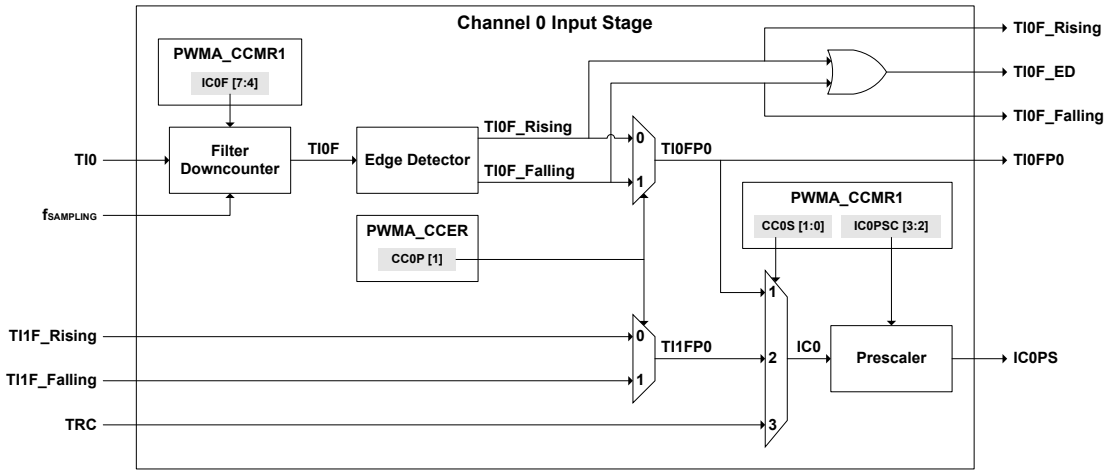

Figure 12-10 Channel 0 Input Stage

As the figure above shows, channel 0 input signal (TI0) is resynchronized and passed through the digital filter. The filter is enabled and configured via the IC0F bit field in the PWMA_CCMR1 register. In this case, TI0 signal is sampled with the sampling frequency ($f_{SAMPLING}$) that also depends on the IC0F bit field value. The resulting filtered signal TI0F then routed to the Edge Detector input.

The Edge Detector generates two output signals: TI0F_Rising and TI0F_Falling. A short pulse is generated on the TI0F_Rising signal upon detection of a rising edge on the TI0F signal, and on the TI0F_Falling signal upon detection of a falling edge on the TI0F signal. A multiplexer, placed after the Edge Detector and controlled by the CC0P bit in the PWMA_CCER register, selects either the TI0F_Rising signal or TI0F_Falling signal to be used as the source of the TI0FP0 signal.

The channels exchange edge detection signals (TIxF_Rising and TIxF_Falling, where x is the channel number) with each other. Channel 0 exchanges signals with channel 1, and channel 2 with channel 3. This expands the range of possible signal sources that can be used to control capture in each channel. Thus, Channel 0 Input Stage receives TI1F_Rising and TI1F_Falling signals from the Edge Detector of the Channel 1 Input Stage.

CC0P bit of the PWMA_CCER register also controls the multiplexer, which is used to select one of the TI1F_Rising and TI1F_Falling signals as the source of the TI1FP0 signal.

TI0FP0 and TI1FP0 signal names indicate how the signals are received:

- TI0 indicates that the signal is received on the channel 0 input, while TI1indicates that the signal is received on the channel 1 input.
- F indicates that the signal is resynchronized and passed through the digital filter.
- P0 indicates that the active level of the signal is controlled via the channel 0 settings, i.e. CC0P bit of the PWMA_CCER register.

As shown in the figure above, the IC0 signal source can be selected from one of the following: TI0FP0, TI1FP0, or the TRC signal (for more information about the TRC signal, see the Trigger Selection section). This selection is controlled by the CC0S bit field of the PWMA_CCMR1 register. The IC0 signal is then routed into a programmable Prescaler, which is controlled by the IC0PSC bit field of the PWMA_CCMR1 register. Finally, the IC0PS signal on the Prescaler output is routed to the input of the Capture/Compare Stage 0.

12.1.1.5.2. Output Stage

This section describes the operation of the Output Stage, using Channel 0 Output Stage as an example. The block diagram of the Channel 0 Output Stage is shown in the figure below.

The Channel 3 Output Stage differs from the Output Stages of the other channels. The main differences are the absence of the complimentary output (OC0N) and a dead-time generator. A block diagram of the Channel 3 Output Stage is provided at the end of this section.

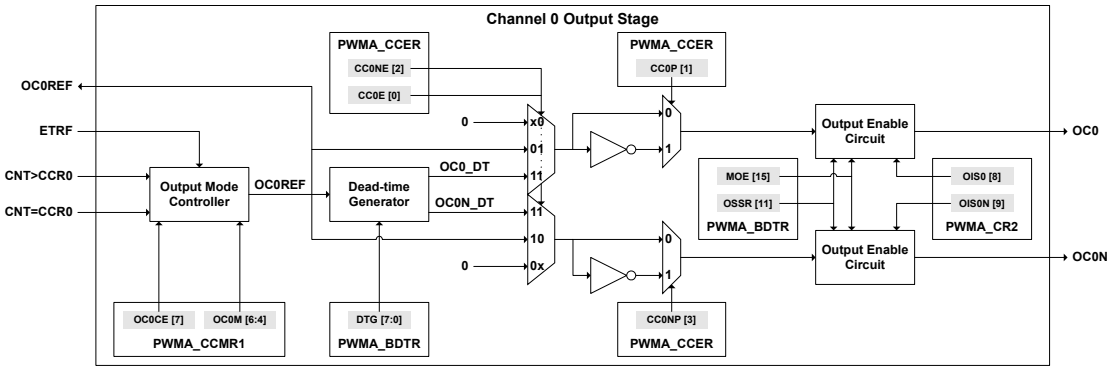

Figure 12-11 Channel 0 Output Stage

As the figure above shows, the OC0REF signal is generated by the Output Mode Controller based on the CNT>CCR0 and CNT=CCR0 signals from the Capture/Compare Stage 0 and settings of the OC0CE and OC0M bits of the PWMA_CCMR1 register.

OC0REF signal can be used by the Dead-time Generator to produce the OC0_DT and OC0N_DT signals, where:

- The OC0_DT output signal is the same as the OC0REF signal except for the rising edges, which are delayed relative to the OC0REF rising edges.
- The OC0N_DT output signal is the opposite of the OC0REF signal except for the rising edges, which are delayed relative to the OC0REF falling edges.

Thus, any switching of the OC0_DT signal is separated from the switching of the OC0N_DT by a delay programmed through the DTG bits of the PWMA_BDTR register. This delay is generally known as dead-time and it must be adjusted depending on the devices connected to the outputs and their characteristics. For more information, refer to the Dead-time Insertion section.

The OC0_DT and OC0N_DT signals are only used when both the CC0E and CC0NE bits in the PWMA_CCER register are set to 0x1. This can be seen in the figure above by observing how the CC0E and CC0NE bit settings control the two multiplexers after the Dead-time Generator. Thus, if, for example CC0NE = 0x1 and CC0E = 0x0 (only OC0N is enabled), OC0REF is used instead of OC0N_DT. In other words, when only one channel is used, the OC0REF signal is redirected to the turned on output, but without additional inversion of the signal when using a complementary output. Therefore, if you need only one channel output, you can use any of the OC0 or OC0N outputs as the main one.

The output polarity (for both the main OC0 and complementary OC0N outputs) is configured independently via the CC0P and CC0NP bits in the PWMA_CCER register. After the polarity selection multiplexers, the signals are routed to the Output Enable Circuit.

The Output Enable Circuit provides the ability to turn on and off the channel outputs, as well as control the level of the disabled outputs. The channel outputs are turned on and off via the MOE bit of the PWMA_BDTR register.

- The PWMA_BDTR.MOE bit can be cleared by software or, if the break is enabled through the PWMA_BDTR.BKE bit, by hardware. Enabling the break allows the MOE bit to be cleared asynchronously, which is used to turn the channel outputs to a known state even if the Timer clock is unavailable (For more information, see Break Function section).
- The PWMA_BDTR.MOE bit can be set by software or, if the PWMA_BDTR.AOE bit is set, automatically at the next Update Event.

The MOE bit affects the state of all Timer outputs (OCx and OCxN, where x is the channel number).

Depending on the state of the MOE bit, the level of the disabled outputs is controlled as follows:

- PWMA_BDTR.MOE = 0x1 — The output level is controlled by the OSSR bit in the PWMA_BDTR register and the CC0P and CC0NP bits in the PWMA_CCER register.

  If OSSR = 0x0, disabled outputs are set to 0. If OSSR = 0x1, disabled outputs are set to their inactive state that is equal to the PWMA_CCER.CC0P bit value for the OC0 output and to the PWMA_CCER.CC0NP bit value for the OC0N output.
- PWMA_BDTR.MOE = 0x0 — The output level is controlled by the CC0P and CC0NP bits in the PWMA_CCER register and the OIS0 and OIS0N bits in the PWMA_CR2 register.

  Disabled outputs are first set to their inactive state that is equal to the PWMA_CCER.CC0P bit value for the OC0 output and to the PWMA_CCER.CC0NP bit value for the OC0N output. Then, if the Timer clock is present, the outputs are set to their idle state after the dead time. The idle state is equal to the PWMA_CR2.OIS0 bit value for the OC0 output and to the PWMA_CR2.OIS0N bit value for the OC0N output.

If the values of both bits OIS0 and OIS0N correspond to active levels of the outputs (OIS0 != CC0P and OIS0N != CC0NP), then both outputs are set to inactive state that is equal to the CC0P and CC0NP bits values.

When both outputs of the channel are not used (CC0E = CC0NE = 0), the OIS0, OIS0N, CC0P and CC0NP bits must be kept cleared.

The table below shows how the output control bits affect the channels with complementary outputs (OCx and OCxN, where x is the channel number).

Table 12-3 Output control bits for complementary OCx and OCxN channels

Control Bits				Output States	
MOE	OSSR	CCxE	CCxNE	OCx output state	OCxN output state
0x1	0x0	0x0	0x0	Output Disabled (OCx=0)	Output Disabled (OCxN=0)
		0x0	0x1	Output Disabled (OCx=0)	OCxREF + Polarity OCxN=OCxREF xor CCxNP
		0x1	0x0	OCxREF + Polarity OCx=OCxREF xor CCxP	Output Disabled (OCxN=0)
		0x1	0x1	OCxREF + Polarity + dead-time	Complementary to OCxREF (not OCxREF) + Polarity + dead-time
	0x1	0x0	0x0	Output Disabled and set to inactive state (OCx=CCxP)	Output Disabled and set to inactive state (OCxN=CCxNP)

Table 12-3 Output control bits for complementary OCx and OCxN channels (continued)

Control Bits				Output States	
		0x0	0x1	Output Disabled and set to inactive state (OCx=CCxP)	OCxREF + Polarity OCxN=OCxREF xor CCxNP
		0x1	0x0	OCxREF + Polarity OCx=OCxREF xor CCxP	Output Disabled and set to inactive state (OCxN=CCxNP)
		0x1	0x1	OCREF + Polarity + dead-time	Complementary to OCREF (not OCREF) + Polarity + dead-time
0x0	X	0x0	0x0	Outputs Disabled and set to their inactive state (OCx=CCxP and OCxN=CCxNP). If the Timer clock is present, after the dead time outputs are set to their idle state (OCx=OISx and OCxN=OISxN) ⚠ If the values of both bits OISx and OISxN correspond to active levels of the outputs (OISx!=CCxP and OISxN!=CCxNP), then both outputs are set to inactive state that is equal to the CCxP and CCxNP bits values.	
		0x0	0x1		
		0x1	0x0		
		0x1	0x1		

The following figure shows the diagram of the Channel 3 Output Stage.

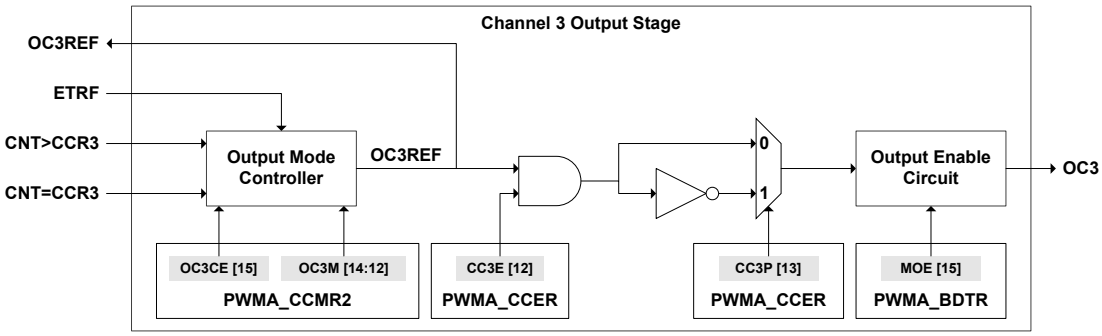

Figure 12-12 Channel 3 Output Stage

12.1.1.5.3. Capture/Compare Stage

A Capture/Compare Stage is used both when the channel is in input mode and when it is in output mode. The figure below shows a block diagram of the Capture/Compare Stage 0 as an example. Capture/Compare Stages of the other channels are fully identical.

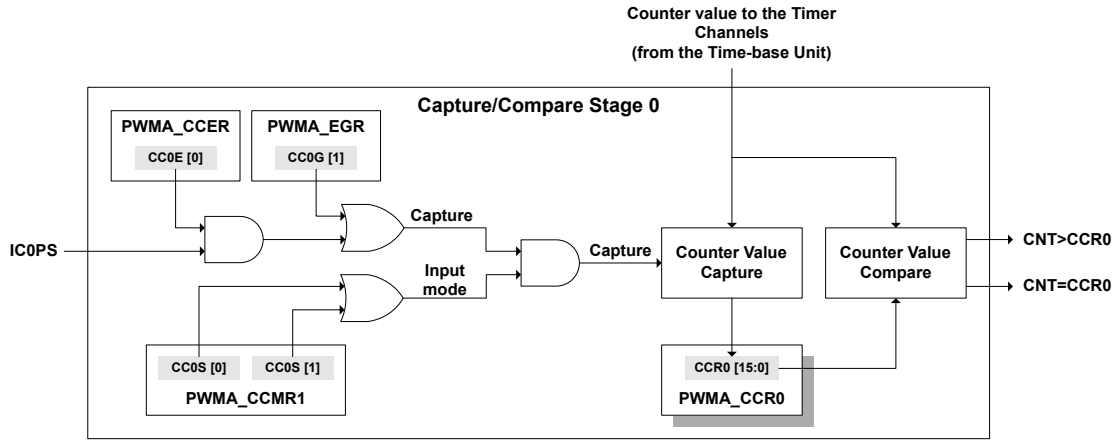

Figure 12-13 Capture/Compare Stage 0

The operation of the Capture/Compare Stage in input and output modes will be described using the Capture/Compare Stage 0 as an example.

The channel is configured as output when the CCxS bits (where x is the channel number) of the PWMA_CCMRn register (where n = 1 for channel 0 and channel 1, n = 2 for channel 2 and channel 3) are set to 0, in other case the channel is configured as input.

Capture/Compare Stage 0 in input mode (PWMA_CCMR1.CC0S != 0x0)

As shown in the figure above, when the PWMA_CCMR1.CC0S != 0x0, the Input mode signal allows the Capture signal to pass to the Counter Value Capture block. The positive pulse of the Capture signal generates the capture event, which causes the counter value (content of the PWMA_CNT register) to be written into the PWMA_CCR0 register.

The positive pulse of the Capture signal can be generated as follows:

- By software, if the CC0G bit of the PWMA_EGR register is set to 1.
- By hardware, as a result of the appearance of the IC0PS signal pulse if the CC0E bit of the PWMA_CCER register is set, where IC0PS is the signal from the Channel 0 Input Stage.

When a capture event occurs, the CC0IF bit in the PWMA_SR register is also set.

Capture/Compare Stage 0 in output mode (PWMA_CCMR1.CC0S = 0x0)

In this mode, the PWMA_CCR0 register value is compared with the PWMA_CNT register value in the Counter Value Compare block. As a result, the Counter Value Compare block generates the CNT=CCR0 and CNT>CCR0 output signals, where:

- The CNT=CCR0 signal is generated when the PWMA_CCR0 register value is equal to the PWMA_CNT register value.
- The CNT>CCR0 signal is generated when the PWMA_CNT register value is greater than the PWMA_CCR0 register value.

The CNT=CCR0 and CNT>CCR0 signals are then used in the Channel 0 Output Stage.

12.1.1.6. XOR Function

The TI0S bit in the PWMA_CR2 register allows the input of channel 0 (TI0) to be connected to the output of a XOR gate, combining the three input pins CH0P, CH1P and CH2P.

Depending on the PWMA_CR2.TI0S bit setting, the TI0 is connected as follows:

- TI0 = CH0P if the TI0S bit of the PWMA_CR2 is set to 0x0.
- TI0 = CH0P xor CH1P xor CH2P if the TI0S bit of the PWMA_CR2 is set to 0x1.

12.1.1.7. Dead-time Insertion

Dead-time insertion is enabled by setting both CCxE and CCxNE bits of the PWMA_CCER register, and the MOE bit of the PWMA_BDTR register if the break circuit is present. DTG[7:0] bits of the PWMA_BDTR register are used to control the dead-time generation for all channels. From a reference waveform OCxREF, the Dead-time Generator produces OCx_DT and OCxN_DT signals (where x corresponds to the channel number), where:

- The OCx_DT output signal is the same as the OCxREF signal except for the rising edges, which are delayed relative to the OCxREF rising edges.
- The OCxN_DT output signal is the opposite of the OCxREF signal except for the rising edges, which are delayed relative to the OCxREF falling edges.

12.1.1.8. Break Function

The break function is designed to prevent damage of the controlled power cascades in case of emergency situations. The break source is the BE-U1000 break input pin (BRK). When the break function is enabled, the channel outputs are forced into a known safe state even if the Timer clock is unavailable. In this case, the channel outputs are modified according to the additional control bits (MOE and OSSR bits of the PWMA_BDTR register, CCxP and CCxNP bits of the PWMA_CCER register, OISx and OISxN bits of the PWMA_CR2 register). If the break function is not used, then a failure of the clocking system leads to the “stuck” state of the outputs. When using a microcontroller, for example, in an inverter circuit of the power supply systems, this will lead to a sharp increase in current through the transformer and damage to the circuit elements. In addition, the external input can be connected to an overcurrent detector and an output overvoltage detector.

For more information on how the bits mentioned above affect the channel outputs, see the **Output control bits for complementary OCx and OCxN channels** table in the Output Stage section.

Upon exit from reset, the break circuit is disabled and the MOE bit is low. The user can enable the break function by setting the BKE bit in the PWMA_BDTR register. The break input polarity can be selected by configuring the BKP bit in the same register. The BKE and BKP bits can be modified at the same time. When the BKE and BKP bits are written, a delay of one Register Interface clock cycle is applied before the writing is effective. Consequently, it is necessary to wait for one Register Interface clock cycle to correctly read back the bit after the write operation.

Because MOE falling edge can be asynchronous, a resynchronization circuit has been inserted between the actual signal (acting on the outputs) and the synchronous control bit (accessed in the PWMA_BDTR register). It results in some delays between the asynchronous and the synchronous signals. In particular, if MOE is written to 1 whereas it was low, a delay (dummy instruction) must be inserted before reading it correctly.

When a break occurs (selected level on the break input):

- The MOE bit is cleared asynchronously. Outputs are set to their inactive state. This feature functions even if the BE-U1000 oscillator is off.
- When complementary outputs are used:
  - The outputs are first set to the inactive state (depending on the polarity). This is done asynchronously so that it works even if no clock is provided to the Timer.
  - If the timer clock is still present, then the Dead-time Generator is reactivated in order to drive the outputs with the level programmed in the PWMA_CR2 register OISx and OISxN bits after a dead-time.

If the values of both bits OISx and OISxN correspond to active levels of the outputs (OISx != CCxP and OISxN != CCxNP), then both outputs are set to inactive state that is equal to the values of the CCxP and CCxNP bits of the PWMA_CCER.

- The break interrupt flag (BIF bit in the PWMA_SR register) is set.
- If the AOE bit in the PWMA_BDTR register is set, the MOE bit is automatically set again at the next *Update Event* (**UEV**).

The break can also be generated by software through the BG bit of the PWMA_EGR register.

12.1.1.9. Shadow Registers

Some of the timer registers are buffered. This register type consists of a pair:

- A software-accessible register (also called the preload register or buffer register).
- A software-inaccessible register (also called the shadow register or active register).

The active registers are used during the Timer operations. They are updated with the values from their corresponding preload registers when an *Update Event* (**UEV**) occurs.

In the figures, the buffered registers are shown as rectangles with the gray background.

12.1.1.10. Quadrature Encoder

The Quadrature Encoder is used to measure the shaft rotation angle. The Quadrature Encoder is enabled by setting the QEN bit of the PWMA_QESET register to 0x1. The following figure shows the block diagram of the Quadrature Encoder.

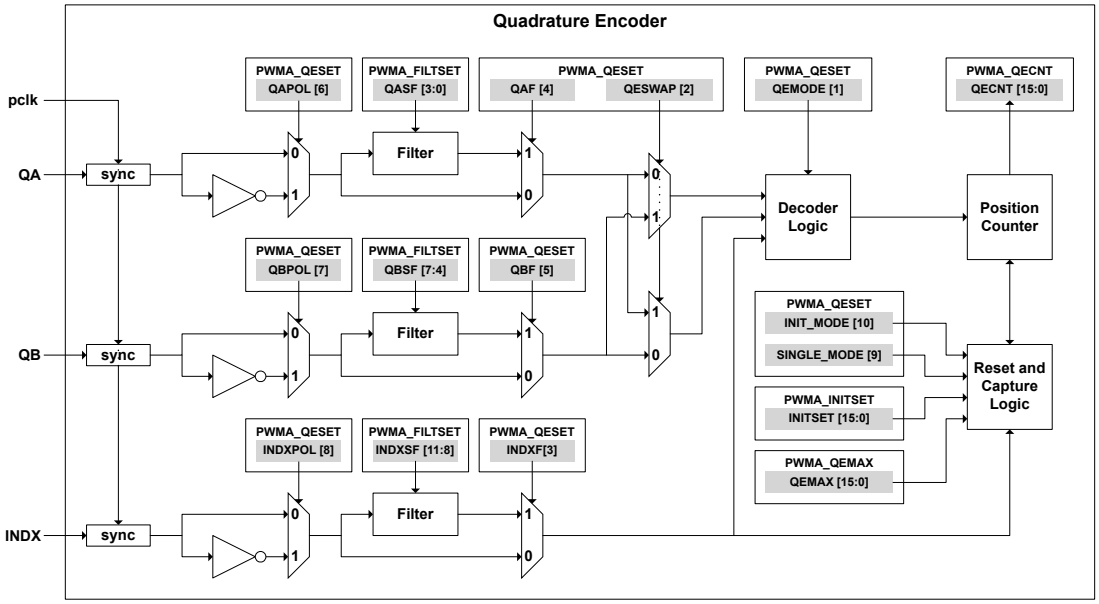

Figure 12-14 Quadrature Encoder

As shown in the figure above, the Quadrature Encoder is connected to the BE-U1000 I/O pins QA, QB and INDX, from which it receives the corresponding input signals. Each input signal is synchronized with the pclk frequency in a sync block, then each signal is sent to a Filter that is controlled via the PWMA_FILTSET register. It is also possible to invert the input signal, bypass the Filter and swap the qa and qb signals via the appropriate bits of the PWMA_QESET register.

12.1.1.10.1. Operation Modes

The Quadrature Encoder can operate in the following modes:

- Quadrature Mode (PWMA_QESET.QEMODE = 0x0)
- qa Rising/Falling Edge Mode (PWMA_QESET.QEMODE = 0x1)

12.1.1.10.1.1. Quadrature Mode

This mode is selected if the QEMODE bit of the PWMA_QESET register is set to 0x0. In Quadrature Mode the counter counts up or down depending on the qa and qb signal phases.

The Quadrature Mode supports the following features:

- It can be used in either single or multiple mode:
  - In single mode (PWMA_QESET.SINGLE_MODE = 0x1), the counter counts up to the PWMA_QEMAX value and then stops, or counts down to 0 and then stops.
  - In multiple mode (PWMA_QESET.SINGLE_MODE = 0x0), the counter can count up to the PWMA_QEMAX value and, if the PWMA_QECNT = PWMA_QEMAX, the counter restarts from 0. The counter can also count down to 0 and, if the PWMA_QECNT = 0, the counter restarts from the PWMA_QEMAX value.
- Counter reset or initialization on the indx rising edge (for more information, see Initialization and Reset).

12.1.1.10.1.2. qa Rising/Falling Edge Mode

This mode is selected if the QEMODE bit of the PWMA_QESET register is set to 0x1. In qa Rising/Falling Edge Mode the counter counts up or down on the qa rising edge (if QAPOL bit of the PWMA_QESET is set to 0x0) or on the qa falling edge (if QAPOL bit of the PWMA_QESET is set to 0x1), where the direction of the counter depends on the qb signal.

This mode can be used in either single or multiple mode:

- In single mode (PWMA_QESET.SINGLE_MODE = 0x1), the counter counts up to the PWMA_QEMAX value and then stops, or counts down to 0 and then stops.
- In multiple mode (PWMA_QESET.SINGLE_MODE = 0x0), the counter counts up to the PWMA_QEMAX value and, if the PWMA_QECNT = PWMA_QEMAX, the counter restarts from 0. The counter can also count down to 0 and, if the PWMA_QECNT= 0, the counter restarts from the PWMA_QEMAX value.

12.1.1.10.2. Initialization and Reset

The Quadrature Encoder uses the indx signal for counter reset or initialization. Depending on the value of the INIT_MODE field of the PWMA_QESET register, one of the following operations will be performed:

- If INIT_MODE = 0x0, the rising edge of the indx signal will cause the counter reset.
- If INIT_MODE = 0x1, the rising edge of the indx signal will cause the counter initialization with the value of the PWMA_INITSET register.

12.1.2. PWMA Registers

The register regions of the PWMA controllers are mapped in the BE-U1000 microcontroller Memory Map as shown in the table below.

Table 12-4 Memory address regions for PWMA registers

Region	Block Name	Size	Start Address	End Address
Periphery 2	PWMA_0	4 KB	0x1400_4000	0x1400_4FFF
	PWMA_1	4 KB	0x1400_5000	0x1400_5FFF
	PWMA_2	4 KB	0x1400_6000	0x1400_6FFF

Table 12-4 Memory address regions for PWMA registers (continued)

Region	Block Name	Size	Start Address	End Address
	PWMA_3	4 KB	0x1400_7000	0x1400_7FFF

For more information, see Memory Map chapter.

This section covers the following topics:

- Register Map
- Register and Field Descriptions

12.1.2.1. Register Map

The following table provides a top-level summary of each register. To view the detailed description of a register, click the register name in the **Description** column.

Table 12-5 PWMA register map

Register	Offset	Description	Default Value
PWMA Timer Registers			
PWMA_CR1	0x00	PWMA Control Register 1	0x0
PWMA_CR2	0x04	PWMA Control Register 2	0x0
PWMA_SMCR	0x08	PWMA Slave Mode Control Register	0x0
PWMA_DIER	0x0C	PWMA DMA/Interrupt Enable Register	0x0
PWMA_SR	0x10	PWMA Status Register	0x0
PWMA_EGR	0x14	PWMA Event Generation Register	0x0
PWMA_CCMR1	0x18	PWMA Capture/Compare Mode Register 1	0x0
PWMA_CCMR2	0x1C	PWMA Capture/Compare Mode Register 2	0x0
PWMA_CCER	0x20	PWMA Capture/Compare Enable Register	0x0
PWMA_CNT	0x24	PWMA Counter	0x0
PWMA_PSC	0x28	PWMA Prescaler	0x0
PWMA_ARR	0x2C	PWMA Auto-reload Register	0x0
PWMA_RCR	0x30	PWMA Repetition Counter Register	0x0
PWMA_CCR0	0x34	PWMA Capture/Compare Register 0	0x0
PWMA_CCR1	0x38	PWMA Capture/Compare Register 1	0x0
PWMA_CCR2	0x3C	PWMA Capture/Compare Register 2	0x0
PWMA_CCR3	0x40	PWMA Capture/Compare Register 3	0x0

Table 12-5 PWMA register map (continued)

Register	Offset	Description	Default Value
PWMA_BDTR	0x44	PWMA Break and Dead-time Register	0x0
PWMA_DCR	0x48	PWMA DMA Control Register	0x0
PWMA_DMAR	0x4C	PWMA DMA Address for Full Transfer	0x0
PWMA Quadrature Encoder Registers			
PWMA_QESET	0x50	PWMA Quadrature Encoder Setting Register	0x0
PWMA_FILTSET	0x54	PWMA Quadrature Encoder Filters Setting	0x0
PWMA_QEMAX	0x58	PWMA Quadrature Encoder Max Count Value	0xFFFF
PWMA_INITSET	0x5C	PWMA Quadrature Encoder Init Value	0xFFFF
PWMA_QECNT	0x60	PWMA Quadrature Encoder Counter Value	0x0

12.1.2.2. Register Descriptions

In the tables below, the **R/W** column specifies how the application and the core can access the register fields.

12.1.2.2.1. PWMA Control Register 1 (PWMA_CR1)

The PWMA_CR1 register is at offset 0x00.

The following table shows the register bit assignments.

Table 12-6 Fields for register: PWMA_CR1

Bits	Name	R/W	Description
[15:10]			Reserved (return 0 when read)
[9:8]	CKD	R/W	Clock division This bit field indicates the division ratio between the timer clock period (t_{CK_INT}) and the dead-time and sampling clock period (t_{DTS}). The dead-time and sampling clock is used by the Dead-time Generators and the digital filters (ETR, TIx). **Values:** **0x0:** $t_{DTS}=t_{CK\_INT}$ **0x1:** $t_{DTS}=2*t_{CK\_INT}$ **0x2:** $t_{DTS}=4*t_{CK\_INT}$ **0x3:** Reserved **Value After Reset:** 0x0
[7]	ARPE	R/W	Auto-reload preload enable **Values:** **0x0:** PWMA_ARR register is not buffered **0x1:** PWMA_ARR register is buffered

Table 12-6 Fields for register: `PWMA_CR1` (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0
[6:5]	`CMS`	R/W	Center-aligned mode selection **Values:** **0x0:** Edge-aligned mode. The Counter counts up or down depending on the direction bit (`DIR`). **0x1:** Center-aligned mode 1. The Counter counts up and down alternatively. Output compare interrupt flags of the channels configured in output (`CCxS`=0x0 in `PWMA_CCMRx` registers) are set only when the Counter is counting down. **0x2:** Center-aligned mode 2. The Counter counts up and down alternatively. Output compare interrupt flags of the channels configured in output (`CCxS`=0x0 in `PWMA_CCMRx` registers) are set only when the Counter is counting up. **0x3:** Center-aligned mode 3. The Counter counts up and down alternatively. Output compare interrupt flags of the channels configured in output (`CCxS`=0x0 in `PWMA_CCMRx` registers) are set both when the Counter is counting up or down. It is not allowed to switch from Edge-aligned mode to Center-aligned mode as long as the Counter is enabled (`CEN`=0x1). **Value After Reset:** 0x0
[4]	`DIR`	R/W	Direction **Values:** **0x0:** Counter used as upcounter **0x1:** Counter used as downcounter This bit is read only when the Timer is configured in Center-aligned mode or Encoder mode. **Value After Reset:** 0x0
[3]	`OPM`	R/W	One pulse mode **Values:** **0x0:** Counter is not stopped at the Update Event **0x1:** Counter stops counting at the next Update Event (clearing the `CEN` bit) **Value After Reset:** 0x0
[2]			Reserved (return 0 when read)
[1]	`UDIS`	R/W	Update disable This bit is set and cleared by software to enable/disable *Update Event* (**UEV**) generation. **Values:**

Table 12-6 Fields for register: PWMA_CR1 (continued)

Bits	Name	R/W	Description
			0x0: UEV enabled. The Update event is generated by one of the following events: ◦ Counter overflow/underflow ◦ Setting the PWMA_EGR.UG bit ◦ Update generation through the Slave Mode Controller **0x1:** UEV disabled. The Update event is not generated, shadow registers keep their value (ARR, PSC, CCRx). However the Counter and the Prescaler are reinitialized if the PWMA_EGR.UG bit is set or if a hardware reset is received from the Slave Mode Controller. **Value After Reset:** 0x0
[0]	CEN	R/W	Counter enable **Values:** **0x0:** Counter disabled **0x1:** Counter enabled • The external clock, gated mode and encoder mode can work only if the CEN bit has been previously set by software. However trigger mode can set the CEN bit automatically by hardware. • The CEN is reset by hardware in one-pulse mode. • The Counter starts counting 1 clock cycle after setting the CEN bit. **Value After Reset:** 0x0

12.1.2.2.2. PWMA Control Register 2 (PWMA_CR2)

The PWMA_CR2 register is at offset 0x04.

The following table shows the register bit assignments.

Table 12-7 Fields for register: PWMA_CR2

Bits	Name	R/W	Description
[15:14]			Reserved (return 0 when read)
[13]	OIS2N	R/W	Output Idle state Channel 2 (OC2N output) **Values:** **0x0:** OC2N=0 after a dead-time when PWMA_BDTR.MOE=0x0 **0x1:** OC2N=1 after a dead-time when PWMA_BDTR.MOE=0x0 If the settings of the OIS2 and OIS2N bits correspond to the active levels at the outputs (i.e. OIS2 = not CC2P and OIS2N = not CC2NP), then in the idle state both specified outputs will be set to inactive level (i.e. OC2 = CC2P and OC2N = CC2NP). It means that the outputs state may not match the specified ones. **Value After Reset:** 0x0

Table 12-7 Fields for register: PWMA_CR2 (continued)

Bits	Name	R/W	Description
[12]	OIS2	R/W	Output Idle state Channel 2 (OC2 output) **Values:** **0x0:** OC2=0 (after a dead-time if OC2N is implemented) when PWMA_BDTR.MOE=0x0 **0x1:** OC2=1 (after a dead-time if OC2N is implemented) when PWMA_BDTR.MOE=0x0 ⚠ Output state in the idle mode may not mach the OIS2 bit setting. For more information, see the OIS2N bit description. **Value After Reset:** 0x0
[11]	OIS1N	R/W	Output Idle state Channel 1 (OC1N output) **Values:** **0x0:** OC1N=0 after a dead-time when PWMA_BDTR.MOE=0x0 **0x1:** OC1N=1 after a dead-time when PWMA_BDTR.MOE=0x0 ⚠ If the settings of the OIS1 and OIS1N bits correspond to the active levels at the outputs (i.e. OIS1 = not CC1P and OIS1N = not CC1NP), then in the idle state both specified outputs will be set to inactive level (i.e. OC1 = CC1P and OC1N = CC1NP). It means that the outputs state may not match the specified ones. **Value After Reset:** 0x0
[10]	OIS1	R/W	Output Idle state Channel 1 (OC1 output) **Values:** **0x0:** OC1=0 (after a dead-time if OC1N is implemented) when PWMA_BDTR.MOE=0x0 **0x1:** OC1=1 (after a dead-time if OC1N is implemented) when PWMA_BDTR.MOE=0x0 ⚠ Output state in the idle mode may not mach the OIS1 bit setting. For more information, see the OIS1N bit description. **Value After Reset:** 0x0
[9]	OIS0N	R/W	Output Idle state Channel 0 (OC0N output) **Values:** **0x0:** OC0N=0 after a dead-time when PWMA_BDTR.MOE=0x0 **0x1:** OC0N=1 after a dead-time when PWMA_BDTR.MOE=0x0 ⚠ If the settings of the OIS0 and OIS0N bits correspond to the active levels at the outputs (i.e. OIS0 = not CC0P and OIS0N = not CC0NP), then in the idle state both specified outputs will be set to inactive level (i.e. OC0 = CC0P and OC0N = CC0NP). It means that the outputs state may not match the specified ones. **Value After Reset:** 0x0
[8]	OIS0	R/W	Output Idle state Channel 0 (OC0 output) **Values:**

Table 12-7 Fields for register: PWMA_CR2 (continued)

Bits	Name	R/W	Description
			0x0: OC0=0 (after a dead-time if OC0N is implemented) when PWMA_BDTR.MOE=0x0 **0x1:** OC1=1 (after a dead-time if OC1N is implemented) when PWMA_BDTR.MOE=0x0 ⚠ Output state in the idle mode may not mach the OIS0 bit setting. For more information, see the OIS0N bit description. **Value After Reset:** 0x0
[7]	TI0S	R/W	TI0 selection **Values:** **0x0:** The CH0P pin is connected to the TI0 input **0x1:** The CH0P, CH1P and CH2P pins are connected to the TI0 input (XOR combination; TI0 = CH0P xor CH1P xor CH2P) ⚠ There is no pin selection for other channels. If TIx, where x = 1 to 3, is used, the input is connected to the appropriate CHxP pin. **Value After Reset:** 0x0
[6:4]	MMS	R/W	Master mode selection These bits select the information to be sent in master mode to slave timers for synchronization (TRGO). **Values:** **0x0** (Reset): The UG bit from the PWMA_EGR register is used as trigger output (TRGO). If the reset is generated by the trigger input (when the Slave Mode Controller is configured in reset mode) then the signal on TRGO is delayed compared to the actual reset. **0x1** (Enable): The Counter Enable signal (CNT_EN) is used as trigger output (TRGO). It is useful to start several timers at the same time or to control the time window during which a slave timer is enabled. **0x2** (Update): The update event is selected as trigger output (TRGO). For example, a master timer can then be used as a prescaler for a slave timer. **0x3** (Compare Pulse): The trigger output sends a positive pulse when the PWMA_SR.CC0IF flag is set (even if it was already high), as soon as a capture or a compare match occurres. **0x4** (Compare): The OC0REF signal is used as trigger output (TRGO). **0x5** (Compare): The OC1REF signal is used as trigger output (TRGO). **0x6** (Compare): The OC2REF signal is used as trigger output (TRGO). **0x7** (Compare): The OC3REF signal is used as trigger output (TRGO).

Table 12-7 Fields for register: PWMA_CR2 (continued)

Bits	Name	R/W	Description
			The clock of the slave timer and ADC must be enabled prior to receiving events from the master timer, and must not be changed on-the-fly while triggers are received from the master timer. **Value After Reset:** 0x0
[3]			Reserved (return 0 when read)
[2]	CCUS	R/W	Capture/compare control update selection **Values:** **0x0:** When capture/compare control bits (CCxE, CCxNE and OCxM) are preloaded (CCPC=0x1), they are updated by setting the COMG bit only. **0x1:** When capture/compare control bits (CCxE, CCxNE and OCxM) are preloaded (CCPC=0x1), they are updated by setting the COMG bit or when an rising edge occurs on TRGI. This bit acts only on channels that have a complementary output. **Value After Reset:** 0x0
[1]			Reserved (return 0 when read)
[0]	CCPC	R/W	Capture/compare preloaded control **Values:** **0x0:** CCxE, CCxNE and OCxM bits are not preloaded. **0x1:** CCxE, CCxNE and OCxM bits are preloaded. After having been written, they are updated only when a *Commutation Event* (**COM**) occurs (PWMA_EGR.COMG bit set or rising edge detected on TRGI, depending on the CCUS bit). This bit acts only on channels that have a complementary output. **Value After Reset:** 0x0

12.1.2.2.3. PWMA Slave Mode Control Register (PWMA_SMCR)

The PWMA_SMCR register is at offset 0x08.

The following table shows the register bit assignments.

Table 12-8 Fields for register: PWMA_SMCR

Bits	Name	R/W	Description
[15]	ETP	R/W	External trigger polarity. This bit selects whether ETR or ETR (inverted ETR) is used for trigger operations. **Values:**

Table 12-8 Fields for register: PWMA_SMCR (continued)

Bits	Name	R/W	Description
			0x0: ETR is non-inverted, active at high level or rising edge. **0x1:** ETR is inverted, active at low level or falling edge. **Value After Reset:** 0x0
[14]	ECE	R/W	External clock enable. This bit enables External clock mode 2. **Values:** **0x0:** External clock mode 2 disabled. **0x1:** External clock mode 2 enabled. The counter is clocked by any active edge on the ETRF signal. 1. Setting the ECE bit has the same effect as selecting the external clock mode 1 with TRGI connected to ETRF (SMS=0x7 and TS=0x7). 2. The external clock mode 2 can be used simultaneously with the following slave modes: reset mode, gated mode and trigger mode. Nevertheless, TRGI must not be connected to ETRF in this case (TS bit field must not be set to 0x7). 3. If the external clock mode 1 and external clock mode 2 are enabled at the same time, the external clock input is ETRF. **Value After Reset:** 0x0
[13:12]	ETPS	R/W	External trigger prescaler The frequency of the ETRP external trigger signal must not exceed 1/4 of the timer clock frequency (CK_INT). A prescaler can be enabled to reduce the ETRP frequency. This is useful when inputting fast external clocks. **Values:** **0x0:** Prescaler OFF **0x1:** ETRP frequency divided by 2 **0x2:** ETRP frequency divided by 4 **0x3:** ETRP frequency divided by 8 **Value After Reset:** 0x0
[11:8]	ETF	R/W	External trigger filter This bit field defines both the sampling frequency for the ETRP signal and the length of the digital filter applied to ETRP. The digital filter consists of an event counter that requires N consecutive events to validate a transition at the output. **Values:** **0x0:** No filter, sampling is done at f_{DTS} **0x1:** $f_{SAMPLING}=f_{CK\_INT}$, N=2 **0x2:** $f_{SAMPLING}=f_{CK\_INT}$, N=4 **0x3:** $f_{SAMPLING}=f_{CK\_INT}$, N=8 **0x4:** $f_{SAMPLING}=f_{DTS}/2$, N=6 **0x5:** $f_{SAMPLING}=f_{DTS}/2$, N=8 **0x6:** $f_{SAMPLING}=f_{DTS}/4$, N=6

Table 12-8 Fields for register: `PWMA_SMCR` (continued)

Bits	Name	R/W	Description
			0x7: $f_{SAMPLING}=f_{DTS}/4$, N=8 **0x8:** $f_{SAMPLING}=f_{DTS}/8$, N=6 **0x9:** $f_{SAMPLING}=f_{DTS}/8$, N=8 **0xA:** $f_{SAMPLING}=f_{DTS}/16$, N=5 **0xB:** $f_{SAMPLING}=f_{DTS}/16$, N=6 **0xC:** $f_{SAMPLING}=f_{DTS}/16$, N=8 **0xD:** $f_{SAMPLING}=f_{DTS}/32$, N=5 **0xE:** $f_{SAMPLING}=f_{DTS}/32$, N=6 **0xF** $f_{SAMPLING}=f_{DTS}/32$, N=8 **Value After Reset:** 0x0
[7]			Reserved
[6:4]	`TS`	R/W	Trigger selection This bit field selects the trigger input to be used to synchronize the Counter. **Values:** **0x0:** Internal Trigger 0 (`ITR0`) **0x1:** Internal Trigger 1 (`ITR1`) **0x2:** Internal Trigger 2 (`ITR2`) **0x3:** Reserved **0x4:** TI1 Edge Detector (`TI0F_ED`) **0x5:** Filtered Timer Input 1 (`TI0FP0`) **0x6:** Filtered Timer Input 2 (`TI1FP1`) **0x7:** External Trigger input (`ETRF`) These bits must be changed only when they are not used (e.g. when `SMS`=0x0) to avoid wrong edge detections at the transition. **Value After Reset:** 0x0
[3]			Reserved
[2:0]	`SMS`	R/W	Slave mode selection When external signals are selected, the active edge of the trigger signal (`TRGI`) is linked to the polarity selected on the external input. **Values:** **0x0** (Slave mode disabled): If `PWMA_CR1.CEN` = 0x1, then the Prescaler is clocked directly by the internal clock. **0x1:** Reserved **0x2:** Reserved **0x3:** Reserved **0x4** (Reset Mode): The rising edge of the selected trigger input (`TRGI`) reinitializes the Counter and generates an update of the registers.

Table 12-8 Fields for register: `PWMA_SMCR` (continued)

Bits	Name	R/W	Description
			0x5 (Gated Mode): The Counter clock is enabled when the trigger input (`TRGI`) is high. The Counter stops (but is not reset) as soon as the trigger becomes low. Both start and stop of the Counter are controlled. **0x6** (Trigger Mode): The Counter starts at a rising edge of the trigger `TRGI` (but it is not reset). Only the start of the Counter is controlled. **0x7** (External Clock Mode 1): The rising edges of the selected trigger (`TRGI`) clock the Counter. The gated mode must not be used if `TI0F_ED` is selected as the trigger input (`TS`=0x4). Indeed, `TI0F_ED` outputs 1 pulse for each transition on `TI0F`, whereas the gated mode checks the level of the trigger signal. **Value After Reset:** 0x0

12.1.2.2.4. PWMA DMA/Interrupt Enable Register (`PWMA_DIER`)

The `PWMA_DIER` register is at offset 0x0C.

The following table shows the register bit assignments.

Table 12-9 Fields for register: `PWMA_DIER`

Bits	Name	R/W	Description
[15]	`INT_CLEAR`	W	Interrupt clear Write 0x1 to this bit to reset the setted interrupt. Writing 0x0 has no effect. **Value After Reset:** 0x0
[14]	`TDE`	R/W	Trigger DMA request enable **Values:** **0x0:** Trigger DMA request disabled **0x1:** Trigger DMA request enabled **Value After Reset:** 0x0
[13]	`COMDE`	R/W	COM DMA request enable **Values:** **0x0:** COM DMA request disabled **0x1:** COM DMA request enabled **Value After Reset:** 0x0
[12]	`CC3DE`	R/W	Capture/Compare Channel 3 DMA request enable **Values:** **0x0:** CC3 DMA request disabled **0x1:** CC3 DMA request enabled **Value After Reset:** 0x0

Table 12-9 Fields for register: `PWMA_DIER` (continued)

Bits	Name	R/W	Description
[11]	`CC2DE`	R/W	Capture/Compare Channel 2 DMA request enable **Values:** **0x0:** CC2 DMA request disabled **0x1:** CC2 DMA request enabled **Value After Reset:** 0x0
[10]	`CC1DE`	R/W	Capture/Compare Channel 1 DMA request enable **Values:** **0x0:** CC1 DMA request disabled **0x1:** CC1 DMA request enabled **Value After Reset:** 0x0
[9]	`CC0DE`	R/W	Capture/Compare Channel 0 DMA request enable **Values:** **0x0:** CC0 DMA request disabled **0x1:** CC0 DMA request enabled **Value After Reset:** 0x0
[8]	`UDE`	R/W	Update DMA request enable **Values:** **0x0:** Update DMA request disabled **0x1:** Update DMA request enabled **Value After Reset:** 0x0
[7]	`BIE`	R/W	Break interrupt enable **Values:** **0x0:** Break interrupt disabled **0x1:** Break interrupt enabled **Value After Reset:** 0x0
[6]	`TIE`	R/W	Trigger interrupt enable **Values:** **0x0:** Trigger interrupt disabled **0x1:** Trigger interrupt enabled **Value After Reset:** 0x0
[5]	`COMIE`	R/W	COM interrupt enable **Values:** **0x0:** COM interrupt disabled **0x1:** COM interrupt enabled **Value After Reset:** 0x0
[4]	`CC3IE`	R/W	Capture/Compare Channel 3 interrupt enable **Values:**

Table 12-9 Fields for register: `PWMA_DIER` (continued)

Bits	Name	R/W	Description
			0x0: CC3 interrupt disabled **0x1:** CC3 interrupt enabled **Value After Reset:** 0x0
[3]	`CC2IE`	R/W	Capture/Compare Channel 2 interrupt enable **Values:** **0x0:** CC2 interrupt disabled **0x1:** CC2 interrupt enabled **Value After Reset:** 0x0
[2]	`CC1IE`	R/W	Capture/Compare Channel 1 interrupt enable **Values:** **0x0:** CC1 interrupt disabled **0x1:** CC1 interrupt enabled **Value After Reset:** 0x0
[1]	`CC0IE`	R/W	Capture/Compare Channel 0 interrupt enable **Values:** **0x0:** CC0 interrupt disabled **0x1:** CC0 interrupt enabled **Value After Reset:** 0x0
[0]	`UIE`	R/W	Update interrupt enable **Values:** **0x0:** Update interrupt disabled **0x1:** Update interrupt enabled **Value After Reset:** 0x0

12.1.2.2.5. PWMA Status Register (`PWMA_SR`)

The `PWMA_SR` register is at offset 0x10.

The following table shows the register bit assignments.

Table 12-10 Fields for register: `PWMA_SR`

Bits	Name	R/W	Description
[15:13]			Reserved
[12]	`CC3OF`	RW0C	Capture/Compare Channel 3 overcapture flag This flag is set by hardware only when the corresponding channel is configured in input capture mode. It is cleared by software by writing it to 0x0. **Values:**

Table 12-10 Fields for register: PWMA_SR (continued)

Bits	Name	R/W	Description
			0x0: No overcapture has been detected **0x1:** The counter value has been captured in PWMA_CCR3 register while CC3IF flag was already set **Value After Reset:** 0x0
[11]	CC2OF	RW0C	Capture/Compare Channel 2 overcapture flag This flag is set by hardware only when the corresponding channel is configured in input capture mode. It is cleared by software by writing it to 0x0. **Values:** **0x0:** No overcapture has been detected **0x1:** The counter value has been captured in PWMA_CCR2 register while CC2IF flag was already set **Value After Reset:** 0x0
[10]	CC1OF	RW0C	Capture/Compare Channel 1 overcapture flag This flag is set by hardware only when the corresponding channel is configured in input capture mode. It is cleared by software by writing it to 0x0. **Values:** **0x0:** No overcapture has been detected **0x1:** The counter value has been captured in PWMA_CCR1 register while CC1IF flag was already set **Value After Reset:** 0x0
[9]	CC0OF	RW0C	Capture/Compare Channel 0 overcapture flag This flag is set by hardware only when the corresponding channel is configured in input capture mode. It is cleared by software by writing it to 0x0. **Values:** **0x0:** No overcapture has been detected **0x1:** The counter value has been captured in PWMA_CCR0 register while CC0IF flag was already set **Value After Reset:** 0x0
[8]			Reserved
[7]	BIF	RW0C	Break interrupt flag This flag is set by hardware as soon as the break input goes active. It can be cleared by software if the break input is not active. **Values:** **0x0:** No break event occurred **0x1:** An active level has been detected on the break input **Value After Reset:** 0x0
[6]	TIF	RW0C	Trigger interrupt flag

Table 12-10 Fields for register: PWMA_SR (continued)

Bits	Name	R/W	Description
			This flag is set by hardware on a trigger event (when an active edge is detected on the TRGI input), when the Slave Mode Controller is enabled in all modes except for a gated mode. The flag is set when the Counter starts or stops when the gated mode is selected. It is cleared by software. **Values:** **0x0:** No trigger event occurred **0x1:** Trigger interrupt pending **Value After Reset:** 0x0
[5]	COMIF	RW0C	COM interrupt flag This flag is set by hardware on COM event (when Capture/compare Control bits (CCxE, CCxNE, OCxM) have been updated). It is cleared by software. **Values:** **0x0:** No COM event occurred **0x1:** COM interrupt pending **Value After Reset:** 0x0
[4]	CC3IF	RW0C	Capture/Compare Channel 3 interrupt flag **If channel CC3 is configured as output** (PWMA_CCMR2.CC3S = 0x0): This flag is set by hardware when the counter matches the compare value, with some exception in center-aligned mode (refer to the CMS bits in the PWMA_CR1 register description). It is cleared by software. **Values:** **0x0:** No match **0x1:** The value in the PWMA_CNT matches the value in the PWMA_CCR3 register. When the value in PWMA_CCR3 is greater than the value in the PWMA_ARR, the CC3IF bit goes high on a counter overflow (in upcounting and up/down-counting modes) or underflow (in downcounting mode). **If channel CC3 is configured as input** (PWMA_CCMR2.CC3S != 0x0): This bit is set by hardware on a capture. It is cleared by software or by reading the PWMA_CCR3 register. **Values:** **0x0:** No input capture occurred **0x1:** The counter value has been captured in PWMA_CCR3 register (An edge has been detected on IC3 which matches the selected polarity). **Value After Reset:** 0x0
[3]	CC2IF	RW0C	Capture/Compare Channel 2 interrupt flag **If channel CC2 is configured as output** (PWMA_CCMR2.CC2S = 0x0): This flag is set by hardware when the counter matches the compare value, with some exception in center-aligned mode (refer to the CMS bits in the PWMA_CR1 register description). It is cleared by software.

Table 12-10 Fields for register: PWMA_SR (continued)

Bits	Name	R/W	Description
			Values: **0x0:** No match **0x1:** The value in the PWMA_CNT matches the value in the PWMA_CCR2 register. When the value in PWMA_CCR2 is greater than the value in the PWMA_ARR, the CC2IF bit goes high on a counter overflow (in upcounting and up/down-counting modes) or underflow (in downcounting mode). **If channel CC2 is configured as input** (PWMA_CCMR2.CC2S != 0x0): This bit is set by hardware on a capture. It is cleared by software or by reading the PWMA_CCR2 register. **Values:** **0x0:** No input capture occurred **0x1:** The counter value has been captured in PWMA_CCR2 register (An edge has been detected on IC2 which matches the selected polarity). **Value After Reset:** 0x0
[2]	CC1IF	RW0C	Capture/Compare Channel 1 interrupt flag **If channel CC1 is configured as output** (PWMA_CCMR1.CC1S = 0x0): This flag is set by hardware when the Counter matches the compare value, with some exception in center-aligned mode (refer to the CMS bits in the PWMA_CR1 register description). It is cleared by software. **Values:** **0x0:** No match **0x1:** The value in the PWMA_CNT matches the value in the PWMA_CCR1 register. When the value in PWMA_CCR1 is greater than the value in the PWMA_ARR, the CC1IF bit goes high on a counter overflow (in upcounting and up/down-counting modes) or underflow (in downcounting mode). **If channel CC1 is configured as input** (PWMA_CCMR1.CC1S != 0x0): This bit is set by hardware on a capture. It is cleared by software or by reading the PWMA_CCR1 register. **Values:** **0x0:** No input capture occurred **0x1:** The counter value has been captured in PWMA_CCR1 register (An edge has been detected on IC1 which matches the selected polarity). **Value After Reset:** 0x0
[1]	CC0IF	RW0C	Capture/Compare Channel 0 interrupt flag **If channel CC0 is configured as output** (PWMA_CCMR1.CC0S = 0x0): This flag is set by hardware when the Counter matches the compare value, with some exception in center-aligned mode (refer to the CMS bits in the PWMA_CR1 register description). It is cleared by software. **Values:**

Table 12-10 Fields for register: PWMA_SR (continued)

Bits	Name	R/W	Description
			0x0: No match **0x1:** The value in the PWMA_CNT matches the value in the PWMA_CCR0 register. When the value in PWMA_CCR0 is greater than the value in the PWMA_ARR, the CC0IF bit goes high on a counter overflow (in upcounting and up/down-counting modes) or underflow (in downcounting mode). **If channel CC0 is configured as input** (PWMA_CCMR1.CC0S != 0x0): This bit is set by hardware on a capture. It is cleared by software or by reading the PWMA_CCR0 register. **Values:** **0x0:** No input capture occurred **0x1:** The Counter value has been captured in PWMA_CCR0 register (An edge has been detected on IC0 which matches the selected polarity). **Value After Reset:** 0x0
[0]	UIF	RW0C	Update interrupt flag This bit is set by hardware on an update event. It is cleared by software. **Values:** **0x0:** No update occurred **0x1:** Update interrupt pending. This bit is set by hardware when the registers are updated: ◦ At overflow or underflow regarding the repetition counter value (update if repetition counter = 0), if UDIS=0x0 in the PWMA_CR1 register. ◦ When CNT is reinitialized by software using the UG bit in PWMA_EGR register, if UDIS=0x0 in the PWMA_CR1 register. ◦ When CNT is reinitialized by a trigger event (see also PWMA_SMCR), if UDIS=0x0 in the PWMA_CR1 register. **Value After Reset:** 0x0

12.1.2.2.6. PWMA Event Generation Register (PWMA_EGR)

The PWMA_EGR register is at offset 0x14.

The following table shows the register bit assignments.

Table 12-11 Fields for register: PWMA_EGR

Bits	Name	R/W	Description
[15:8]			Reserved
[7]	BG	W	Break generation This bit is set by software to generate an event and is automatically cleared by hardware. **Values:**

Table 12-11 Fields for register: PWMA_EGR (continued)

Bits	Name	R/W	Description
			0x0: No action **0x1:** A break event is generated. PWMA_BDTR.MOE bit is cleared and PWMA_SR.BIF flag is set. Related interrupt or DMA transfer can occur if enabled. **Value After Reset:** 0x0
[6]	TG	W	Trigger generation This bit is set by software to generate an event and is automatically cleared by hardware. **Values:** **0x0:** No action **0x1:** The TIF flag is set in PWMA_SR register. Related interrupt or DMA transfer can occur if enabled. **Value After Reset:** 0x0
[5]	COMG	W	Capture/Compare control update generation This bit can be set by software. It is automatically cleared by hardware. **Values:** **0x0:** No action **0x1:** When PWMA_CR2.CCPC bit is set, it allows to update the CCxE, CCxNE and OCxM bits. This bit acts only on channels having a complementary output. **Value After Reset:** 0x0
[4]	CC3G	W	Capture/Compare 3 generation This bit is set by software to generate an event and is automatically cleared by hardware. **Values:** **0x0:** No action **0x1:** A capture/compare event is generated on channel 3: **If channel CC3 is configured as output** (PWMA_CCMR2.CC3S = 0x0): The PWMA_SR.CC3IF flag is set. The corresponding interrupt or DMA request is sent if enabled. **If channel CC3 is configured as input** (PWMA_CCMR2.CC3S != 0x0): The current value of the Counter is captured in PWMA_CCR3 register. The PWMA_SR.CC3IF flag is set, the corresponding interrupt or DMA request is sent if enabled. The PWMA_SR.CC3OF flag is set if the CC3IF flag was already high. **Value After Reset:** 0x0
[3]	CC2G	W	Capture/Compare 2 generation This bit is set by software to generate an event and is automatically cleared by hardware.

Table 12-11 Fields for register: PWMA_EGR (continued)

Bits	Name	R/W	Description
			Values: **0x0:** No action **0x1:** A capture/compare event is generated on channel 2: **If channel CC2 is configured as output** (PWMA_CCMR2.CC2S = 0x0): The PWMA_SR.CC2IF flag is set. The corresponding interrupt or DMA request is sent if enabled. **If channel CC2 is configured as input** (PWMA_CCMR2.CC2S != 0x0): The current value of the Counter is captured in PWMA_CCR2 register. The PWMA_SR.CC2IF flag is set, the corresponding interrupt or DMA request is sent if enabled. The PWMA_SR.CC2OF flag is set if the CC2IF flag was already high. **Value After Reset:** 0x0
[2]	CC1G	W	Capture/Compare 1 generation This bit is set by software to generate an event and is automatically cleared by hardware. **Values:** **0x0:** No action **0x1:** A capture/compare event is generated on channel 1: **If channel CC1 is configured as output** (PWMA_CCMR1.CC1S = 0x0): The PWMA_SR.CC1IF flag is set. The corresponding interrupt or DMA request is sent if enabled. **If channel CC1 is configured as input** (PWMA_CCMR1.CC1S != 0x0): The current value of the Counter is captured in PWMA_CCR1 register. The PWMA_SR.CC1IF flag is set, the corresponding interrupt or DMA request is sent if enabled. The PWMA_SR.CC1OF flag is set if the CC1IF flag was already high. **Value After Reset:** 0x0
[1]	CC0G	W	Capture/Compare 0 generation This bit is set by software to generate an event and is automatically cleared by hardware. **Values:** **0x0:** No action **0x1:** A capture/compare event is generated on channel 0: **If channel CC0 is configured as output** (PWMA_CCMR1.CC0S = 0x0): The PWMA_SR.CC0IF flag is set. The corresponding interrupt or DMA request is sent if enabled. **If channel CC0 is configured as input** (PWMA_CCMR1.CC0S != 0x0):

Table 12-11 Fields for register: PWMA_EGR (continued)

Bits	Name	R/W	Description
			The current value of the Counter is captured in PWMA_CCR0 register. The PWMA_SR.CC0IF flag is set, the corresponding interrupt or DMA request is sent if enabled. The PWMA_SR.CC0OF flag is set if the CC0IF flag was already high. **Value After Reset:** 0x0
[0]	UG	W	Update generation This bit can be set by software. It is automatically cleared by hardware. **Values:** **0x0:** No action **0x1:** Reinitialize the counter and generates an update of the registers. Note that the prescaler counter is cleared too (anyway the prescaler ratio is not affected). The counter is cleared if the center-aligned mode is selected or if PWMA_CR1.DIR=0x0 (upcounting), else it takes the auto-reload value (PWMA_ARR) if PWMA_CR1.DIR=0x1 (downcounting). **Value After Reset:** 0x0

12.1.2.2.7. PWMA Capture/Compare Mode Register 1 (PWMA_CCMR1)

The PWMA_CCMR1 register is at offset 0x18.

⚠ The channels can be configured in either input (capture) or output (compare) mode. The direction of the channel is defined by configuring the CCxS bit field. The other bits in this register have different functions depending on the mode (input/output). For a given bit, OCxx describes its function when the channel 0 is configured as output, while ICxx describes describes its function when the channel is configured as input. Therefore, the user must be aware that the same bit can have a different meaning for the input stage and for the output stage.

The following table shows the register bit assignments.

Table 12-12 Fields for register: PWMA_CCMR1

Bits	Name	R/W	Description
[15:10]	IC1* or OC1*	R/W	⚠ The functions of these bits depend on the value of the CC1S bit field. For details, see the table **PWMA_CCMR1[15:10] bits functions for different CC1S values** below.
[9:8]	CC1S	R/W	Capture/Compare Channel 1 selection This bit field defines the direction of the channel 1 (input/output) as well as the used input. **Values:** **0x0:** CC1 channel is configured as output **0x1:** CC1 channel is configured as input, IC1 is mapped on TI1 **0x2:** CC1 channel is configured as input, IC1 is mapped on TI0 **0x3:** CC1 channel is configured as input, IC1 is mapped on TRC This mode works only if an internal trigger input is selected via the TS bit (PWMA_SMCR register).

Table 12-12 Fields for register: `PWMA_CCMR1` (continued)

Bits	Name	R/W	Description
			`CC1S` bits are writable only when the channel is OFF (`CC1E` = 0x0 in `PWMA_CCER`) **Value After Reset:** 0x0
[7:2]	`IC0*` or `OC0*`	R/W	The functions of these bits depend on the value of the `CC0S` bit field. For details, see the table **`PWMA_CCMR1[7:2]` bits functions for different `CC0S` values** below.
[1:0]	`CC0S`	R/W	Capture/Compare Channel 0 selection This bit field defines the direction of the channel 0 (input/output) as well as the used input. **Values:** **0x0:** CC0 channel is configured as output **0x1:** CC0 channel is configured as input, `IC0` is mapped on `TI0` **0x2:** CC0 channel is configured as input, `IC0` is mapped on `TI1` **0x3:** CC0 channel is configured as input, `IC0` is mapped on `TRC` This mode works only if an internal trigger input is selected via the `TS` bit (`PWMA_SMCR` register). `CC0S` bits are writable only when the channel is OFF (`CC0E` = 0x0 in `PWMA_CCER`) **Value After Reset:** 0x0

The table below describes the `PWMA_CCMR1[15:10]` bits functions for different `PWMA_CCMR1.CC1S` bit values

Table 12-13 `PWMA_CCMR1[15:10]` bits functions for different `CC1S` values

Bits	Name	R/W	Description
Channel 1 Output Compare mode (`CC1S` = 0x0)			
[15]	`OC1CE`	R/W	Output Compare Channel 1 clear enable **Values:** **0x0:** `OC1REF` is not affected by the `ETRF` input. **0x1:** `OC1REF` is cleared as soon as a High level is detected on the `ETRF` input. **Value After Reset:** 0x0
[14:12]	`OC1M`	R/W	Output Compare Channel 1 mode These bits define the behavior of the output reference signal `OC1REF`, from which the `OC1` and `OC1N` are derived. The `OC1REF` is active high, whereas the active levels of `OC1` and `OC1N` depend on `CC1P` and `CC1NP` bits. The possible values of this bit are the same as the `OC0M` bit.

Table 12-13 `PWMA_CCMR1[15:10]` bits functions for different `CC1S` values (continued)

Bits	Name	R/W	Description
			• If `OC1M` bit is set to 0x6 or 0x7, it is not recommended to set the `PWMA_CCR1` register to 0x0. If these values are written, the timer output will not give a 100% duty cycle, and, when the counter value is equal to zero, a short pulse will be emitted on each counter period. • It is recommended to set the `OC1M` bit to 0x4 or 0x5 to get a 100% duty cycle. **Value After Reset:** 0x0
[11]	`OC1PE`	R/W	Output Compare Channel 1 preload enable **Values:** **0x0:** Preload register for `PWMA_CCR1` disabled. `PWMA_CCR1` can be written at anytime, the new value is taken into account immediately. **0x1:** Preload register on `PWMA_CCR1` enabled. Read/Write operations access the preload register. `PWMA_CCR1` preload value is loaded into the active register at each *Update Event* (**UEV**). The PWM mode can be used without validating the preload register only in one pulse mode (`OPM` bit set in `PWMA_CR1` register). Otherwise, the behavior is not guaranteed. **Value After Reset:** 0x0
[10]			Reserved
Channel 1 Input Capture mode (`CC1S` != 0x0)			
[15:12]	`IC1F`	R/W	Input Capture Channel 1 filter This bit field defines both the sampling frequency for the `TI1` input and the length of the digital filter applied to `TI1`. The digital filter consists of an event counter that requires N consecutive events to validate a transition at the output. The possible values of this bit are the same as the `IC0F` bit. **Value After Reset:** 0x0
[11:10]	`IC1PSC`	R/W	Input Capture Channel 1 prescaler This bit field defines the Prescaler ratio for the CC1 input (`IC1`). The Prescaler is reset as soon as `CC1E`=0x0 (`PWMA_CCER` register). **Values:** **0x0:** No prescaler, capture is done each time an edge is detected on the capture input **0x1:** Capture is done once every 2 events **0x2:** Capture is done once every 4 events **0x3:** Capture is done once every 8 events **Value After Reset:** 0x0

`PWMA_CCMR1[7:2]` bit functions for different `PWMA_CCMR1.CC0S` bit field values are shown in the table below.

Table 12-14 PWMA_CCMR1[7:2] bits functions for different CC0S values

Bits	Name	R/W	Description
Channel 0 Output Compare mode (CC0S = 0x0)			
[7]	OC0CE	R/W	Output Compare Channel 0 clear enable **Values:** **0x0:** OC0REF is not affected by the ETRF input **0x1:** OC0REF is cleared as soon as a High level is detected on ETRF input **Value After Reset:** 0x0
[6:4]	OC0M	R/W	Output Compare Channel 0 mode These bits define the behavior of the output reference signal OC0REF, from which the OC0 and OC0N are derived. The OC0REF is active high, whereas the active levels of OC0 and OC0N depend on CC0P and CC0NP bits. **Values:** **0x0** (Frozen): The comparison between the output compare register PWMA_CCR0 and the counter PWMA_CNT has no effect on the outputs. (This mode is used to generate a timing base). **0x1:** Set channel 0 to active level on match. OC0REF signal is forced high when the counter PWMA_CNT matches the PWMA_CCR0 register. **0x2:** Set channel 0 to inactive level on match. OC0REF signal is forced low when the counter PWMA_CNT matches the PWMA_CCR0 register. **0x3** (Toggle): The OC0REF toggles when PWMA_CNT=PWMA_CCR0. **0x4** (Force inactive level): The OC0REF is forced low. **0x5** (Force active level): The OC0REF is forced high. **0x6** (PWM mode 1): In upcounting, the channel 0 is active as long as PWMA_CNT<PWMA_CCR0. Otherwise, it is inactive. In downcounting, the channel 0 is inactive (OC0REF='0') as long as PWMA_CNT>PWMA_CCR0. Otherwise, it is active (OC0REF='1'). **0x7** (PWM mode 2): In upcounting, the channel 0 is inactive as long as PWMA_CNT<PWMA_CCR0. Otherwise, it is active. In downcounting, the channel 0 is active as long as PWMA_CNT>PWMA_CCR0. Otherwise, it is inactive. 1. In PWM mode 1 or 2, the OCR0EF level changes only when the result of the comparison changes or when the output compare mode switches from "frozen" mode to "PWM" mode. 2. On channels having a complementary output, this bit field is preloaded. If the CCPC bit is set in the PWMA_CR2 register then the OC0M active bits take the new value from the preloaded bits only when a COM event is generated. • If OC0M bit is set to 0x6 or 0x7, it is not recommended to set the PWMA_CCR0 register to 0x0. If these values are written, the timer output will not give a 100% duty cycle, and, when the

Table 12-14 `PWMA_CCMR1[7:2]` bits functions for different `CC0S` values (continued)

Bits	Name	R/W	Description
			counter value is equal to zero, a short pulse will be emitted on each counter period. • It is recommended to set the `OC0M` bit to 0x4 or 0x5 to get a 100% duty cycle. **Value After Reset:** 0x0
[3]	`OC0PE`	R/W	Output Compare Channel 0 preload enable **Values:** **0x0:** Preload register for `PWMA_CCR0` disabled. `PWMA_CCR0` can be written at anytime, the new value is taken into account immediately. **0x1:** Preload register for `PWMA_CCR0` enabled. Read/Write operations access the preload register. `PWMA_CCR0` preload value is loaded into the active register at each update event. The PWM mode can be used without validating the preload register only in one pulse mode (`OPM` bit set in `PWMA_CR1` register). Otherwise, the behavior is not guaranteed. **Value After Reset:** 0x0
[2]			Reserved
Channel 0 Input Capture mode (`CC0S` != 0x0)			
[7:4]	`IC0F`	R/W	Input Capture Channel 0 filter This bit field defines both the sampling frequency for the `TI0` input and the length of the digital filter applied to `TI0`. The digital filter consists of an event counter that requires N consecutive events to validate a transition at the output. **Values:** **0x0:** No filter, sampling is done at f_{DTS} ($f_{SAMPLING}=f_{DTS}$) **0x1:** $f_{SAMPLING}=f_{CK\_INT}$, N=2 **0x2:** $f_{SAMPLING}=f_{CK\_INT}$, N=4 **0x3:** $f_{SAMPLING}=f_{CK\_INT}$, N=8 **0x4:** $f_{SAMPLING}=f_{DTS}/2$, N=6 **0x5:** $f_{SAMPLING}=f_{DTS}/2$, N=8 **0x6:** $f_{SAMPLING}=f_{DTS}/4$, N=6 **0x7:** $f_{SAMPLING}=f_{DTS}/4$, N=8 **0x8:** $f_{SAMPLING}=f_{DTS}/8$, N=6 **0x9:** $f_{SAMPLING}=f_{DTS}/8$, N=8 **0xA:** $f_{SAMPLING}=f_{DTS}/16$, N=5 **0xB:** $f_{SAMPLING}=f_{DTS}/16$, N=6 **0xC:** $f_{SAMPLING}=f_{DTS}/16$, N=8 **0xD:** $f_{SAMPLING}=f_{DTS}/32$, N=5 **0xE:** $f_{SAMPLING}=f_{DTS}/32$, N=6 **0xF:** $f_{SAMPLING}=f_{DTS}/32$, N=8 **Value After Reset:** 0x0
[3:2]	`IC0PSC`	R/W	Input Capture Channel 0 prescaler

Table 12-14 `PWMA_CCMR1[7:2]` bits functions for different `CC0S` values (continued)

Bits	Name	R/W	Description
			This bit field defines the Prescaler ratio for the CC0 input (`IC0`). The Prescaler is reset as soon as `CC0E`=0x0 (`PWMA_CCER` register). **Values:** **0x0:** No prescaler, capture is done each time an edge is detected on the capture input **0x1:** Capture is done once every 2 events **0x2:** Capture is done once every 4 events **0x3:** Capture is done once every 8 events **Value After Reset:** 0x0

12.1.2.2.8. PWMA Capture/Compare Mode Register 2 (`PWMA_CCMR2`)

The `PWMA_CCMR2` register is at offset 0x1C.

The channels can be configured in either input (capture) or output (compare) mode. The direction of the channel is defined by configuring the `CCxS` bit field. The other bits in this register have different functions depending on the mode (input/output). For a given bit, `OCxx` describes its function when the channel 0 is configured as output, while `ICxx` describes its function when the channel is configured as input. Therefore, the user must be aware that the same bit can have a different meaning for the input stage and for the output stage.

The following table shows the register bit assignments.

Table 12-15 Fields for register: `PWMA_CCMR2`

Bits	Name	R/W	Description

[15:10]	`IC3*`
	or
	`OC3*`
		R/W	The functions of these bits depend on the value of the `CC3S` bit field. For details, see the table **`PWMA_CCMR2[15:10]` bits functions for different `CC3S` values** below.

[9:8]	`CC3S`	R/W	Capture/Compare Channel 3 selection This bit field defines the direction of the channel 3 (input/output) as well as the used input. **Values:** **0x0:** CC3 channel is configured as output **0x1:** CC3 channel is configured as input, `IC3` is mapped on `TI3` **0x2:** CC3 channel is configured as input, `IC3` is mapped on `TI2` **0x3:** CC3 channel is configured as input, `IC3` is mapped on `TRC` This mode works only if an internal trigger input is selected through the `TS` bit (`PWMA_SMCR` register). The `CC3S` bits are writable only when the channel is OFF (`CC3E` = 0x0 in `PWMA_CCER`). **Value After Reset:** 0x0

[7:2]	`IC2*`
	or
	`OC2*`
		R/W	The functions of these bits depend on the value of the `CC2S` bit field. For details, see the table **`PWMA_CCMR2[7:2]` bits functions for different `CC2S` values** below.

Table 12-15 Fields for register: PWMA_CCMR2 (continued)

Bits	Name	R/W	Description
[1:0]	CC2S	R/W	Capture/Compare Channel 2 selection This bit field defines the direction of the channel 2 (input/output) as well as the used input. **Values:** **0x0:** CC2 channel is configured as output **0x1:** CC2 channel is configured as input, IC2 is mapped on TI2 **0x2:** CC2 channel is configured as input, IC2 is mapped on TI3 **0x3:** CC2 channel is configured as input, IC2 is mapped on TRC This mode works only if an internal trigger input is selected through the TS bit (PWMA_SMCR register). The CC2S bits are writable only when the channel is OFF (CC2E = 0x0 in PWMA_CCER). **Value After Reset:** 0x0

The table below describes the PWMA_CCMR2[15:10] bits functions for different PWMA_CCMR2.CC3S bit values

Table 12-16 PWMA_CCMR2[15:10] bits functions for different CC3S values

Bits	Name	R/W	Description
Channel 3 Output Compare mode (CC3S = 0x0)			
[15]	OC3CE	R/W	Output Compare Channel 3 clear enable **Values:** **0x0:** OC3REF is not affected by the ETRF input **0x1:** OC3REF is cleared as soon as a High level is detected on ETRF input **Value After Reset:** 0x0
[14:12]	OC3M	R/W	Output Compare Channel 3 mode These bits define the behavior of the output reference signal OC3REF, from which the OC3 is derived. The OC3REF is active high, whereas OC3 active level depends on CC3P bit. The possible values of this bit are the same as the values of the OC2M bit. • If OC3M bit is set to 0x6 or 0x7, it is not recommended to set the PWMA_CCR3 register to 0x0. If these values are written, the timer output will not give a 100% duty cycle, and, when the counter value is equal to zero, a short pulse will be emitted on each counter period. • It is recommended to set the OC3M bit to 0x4 or 0x5 to get a 100% duty cycle. **Value After Reset:** 0x0
[11]	OC3PE	R/W	Output Compare Channel 3 preload enable **Values:**

Table 12-16 PWMA_CCMR2[15:10] bits functions for different CC3S values (continued)

Bits	Name	R/W	Description
			0x0: Preload register for PWMA_CCR3 disabled. PWMA_CCR3 can be written at anytime, the new value is taken into account immediately. **0x1:** Preload register for PWMA_CCR3 enabled. Read/Write operations access the preload register. PWMA_CCR3 preload value is loaded into the active register at each update event. The PWM mode can be used without validating the preload register only in one pulse mode (OPM bit set in PWMA_CR1 register). Otherwise, the behavior is not guaranteed. **Value After Reset:** 0x0
[10]			Reserved
Channel 3 Input Capture mode (CC3S != 0x0)			
[15:12]	IC3F	R/W	Input Capture Channel 3 filter This bit field defines both the sampling frequency for the TI3 input and the length of the digital filter applied to TI3. The digital filter consists of an event counter that requires N consecutive events to validate a transition at the output. The possible values of this bit are the same as the IC2F bit. **Value After Reset:** 0x0
[11:10]	IC3PSC	R/W	Input Capture Channel 3 prescaler This bit field defines the Prescaler ratio for the CC3 input (IC3). The Prescaler is reset as soon as CC3E=0x0 (PWMA_CCER register). **Values:** **0x0:** No prescaler, capture is done each time an edge is detected on the capture input **0x1:** Capture is done once every 2 events **0x2:** Capture is done once every 4 events **0x3:** Capture is done once every 8 events **Value After Reset:** 0x0

PWMA_CCMR2[7:2] bit functions for different PWMA_CCMR2.CC2S bit field values are shown in the table below.

Table 12-17 PWMA_CCMR2[7:2] bits functions for different CC2S values

Bits	Name	R/W	Description
Channel 2 Output Compare mode (CC2S = 0x0)			
[7]	OC2CE	R/W	Output Compare Channel 2 clear enable **Values:** **0x0:** OC2REF is not affected by the ETRF input **0x1:** OC2REF is cleared as soon as a High level is detected on ETRF input **Value After Reset:** 0x0

Table 12-17 PWMA_CCMR2[7:2] bits functions for different CC2S values (continued)

Bits	Name	R/W	Description
[6:4]	OC2M	R/W	Output Compare Channel 2 mode These bits define the behavior of the output reference signal OC2REF, from which the OC2 and OC2N are derived. The OC2REF is active high, whereas the active levels of OC2 and OC2N depend on CC2P and CC2NP bits. **Values:** **0x0** (Frozen): The comparison between the output compare register PWMA_CCR2 and the counter PWMA_CNT has no effect on the outputs. (This mode is used to generate a timing base). **0x1:** Set channel 2 to active level on match. OC2REF signal is forced high when the counter PWMA_CNTmatches the PWMA_CCR2 register. **0x2:** Set channel 2 to inactive level on match. OC2REF signal is forced low when the counter PWMA_CNT matches the PWMA_CCR2 register.. **0x3** (Toggle): The OC2REF toggles when PWMA_CNT=PWMA_CCR2. **0x4** (Force inactive level): The OC2REF is forced low. **0x5** (Force active level): The OC2REF is forced high. **0x6** (PWM mode 1): In upcounting, the channel 2 is active as long as PWMA_CNT<PWMA_CCR2. Otherwise, it is inactive. In downcounting, the channel 2 is inactive (OC2REF=‘0’) as long as PWMA_CNT>PWMA_CCR2. Otherwise, it is active (OC2REF=’1’). **0x7** (PWM mode 2): In upcounting, the channel 2 is inactive as long as PWMA_CNT<PWMA_CCR2. Otherwise, it is active. In downcounting, the channel 2 is active as long as PWMA_CNT>PWMA_CCR2. Otherwise, it is inactive.  1. In PWM mode 1 or 2, the OCR2EF level changes only when the result of the comparison changes or when the output compare mode switches from “frozen” mode to “PWM” mode. 2. On channels having a complementary output, this bit field is preloaded. If the CCPC bit is set in the PWMA_CR2 register then the OC2M active bits take the new value from the preloaded bits only when a COM event is generated.  • If OC2M bit is set to 0x6 or 0x7, it is not recommended to set the PWMA_CCR2 register to 0x0. If these values are written, the timer output will not give a 100% duty cycle, and, when the counter value is equal to zero, a short pulse will be emitted on each counter period. • It is recommended to set the OC2M bit to 0x4 or 0x5 to get a 100% duty cycle. **Value After Reset:** 0x0
[3]	OC2PE	R/W	Output Compare Channel 2 preload enable **Values:**

Table 12-17 PWMA_CCMR2[7:2] bits functions for different CC2S values (continued)

Bits	Name	R/W	Description
			0x0: Preload register for PWMA_CCR2 disabled. PWMA_CCR2 can be written at anytime, the new value is taken into account immediately. **0x1:** Preload register for PWMA_CCR2 enabled. Read/Write operations access the preload register. PWMA_CCR2 preload value is loaded into the active register at each update event. The PWM mode can be used without validating the preload register only in one pulse mode (OPM bit set in PWMA_CR1 register). Otherwise, the behavior is not guaranteed. **Value After Reset:** 0x0
[2]			Reserved
Channel 2 Input Capture mode (CC2S != 0x0)			
[7:4]	IC2F	R/W	Input Capture Channel 2 filter This bit field defines both the sampling frequency for the TI2 input and the length of the digital filter applied to TI2. The digital filter consists of an event counter that requires N consecutive events to validate a transition at the output. **Values:** **0x0:** No filter, sampling is done at f_{DTS} ($f_{SAMPLING}=f_{DTS}$) **0x1:** $f_{SAMPLING}=f_{CK\_INT}$, N=2 **0x2:** $f_{SAMPLING}=f_{CK\_INT}$, N=4 **0x3:** $f_{SAMPLING}=f_{CK\_INT}$, N=8 **0x4:** $f_{SAMPLING}=f_{DTS}/2$, N=6 **0x5:** $f_{SAMPLING}=f_{DTS}/2$, N=8 **0x6:** $f_{SAMPLING}=f_{DTS}/4$, N=6 **0x7:** $f_{SAMPLING}=f_{DTS}/4$, N=8 **0x8:** $f_{SAMPLING}=f_{DTS}/8$, N=6 **0x9:** $f_{SAMPLING}=f_{DTS}/8$, N=8 **0xA:** $f_{SAMPLING}=f_{DTS}/16$, N=5 **0xB:** $f_{SAMPLING}=f_{DTS}/16$, N=6 **0xC:** $f_{SAMPLING}=f_{DTS}/16$, N=8 **0xD:** $f_{SAMPLING}=f_{DTS}/32$, N=5 **0xE:** $f_{SAMPLING}=f_{DTS}/32$, N=6 **0xF:** $f_{SAMPLING}=f_{DTS}/32$, N=8 **Value After Reset:** 0x0
[3:2]	IC2PSC	R/W	Input Capture Channel 2 prescaler This bit field defines the Prescaler ratio for the CC2 input (IC20). The Prescaler is reset as soon as CC2E=0x0 (PWMA_CCER register). **Values:** **0x0:** No prescaler, capture is done each time an edge is detected on the capture input **0x1:** Capture is done once every 2 events **0x2:** Capture is done once every 4 events **0x3:** Capture is done once every 8 events

Table 12-17 `PWMA_CCMR2[7:2]` bits functions for different `CC2S` values (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0

12.1.2.2.9. PWMA Capture/Compare Enable Register (`PWMA_CCER`)

The `PWMA_CCER` register is at offset 0x20.

The following table shows the register bit assignments.

Table 12-18 Fields for register: `PWMA_CCER`

Bits	Name	R/W	Description
[15:14]			Reserved
[13]	`CC3P`	R/W	Capture/Compare Channel 3 output polarity This bit field description and its valid values are similar to the `CC0P` bit filed. **Value After Reset:** 0x0
[12]	`CC3E`	R/W	Capture/Compare Channel 3 output enable This bit field description and its valid values are similar to the `CC0E` bit filed. **Value After Reset:** 0x0
[11]	`CC2NP`	R/W	Capture/Compare Channel 2 complementary output polarity This bit field description and its valid values are similar to the `CC0NP` bit filed. **Value After Reset:** 0x0
[10]	`CC2NE`	R/W	Capture/Compare Channel 2 complementary output enable This bit field description and its valid values are similar to the `CC0NE` bit filed. **Value After Reset:** 0x0
[9]	`CC2P`	R/W	Capture/Compare Channel 2 output polarity This bit field description and its valid values are similar to the `CC0P` bit filed. **Value After Reset:** 0x0
[8]	`CC2E`	R/W	Capture/Compare Channel 2 output enable This bit field description and its valid values are similar to the `CC0E` bit filed. **Value After Reset:** 0x0
[7]	`CC1NP`	R/W	Capture/Compare Channel 1 complementary output polarity This bit field description and its valid values are similar to the `CC0NP` bit filed. **Value After Reset:** 0x0
[6]	`CC1NE`	R/W	Capture/Compare Channel 1 complementary output enable This bit field description and its valid values are similar to the `CC0NE` bit filed.

Table 12-18 Fields for register: `PWMA_CCER` (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0
[5]	`CC1P`	R/W	Capture/Compare Channel 1 output polarity This bit field description and its valid values are similar to the `CC0P` bit filed. **Value After Reset:** 0x0
[4]	`CC1E`	R/W	Capture/Compare Channel 1 output enable This bit field description and its valid values are similar to the `CC0E` bit filed. **Value After Reset:** 0x0
[3]	`CC0NP`	R/W	Capture/Compare Channel 0 complementary output polarity **Values:** **0x0:** `OC0N` active high **0x1:** `OC0N` active low **Value After Reset:** 0x0
[2]	`CC0NE`	R/W	Capture/Compare Channel 0 complementary output enable **Values:** **0x0** (Off): The `OC0N` is not active. The `OC0N` level is then function of `MOE`, `OSSR`, `OIS0`, `OIS0N`, and `CC0E` bits. **0x1** (On): The `OC0N` signal is output on the corresponding output pin depending on `MOE`, `OSSR`, `OIS0`, `OIS0N`, and `CC0E` bits. **Value After Reset:** 0x0
[1]	`CC0P`	R/W	Capture/Compare Channel 0 output polarity **If channel CC0 is configured as output** (`PWMA_CCMR1.CC0S` = 0x0): **Values:** **0x0:** `OC0` active high **0x1:** `OC0` active low **If channel CC0 is configured as input** (`PWMA_CCMR1.CC0S` != 0x0): This bit selects the active polarity of `TI0FP0` and `TI1FP0`. **Values:** **0x0:** Non-inverted/rising edge **0x1:** Inverted/falling edge **Value After Reset:** 0x0
[0]	`CC0E`	R/W	Capture/Compare Channel 0 output enable **If channel CC0 is configured as output** (`PWMA_CCMR1.CC0S` = 0x0): **Values:** **0x0** (Off): The `OC0` is not active. The `OC0` level is then function of `MOE`, `OSSR`, `OIS0`, `OIS0N`, and `CC0NE` bits. **0x1** (On): The `OC0` signal is output on the corresponding output pin depending on `MOE`, `OSSR`, `OIS0`, `OIS0N`, and `CC0NE` bits. **If channel CC0 is configured as input** (`PWMA_CCMR1.CC0S` != 0x0):

Table 12-18 Fields for register: PWMA_CCER (continued)

Bits	Name	R/W	Description
			This bit determines if a capture of the counter value can actually be done into the input capture/compare register 0 (PWMA_CCR0) or not. **Values:** **0x0:** Capture disabled **0x1:** Capture enabled **Value After Reset:** 0x0

12.1.2.2.10. PWMA Counter (PWMA_CNT)

The PWMA_CNT register is at offset 0x24.

The following table shows the register bit assignments.

Table 12-19 Fields for register: PWMA_CNT

Bits	Name	R/W	Description
[15:0]	CNT	R/W	Counter value **Value After Reset:** 0x0

12.1.2.2.11. PWMA Prescaler (PWMA_PSC)

The PWMA_PSC register is at offset 0x28.

The following table shows the register bit assignments.

Table 12-20 Fields for register: PWMA_PSC

Bits	Name	R/W	Description
[15:0]	PSC	R/W	Prescaler value The counter clock frequency (CK_CNT) is equal to $f_{CK\_PSC} / (PSC + 1)$. The PSC contains the value to be loaded in the active prescaler register at each Update Event (including when the Counter is cleared through the UG bit of the PWMA_EGR register or through the trigger controller, when configured in “reset mode”). **Value After Reset:** 0x0

12.1.2.2.12. PWMA Auto-reload Register (PWMA_ARR)

The PWMA_ARR register is at offset 0x2C.

The following table shows the register bit assignments.

Table 12-21 Fields for register: PWMA_ARR

Bits	Name	R/W	Description
[15:0]	ARR	R/W	Auto-reload value ARR is the value to be loaded in the actual auto-reload register.

Table 12-21 Fields for register: PWMA_ARR (continued)

Bits	Name	R/W	Description
			This register can be buffered (see ARPE bit description in the PWMA_CR1 register). The counter is blocked while the Auto-reload value (ARR) is null. **Value After Reset:** 0x0

12.1.2.2.13. PWMA Repetition Counter Register (PWMA_RCR)

The PWMA_RCR register is at offset 0x30.

The following table shows the register bit assignments.

Table 12-22 Fields for register: PWMA_RCR

Bits	Name	R/W	Description
[15:8]			Reserved
[7:0]	REP	R/W	Repetition counter value These bits allow the user to configure the update rate of the compare registers (i.e., the rate of periodic transfers from the preload registers to the active registers) when the preload registers are enabled, as well as the update interrupt generation rate, if this interrupt is enabled. Each time the Repetition Counter related downcounter reaches zero, an Update Event is generated and it restarts counting from the REP value. Since the Repetition Counter is reloaded with the REP value only at the Update Event, any write to the PWMA_RCR register is not taken into account until the next Update Event. When the Update Event is generated by software (by setting the UG bit in PWMA_EGR register) or by hardware through the slave mode controller, the repetition counter is reloaded with the content of the PWMA_RCR register. This means that in PWM mode (REP+1) corresponds to: • The number of PWM periods in edge-aligned mode • The number of half PWM periods in center-aligned mode **Value After Reset:** 0x0

12.1.2.2.14. PWMA Capture/Compare Register 0 (PWMA_CCR0)

The PWMA_CCR0 register is at offset 0x34.

The following table shows the register bit assignments.

Table 12-23 Fields for register: PWMA_CCR0

Bits	Name	R/W	Description
[15:0]	CCR0	R/W	Capture/Compare Channel 0 value **If the channel CC0 is configured as output** (PWMA_CCMR1.CC0S = 0x0): CCR0 contains the value to be loaded into the actual capture/compare 0 register (preload value).

Table 12-23 Fields for register: PWMA_CCR0 (continued)

Bits	Name	R/W	Description
			This value is loaded permanently if the preload feature is not selected in the PWMA_CCMR1 register (bit OC0PE). Otherwise, the preload value is copied into the active capture/compare 0 register when an Update Event occurs. The active capture/compare register contains the value to be compared to the counter PWMA_CNT and signaled on OC0 output. STOP If PWMA_CCMR1.OC0M bit is set to 0x6 or 0x7, it is not recommended to set the CCR0 bit to 0x0. If these values are written, the timer output will not give a 100% duty cycle, and, when the counter value is equal to zero, a short pulse will be emitted on each counter period. **If the channel CC0 is configured as input** (PWMA_CCMR1.CC0S != 0x0): CCR0 contains the counter value transferred by the last input capture 0 event (IC0). The PWMA_CCR0 register is read-only and cannot be programmed. **Value After Reset:** 0x0

12.1.2.2.15. PWMA Capture/Compare Register 1 (PWMA_CCR1)

The PWMA_CCR1 register is at offset 0x38.

The following table shows the register bit assignments.

Table 12-24 Fields for register: PWMA_CCR1

Bits	Name	R/W	Description
[15:0]	CCR1	R/W	Capture/Compare Channel 1 value **If the channel CC1 is configured as output** (PWMA_CCMR1.CC1S = 0x0): CCR1 contains the value to be loaded into the actual capture/compare 1 register (preload value). This value is loaded permanently if the preload feature is not selected in the PWMA_CCMR1 register (bit OC1PE). Otherwise, the preload value is copied into the active capture/compare 1 register when an Update Event occurs. The active capture/compare register contains the value to be compared to the counter PWMA_CNT and signaled on OC1 output. STOP If PWMA_CCMR1.OC1M bit is set to 0x6 or 0x7, it is not recommended to set the CCR1 bit to 0x0. If these values are written, the timer output will not give a 100% duty cycle, and, when the counter value is equal to zero, a short pulse will be emitted on each counter period. **If the channel CC1 is configured as input** (PWMA_CCMR1.CC1S != 0x0): CCR1 contains the counter value transferred by the last input capture 1 event (IC1). The PWMA_CCR1 register is read-only and cannot be programmed. **Value After Reset:** 0x0

12.1.2.2.16. PWMA Capture/Compare Register 2 (PWMA_CCR2)

The PWMA_CCR2 register is at offset 0x3C.

The following table shows the register bit assignments.

Table 12-25 Fields for register: PWMA_CCR2

Bits	Name	R/W	Description
[15:0]	CCR2	R/W	Capture/Compare Channel 2 value **If the channel CC2 is configured as output** (PWMA_CCMR2.CC2S = 0x0): CCR2 contains the value to be loaded into the actual capture/compare 2 register (preload value). This value is loaded permanently if the preload feature is not selected in the PWMA_CCMR2 register (bit OC2PE). Otherwise, the preload value is copied into the active capture/compare 2 register when an Update Event occurs. The active capture/compare register contains the value to be compared to the counter PWMA_CNT and signaled on OC2 output. STOP If PWMA_CCMR2.OC2M bit is set to 0x6 or 0x7, it is not recommended to set the CCR2 bit to 0x0. If these values are written, the timer output will not give a 100% duty cycle, and, when the counter value is equal to zero, a short pulse will be emitted on each counter period. **If the channel CC2 is configured as input** (PWMA_CCMR2.CC2S != 0x0): CCR2 contains the counter value transferred by the last input capture 2 event (IC2). The PWMA_CCR2 register is read-only and cannot be programmed. **Value After Reset:** 0x0

12.1.2.2.17. PWMA Capture/Compare Register 3 (PWMA_CCR3)

The PWMA_CCR3 register is at offset 0x40.

The following table shows the register bit assignments.

Table 12-26 Fields for register: PWMA_CCR3

Bits	Name	R/W	Description
[15:0]	CCR3	R/W	Capture/Compare Channel 3 value **If the channel CC3 is configured as output** (PWMA_CCMR2.CC3S = 0x0): CCR3 contains the value to be loaded into the actual capture/compare 3 register (preload value). This value is loaded permanently if the preload feature is not selected in the PWMA_CCMR2 register (bit OC3PE). Otherwise, the preload value is copied into the active capture/compare 3 register when an Update Event occurs. The active capture/compare register contains the value to be compared to the counter PWMA_CNT and signaled on OC3 output. STOP If PWMA_CCMR2.OC3M bit is set to 0x6 or 0x7, it is not recommended to set the CCR3 bit to 0x0. If these values are

Table 12-26 Fields for register: `PWMA_CCR3` (continued)

Bits	Name	R/W	Description
			written, the timer output will not give a 100% duty cycle, and, when the counter value is equal to zero, a short pulse will be emitted on each counter period. **If the channel CC3 is configured as input** (`PWMA_CCMR2.CC3S` != 0x0): `CCR3` contains the counter value transferred by the last input capture 3 event (`IC3`). The `PWMA_CCR3` register is read-only and cannot be programmed. **Value After Reset:** 0x0

12.1.2.2.18. PWMA Break and Dead-Time Register (`PWMA_BDTR`)

The `PWMA_BDTR` register is at offset 0x44.

The following table shows the register bit assignments.

Table 12-27 Fields for register: `PWMA_BDTR`

Bits	Name	R/W	Description
[15]	`MOE`	R/W	Main output enable This bit is cleared asynchronously by hardware when the break input becomes active (if the break function is enabled, see `BKE` for details). It is set by software or automatically, depending on the `AOE` bit. It acts only on the channels that are configured as outputs. **Values:** **0x0:** `OCx` and `OCxN` outputs are disabled or forced to idle state **0x1:** `OCx` and `OCxN` outputs are enabled if their respective enable bits are set (`CCxE`, `CCxNE` in `PWMA_CCER` register) **Value After Reset:** 0x0
[14]	`AOE`	R/W	Automatic output enable **Values:** **0x0:** `MOE` can be set only by software **0x1:** `MOE` can be set by software or automatically at the next update event (if the break input is not be active) **Value After Reset:** 0x0
[13]	`BKP`	R/W	Break polarity **Values:** **0x0:** Break input (`BRK`) is active low **0x1:** Break input (`BRK`) is active high Any write operation to this bit takes a delay of one Register Interface clock cycle to become effective. **Value After Reset:** 0x0
[12]	`BKE`	R/W	Break enable **Values:**

Table 12-27 Fields for register: PWMA_BDTR (continued)

Bits	Name	R/W	Description
			0x0: Break input (BRK) disabled **0x1:** Break input (BRK) enabled. An active signal level on input resets the MOE bit. ⚠ Any write operation to this bit takes a delay of one Register Interface clock cycle to become effective. **Value After Reset:** 0x0
[11]	OSSR	R/W	Off-state selection for Run mode This bit determines the state of disabled outputs when MOE=0x1 on channels having a complementary output which are configured as outputs. OSSR is not implemented if no complementary output is implemented in the timer. **Values:** **0x0:** The disabled outputs OCx/OCxN are set to 0. **0x1:** The disabled outputs OCx/OCxN are set to their inactive level (inactive level depends on settings of the CCxP and CCxNP bits in the PWMA_CCER register). 📝 • An output is considered to be disabled when the appropriate CCxE or CCxNE bit of the PWMA_CCER is set to 0x0. • The disabled output level can be set to 1 if that level is specified as inactive. ⚠ Avoid the configuration when both outputs of one channel are disabled (appropriate CCxE and CCxNE bits in the PWMA_CCER register are set to 0x0), and break function is enabled (protective shutdown of timer outputs). In this case, reset of the MOE bit can lead to the unpredictable behaviour of the channel outputs. **Value After Reset:** 0x0
[10:8]			Reserved
[7:0]	DTG	R/W	Dead-time generator setup This bit field defines the duration of the *dead-time* (**DT**) inserted between the complementary outputs. DT correspond to this duration. **Values:** **0xxxxxxx:** DT=DTG[7:0]*t_{dtg} with t_{dtg}=t_{DTS} **10xxxxxx:** DT=(64+DTG[5:0])*t_{dtg} with t_{dtg}=2*t_{DTS} **110xxxxx:** DT=(32+DTG[4:0])*t_{dtg} with t_{dtg}=8*t_{DTS} **111xxxxx:** DT=(32+DTG[4:0])*t_{dtg} with t_{dtg}=16*t_{DTS} **Example:** If t_{DTS}=125 ns (8 MHz), dead-time possible values are: • 0 to 15875 ns by 125 ns steps • 16 µs to 31750 ns by 250 ns steps • 32 µs to 63 µs by 1 µs steps • 64 µs to 126 µs by 2 µs steps

Table 12-27 Fields for register: PWMA_BDTR (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0

12.1.2.2.19. PWMA DMA Control Register (PWMA_DCR)

The PWMA_DCR register is at offset 0x48.

The following table shows the register bit assignments.

Table 12-28 Fields for register: PWMA_DCR

Bits	Name	R/W	Description
[15:13]			Reserved (return 0 when read)
[12:8]	DBL	R/W	DMA burst length This 5-bit vector defines the number n of DMA transfers, where n=DBL (the timer detects a burst transfer when a read or a write access to the PWMA_DMAR register address is performed). **Values:** **0x0 or 0x1:** 1 transfer **0x2:** 2 transfers **0x3:** 3 transfers ... **0x11:** 17 transfers **Value After Reset:** 0x0
[7:5]			Reserved (return 0 when read)
[4:0]	DBA	R/W	DMA base address This 5-bits vector defines the base address for DMA transfers (when read/write access is performed via the PWMA_DMAR address). The DBA is defined as an offset starting from the address of the PWMA_CR1 register. **An example of values:** **0x0:** The offset corresponds to the PWMA_CR1 register **0x1:** The offset corresponds to the PWMA_CR2 register **0x2:** The offset corresponds to the PWMA_SMCR register **An example of usage:** Let's consider a transfer with DBL = 0x7 (7 transfers) and DBA = 0x0 (the offset corresponds to the PWMA_CR1 register). This configuration will perform a transfer to/from 7 registers starting from the PWMA_CR1 address. **Value After Reset:** 0x0

12.1.2.2.20. PWMA DMA Address for Full Transfer (PWMA_DMAR)

The PWMA_DMAR register is at offset 0x4C.

The following table shows the register bit assignments.

Table 12-29 Fields for register: PWMA_DMAR

Bits	Name	R/W	Description
[31:0]	DMAB	R/W	DMA register for burst accesses A read or write to the PWMA_DMAR register accesses the register at the address: (PWMA_CR1 address) + (DBA + DMA index) * 4, where: PWMA_CR1 address is the offset of the PWMA Control Register 1; DBA is the DMA base address configured in PWMA_DCR register; DMA index is automatically controlled by the DMA transfer, and ranges from 0 to DBL (DBL configured in PWMA_DCR). **Value After Reset:** 0x0

12.1.2.2.21. PWMA Quadrature Encoder Setting Register (PWMA_QESET)

The PWMA_QESET register is at offset 0x50.

The following table shows the register bit assignments.

Table 12-30 Fields for register: PWMA_QESET

Bits	Name	R/W	Description
[15:14]			Reserved (return 0 when read)
[13]	INTRPT_CLR	W	Quadrature Encoder interrupt clear Write 0x1 to this bit to reset the setted interrupt. Writing 0x0 has no effect. **Value After Reset:** 0x0
[12]	INIE	R/W	indx interrupt enable This bit enables the interrupt on the indx signal edge. **Values:** **0x0:** indx interrupt disabled **0x1:** indx interrupt enabled **Value After Reset:** 0x0
[11]	OFIE	R/W	Overflow interrupt enable This bit enables the interrupt on the counter overflow and counter underflow. **Values:** **0x0:** Overflow interrupt disabled **0x1:** Overflow interrupt enabled **Value After Reset:** 0x0
[10]	INIT_MODE	R/W	Initialization mode **Values:** **0x0:** A rising edge of the indx signal resets the counter. **0x1:** A rising edge of the indx signal will cause the counter initialization with the value of the PWMA_INITSET register. **Value After Reset:** 0x0

Table 12-30 Fields for register: `PWMA_QESET` (continued)

Bits	Name	R/W	Description
[9]	`SINGLE_MODE`	R/W	Single mode **Values:** **0x0** (Multiple mode): The counter can count up to `PWMA_QEMAX` value and, if the `PWMA_QECNT` = `PWMA_QEMAX`, the counter restarts with 0. The counter can also count down to 0 and, if the `PWMA_QECNT` = 0, the counter restarts with `PWMA_QEMAX` value. **0x1** (Single mode): The counter can count up to `PWMA_QEMAX` value and then stop, or count up to 0 and then stop. **Value After Reset:** 0x0
[8]	`INDXPOL`	R/W	`indx` polarity **Values:** **0x0:** `indx` signal is not inverted **0x1:** `indx` signal is inverted **Value After Reset:** 0x0
[7]	`QBPOL`	R/W	`qb` polarity **Values:** **0x0:** `qb` signal is not inverted **0x1:** `qb` signal is inverted **Value After Reset:** 0x0
[6]	`QAPOL`	R/W	`qa` polarity **Values:** **0x0:** `qa` signal is not inverted **0x1:** `qa` signal is inverted **Value After Reset:** 0x0
[5]	`QBF`	R/W	`qb` Filtering **Values:** **0x0:** Disable the `qb` filtering **0x1:** Enable the `qb` filtering Filter settings are controlled through the `QBSF` bit field of the `PWMA_FILTSET` register. **Value After Reset:** 0x0
[4]	`QAF`	R/W	`qa` Filtering **Values:** **0x0:** Disable the `qa` filtering **0x1:** Enable the `qa` filtering Filter settings are controlled through the `QASF` bit field of the `PWMA_FILTSET` register. **Value After Reset:** 0x0

Table 12-30 Fields for register: `PWMA_QESET` (continued)

Bits	Name	R/W	Description
[3]	`INDXF`	R/W	`indx` Filtering **Values:** **0x0:** Disable the `indx` filtering **0x1:** Enable the `indx` filtering Filter settings are controlled through the `INDXSF` bit field of the `PWMA_FILTSET` register. **Value After Reset:** 0x0
[2]	`QESWAP`	R/W	Swap `qa` and `qb` signals **Values:** **0x0:** No swap **0x1:** `qa` is repaced by `qb`, and `qb` is replaced by `qa` **Value After Reset:** 0x0
[1]	`QEMODE`	R/W	Selection of counting mode **Values:** **0x0:** Quadrature Mode **0x1:** `qa` Rising/Falling Edge Mode **Value After Reset:** 0x0
[0]	`QEN`	R/W	Quadrature Encoder enable **Values:** **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x0

12.1.2.2.22. PWMA Quadrature Encoder Filters Setting (`PWMA_FILTSET`)

The `PWMA_FILTSET` register is at offset 0x54.

The following table shows the register bit assignments.

Table 12-31 Fields for register: `PWMA_FILTSET`

Bits	Name	R/W	Description
[15:12]			Reserved (return 0 when read)
[11:8]	`INDXSF`	R/W	`indx` filter setting This bit field defines both the sampling frequency for the indx signal and the length of the digital filter applied to `indx`. The digital filter consists of an event counter that requires N consecutive events to validate a transition at the output. The possible values of this bit are the same as the values of the `QASF` bit. **Value After Reset:** 0x0

Table 12-31 Fields for register: PWMA_FILTSET (continued)

Bits	Name	R/W	Description
[7:4]	QBSF	R/W	qb filter setting This bit field defines both the sampling frequency for the qb signal and the length of the digital filter applied to qb. The digital filter consists of an event counter that requires N consecutive events to validate a transition at the output. The possible values of this bit are the same as the values of the QASF bit **Value After Reset:** 0x0
[3:0]	QASF	R/W	qa filter setting This bit field defines both the sampling frequency for the qa signal and the length of the digital filter applied to qa. The digital filter consists of an event counter that requires N consecutive events to validate a transition at the output. **Values:** **0x0:** No filter, sampling is done at f_{DTS} **0x1:** $f_{SAMPLING}=f_{CK\_INT}$, N=2 **0x2:** $f_{SAMPLING}=f_{CK\_INT}$, N=4 **0x3:** $f_{SAMPLING}=f_{CK\_INT}$, N=8 **0x4:** $f_{SAMPLING}=f_{DTS}/2$, N=6 **0x5:** $f_{SAMPLING}=f_{DTS}/2$, N=8 **0x6:** $f_{SAMPLING}=f_{DTS}/4$, N=6 **0x7:** $f_{SAMPLING}=f_{DTS}/4$, N=8 **0x8:** $f_{SAMPLING}=f_{DTS}/8$, N=6 **0x9:** $f_{SAMPLING}=f_{DTS}/8$, N=8 **0xA:** $f_{SAMPLING}=f_{DTS}/16$, N=5 **0xB:** $f_{SAMPLING}=f_{DTS}/16$, N=6 **0xC:** $f_{SAMPLING}=f_{DTS}/16$, N=8 **0xD:** $f_{SAMPLING}=f_{DTS}/32$, N=5 **0xE:** $f_{SAMPLING}=f_{DTS}/32$, N=6 **0xF:** $f_{SAMPLING}=f_{DTS}/32$, N=8 **Value After Reset:** 0x0

12.1.2.2.23. PWMA Quadrature Encoder Max Count Value (PWMA_QEMAX)

The PWMA_QEMAX register is at offset 0x58.

The following table shows the register bit assignments.

Table 12-32 Fields for register: PWMA_QEMAX

Bits	Name	R/W	Description
[15:0]	QEMAX	R/W	Max count value

Table 12-32 Fields for register: `PWMA_QEMAX` (continued)

Bits	Name	R/W	Description
			This bit field defines the max count value after which the counter will stop in single mode (`PWMA_QESET.SINGLE_MODE` = 0x1) or reset in multiple mode (`PWMA_QESET.SINGLE_MODE` = 0x0). **Value After Reset:** 0xFFFF

12.1.2.2.24. PWMA Quadrature Encoder Init Value (`PWMA_INITSET`)

The `PWMA_INITSET` register is at offset 0x5C.

The following table shows the register bit assignments.

Table 12-33 Fields for register: `PWMA_INITSET`

Bits	Name	R/W	Description
[15:0]	`INITSET`	R/W	Initialization value for the counter If the `PWMA_QESET.INIT_MODE` = 0x1, the rising edge of the `indx` signal will cause the counter initialization with the value of this register. **Value After Reset:** 0xFFFF

12.1.2.2.25. PWMA Quadrature Encoder Counter Value (`PWMA_QECNT`)

The `PWMA_QECNT` register is at offset 0x60.

The following table shows the register bit assignments.

Table 12-34 Fields for register: `PWMA_QECNT`

Bits	Name	R/W	Description
[15:0]	`QECNT`	R/W	Quadrature Encoder Counter value **Value After Reset:** 0x0

12.2. PWMG

This section describes the general-purpose timer (below referred to as PWMG) of the BE-U1000 microcontroller. The BE-U1000 contains two PWMGs.

A simplified block diagram of the PWMG is shown in the following figure.

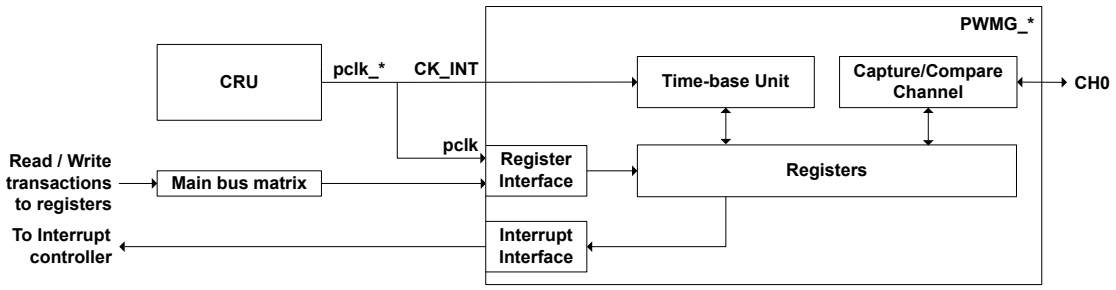

Figure 12-15 PWMG block diagram

In the figure above, an asterisk symbol (*) indicates 0 for PWMG_0 and 1 for PWMG_1.

Signal ID, shown in the right part of the diagram above, corresponds to the appropriate I/O pin ID. For more information, refer to the ***BE-U1000 Datasheet***.

PWMG features:

- Single channel for input signal capture and output signal generation
- Programmable prescaler for the counter clock (`CK_CNT`) frequency
- Interrupt generation on the following events:
  - Update event (counter overflow, counter initialization)
  - Input capture
  - Output compare

12.2.1. PWMG Functional Description

This section covers the following topics:

- Time-base Unit
- Counter Upcounting Mode
- Capture/Compare Channel
- Shadow Registers

12.2.1.1. Time-Base Unit

The Time-Base Unit consists of the following blocks, as the figure below shows:

- Counter
- Prescaler

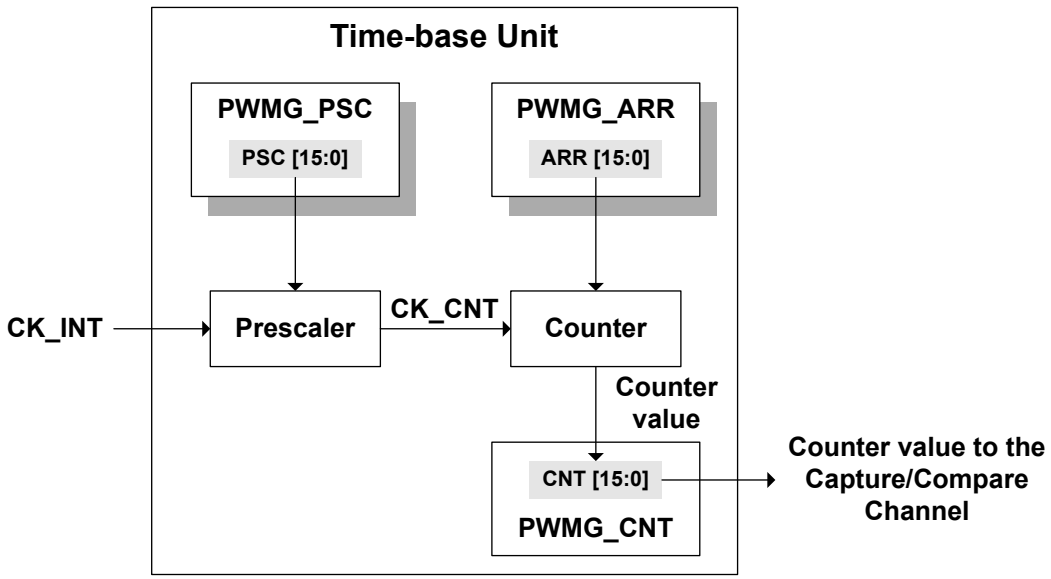

Figure 12-16 Time-Base Unit

As shown in the figure above, the blocks of the Time-Base Unit can be controlled through the following registers:

- PWMG Counter (`PWMG_CNT`)
- PWMG Prescaler (`PWMG_PSC`)
- PWMG Auto-reload Register (`PWMG_ARR`)

These registers can be written or read by software even when the counter is running.

In the figure above, some registers are shown as rectangles with the gray background. This means that the register has a software inaccessible shadow register. For more information, refer to Shadow Registers section.

12.2.1.1.1. Counter

The main block of the PWMG is a 16-bit counter with its related auto-reload register (PWMG_ARR). The counter counts up. The Counter clock (CK_CNT) can be divided by the Prescaler.

The Counter is enabled only when the CEN bit in the PWMG_CR1 register is set. Note that the Counter starts counting 1 clock cycle after setting the CEN bit in the PWMG_CR1 register.

The auto-reload register (PWMG_ARR) is preloaded. Writing to or reading from the auto-reload register accesses the preload register. The content of the preload register is transferred into the shadow register either permanently or at each *Update Event* (**UEV**), depending on the auto-reload preload enable bit (ARPE) in PWMG_CR1 register. The Update Event is sent when the counter reaches the overflow and if the UDIS bit equals 0 in the PWMG_CR1 register. It can also be generated by software by setting the UG bit in the PWMG_EGR register. For information about the UEV generation and PWMG_ARR usage, see Counter Upcounting Mode section.

12.2.1.1.2. Prescaler

The Prescaler can divide the counter clock frequency by any factor between 1 and 65536. It is based on a 16-bit counter controlled through a 16-bit PWMG_PSC register. It can be changed on the fly as this control register is buffered. The new prescaler ratio is taken into account at the next *Update Event* (**UEV**).

- The UEV can be disabled by software by setting the UDIS bit in the PWMG_CR1 register.
- The UEV can be generated by software by setting the UG bit in the PWMG_EGR register.

12.2.1.2. Counter Upcounting Mode

The Counter can operate in upcounting mode. In upcounting mode, the Counter counts from 0 to the auto-reload value (content of the PWMG_ARR register), then restarts from 0 and generates a counter overflow event.

Setting the UG bit in the PWMG_EGR register also generates an *Update Event* (**UEV**).

When the UEV is generated, the Counter restarts from 0, as well as the counter of the Prescaler (but the prescale rate does not change).

The UEV can be disabled by software by setting the UDIS bit in the PWMG_CR1 register. This prevents updating the shadow registers while writing new values into the preload registers. Then no Update Event occurs until the UDIS bit has been written to 0. However, the Counter continues counting up based on the current auto-reload value.

When an UEV occurs, all the registers are updated and the update flag (UIF bit in PWMG_SR register) is set:

- The auto-reload shadow register is updated with the preload value (PWMG_ARR).
- The buffer of the Prescaler is reloaded with the preload value (content of the PWMG_PSC register).

Depending on the PWMG settings, the occurence of an UEV may lead to the interrupt generation.

12.2.1.3. Capture/Compare Channel

Capture/Compare Channel is used for input signal capture and output signal generation. Capture/Compare Channel has one input and one output that are connected to the BE-U1000 I/O pin.

The Capture/Compare Channel combines two functional blocks, one used when the channel operates in input mode (to capture an input signal) and the other used in output mode (to generate an output signal). The Capture/Compare Channel is built around a Capture/Compare Stage that includes a PWMG Capture/Compare Register (with a shadow register), an Input Stage for signal capture (with a digital filter and the Prescaler), and an Output Stage (with output control).

Since the channel can't work in input and output mode at the same time, the input and output of the channel are physically combined into a single `CH0` I/O pin.

A simplified block diagram of the PWMG Capture/Compare Channel is shown in the figure below.

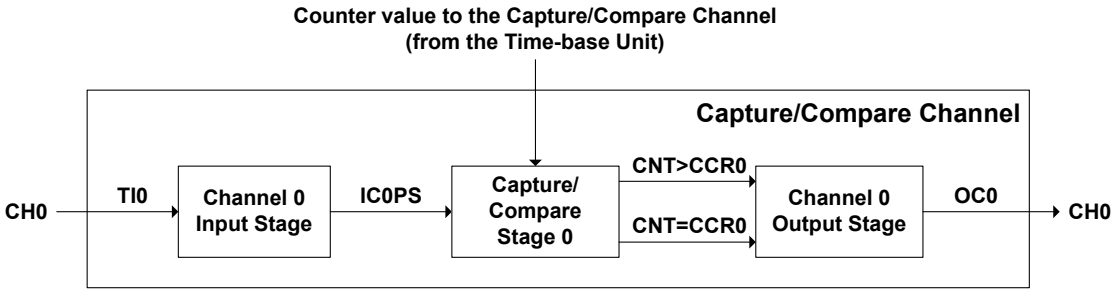

Figure 12-17 Capture/Compare Channel

12.2.1.3.1. Input Stage

A block diagram of the Channel 0 Input Stage is shown in the figure below.

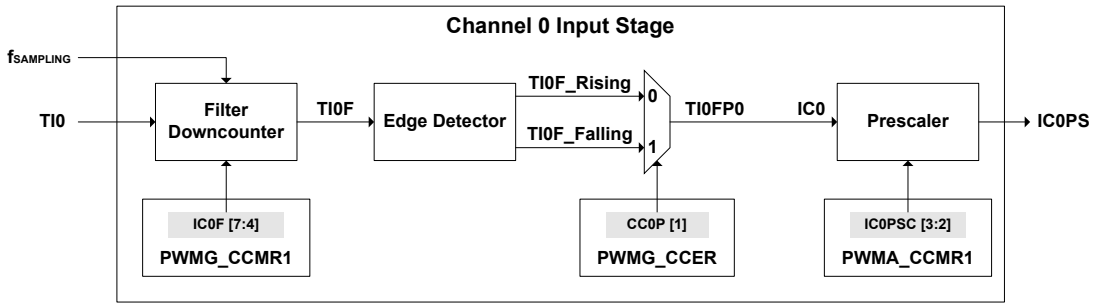

Figure 12-18 Channel 0 Input Stage

As the figure above shows, channel 0 input signal (`TI0`) is resynchronized and passed through the digital filter, which is enabled and configured through the `IC0F` bit field of the `PWMG_CCMR1` register. In this case, `TI0` signal is sampled with the sampling frequency ($f_{SAMPLING}$) that also depends on the `IC0F` bit field value. Thus, we receive the filtered signal `TI0F` that is connected to the Edge Detector input.

Edge Detector has `TI0F_Rising` and `TI0F_Falling` output signals. A short pulse appears on the `TI0F_Rising` signal when a rising edge of the `TI0F` signal is detected and on the `TI0F_Falling` when a falling edge of the `TI0F` signal is detected. Multiplexor, which is used after the Edge Detector and controlled through the `CC0P` bit of the `PWMG_CCER` register, allows one of the `TI0F_Rising` and `TI0F_Falling` signals to be used as the source of the `TI0FP0` signal.

The TI0FP0 indicates how the signal is received:

- TI0 indicates that the signal is received on the channel 0 input.
- F indicates that the signal is resynchronized and passed through the digital filter.
- P0 indicates that the active level of the signal is controlled via the channel 0 settings, i.e. CC0P bit of the PWMG_CCER register.

As shown in the figure above, the TI0FP0 signal is used as the IC0 signal source. IC0 signal then is connected to the programmable prescaler that is controlled through the IC0PSC bit field of the PWMG_CCMR1 register. IC0PS signal on the prescaler output is connected to the Capture/Compare Stage 0.

12.2.1.3.2. Output Stage

The following figure shows the diagram of the Channel 0 Output Stage.

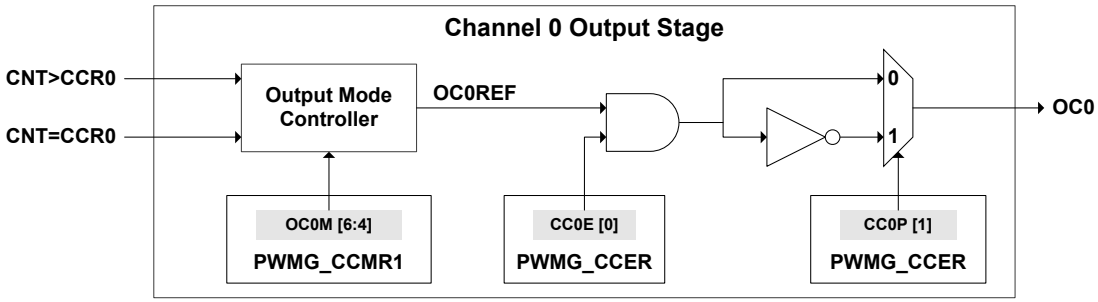

Figure 12-19 Channel 0 Output Stage

As the figure above shows, OC0REF signal is generated by the Output Mode Controller based on the CNT>CCR0 and CNT=CCR0 signals from the Capture/Compare Stage 0 and PWMG_CCMR1.OC0M bit setting.

You can also select the polarity of the OC0 output. This is done by writing to the CC0P bit of the PWMG_CCER register. Note than the OC0 signal is used only if the CC0E bit of the PWMG_CCER register is set to 0x1.

12.2.1.3.3. Capture/Compare Stage

A Capture/Compare Stage is used both when the channel is in input mode and when it is in output mode.

The channel is configured as output when the CC0S bits of the PWMG_CCMR1 register are set to 0. If the CC0S bits are set to 1, the channel is configured as input.

The figure below shows the diagram of the Capture/Compare Stage 0.

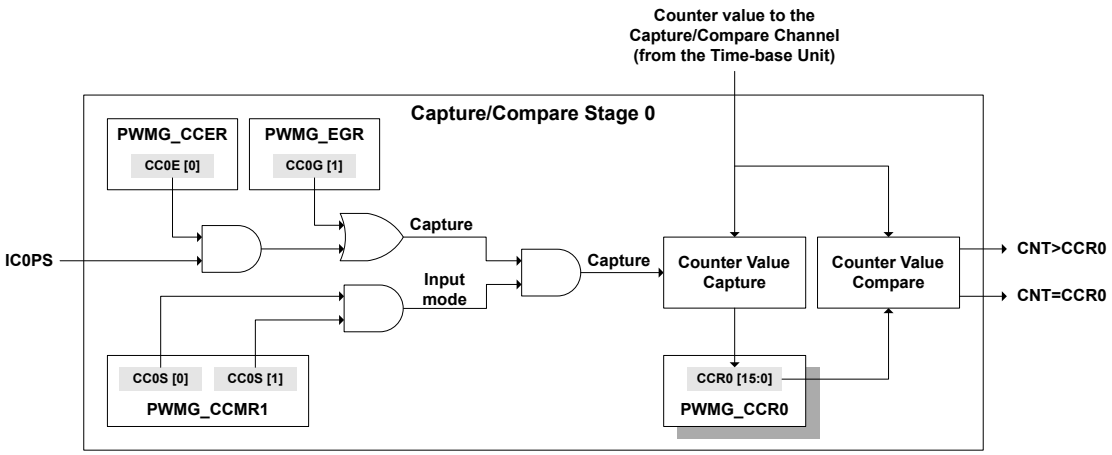

Figure 12-20 Capture/Compare Stage 0

Capture/Compare Stage 0 in input mode (`PWMG_CCMR1.CC0S` = 0x3)

As shown in the figure above, when the `PWMG_CCMR1.CC0S` = 0x3, the `Input mode` signal allows the `Capture` signal to pass to the Counter Value Capture block. The positive pulse of the `Capture` signal generates the capture event, which causes the counter value (content of the `PWMG_CNT` register) to be written to the `PWMG_CCR0` register.

Positive pulse of the `Capture` signal can be generated as follows:

- By software, if the `CC0G` bit of the `PWMG_EGR` register is set to 1.
- By hardware, as a result of the appearance of the `IC0PS` signal pulse if the `CC0E` bit of the `PWMG_CCER` register is set, where `IC0PS` is the signal from the Channel 0 Input Stage.

When a capture event occurs, the `CC0IF` bit in the `PWMG_SR` register is also set.

Capture/Compare Stage 0 in output mode (`PWMG_CCMR1.CC0S` = 0x0)

In this mode, the `PWMG_CCR0` register value is compared with the `PWMG_CNT` register value in the Counter Value Compare block. As a result, the Counter Value Compare block generates the `CNT=CCR0` and `CNT>CCR0` output signals:

- `CNT=CCR0` signal is generated when the `PWMG_CCR0` register value is equal to the `PWMG_CNT` register value.
- `CNT>CCR0` signal is generated when the `PWMG_CNT` register value is greater than the `PWMG_CCR0` register value.

The `CNT=CCR0` and `CNT>CCR0` signals are then used in the Channel 0 Output Stage.

12.2.1.4. Shadow Registers

Some of the PWMG registers are buffered. This register type consists of a pair:

- A software-accessible register (also called the preload register or buffer register).
- A software-inaccessible register (also called the shadow register or active register).

The active registers are used during the PWMG operations. They are updated with the values from their corresponding preload registers when an *Update Event* (**UEV**) occurs.

In the figures, the buffered registers are shown as rectangles with the gray background.

12.2.2. PWMG Registers

The register regions of the PWMG are mapped in the BE-U1000 microcontroller Memory Map as shown in the table below.

Table 12-35 Memory address regions for PWMG registers

Region	Block Name	Size	Start Address	End Address
Periphery 0	PWMG_0	4 KB	0x1100_A000	0x1100_AFFF
Periphery 1	PWMG_1	4 KB	0x1300_A000	0x1300_AFFF

For more information, see Memory Map chapter.

This section covers the following topics:

- Register Map
- Register and Field Descriptions

12.2.2.1. Register Map

The following table provides a top-level summary of each register. To view the detailed description of a register, click the register name in the **Description** column.

Table 12-36 PWMG register map

Register	Offset	Description	Default Value
`PWMG_CR1`	0x00	PWMG Control Register 1	0x0
`PWMG_IER`	0x0C	PWMG Interrupt Enable Register	0x0
`PWMG_SR`	0x10	PWMG Status Register	0x0
`PWMG_EGR`	0x14	PWMG Event Generation Register	0x0
`PWMG_CCMR1`	0x18	PWMG Capture/Compare Mode Register 1	0x0
`PWMG_CCER`	0x20	PWMG Capture/Compare Enable Register	0x0
`PWMG_CNT`	0x24	PWMG Counter	0x0
`PWMG_PSC`	0x28	PWMG Prescaler	0x0
`PWMG_ARR`	0x2C	PWMG Auto-reload Register	0x0
`PWMG_CCR0`	0x34	PWMG Capture/Compare Register 0	0x0

12.2.2.2. Register Descriptions

In the tables below, the **R/W** column specifies how the application and the core can access the register fields.

12.2.2.2.1. PWMG Control Register 1 (`PWMG_CR1`)

The `PWMG_CR1` register is at offset 0x00.

The following table shows the register bit assignments.

Table 12-37 Fields for register: `PWMG_CR1`

Bits	Name	R/W	Description
[15:10]			Reserved (return 0 when read)
[9:8]	`CKD`	R/W	Clock division This bit field indicates the division ratio between the PWMG clock period (t_{CK_INT}) and the sampling clock period (t_S). The sampling clock is used by the digital filter (`TI0`). **Values:** **0x0:** $t_S=t_{CK\_INT}$ **0x1:** $t_S=2*t_{CK\_INT}$ **0x2:** $t_S=4*t_{CK\_INT}$ **0x3:** Reserved **Value After Reset:** 0x0
[7]	`ARPE`	R/W	Auto-reload preload enable **Values:** **0x0:** `PWMG_ARR` register is not buffered **0x1:** `PWMG_ARR` register is buffered **Value After Reset:** 0x0
[6:4]			Reserved (return 0 when read)
[3]	`OPM`	R/W	One pulse mode **Values:** **0x0:** The Counter is not stopped at an Update Event **0x1:** The Counter stops counting at the next Update Event (clearing the `CEN` bit) **Value After Reset:** 0x0
[2]			Reserved (return 0 when read)
[1]	`UDIS`	R/W	Update disable This bit is set and cleared by software to enable/disable *Update Event* (**UEV**) generation. **Values:** **0x0:** UEV enabled. The UEV is generated by one of the following events: ◦ Counter overflow ◦ Setting the `PWMG_EGR.UG` bit **0x1:** UEV disabled. The UEV is not generated, shadow registers keep their value (`ARR`, `PSC`, `CCR0`). However, the Counter and the Prescaler are reinitialized if the `PWMG_EGR.UG` bit is set. **Value After Reset:** 0x0
[0]	`CEN`	R/W	Counter enable

Table 12-37 Fields for register: `PWMG_CR1` (continued)

Bits	Name	R/W	Description
			Values: **0x0:** Counter disabled **0x1:** Counter enabled **Value After Reset:** 0x0

12.2.2.2.2. PWMG Interrupt Enable Register (`PWMG_IER`)

The `PWMG_IER` register is at offset 0x0C.

The following table shows the register bit assignments.

Table 12-38 Fields for register: `PWMG_IER`

Bits	Name	R/W	Description
[15]	`INT_CLEAR`	W	Interrupt clear Write 0x1 to this bit to reset the setted interrupt. Writing 0x0 has no effect. **Value After Reset:** 0x0
[14:2]			Reserved
[1]	`CC0IE`	R/W	Capture/Compare Channel 0 interrupt enable **Values:** **0x0:** CC0 interrupt disabled **0x1:** CC0 interrupt enabled **Value After Reset:** 0x0
[0]	`UIE`	R/W	Update interrupt enable **Values:** **0x0:** Update interrupt disabled **0x1:** Update interrupt enabled **Value After Reset:** 0x0

12.2.2.2.3. PWMG Status Register (`PWMG_SR`)

The `PWMG_SR` register is at offset 0x10.

The following table shows the register bit assignments.

Table 12-39 Fields for register: `PWMG_SR`

Bits	Name	R/W	Description
[15:10]			Reserved
[9]	`CC0OF`	RW0C	Capture/Compare Channel 0 overcapture flag This flag is set by hardware only when the corresponding channel is configured in input capture mode. It is cleared by software by writing it to 0x0. **Values:**

Table 12-39 Fields for register: PWMG_SR (continued)

Bits	Name	R/W	Description
			0x0: No overcapture has been detected **0x1:** The counter value has been captured in PWMG_CCR0 register while CC0IF flag was already set **Value After Reset:** 0x0
[8:2]			Reserved
[1]	CC0IF	RW0C	Capture/Compare Channel 0 interrupt flag **If channel CC0 is configured as output** (PWMG_CCMR1.CC0S = 0x0): This flag is set by hardware when the counter matches the value for compare. It is cleared by software. **Values:** **0x0:** No match **0x1:** The value in the PWMG_CNT counter register matches the value in the PWMG_CCR0 capture/compare register. When the value in PWMG_CCR0 is greater than the value in the PWMG_ARR auto-reload register, the CC0IF bit goes high on a counter overflow. **If channel CC0 is configured as input** (PWMG_CCMR1.CC0S = 0x3): This bit is set by hardware on a capture. It is cleared by software or by reading the PWMG_CCR0 register. **Values:** **0x0:** No input capture occurred **0x1:** The counter value has been captured in PWMG_CCR0 register (An edge has been detected on IC0 which matches the selected polarity). **Value After Reset:** 0x0
[0]	UIF	RW0C	Update interrupt flag This bit is set by hardware on an Update Event. It is cleared by software. **Values:** **0x0:** No update occurred **0x1:** Update interrupt pending. This bit is set by hardware when the registers are updated: ◦ At overflow, if UDIS=0x0 in the PWMG_CR1 register ◦ When the counter is reinitialized by software using the UG bit in PWMG_EGR register, if UDIS=0x0 in the PWMG_CR1 register. **Value After Reset:** 0x0

12.2.2.2.4. PWMG Event Generation Register (PWMG_EGR)

The PWMG_EGR register is at offset 0x14.

The following table shows the register bit assignments.

Table 12-40 Fields for register: PWMG_EGR

Bits	Name	R/W	Description
[15:2]			Reserved
[1]	CC0G	W	Capture/Compare 0 generation This bit is set by software to generate an event and is automatically cleared by hardware. **Values:** **0x0:** No action **0x1:** A capture/compare event is generated on channel 0: **If the channel CC0 is configured as output** (PWMG_CCMR1.CC0S = 0x0): PWMG_SR.CC0IF flag is set. Corresponding interrupt is sent if enabled. **If channel CC0 is configured as input** (PWMG_CCMR1.CC0S = 0x3): The current value of the counter is captured in PWMG_CCR0 register. The PWMG_SR.CC0IF flag is set, the corresponding interrupt is sent if enabled. The PWMG_SR.CC0OF flag is set if the CC0IF flag was already high. **Value After Reset:** 0x0
[0]	UG	W	Update generation This bit can be set by software. It is automatically cleared by hardware. **Values:** **0x0:** No action **0x1:** It reinitializes the counter and generates an update of the registers. Note that the Prescaler counter is also cleared; however, the Prescaler ratio is not affected. **Value After Reset:** 0x0

12.2.2.2.5. PWMG Capture/Compare Mode Register 1 (PWMG_CCMR1)

The PWMG_CCMR1 register is at offset 0x18.

The channel 0 can be configured in either input (capture) or output (compare) mode. The direction of the channel is defined by configuring the CC0S bit field. The other bits in this register have different functions depending on the mode (input/output). For a given bit, OC0x describes its function when the channel 0 is configured as output, while IC0x describes its function when the channel 0 is configured as input. Therefore, the user must be aware that the same bit can have a different meaning for the input stage and for the output stage.

The following table shows the register bit assignments.

Table 12-41 Fields for register: PWMG_CCMR1

Bits	Name	R/W	Description
[15:8]			Reserved

Table 12-41 Fields for register: PWMG_CCMR1 (continued)

Bits	Name	R/W	Description
[7:2]	IC0* or OC0*	R/W	The functions of these bits depend on the value of the CC0S bit field. For details, see the table **PWMG_CCMR1[7:2] bit functions for different CC0S values** below.
[1:0]	CC0S	R/W	Capture/Compare Channel 0 selection This bit field defines the direction of the channel 0 (input/output) as well as the used input. **Values:** **0x0:** CC0 channel is configured as output **0x1:** Reserved **0x2:** Reserved **0x3:** CC0 channel is configured as input, IC0 is mapped on TI0 CC0S bits are writable only when the channel is OFF (CC0E = 0x0 in PWMG_CCER). **Value After Reset:** 0x0

PWMG_CCMR1[7:2] functions for different PWMG_CCMR1.CC0S bit filed values are shown in the table below.

Table 12-42 PWMG_CCMR1[7:2] bit functions for different CC0S values

Bits	Name	R/W	Description
Channel 0 Output Compare mode (CC0S = 0x0)			
[7]			Reserved
[6:4]	OC0M	R/W	Output Compare Channel 0 mode These bits define the behavior of the output reference signal OC0REF, from which the OC0 is derived. The OC0REF is active high, whereas the active level of the OC0 depends on CC0P bit. **Values:** **0x0** (Frozen): The comparison between the output compare register PWMG_CCR0 and the counter PWMG_CNT has no effect on the outputs (This mode is used to generate a timing base). **0x1:** Set channel 0 to active level on match. The OC0REF signal is forced high when the counter PWMG_CNT matches the PWMG_CCR0 register. **0x2:** Set channel 0 to inactive level on match. The OC0REF signal is forced low when the counter PWMG_CNT matches the PWMG_CCR0 register. **0x3** (Toggle): The OC0REF toggles when PWMG_CNT=PWMG_CCR0. **0x4** (Force inactive level): The OC0REF is forced low. **0x5** (Force active level): The OC0REF is forced high. **0x6** (PWM mode 1): The channel 0 is active as long as PWMG_CNT<PWMG_CCR0. Otherwise, it is inactive. **0x7** (PWM mode 2): The channel 0 is inactive as long as PWMG_CNT<PWMG_CCR0. Otherwise, it is active.

Table 12-42 PWMG_CCMR1[7:2] bit functions for different CC0S values (continued)

Bits	Name	R/W	Description
			In PWM mode 1 or 2, the OCR0EF level changes only when the result of the comparison changes or when the output compare mode switches from the “Frozen” mode to the “PWM” mode. **Value After Reset:** 0x0
[3]	OC0PE	R/W	Output Compare Channel 0 preload enable **Values:** **0x0:** Preload register for PWMG_CCR0 disabled. PWMG_CCR0 can be written at anytime; the new value is taken into account immediately. **0x1:** Preload register for PWMG_CCR0 enabled. Read/Write operations access the preload register. PWMG_CCR0 preload value is loaded into the active register at each Update Event. The PWM mode can be used without validating the preload register only in one pulse mode (OPM bit set in PWMG_CR1 register). Otherwise, the behavior is not guaranteed. **Value After Reset:** 0x0
[2]			Reserved
Channel 0 Input Capture mode (CC0S = 0x3)			
[7:4]	IC0F	R/W	Input Capture Channel 0 filter This bit field defines both the sampling frequency for the TI0 input and the length of the digital filter applied to TI0. The digital filter consists of an event counter that requires N consecutive events to validate a transition at the output. **Values:** **0x0:** No filter, sampling is done at f_S ($f_{SAMPLING}=f_S$) **0x1:** $f_{SAMPLING}=f_{CK\_INT}$, N=2 **0x2:** $f_{SAMPLING}=f_{CK\_INT}$, N=4 **0x3:** $f_{SAMPLING}=f_{CK\_INT}$, N=8 **0x4:** $f_{SAMPLING}=f_S/2$, N=6 **0x5:** $f_{SAMPLING}=f_S/2$, N=8 **0x6:** $f_{SAMPLING}=f_S/4$, N=6 **0x7:** $f_{SAMPLING}=f_S/4$, N=8 **0x8:** $f_{SAMPLING}=f_S/8$, N=6 **0x9:** $f_{SAMPLING}=f_S/8$, N=8 **0xA:** $f_{SAMPLING}=f_S/16$, N=5 **0xB:** $f_{SAMPLING}=f_S/16$, N=6 **0xC:** $f_{SAMPLING}=f_S/16$, N=8 **0xD:** $f_{SAMPLING}=f_S/32$, N=5 **0xE:** $f_{SAMPLING}=f_S/32$, N=6 **0xF:** $f_{SAMPLING}=f_S/32$, N=8 **Value After Reset:** 0x0
[3:2]	IC0PSC	R/W	Input Capture Channel 0 prescaler This bit field defines the Prescaler ratio for the CC0 input (IC0).

Table 12-42 `PWMG_CCMR1[7:2]` bit functions for different `CC0S` values (continued)

Bits	Name	R/W	Description
			The Prescaler is reset when the `PWMG_CCER.CC0E`=0x0. **Values:** **0x0:** No Prescaler, capture is done each time an edge is detected at the capture input **0x1:** Capture is done once every 2 events **0x2:** Capture is done once every 4 events **0x3:** Capture is done once every 8 events **Value After Reset:** 0x0

12.2.2.2.6. PWMG Capture/Compare Enable Register (`PWMG_CCER`)

The `PWMG_CCER` register is at offset 0x20.

The following table shows the register bit assignments.

Table 12-43 Fields for register: `PWMG_CCER`

Bits	Name	R/W	Description
[15:2]			Reserved
[1]	`CC0P`	R/W	Capture/Compare Channel 0 output polarity **If the channel CC0 is configured as output** (`PWMG_CCMR1.CC0S` = 0x0): **Values:** **0x0:** `OC0` active high **0x1:** `OC0` active low **If the channel CC0 is configured as input** (`PWMG_CCMR1.CC0S` = 0x3): This bit selects the `TI0FP0` active polarity. **Values:** **0x0:** Non-inverted/rising edge **0x1:** Inverted/falling edge **Value After Reset:** 0x0
[0]	`CC0E`	R/W	Capture/Compare Channel 0 output enable **If the channel CC0 is configured as output** (`PWMG_CCMR1.CC0S` = 0x0): **Values:** **0x0** (Off): The `OC0` is not active **0x1** (On): The `OC0` signal is output on the corresponding output pin **If the channel CC0 is configured as input** (`PWMG_CCMR1.CC0S` = 0x3): This bit determines if a capture of the counter value can actually be done into the input capture/compare register 0 (`PWMG_CCR0`) or not. **Values:** **0x0:** Capture disabled **0x1:** Capture enabled **Value After Reset:** 0x0

12.2.2.2.7. PWMG Counter (PWMG_CNT)

The PWMG_CNT register is at offset 0x24.

The following table shows the register bit assignments.

Table 12-44 Fields for register: PWMG_CNT

Bits	Name	R/W	Description
[15:0]	CNT	R/W	Counter value **Value After Reset:** 0x0

12.2.2.2.8. PWMG Prescaler (PWMG_PSC)

The PWMG_PSC register is at offset 0x28.

The following table shows the register bit assignments.

Table 12-45 Fields for register: PWMG_PSC

Bits	Name	R/W	Description
[15:0]	PSC	R/W	Prescaler value The counter clock frequency (CK_CNT) is equal to f_{CK_INT} / (PSC + 1). The PSC contains the value to be loaded in the active prescaler register at each Update Event (including when the Counter is cleared through the UG bit of the PWMG_EGR register). **Value After Reset:** 0x0

12.2.2.2.9. PWMG Auto-reload Register (PWMG_ARR)

The PWMG_ARR register is at offset 0x2C.

The following table shows the register bit assignments.

Table 12-46 Fields for register: PWMG_ARR

Bits	Name	R/W	Description
[15:0]	ARR	R/W	Auto-reload value ARR contains the value to be loaded in the actual auto-reload register. This register can be buffered (see ARPE bit description in the PWMG_CR1 register). The Counter is blocked while the auto-reload value (ARR) is null. **Value After Reset:** 0x0

12.2.2.2.10. PWMG Capture/Compare Register 0 (PWMG_CCR0)

The PWMG_CCR0 register is at offset 0x34.

The following table shows the register bit assignments.

Table 12-47 Fields for register: `PWMG_CCR0`

Bits	Name	R/W	Description
[15:0]	`CCR0`	R/W	Capture/Compare Channel 0 value **If the channel CC0 is configured as output** (`PWMG_CCMR1.CC0S` = 0x0): `CCR0` contains the value to be loaded in the actual capture/compare 0 register (preload value). This value is loaded permanently if the preload feature is not selected in the `PWMG_CCMR1` register (bit `OC0PE`). Otherwise, the preload value is copied into the active capture/compare 0 register when an *Update Event* (**UEV**) occurs. The active capture/compare register contains the value to be compared to the counter `PWMG_CNT` and signaled on `OC0` output. **If the channel CC0 is configured as input** (`PWMG_CCMR1.CC0S` = 0x3): `CCR0` contains the counter value transferred by the last input capture 0 event (`IC0`). The `PWMG_CCR0` register is read-only and cannot be programmed. **Value After Reset:** 0x0

12.3. TIM

This chapter describes the *Timer Module* (**TIM**) of the BE-U1000 microcontroller. The BE-U1000 microcontroller contains two TIMs.

Each TIM implements four identical but independently programmable timers. These timers are accessed through the Register Interface, as shown in the diagram below.

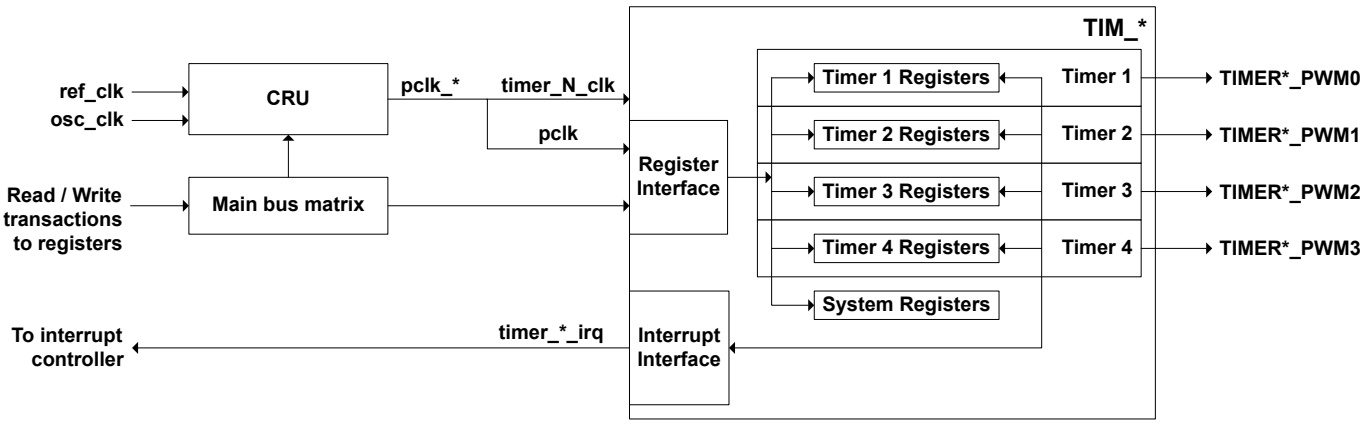

Figure 12-21 TIM block diagram

In the figure above an asterisk symbol (*) indicates 0 for TIM_0 and 1 for TIM_1.

TIM features:

- Timer width: 32 bits
- Two operation modes: free-running and user-defined count
- Independent clocking of timers
- The LOW and HIGH period widths can be programmed separately via the TIM registers
- Option to include the toggle output that toggles whenever the timer counter reloads

- Option to enable programmable *Pulse Width Modulation* (**PWM**) of timer toggle outputs
- Option to include PWM of the timer toggle outputs with 0% and 100% duty cycles

12.3.1. TIM Functional Description

12.3.1.1. Timer Operation

Each timer counts down from a programmed value and generates an interrupt when the count reaches zero.

The initial value for each timer — the value from which it counts down — is loaded into the timer using the appropriate load count register (`TIMER<N>LOAD_COUNT`). Two events can cause a timer to load the initial count from its (`TIMER<N>LOAD_COUNT`) register:

- A timer is enabled after being reset or disabled.
- A timer counts down to 0.

All interrupt status registers (`TIMER<N>INT_STATUS`) and end-of-interrupt registers (`TIMER<N>EOI`) can be accessed at any time.

12.3.1.1.1. Using a Timer

The procedure illustrated in the following figure is a basic flow to follow when programming the Timer.

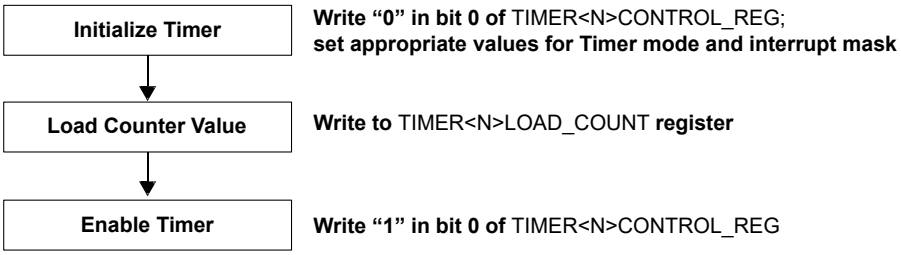

Figure 12-22 Timer usage flow

Before writing to the `TIMER<N>LOAD_COUNT` register, disable the timer by writing a 0 to the `TMR_EN` bit of `TIMER<N>CONTROL_REG` in order to avoid potential synchronization problems.

To use a timer, perform the following operations:

1. Initialize the timer through the `TIMER<N>CONTROL_REG` register:
    a. Disable the timer by writing a 0 to the `TMR_EN` bit (bit 0).
    b. Program the timer mode, user-defined or free-running, by writing 1 or 0, respectively, into the timer mode bit (bit 1).
    c. Set the interrupt mask as either masked or not masked by writing 1 or 0, respectively, into the timer interrupt mask bit (bit 2).
2. Load the timer counter value into the `TIMER<N>LOAD_COUNT` register.
3. Enable the timer by writing 1 to bit 0 of `TIMER<N>CONTROL_REG`.

12.3.1.1.2. Enabling a Timer

You must use the bit 0 of the `TIMER<N>CONTROL_REG`, where N is in the range from 1 to 4, to either enable or disable a timer.

To enable a timer, write 1 into the bit 0 of the `TIMER<N>CONTROL_REG` register.

12.3.1.1.3. Disabling a Timer

You must use the bit 0 of the TIMER<N>CONTROL_REG, where N is in the range from 1 to 4, to either enable or disable a timer.

To disable a timer, write 0 into the bit 0 of the TIMER<N>CONTROL_REG register.

When a timer is enabled and running, its counter decrements on each rising edge of its clock signal. When a timer transitions from disabled to enabled, the current value of its TIMER<N>LOAD_COUNT register is loaded into the timer counter on the next rising edge of the timer clock.

When the timer enable bit is de-asserted and the timer stops running, the timer counter and any associated registers in the timer clock domain, are asynchronously reset.

When the timer enable bit is asserted, the initial value is loaded into the timer counter. 0 is always read back when the timer is not enabled; otherwise, the current value of the timer (TIMER<N>CURRENT_VALUE register) is read back.

12.3.1.1.4. Loading a Timer Countdown Value

When a timer counter is enabled after being reset or disabled, the count value is loaded from the TIMER<N>LOAD_COUNT register; this occurs in both free-running and user-defined count modes.

When a timer counts down to 0, it loads one of two values, depending on the timer operating mode:

- **User-defined count mode** — the timer loads the current value of the TIMER<N>LOAD_COUNT register. Use this mode if you want a fixed, timed interrupt. Designate this mode by writing 1 into the bit 1 of TIMER<N>CONTROL_REG.
- **Free-running mode** — the timer loads the maximum value, which is dependent on the timer counter width. Therefore each bit of the TIMER<N>LOAD_COUNT register is loaded with 1. The timer counter wrapping to its maximum value allows time to reprogram or disable the timer before another interrupt occurs. Use this mode if you want a single timed interrupt. Designate this mode by writing 0 to bit 1 of TIMER<N>CONTROL_REG.

12.3.1.1.5. Working with Interrupts

The TIMER<N>INT_STATUS and TIMER<N>EOI registers handle interrupts in order to ensure safe operation of the interrupt clearing.

Provided the timer is enabled, its interrupt remains asserted until it is cleared by reading one of two end-of-interrupt registers (the TIMER<N>EOI individual register or TIMERS_EOI global end-of-interrupt register). When the timer is disabled, the timer interrupt is cleared. You can clear an individual timer interrupt by reading its TIMER<N>EOI register. You can clear all active timer interrupts at once by reading the TIMERS_EOI global register or by disabling the timer.

When reading the TIMERS_EOI register, timer interrupts are cleared. If an end-of-interrupt register is read during the time when the internal interrupt signal is high, the timer interrupt is set. This occurs because setting the timer interrupts takes precedence over clearing them.

You can query the interrupt status of an individual timer without clearing its interrupt by reading the TIMER<N>INT_STATUS register. You can query the interrupt status of all timers without clearing the interrupts by reading the global TIMERS_INT_STATUS register.

Each individual timer interrupt can be masked using its TIMER<N>CONTROL_REG register. To mask an interrupt, write 1 into the TMR_INT_MASK bit of TIMER<N>CONTROL_REG. If all individual timer interrupts are masked, then the combined interrupt is also masked.

12.3.1.1.6. Generating Toggled Outputs

You can configure a timer in order to generate an output that toggles whenever the timer counter reaches 0.

12.3.1.1.6.1. Pulse Width Modulation of Toggle Outputs

The `timer_N_toggle` allows the toggle output from each of the timers to be pulse-width modulated. If `TIMER<N>CONTROL_REG[4]` (`TIMER_PWM` bit) is set to 1, the HIGH and LOW periods of the toggle outputs can be controlled separately by programming the `TIMER<N>LOAD_COUNT2` and `TIMER<N>LOAD_COUNT` registers.

The pulse widths of the toggle outputs are controlled as follows:

- Width of `timer_N_toggle` `HIGH period = (TIMER<N>LOAD_COUNT2 + 1) * timer_N_clk clock period`
- Width of `timer_N_toggle` `LOW period = (TIMER<N>LOAD_COUNT + 1) * timer_N_clk clock period`

12.3.1.1.6.2. Pulse Width Modulation with 0% and 100% Duty Cycles

The TIM supports programming the 0% duty cycle and 100% duty cycle *Pulse Width Modulation* (**PWM**) for toggle outputs (`timer_N_toggle`) through the `TIMER<N>LOAD_COUNT` and `TIMER<N>LOAD_COUNT2` registers, when 0% and 100% duty cycle mode is enabled. You can enable the duty cycle mode either by setting the `TIMER<N>CONTROL_REG[4]` register.

The `TIMER<N>LOAD_COUNT` register defines the LOW period and the `TIMER<N>LOAD_COUNT2` register defines the HIGH period values.

The definition of the duty cycles (with `TIMER<N>LOAD_COUNT` and `TIMER<N>LOAD_COUNT2`) is as follows (note that the HIGH period signifies the duty cycle number):

- 0% duty cycle — Continuous LOW and no HIGH
  - `TIMER<N>LOAD_COUNT` = Do not care
  - `TIMER<N>LOAD_COUNT2` = 0
- 100% duty cycle — No LOW period and continuous HIGH
  - `TIMER<N>LOAD_COUNT` = 0
  - `TIMER<N>LOAD_COUNT2` = Do not care
- Other duty cycle – When 0% and 100% duty cycle mode is enabled (with timer PWM mode and the user-defined count mode enabled), the definition of the toggle HIGH and LOW period changes as follows for a duty cycle other than 0% or 100%:
  - Width of `timer_N_toggle` `HIGH period = TIMER<N>LOAD_COUNT2 * timer_N_clk clock period`
  - Width of `timer_N_toggle` `LOW period = TIMER<N>LOAD_COUNT * timer_N_clk clock period`

The above definition is applicable only if the 0% and 100% duty cycle mode is enabled along with the timer *Pulse Width Modulation* (**PWM**) mode and the user-defined count mode. If any of these modes are not enabled, that is, if the PWM mode or user-defined mode is not enabled, then the new definition of the HIGH and LOW period is not applicable. The previous definition of the HIGH and LOW period is applicable.

The following table provides information on the relation between the duty cycle, TIMER<N>LOAD_COUNT and TIMER<N>LOAD_COUNT2 values, considering that the maximum value is 100.

Table 12-48 Duty cycle, TIMER<N>LOAD_COUNT and TIMER<N>LOAD_COUNT2 relationship

Duty Cycle (%)	TIMER<N>LOAD_COUNT	TIMER<N>LOAD_COUNT2
0	X	0
1	99	1
2	98	2
3	97	3
4	96	4
...	...	...
96	4	96
97	3	97
98	2	98
99	1	99
100	0	100

When TIMER<N>LOAD_COUNT=0 and TIMER<N>LOAD_COUNT2=0, the timer considers this as 100% by providing higher priority to the TIMER<N>LOAD_COUNT.

Here are some points to consider when configuring the *Pulse Width Modulation* (**PWM**) with 0% and 100% duty cycle feature:

- The 0% and 100% duty cycle mode is enabled when the TIM is programmed with the following:
  - TIMER<N>CONTROL_REG[4] (0% and 100% duty cycle mode enable bit)
  - TIMER<N>CONTROL_REG[3] (PWM enable bit)
- When toggle output is in 0% and 100% duty cycle mode, the timer interrupts are generated.
- The 0% and 100% duty cycle mode can be enabled only if the following bits are set:
  - TIMER<N>CONTROL_REG[3] (PWM enable bit)
  - TIMER<N>CONTROL_REG[1] (Timer mode bit)
  - TIMER<N>CONTROL_REG[4] (0% and 100% duty cycle mode bit)

  If any of these modes are not enabled, that is if the PWM mode or user-defined mode is not enabled, then the timer operates in the normal PWM mode or free running mode.
- The 0% and 100% duty cycle mode can be enabled by programming 0x0 into the TIMER<N>LOAD_COUNT or TIMER<N>LOAD_COUNT2 register. If any other combination of values is programmed, then it operates in the normal PWM mode.
- The TIMER<N>CONTROL_REG[4] bit enables the 0% and 100% duty cycle mode. You must not set any random value to this bit.

12.3.2. TIM Registers

Register regions of the TIMs are mapped in the BE-U1000 memory map as the following table shows.

Table 12-49 Memory address regions for TIM registers

Region	Block Name	Size	Start Address	End Address
Periphery 0	TIM_0	4 KB	0x1100_7000	0x1100_7FFF
Periphery 1	TIM_1	4 KB	0x1300_7000	0x1300_7FFF

 For more information, see Memory Map chapter.

This section covers the following topics:

- Register Map
- Register Descriptions

12.3.2.1. Register Map

This section tables contain information about the register memory map.

The following table lists the address ranges of the registers for each timer; they are aligned to 32-bit boundaries.

Table 12-50 TIM address range

Address Range (Base +)	Function
0x00 to 0x10	Timer 1 registers
0x14 to 0x24	Timer 2 registers
0x28 to 0x38	Timer 3 registers
0x3C to 0x4C	Timer 4 registers
0xA0 to 0xA8	System registers

The following table provides high-level summary of each register. To view the detailed description of a register, click the register name in the **Description** column.

Table 12-51 TIM register map

Name	Offset	Description	Reset Value
TIMER_1_LOAD_COUNT	0x00	Timer N Load Count Register Used to specify the value to be loaded into timer N.	0x0
TIMER_2_LOAD_COUNT	0x14		
TIMER_3_LOAD_COUNT	0x28		
TIMER_4_LOAD_COUNT	0x3C		
TIMER_1_LOAD_COUNT2	0xB0	Timer N Load Count2 Register	0x0

Table 12-51 TIM register map (continued)

Name	Offset	Description	Reset Value
TIMER_2_LOAD_COUNT2	0xB4	Used to specify the value to be loaded into timer N when toggle output changes from 0 to 1.	
TIMER_3_LOAD_COUNT2	0xB8		
TIMER_4_LOAD_COUNT2	0xBC		
TIMER_1_CURRENT_VALUE	0x04	Timer N Current Value Register Holds the Current Value of timer N.	0xFFFFFFFF
TIMER_2_CURRENT_VALUE	0x18		
TIMER_3_CURRENT_VALUE	0x2C		
TIMER_4_CURRENT_VALUE	0x40		
TIMER_1_CONTROL_REG	0x08	Timer N Control Register Controls enabling, operating mode (free-running or defined-count), and interrupt mask of timer N.	0x0
TIMER_2_CONTROL_REG	0x1C		
TIMER_3_CONTROL_REG	0x30		
TIMER_4_CONTROL_REG	0x44		
TIMER_1_EOI	0x0C	Timer N End-of-Interrupt Register Clears the interrupt from timer N.	0x0
TIMER_2_EOI	0x20		
TIMER_3_EOI	0x34		
TIMER_4_EOI	0x48		
TIMER_1_INT_STATUS	0x10	Timer N Interrupt Status Register Contains the interrupt status for timer N.	0x0
TIMER_2_INT_STATUS	0x24		
TIMER_3_INT_STATUS	0x38		
TIMER_4_INT_STATUS	0x4C		
TIMERS_INT_STATUS	0xA0	Timers Interrupt Status Register Contains the interrupt status of all timers in the component.	0x0
TIMERS_EOI	0xA4	Timers End-of-Interrupt Register Returns all zeroes (0) and clears all active interrupts.	0x0
TIMERS_RAW_INT_STATUS	0xA8	Timers Raw Interrupt Status Register Contains the unmasked interrupt status of all timers in the subsystems.	0x0

12.3.2.2. Register Descriptions

In the tables below, the **R/W** column specifies how the application and the core can access the register fields.

⚠ Reserved bits in the TIM registers cannot be used.

12.3.2.2.1. Timer N Load Count Register (`TIMER<N>LOAD_COUNT`)

The `TIMER<N>LOAD_COUNT` register is at offset 0x00 + (n*0x14) where n is from 0 to 3.

This register is used to specify the value to be loaded into timer N.

The following table shows the register bit assignments.

Table 12-52 Fields for register: `TIMER<N>LOAD_COUNT`

Bits	Name	R/W	Description
[31:0]	`TIMER<N>LOADCOUNT`	R/W	Value to be loaded into Timer N. This is the value from which the timer starts counting. Any value written to this register is loaded into the associated timer. **Value After Reset:** 0x0

12.3.2.2.2. Timer N Load Count 2 Register (`TIMER<N>LOAD_COUNT2`)

The `TIMER<N>LOAD_COUNT2` register is at offset 0xB0 + (n*0x4) where n is from 0 to 3.

This register is used to specify the value to be loaded into timer N when the toggle output changes from 0 to 1.

The LOW and HIGH period widths of the timer toggle outputs can be separately programmed via the registers `TIMER<N>LOAD_COUNT` and `TIMER<N>LOAD_COUNT2` respectively, if `TIMER<N>CONTROL_REG[3]` is set to '1'.

The following table shows the register bit assignments.

Table 12-53 Fields for register: `TIMER<N>LOAD_COUNT2`

Bits	Name	R/W	Description
[31:0]	`TIMER<N>LOADCOUNT2`	R/W	Value to be loaded into timer N when `timer_N_toggle` output changes from 0 to 1. This value determines the width of the HIGH period of the `timer_N_toggle` output. **Value After Reset:** 0x0

12.3.2.2.3. Timer N Current Value Register (`TIMER<N>CURRENT_VALUE`)

The `TIMER<N>CURRENT_VALUE` register is at offset 0x04 + (n*0x14) where n is from 0 to 3.

This register holds the Current Value of timer N.

The following table shows the register bit assignments.

Table 12-54 Fields for register: `TIMER<N>CURRENT_VALUE`

Bits	Name	R/W	Description
[31:0]	`TIMER<N>CURRENTVAL`	R	Current value of Timer.

Table 12-54 Fields for register: `TIMER<N>CURRENT_VALUE` (continued)

Bits	Name	R/W	Description
			Value After Reset: 0xFFFFFFFF **Volatile:** true

12.3.2.2.4. Timer N Control Register (`TIMER<N>CONTROL_REG`)

The `TIMER<N>CONTROL_REG` register is at offset 0x08 + (n*0x14) where n is from 0 to 3.

This register controls enabling, operating mode (free-running or defined-count), and interrupt mask of timer N.

The following table shows the register bit assignments.

Table 12-55 Fields for register: `TIMER<N>CONTROL_REG`

Bits	Name	R/W	Description
[31:5]			Reserved
[4]	`TIMER_0N100PWM_EN`	R/W	This bit allows a user to enable or disable the 0% and 100% PWM duty cycle mode for the timer. **0x0**: Mode disabled **0x1**: Mode enabled **Value After Reset:** 0x0
[3]	`TIMER_PWM`	R/W	*Pulse Width Modulation* (**PWM**) of the `timer_N_toggle` output **0x0**: PWM for `timer_N_toggle` o/p is disabled **0x1**: PWM for `timer_N_toggle` o/p is enabled **Value After Reset:** 0x0
[2]	`TMR_INT_MASK`	R/W	Interrupt mask **0x0:** Timer N interrupt is unmasked **0x1:** Timer N interrupt is masked **Value After Reset:** 0x0
[1]	`TMR_MODE`	R/W	Operating mode **0x0:** Free-running mode of operation **0x1:** User-defined mode of operation For more information about these modes, see Timer Operation You must set the `TIMER<N>LOAD_COUNT` to all 1s before enabling the timer in free-running mode. **Value After Reset:** 0x0
[0]	`TMR_EN`	R/W	Timer enable **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x0

12.3.2.2.5. Timer N End-of-Interrupt Register (`TIMER<N>EOI`)

The `TIMER<N>EOI` register is at offset 0x0C + (n*0x14) where n is from 0 to 3.

This register clears the interrupt from timer N.

The following table shows the register bit assignments.

Table 12-56 Fields for register: `TIMER<N>EOI`

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	`TIMER<N>EOI`	R	A read from this register returns zero (0) and clears the timer N interrupt. **Value After Reset:** 0x0

12.3.2.2.6. Timer N Interrupt Status Register (`TIMER<N>INT_STATUS`)

The `TIMER<N>INT_STATUS` register is at offset 0x10 + (n*0x14) where n is from 0 to 3.

This register contains the interrupt status for timer N.

The following table shows the register bit assignments.

Table 12-57 Fields for register: `TIMER<N>INT_STATUS`

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	`TIMER<N>INTSTAT`	R	This bit contains the interrupt status for timer N. **0x0**: Timer N interrupt is inactive **0x1**: Timer N interrupt is active **Value After Reset:** 0x0 **Volatile:** true

12.3.2.2.7. Timers Interrupt Status Register (`TIMERS_INT_STATUS`)

The `TIMERS_INT_STATUS` register is at offset 0xA0.

This register contains the interrupt status of all timers in the subsystem.

The following table shows the register bit assignments.

Table 12-58 Fields for register: `TIMERS_INT_STATUS`

Bits	Name	R/W	Description
[31:4]			Reserved
[3:0]	`TIMERSINTSTATUS`	R	This bit field contains the interrupt status of all timers in the subsystem. If a bit of this register is 0, then the corresponding timer interrupt is not active – and the corresponding interrupt could be on the timer interrupt bus. Similarly, if a bit of this register is 1, then the corresponding interrupt bit has been set in the interrupt bus. In both cases, the status reported is the status

Table 12-58 Fields for register: `TIMERS_INT_STATUS` (continued)

Bits	Name	R/W	Description
			after the interrupt mask has been applied. A read from this register does not clear any active interrupts: **0x0:** Timer interrupt is not active after masking **0x1:** Timer interrupt is active after masking **Value After Reset:** 0x0 **Volatile:** true

12.3.2.2.8. Timers End-of-Interrupt Register (`TIMERS_EOI`)

The `TIMERS_EOI` register is at offset 0xA4.

This register returns zero (0) and clears all active interrupts.

The following table shows the register bit assignments.

Table 12-59 Fields for register: `TIMERS_EOI`

Bits	Name	R/W	Description
[31:4]			Reserved
[3:0]	`TIMERSEOI`	R	A read from this register returns zero (0) and clears all active interrupts. **Value After Reset:** 0x0

12.3.2.2.9. Timers Raw Interrupt Status Register (`TIMERS_RAW_INT_STATUS`)

The `TIMERS_RAW_INT_STATUS` register is at offset 0xA8.

This register contains the unmasked interrupt status of all timers in the subsystem.

The following table shows the register bit assignments.

Table 12-60 Fields for register: `TIMERS_RAW_INT_STATUS`

Bits	Name	R/W	Description
[31:4]			Reserved
[3:0]	`TIMERSRAWINTSTAT`	R	The register contains the unmasked interrupt status of all timers in the subsystem. **0x0:** Timer interrupt is not active prior to masking **0x1:** Timer interrupt is active prior to masking **Value After Reset:** 0x0 **Volatile:** true

12.4. WDT

This section describes the *Watchdog Timer* (below referred to as **WDT**) of the BE-U1000 microcontroller. The WDT prevents system lockups that may be caused by conflicting parts or programs in the microcontroller. The WDT can be reset independently of other components.

The interrupt generated by WDT_0 (`wdt_0_irq`) is passed to the Core 0/1 *Core-Local Interrupt Controller* (**CLIC**). The reset signal generated by WDT_0 (`wdt0_sys_rst`) is passed to the Reset Sources Block of the Control Registers Unit. The state of the WDT_0 reset (`wdt0_sys_rst`) is also fixed in the `WDTHIT` bit of the `SYSSR0` CRU register.

The interrupt generated by WDT_1 (`wdt_1_irq`) is passed to the Core 1 command execution subsystem.

The following figures show the simplified block diagrams of the WDT_0 and WDT_1.

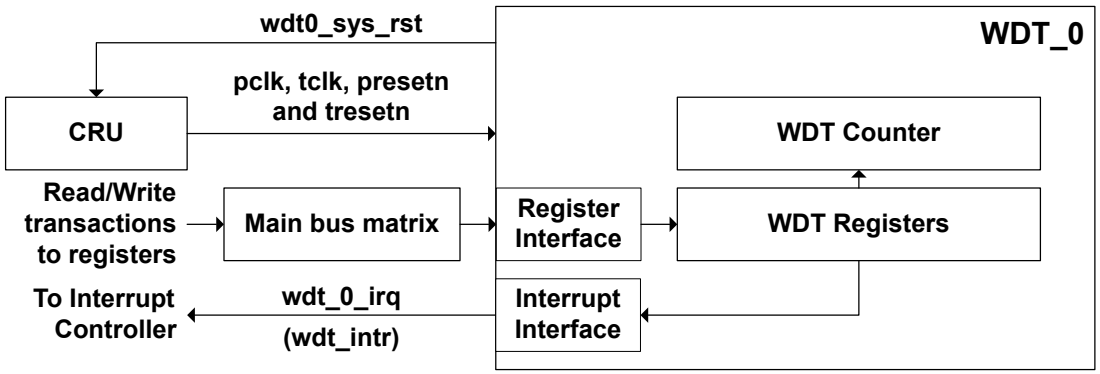

Figure 12-23 WDT_0 block diagram

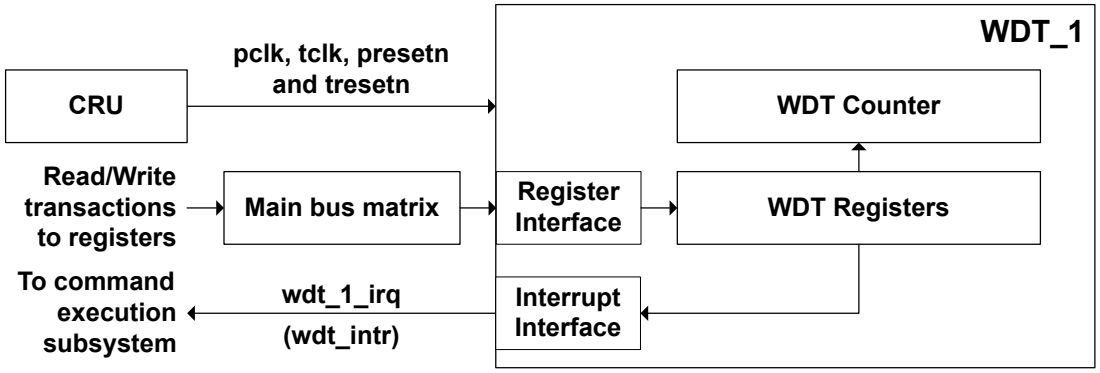

Figure 12-24 WDT_1 block diagram

WDT features:

- WDT counter width: 32 bits
- The counter counts down from a preset value to 0, indicating a timeout
- On a timeout, the WDT first generates an interrupt. If this interrupt is not cleared before a second timeout occurs, the WDT triggers a system reset.
- Programmable timeout period
- Programmable reset pulse length
- Protection against accidental restart of the WDT counter
- Protection against accidental disabling of the WDT

12.4.1. WDT Functional Description

This section describes the functional operation of the WDT.

This section covers the following topics:

- Interrupts
- Counter
- System Resets
- Reset Pulse Length

12.4.1.1. Interrupts

The WDT can be programmed to generate an interrupt (and then a system reset) when a timeout has occurred. If the response mode field (`RMOD`, bit 1) of the Control Register (`WDT_CR`) is set to 1, the WDT generates an interrupt. Even if the interrupt is cleared by the time a second timeout occurs, it generates a system reset.

`wdt_intr` is asserted as soon as the timer counter rolls over to its maximum value, and it stays asserted until it is cleared from the `pclk` domain by either a read of the Interrupt Clear Register (`WDT_EOI`) or by writing 0x76 to the Counter Restart Register (`WDT_CRR`).

12.4.1.2. Counter

The WDT counts from a preset (timeout) value in descending order to zero. When the counter reaches zero, depending on the output response mode selected, either a system reset or an interrupt occurs. When the counter reaches zero, it wraps to the selected timeout value and continues decrementing. You can restart the counter to its initial value. This is programmed by writing to the Counter Restart Register (`WDT_CRR`) at any time. The process of restarting the watchdog counter is sometimes referred to as *kicking the dog*. As a safety feature to prevent accidental restarts, the value 0x76 must be written to the Counter Restart Register (`WDT_CRR`).

12.4.1.3. System Resets

When a 0 is written to the output response mode field (`RMOD`, bit 1) of the Control Register (`WDT_CR`), the WDT generates a system reset when a timeout occurs.

If a restart occurs at the same time the watchdog counter reaches zero, a system reset is not generated.

WDT can be programmed to generate the `wdt_intr` (and then a system reset) when a timeout occurs, that is when the `RMOD` field of the WDT Control Register (`WDT_CR`) is set to 1.

Irrespective of whether the `wdt_intr` is cleared (by reading the `WDT_EOI` register) and if the second timeout occurs, WDT generates a system reset. For subsequent timeouts, both the interrupt and the system reset are generated. However, this system reset generation can be avoided by performing a restart, that is by writing 0x76 into the `WDT_CRR` register before the second or subsequent timeout occurs. This behavior of a system reset is enabled by programming the `WDT_CR`.`RMOD` register bit to 1.

12.4.1.4. Reset Pulse Length

The reset pulse length is the number of Register Interface clock (`pclk`) cycles for which a system reset is asserted. When a system reset is generated, it remains asserted for the number of cycles specified by the reset pulse length (`WDT_CR`.`RPL` field setting) plus two cycles of synchronization delay, or until the system is reset by CRU. A counter restart has no effect on the system reset once it has been asserted.

12.4.2. WDT Registers

The BE-U1000 microcontroller contains two WDTs. The following table describes address regions for the WDTs.

Table 12-61 Memory address regions for WDT registers

Region	Block Name	Size	Start Address	End Address
Periphery 0	WDT_0	4 KB	0x1100_8000	0x1100_8FFF
Periphery 1	WDT_1	4 KB	0x1300_8000	0x1300_8FFF

For more information, see Memory Map chapter.

This section covers the following topics:

- Register Map
- Register Descriptions

12.4.2.1. Register Map

The following table provides a high-level summary of each WDT register. To view the detailed description of a register, click the register name in the **Description** column.

Table 12-62 WDT register map

Name	Offset	Description	Default Value
WDT_CR	0x0	Control Register	0x10
WDT_TORR	0x4	Timeout Range Register	0x4
WDT_CCVR	0x8	Current Counter Value Register	0xFFFF
WDT_CRR	0xC	Counter Restart Register	0x0
WDT_STAT	0x10	Interrupt Status Register	0x0
WDT_EOI	0x14	Interrupt Clear Register	0x0
WDT_COMP_PARAM_5	0xE4	Component Parameters Register 5	0x7FFFFFF
WDT_COMP_PARAM_4	0xE8	Component Parameters Register 4	0x0
WDT_COMP_PARAM_3	0xEC	Component Parameters Register 3	0x4
WDT_COMP_PARAM_2	0xF0	Component Parameters Register 2	0xFFFF
WDT_COMP_PARAM_1	0xF4	Component Parameters Register 1	0x10041280

12.4.2.2. Register Descriptions

The following sections contain the register descriptions.

In the tables below, the **R/W** column specifies how the application and the core can access the register fields:

- **R:** Read Only Access
- **R/W:** Read/Write Access
- **W:** Write Access

12.4.2.2.1. Control Register (`WDT_CR`)

The `WDT_CR` register is at offset 0x0

The following table shows the register bit assignments.

Table 12-63 Fields for register: `WDT_CR`

Bits	Name	R/W	Description
[31:5]			Reserved
[4:2]	`RPL`	R/W	Reset pulse length This bit field selects the number of `pclk` cycles for which the system reset remains asserted. The available range is 2 to 256 `pclk` cycles. **Values:** **0x0:** 2 `pclk` cycles **0x1:** 4 `pclk` cycles **0x2:** 8 `pclk` cycles **0x3:** 16 `pclk` cycles **0x4:** 32 `pclk` cycles **0x5:** 64 `pclk` cycles **0x6:** 128 `pclk` cycles **0x7:** 256 `pclk` cycles **Value After Reset:** 0x4
[1]	`RMOD`	R/W	Response mode Selects the output response generated to a timeout. **Values:** **0x0:** Generate a system reset. **0x1:** First generate an interrupt and if it is not cleared by the time a second timeout occurs then generate a system reset. **Value After Reset:**0x0
[0]	`WDT_EN`	R/W	WDT enable This bit is used to enable and disable the WDT. When disabled, the counter does not decrement. Thus, no interrupts or system resets are generated. The WDT is used to prevent system lock-up. To prevent a software bug from disabling the WDT, once this bit has been enabled, it can be cleared only by a system reset. **Values:** **0x0:** WDT disabled **0x1:** WDT enabled **Value After Reset:** 0x0

12.4.2.2.2. Timeout Range Register (`WDT_TORR`)

The `WDT_TORR` register is at offset 0x4.

The following table shows the register bit assignments.

Table 12-64 Fields for register: `WDT_TORR`

Bits	Name	R/W	Description
[31:4]			Reserved
[3:0]	`TOP`	R/W	Timeout period This bit field selects the timeout period after which the WDT counter restarts. A change to the timeout period takes effect only after the next counter restart (kick). The range of values is limited by the WDT counter width (32 bits). If `TOP` is programmed to select a range greater than the counter width, the timeout period is truncated to fit to the counter width. **Values:** **0x0:** 0xFF **0x1:** 0x1FF **0x2:** 0x3FF **0x3:** 0x7FF **0x4:** 0xFFFF **0x5:** 0x1FFFF ... **0xE:** 0x3FFFFFF **0xF:** 0x7FFFFFF **Value After Reset:** 0x4

12.4.2.2.3. Current Counter Value Register (`WDT_CCVR`)

The `WDT_CCVR` register is at offset 0x08.

The following table shows the register bit assignments.

Table 12-65 Fields for register: `WDT_CCVR`

Bits	Name	R/W	Description
[31:0]	`WDT_CCVR`	R	Current Counter Value Register This register, when read, is the current value of the internal counter. This value is read coherently whenever it is read. **Value After Reset:** 0xFFFF

12.4.2.2.4. Counter Restart Register (`WDT_CRR`)

The `WDT_CRR` register is at offset 0x0C.

The following table shows the register bit assignments.

Table 12-66 Fields for register: `WDT_CRR`

Bits	Name	R/W	Description
[31:8]			Reserved
[7:0]	`WDT_CRR`	W	Counter Restart Register

Table 12-66 Fields for register: `WDT_CRR` (continued)

Bits	Name	R/W	Description
			This register is used to restart the WDT counter. As a safety feature to prevent accidental restarts, the value 0x76 must be written. A restart also clears the WDT interrupt. Reading this register returns zero. **Value After Reset:** 0x0

12.4.2.2.5. Interrupt Status Register (`WDT_STAT`)

The `WDT_STAT` register is at offset 0x10.

The following table shows the register bit assignments.

Table 12-67 Fields for register: `WDT_STAT`

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	`WDT_STAT`	R	Interrupt Status Register This register shows the interrupt status of the WDT. **Values:** **0x0:** Interrupt is inactive **0x1:** Interrupt is active regardless of polarity **Value After Reset:** 0x0

12.4.2.2.6. Interrupt Clear Register (`WDT_EOI`)

The `WDT_EOI` register is at offset 0x14.

The following table shows the register bit assignments.

Table 12-68 Fields for register: `WDT_EOI`

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	`WDT_EOI`	R	Interrupt Clear Register This bit clears the WDT interrupt. This can be used to clear the interrupt without restarting the WDT counter. **Value After Reset:** 0x0

12.4.2.2.7. Component Parameters Register 5 (`WDT_COMP_PARAM_5`)

The `WDT_COMP_PARAM_5` register is at offset 0xE4.

This read-only register contains encoded information about the WDT configuration settings.

The following table shows the register bit assignments.

Table 12-69 Fields for register: `WDT_COMP_PARAM_5`

Bits	Name	R/W	Description
[31:0]	`CP_WDT_USER_TOP_MAX`	R	Upper limit of timeout period

Table 12-69 Fields for register: `WDT_COMP_PARAM_5` (continued)

Bits	Name	R/W	Description
			Values: **0xFF:** range 0 **0x1FF:** range 1 **0x3FF:** range 2 **0x7FF:** range 3 **0xFFFF:** range 4 **0x1FFFF:** range 5 ... **0x3FFFFFF:** range 14 **0x7FFFFFF:** range 15 **Value After Reset:** 0x7FFFFFF

12.4.2.2.8. Component Parameters Register 4 (`WDT_COMP_PARAM_4`)

The `WDT_COMP_PARAM_4` register is at offset 0xE8.

This read-only register contains encoded information about the WDT configuration settings.

The following table shows the register bit assignments.

Table 12-70 Fields for register: `WDT_COMP_PARAM_4`

Bits	Name	R/W	Description
[31:0]	`CP_WDT_USER_TOP_INIT_MAX`	R	Upper limit of initial timeout period **Value After Reset:** 0x0

12.4.2.2.9. Component Parameters Register 3 (`WDT_COMP_PARAM_3`)

The `WDT_COMP_PARAM_3` register is at offset 0xEC.

This read-only register contains encoded information about the WDT configuration settings.

The following table shows the register bit assignments.

Table 12-71 Fields for register: `WDT_COMP_PARAM_3`

Bits	Name	R/W	Description
[31:0]	`CD_WDT_TOP_RST`	R	WDT default counter value The value of this register is derived from the configuration settings that can be read in the `WDT_DFLT_TOP` field of the `WDT_COMP_PARAM_1` register. **Value After Reset:** 0x4

12.4.2.2.10. Component Parameters Register 2 (`WDT_COMP_PARAM_2`)

The `WDT_COMP_PARAM_2` register is at offset 0xF0.

This read-only register contains encoded information about the WDT configuration settings.

The following table shows the register bit assignments.

Table 12-72 Fields for register: `WDT_COMP_PARAM_2`

Bits	Name	R/W	Description
[31:0]	`CP_WDT_CNT_RST`	R	WDT Counter Value After Reset **Value After Reset:** 0xFFFF

12.4.2.2.11. Component Parameters Register 1 (`WDT_COMP_PARAM_1`)

The `WDT_COMP_PARAM_1` register is at offset 0xF4.

This read-only register contains encoded information about the WDT configuration settings.

The following table shows the register bit assignments.

Table 12-73 Fields for register: `WDT_COMP_PARAM_1`

Bits	Name	R/W	Description
[31:29]			Reserved
[28:24]	`WDT_CNT_WIDTH`	R	WDT counter width **0x10:** 32 bits **Value After Reset:** 0x10
[23:20]	`WDT_DFLT_TOP_INIT`	R	Initial timeout period range selected as default **Value After Reset:** 0x0
[19:16]	`WDT_DFLT_TOP`	R	Main timeout period range selected as default **Value After Reset:** 0x4
[15:13]			Reserved
[12:10]	`WDT_DFLT_RPL`	R	Default reset pulse length **0x4:** 32 `pclk` cycles **Value After Reset:** 0x4
[9:8]	`REG_IF_DATA_WIDTH`	R	Register interface data bus width **0x2:** 32 bits **Value After Reset:** 0x2
[7]	`WDT_PAUSE`	R	Pause enable input on interface **0x1:** Included ⚠ `pause` signal tied off to 0 in the BE-U1000 **Value After Reset:** 0x1
[6]	`WDT_USE_FIX_TOP`	R	Predefined timeout period ranges Since this parameter is set to 0, the range of the timeout period is not fixed and you can define it. **0x0:** No **Value After Reset:** 0x0
[5]	`WDT_HC_TOP`	R	Hard code timeout period range Since the parameter is set to 0, the timeout period is programmable. **0x0:** No

Table 12-73 Fields for register: `WDT_COMP_PARAM_1` (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0
[4]	`WDT_HC_RPL`	R	Hard code reset pulse length Since the parameter is set to 0, the reset pulse length is programmable. **0x0:** No **Value After Reset:** 0x0
[3]	`WDT_HC_RMOD`	R	Hard code output response mode Since the parameter is set to 0, the output response mode is programmable. **0x0:** No **Value After Reset:** 0x0
[2]	`WDT_DUAL_TOP`	R	A second timeout period for initialization is included **0x0:** False **Value After Reset:** 0x0
[1]	`WDT_DFLT_RMOD`	R	Output response mode Describes the output response mode that is available directly after reset. Indicates the output response the WDT gives if a zero count is reached. **0x0:** System reset only **Value After Reset:** 0x0
[0]	`WDT_ALWAYS_EN`	R	Enable WDT from reset Since the parameter is set to 0, the WDT is disabled on reset. **0x0:** No **Value After Reset:** 0x0

13. Low Speed Peripherals

This chapter covers the following topics:

- GPIO
- CAN FD
- UART
- SPI & QSPI
- I2C
- I2S

13.1. GPIO

General Purpose Input-Output (**GPIO**) Controller is a run-time programmable interface for external communications.

The BE-U1000 microcontroller contains three GPIO Controllers.

A simplified block diagram of the GPIO is shown in the following figure.

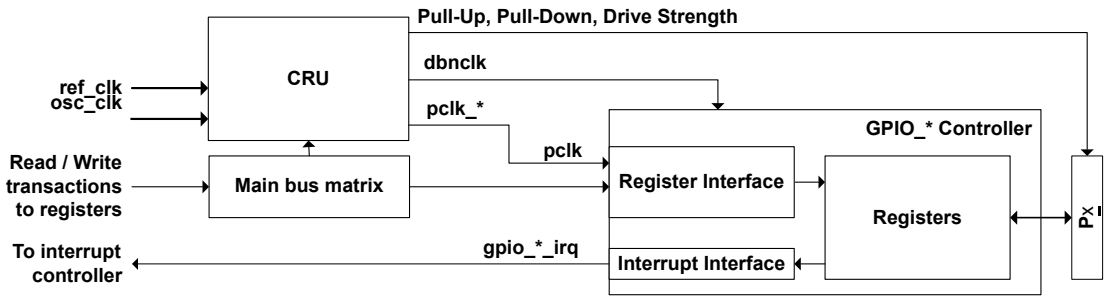

Figure 13-1 GPIO block diagram

In the figure above, x shows corresponding I/O port, where x = A for GPIO_0, x = B for GPIO_1 and x = C for GPIO_2. An asterisk symbol (*) indicates 0 for GPIO_0, 1 for GPIO_1 and 2 for GPIO_2.

GPIO Controller features:

- 16 independently configurable signals with Pull-Up control (via the `IOPUCRx` CRU registers), Pull-Down control (via the `IOPDCRx` CRU registers) and Drive Strength control (via the `IODSCRx` CRU registers).
- Debounce logic with an external clock to debounce interrupts.
- Combined interrupt signal.

13.1.1. GPIO Functional Description

This section covers the following topics:

- Data and Control Flow
- Interrupts

13.1.1.1. Data and Control Flow

The GPIO Controller controls the output data and direction of external I/O pads with Pull-Up control (via the `IOPUCRx` CRU registers), Pull-Down control (via the `IOPDCRx` CRU registers) and Drive

Strength control (via the IODSCRx CRU registers). It also can read back the data on external pads using memory-mapped registers.

Setting the default source of the input data, the output data, and the control of each signal are available through Software Control over the Register Interface.

13.1.1.1.1. Software Control

The data and direction control for the signal are sourced from the Data Output Register (GPIO_DR) and Data Direction Control Register (GPIO_DDR).

Data Direction Control Register controls the direction of the external I/O pad.

The data written to the Data Output Register (GPIO_DR) drives the output buffer of the I/O pad.

External data are input on the Input Data Register (GPIO_EXT). This register is read-only, meaning that it cannot be written from the Register Interface.

13.1.1.1.2. Reading External Signals

The data on the external GPIO signal is read by an Register Interface read of the memory-mapped register, GPIO_EXT. A Register Interface read to the GPIO_EXT register provides either the data on the External Port control lines or the contents of the GPIO_DR, depending on the value of GPIO_DDR.

Reading from an unused location or unused bits in a particular register always returns zeros. There is no error mechanism in the Register Interface.

13.1.1.2. Interrupts

Each GPIO signal can be programmed to accept external signals as interrupt sources on any of signal bits. The type of interrupt is programmable with one of the following settings:

- Active-high and level
- Active-low and level
- Rising edge
- Falling edge

Programming the registers (GPIO_INTEN, GPIO_INTLVL and GPIO_INTPOLARITY) for interrupt capability, level-sensitive interrupts, and interrupt polarity should be completed prior to enabling the interrupts on the port in order to prevent spurious glitches on the interrupt lines to the interrupt controller.

You can mask the interrupts by programming the GPIO_INTMSK register. The interrupt status can be read before masking (called *raw status*) and after masking.

This section covers the following topics:

- Debounce Operation
- Level-sensitive Interrupts

13.1.1.2.1. Debounce Operation

The external signal can be debounced to remove any spurious glitches that are less than one period of the external debouncing clock.

The GPIO Controller supports the logic to debounce both rising edge and falling edge. The signal is debounced on either a single edge or on both edges, depending on the value of the `GPIO_BOTHEDGE[n]` register.

The use of the debounce circuitry increases interrupt latency by two clock cycles of the debounce clock.

13.1.1.2.2. Level-sensitive Interrupts

With level-sensitive interrupts, there is a choice of whether they are synchronized to the `pclk` or are entirely combinational (aside from the debounce circuitry). The selection is done by programming the Synchronization Level Register (`GPIO_SYNC`).

13.1.2. GPIO Registers

Register regions of the GPIO Controllers are mapped in the BE-U1000 microcontroller memory map as the following table shows.

Table 13-1 Memory address regions for GPIO Controller registers

Region	Block Name	Size	Start Address	End Address
Periphery 0	GPIO_0	4 KB	0x1100_9000	0x1100_9FFF
Periphery 1	GPIO_1	4 KB	0x1300_9000	0x1300_9FFF
Periphery 2	GPIO_2	4 KB	0x1400_8000	0x1400_8FFF

For details, refer to Memory Map chapter.

This section covers the following topics:

- Programming Considerations
- Register Map
- Register Descriptions

13.1.2.1. Programming Considerations

- Reading from an unused location or unused bit in a register always returns zero. There is no error mechanism in the Register Interface.
- Programming the registers (`GPIO_INTEN`, `GPIO_INTLVL` and `GPIO_INTPOLARITY`) for interrupt capability, level-sensitive interrupts, and interrupt polarity should be completed prior to enabling the interrupts on the port in order to prevent spurious glitches on the interrupt lines to the interrupt controller.
- Writing to the `EOI` register clears an edge-detected interrupt and has no effect on a level-sensitive interrupt.

13.1.2.2. Register Map

This section tables contain information about the register map.

The following table provides a high-level summary of each register. To view the detailed description of a register, click the register name in the **Description** column.

Table 13-2 GPIO register map

Name	Offset	Description	Default Value
GPIO_DR	0x00	Data Output Register	0x0
GPIO_DDR	0x04	Data Direction Control Register	0x0
GPIO_INTEN	0x30	Interrupt Enable Register	0x0
GPIO_INTMSK	0x34	Interrupt Mask Register	0x0
GPIO_INTLVL	0x38	Interrupt Level Register	0x0
GPIO_INTPOLARITY	0x3C	Interrupt Polarity Register	0x0
GPIO_INTSTAT	0x40	Interrupt Status Register	0x0
GPIO_INTSTATRAW	0x44	Raw Interrupt Status Register	0x0
GPIO_DEBOUNCE	0x48	Debounce Enable Register	0x0
GPIO_EOI	0x4C	Clear Interrupt Register	0x0
GPIO_EXT	0x50	External Port Register	0x0
GPIO_SYNC	0x60	Synchronization Level Register	0x0
GPIO_BOTHEDGE	0x68	Interrupt Both Edge Type Register	0x0

13.1.2.3. Register Descriptions

The following subsections describe the data fields of the GPIO Controller registers.

13.1.2.3.1. Data Output Register (GPIO_DR)

The GPIO_DR register is at offset 0x00.

The following table shows the register bit assignments.

Table 13-3 Fields for register: GPIO_DR

Bits	Name	R/W	Description
[31:16]			Reserved
[15:0]	GPIO_DR	R/W	Output Data Register Values written to this register are output on the I/O signals if the DDR bits of the GPIO_DDR register are set to Output mode. **Value After Reset**: 0x0

13.1.2.3.2. Data Direction Control Register (GPIO_DDR)

The GPIO_DDR register is at offset 0x04.

The following table shows the register bit assignments.

Table 13-4 Fields for register: `GPIO_DDR`

Bits	Name	R/W	Description
[31:16]			Reserved
[15:0]	`DDR`	R/W	Data Direction Register Controls the direction of the corresponding data bit. **0x0**: Input data **0x1**: Output data **Value After Reset**: 0x0

13.1.2.3.3. Interrupt Enable Register (`GPIO_INTEN`)

The `GPIO_INTEN` register is at offset 0x30.

The following table shows the register bit assignments.

Table 13-5 Fields for register: `GPIO_INTEN`

Bits	Name	R/W	Description
[31:16]			Reserved
[15:0]	`GPIO_INTEN`	R/W	Interrupt Enable Register Allows each bit of `GPIO_DR` to be configured for interrupts. **0x0**: Configure bit as normal GPIO signal (default) **0x1**: Configure bit as interrupt To disable interrupts on the corresponding bits set the value 1 to the corresponding bits of the `GPIO_DDR`. **Value After Reset**: 0x0

13.1.2.3.4. Interrupt Mask Register (`GPIO_INTMSK`)

The `GPIO_INTMSK` register is at offset 0x34.

The following table shows the register bit assignments.

Table 13-6 Fields for register: `GPIO_INTMSK`

Bits	Name	R/W	Description
[31:16]			Reserved
[15:0]	`GPIO_INTMSK`	R/W	Interrupt Mask Register Masks the corresponding interrupt generation. By default, all interrupts bits are unmasked. The unmasked status can be read as well as the resultant status after masking. **0x0**: Interrupt bits are unmasked (default) **0x1**: Mask interrupt **Value After Reset**: 0x0

13.1.2.3.5. Interrupt Level Register (`GPIO_INTLVL`)

The `GPIO_INTLVL` register is at offset 0x38.

The following table shows the register bit assignments.

Table 13-7 Fields for register: `GPIO_INTLVL`

Bits	Name	R/W	Description
[31:16]			Reserved
[15:0]	`GPIO_INTLVL`	R/W	Interrupt Level Register Controls the type of interrupt that can occur on GPIO Port: **0x0**: Level-sensitive (default) **0x1**: Edge-sensitive **Value After Reset**: 0x0

13.1.2.3.6. Interrupt Polarity Register (`GPIO_INTPOLARITY`)

The `GPIO_INTPOLARITY` register is at offset 0x3C.

The following table shows the register bit assignments.

Table 13-8 Fields for register: `GPIO_INTPOLARITY`

Bits	Name	R/W	Description
[31:16]			Reserved
[15:0]	`GPIO_INTPOLARITY`	R/W	Interrupt Polarity Register Controls the polarity of edge or level sensitivity (depends on `INTTYPE_LEVEL` value) that can occur on External Port Register (`GPOI_EXT`). Whenever a 0 is written to a bit of this register, it configures the interrupt type to Falling-Edge or Active-Low sensitive, otherwise, it is Rising-Edge or Active-High sensitive. **0x0**: Active-low (default) **0x1**: Active-high **Value After Reset**: 0x0

13.1.2.3.7. Interrupt Status Register (`GPIO_INTSTAT`)

The `GPIO_INTSTAT` register is at offset 0x40.

The following table shows the register bit assignments.

Table 13-9 Fields for register: `GPIO_INTSTAT`

Bits	Name	R/W	Description
[31:16]			Reserved
[15:0]	`GPIO_INTSTAT`	R	Interrupt Status of GPIO Port **0x0**: Inactive (default) **0x1**: Active **Value After Reset**: 0x0

13.1.2.3.8. Raw Interrupt Status Register (`GPIO_INTSTATRAW`)

The `GPIO_INTSTATRAW` register is at offset 0x44.

This is the Raw Interrupt Status Register for GPIO signals. It is used with the Interrupt Mask Register to allow interrupts from GPIO signals.

The following table shows the register bit assignments.

Table 13-10 Fields for register: `GPIO_INTSTATRAW`

Bits	Name	R/W	Description
[31:16]			Reserved
[15:0]	`GPIO_INTSTATRAW`	R	Raw interrupt of status of GPIO Port signals (premasking bits) **0x0**: Inactive (default) **0x1**: Active **Value After Reset**: 0x0

13.1.2.3.9. Debounce Enable Register (`GPIO_DEBOUNCE`)

The `GPIO_DEBOUNCE` register is at offset 0x48.

The following table shows the register bit assignments.

Table 13-11 Fields for register: `GPIO_DEBOUNCE`

Bits	Name	R/W	Description
[31:16]			Reserved
[15:0]	`GPIO_DEBOUNCE`	R/W	Debounce enable Controls whether an external signal that is the source of an interrupt needs to be debounced to remove any spurious glitches. **0x0**: No debounce (default) **0x1**: Enables the denouncing circuitry A signal must be valid for two periods of an external clock before it is internally processed. **Value After Reset**: 0x0

13.1.2.3.10. Clear Interrupt Register (`GPIO_EOI`)

The `GPIO_EOI` register is at offset 0x4C.

The following table shows the register bit assignments.

Table 13-12 Fields for register: `GPIO_EOI`

Bits	Name	R/W	Description
[31:16]			Reserved
[15:0]	`GPIO_EOI`	W	Controls the clearing of edge type interrupts from GPIO Port. When a 1 is written into a corresponding bit of this register, the interrupt is cleared. All interrupts are cleared when GPIO Port is not configured for interrupts **0x0**: No interrupt clear **0x1**: Clear Interrupt **Value After Reset**: 0x0

13.1.2.3.11. External Port Register (GPIO_EXT)

The GPIO_EXT register is at offset 0x50.

The following table shows the register bit assignments.

Table 13-13 Fields for register: GPIO_EXT

Bits	Name	R/W	Description
[31:16]			Reserved
[15:0]	GPIO_EXT	R	This register always reflects the signals value on the External Port. **Value After Reset**: 0x0

13.1.2.3.12. Synchronization Level Register (GPIO_SYNC)

The GPIO_SYNC register is at offset 0x60.

The following table shows the register bit assignments.

Table 13-14 Fields for register: GPIO_SYNC

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	GPIO_SYNC	R/W	Synchronization level Level-sensitive interrupts synchronization to pclk: **0x0**: No synchronization to pclk (default) **0x1**: Synchronize to pclk **Value After Reset**: 0x0

13.1.2.3.13. Interrupt Both Edge Type Register (GPIO_BOTHEDGE)

The GPIO_BOTHEDGE register is at offset 0x68.

The following table shows the register bit assignments.

Table 13-15 Fields for register: GPIO_BOTHEDGE

Bits	Name	R/W	Description
[31:16]			Reserved
[15:0]	GPIO_BOTHEDGE	R/W	Controls the edge type of interrupt that can occur on GPIO Port: Whenever a particular bit is programmed to 1, it enables the generation of interrupts on both the rising edge and the falling edge of an external input signal corresponding to that bit on GPIO Port. The values programmed in the registers GPIO_INTLVL and GPIO_INTPOLARITY for this particular bit are not considered when the corresponding bit of this register is set to 1. Whenever a particular bit is programmed to 0, the interrupt type depends on the value of the corresponding bits in the GPIO_INTLVL and GPIO_INTPOLARITY registers. **0x0**: Single edge sensitive **0x1**: Both edge sensitive

Table 13-15 Fields for register: `GPIO_BOTHEDGE` (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0

13.2. CAN FD

The BE-U1000 integrates two **CAN FD** (*Controller Area Network with Flexible Data-Rate*) Controllers.

A simplified block diagram of the CAN FD is shown in the following figure.

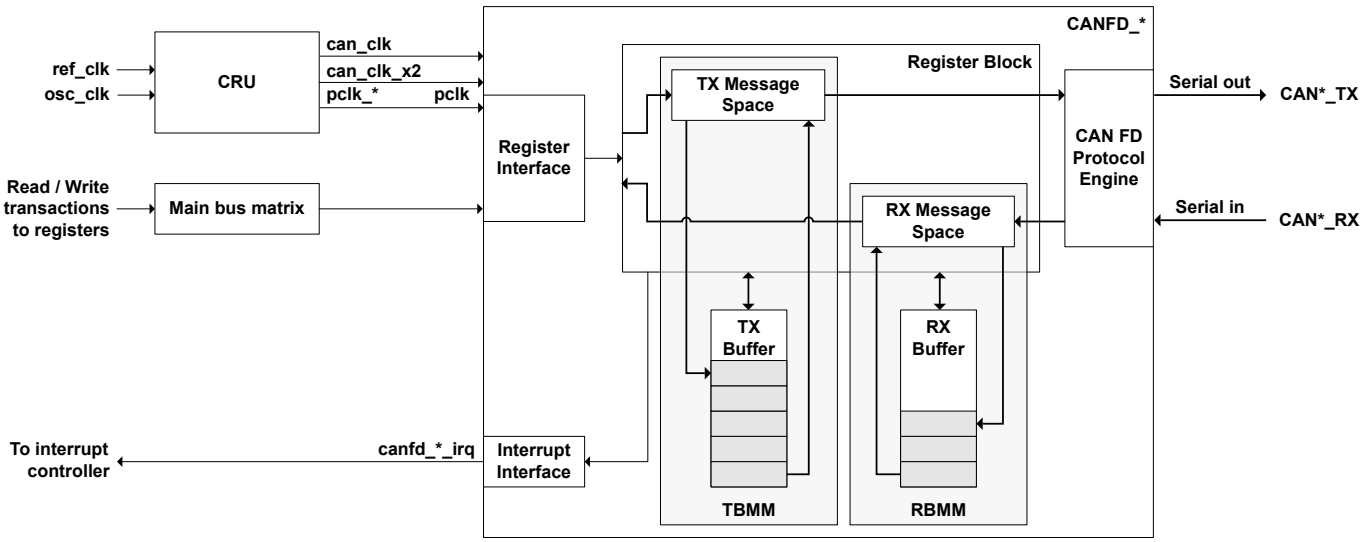

Figure 13-2 CAN FD block diagram

In the figure above: an asterisk symbol (*) indicates 0 for CANFD_0 and 1 for CANFD_1.

In the figure above:

- *Tx Buffer Management Module* (**TBMM**) — consists of the Tx Message Space and Tx Buffer and interfaces with the CAN FD protocol engine to provide the next buffer to transmit on the CAN bus.
- *Rx Buffer Management Module* (**RBMM**) — consists of the Rx Message Space and Rx Buffer and interfaces with the CAN FD protocol engine to provide storage for message reception from the CAN bus. It manages the host access to the Rx block RAM.
- CAN FD Protocol Engine functions:
  - Initiation of the transmission process after recognizing bus idle (compliance with inter-frame space)
  - Bit timing functions
  - Error detection and signaling
  - Recognition of an overload condition and reaction

CAN FD features:

- Supports both CAN and CAN FD frames
- Supports flexible data rates up to 8 Mb/s
- Supports nominal data rates up to 1 Mb/s
- 64-deep Rx FIFO with 32 ID Filter-Mask pairs

- Message with lowest ID transmitted first
- Supports Tx message cancellation
- Separate error logging for fast data rate
- Timestamp for transmitted and received messages

13.2.1. CAN FD Functional Description

This section covers the following topics:

- Operating Modes and States
- Interrupts

13.2.1.1. Operating Modes and States

The CAN FD supports the following modes and states:

- Configuration Mode
- Normal Mode
- Loopback Mode
- Sleep Mode
- Snoop Mode
- *Protocol Exception Event* (**PEE**) State
- Bus-Off Recovery State

The following table defines the modes of operation with corresponding control and status bits.

Table 13-16 CAN FD operation mode transitions

System (Hard) Reset	SRR Register Bits		MSR Register Bits			SR Register Bits							Operation Mode
	SRST	CEN	LBACK	SLEEP	SNOOP	CONFIG	BSFR_CONFIG	PEE_CONFIG	LBACK	SLEEP	NORMAL	SNOOP	
0 (reset)	X	X	1	X	X	1	0	0	0	0	0	0	CAN FD is under reset
1	1 (reset)	X	1	X	X	1	0	0	0	0	0	0	CAN FD is under reset
1	0	0	X	X	X	1	0	0	0	0	0	0	Configuration
		1	0	0	0	0	0	0	0	0	1	0	Normal
			0	0	1	0	0	0	0	0	1	1	Snoop
			0	1	0	0	0	0	0	1	0	0	Sleep
			1	0	0	0	0	0	1	0	0	0	Loopback
			0	X	0	0	1	0	0	0	X	0	Bus-Off Recovery
			0	X	X	0	0	1	0	0	X	X	PEE

- X-Control bit don't care. Status bit does not mean anything.
- Transition to Bus-Off state depends on Transmit Error Count value (ECR.TEC) as per standard specification. Recovery from Bus-Off state depends on SBR and ABR bit settings in the MSR register (as per respective bit behavior description). Bus-Off Recovery can be tracked through status bit

BSFR_CONFIG in SR register and REC field in ECR register. Entry and exit from Bus-Off state can also generate interrupt.

- Transition to CAN FD *Protocol Exception Event* (**PEE**) depends on the DPEE bit in MSR register. The CAN FD enters and exits PEE state as per ISO standard specification and this is reflected by status bit PEE_CONFIG in the SR register. Entry to PEE state can also generate interrupt.

Configuration Mode

The CAN FD enters Configuration mode, irrespective of the operation mode, when any of the following actions is performed:

- Writing a 0 to the CEN bit in the SRR register.
- Writing a 1 to the SRST bit in the SRR register.

After reset, the CAN FD exits Configuration mode after the SRR.CEN bit is set and 11 consecutive nominal recessive bits are seen on the CAN bus.

The CAN FD has the following Configuration mode characteristics:

- The Controller loses synchronization with the CAN bus and drives a constant recessive bit on the Tx line.
- The Error Count Register is reset.
- The Error Status Register is reset.
- The Arbitration Phase (Nominal) Bit Timing Register and Arbitration Phase (Nominal) Baud Rate Prescaler Registercan be modified.
- The CONFIG bit in the Error Status Register is 1.
- The CAN FD does not receive any new messages.
- The CAN FD does not transmit any messages.
- All configuration registers are accessible.
- If there are messages pending for transmission when CEN in SRR is written 0, they are preserved (unless cancelled) and transmitted when normal operation is resumed.
- Message cancellation is permitted.
- New messages can be added for transmission (provided the SNOOP bit is not set in the MSR register).
- If there are new received messages available, they are preserved until the host reads them.
- The Interrupt Status Register bits ARBLST, TXOK, RXOK, RXOFLW, ERROR, BSOFF, SLP and WKUP are cleared.
- Interrupt Status Register bits TXRRS and TXCRS can be set due to cancellation.
- Interrupts are generated if the corresponding bits in the IER are 1.
- When in Configuration mode, the Controller stays in this mode until the CEN bit in the SRR register is set to 1.
- After the CEN bit is set to 1, the Controller waits for a sequence of 11 nominal recessive bits before exiting Configuration mode.
- CAN FD enters Normal, Loopback, Snoop or Sleep modes from Configuration mode, depending on the LBACK, SNOOP, and SLEEP bits in the MSR register.

Normal Mode

The CAN FD can enter Normal mode only if SLEEP, LBACK and SNOOP bits are 0 in the MSR register.

In Normal mode, the CAN FD participates in bus communication by transmitting and receiving messages.

In Normal mode, CAN FD does not store its own transmitted messages.

Loopback Mode

This mode is used for diagnostic purposes.

The CAN FD enters Loopback mode when the LBACK bit is 1 in the MSR register. In Loopback mode, the CAN FD receives any messages that it transmits by an internal loopback to the RX line and acknowledges them (the external TX line is ignored).

Sleep Mode

The CAN FD enters Sleep mode from Configuration or Normal mode when the SLEEP bit is 1 in the MSR register, the CAN bus is idle, and there are no pending transmission requests. The CAN FD enters Configuration mode when any configuration condition is satisfied. The CAN FD enters Normal mode (clearing the SLEEP request bit in the MSR register and also clearing the corresponding status bit) under the following (wake-up) conditions:

- Whenever the SLEEP bit in the MSR register is set to 0.
- Whenever the SLEEP bit is 1, and bus activity is detected.
- Whenever there is a new message for transmission.

Interrupts are generated when the CAN FD enters Sleep mode or wakes up from Sleep mode.

Snoop (Bus Monitoring) Mode

The features of Snoop mode are as follows:

- The CAN FD transmits recessive bits onto the CAN bus.
- The CAN FD receives messages that are transmitted by other nodes but does not ACK. Stores received messages based on programmed ID filtering.
- Error counters are disabled and cleared to 0. Reading the Error Count Register returns 0.

Recommended that Snoop mode is programmed only after system reset or software reset.

Protocol Exception State

The CAN FD enters CAN FD *Protocol Exception Event* (**PEE**) state if it receives the res bit to be recessive in the CAN FD frame (provided the DPEE bit is not set in the MSR register). It comes out of this state after detecting a sequence of 11 nominal recessive bits on the CAN bus and, as per protocol specification, transmit and receive error count remains unchanged in this state.

When a CAN FD node detects a PEE, the software is required to perform the following steps to recover the transmission:

1. Read the Tx Buffer Ready Request (TRR) value.
2. Write the TRR read value to Tx Buffer Cancel Request (TCR).
3. Wait for the TRR to become 0.
4. Write the TRR with the value read in step 1.

Bus-Off Recovery State

The CAN FD enters Bus-Off state if the Transmit Error count (ECR.TEC) reaches or exceeds its terminal point. Recovery from Bus-Off states is governed by the auto-recovery (ABR) or manual recovery (SBR) bit setting in the MSR register and is done according to protocol specification. When a CAN FD

node detects a Bus-Off state, the software is required to perform the following steps to recover the transmission:

1. Read the Tx Buffer Ready Request (TRR) value.
2. Write the TRR read value to Tx Buffer Cancel Request (TCR).
3. Wait for the TRR to become 0.
4. Write the TRR with the value read in step 1

13.2.1.2. Interrupts

The CAN FD has a single level-sensitive interrupt output to indicate an interrupt. Interrupts are indicated by asserting the canfd_*_irq (where * indicates the number of CAN FD) line. Interrupt assertion and deassertion is synchronous to the Register Interface clock.

Events such as errors on the bus line, message transmission and reception, and various other conditions can generate interrupts. During power on, the interrupt line is driven Low. The Interrupt Status register (ISR) indicates the interrupt status bits. These bits are set and cleared regardless of the status of the corresponding bit in the Interrupt Enable register (IER). The IER handles the interrupt-enable functionality. The clearing of a status bit in the ISR is handled by writing a 1 to the corresponding bit in the Interrupt Clear register (ICR).

Two conditions cause the interrupt line to be asserted:

- If a bit in the ISR is 1 and the corresponding bit in the IER is 1.
- Changing an IER bit from a 0 to 1 when the corresponding bit in the ISR is already 1.

Two conditions cause the interrupt line to be deasserted:

- Clearing a 1 bit in the ISR (by writing a 1 to the corresponding bit in the ICR provided the corresponding bit in the IER is 1).
- Changing an IER bit from 1 to 0 when the corresponding bit in the ISR is 1.

When both deassertion and assertion conditions occur simultaneously, the interrupt line is deasserted first, and is reasserted if the assert condition remains TRUE.

13.2.2. CAN FD Registers

Register regions of the CAN FD are mapped in the BE-U1000 memory map as the following table shows.

Table 13-17 Memory address regions for CAN FD registers

Region	Block Name	Size	Start Address	End Address
Periphery 0	CANFD_0	32 KB	0x1101_0000	0x1101_7FFF
Periphery 1	CANFD_1	32 KB	0x1301_0000	0x1301_7FFF

For details, refer to Memory Map chapter.

This section covers the following topics:

- Register Map
- Register Descriptions

13.2.2.1. Register Map

The following tables provide A top-level summary of each register space of the CAN FD:

- Core Register Space is implemented with flip-flops.
- Tx Message Space is implemented with Tx block RAM and provides storage for the 32 Tx buffers. It also provides storage for 32 ID Filter-Mask pairs.
- CAN FD Tx Event FIFO Status Register Space
- Rx Message Space is implemented with Rx block RAM. It provides storage for 64-deep message Rx FIFO and for 32 deep Tx Event FIFO.

To view the detailed description of a register, click the register name in the **Description** column.

Table 13-18 CAN FD register map

Register	Offset	Description	Default Value
Core Register Space			
SRR	0x00	Software Reset Register	0x0
MSR	0x04	Mode Select Register	0x0
BRPR	0x08	Arbitration Phase (Nominal) Baud Rate Prescaler Register	0x0
BTR	0x0C	Arbitration Phase (Nominal) Bit Timing Register	0x0
ECR	0x10	Error Counter Register	0x0
ESR	0x14	Error Status Register	0x0
SR	0x18	Status Register	0x1
ISR	0x1C	Interrupt Status Register	0x0
IER	0x20	Interrupt Enable Register	0x0
ICR	0x24	Interrupt Clear Register	0x0
TSR	0x28	Timestamp Register	0x0
DP_BRPR	0x88	Data Phase Baud Rate Prescaler Register	0x0
DP_BTR	0x8C	Data Phase Bit Timing Register	0x0
TRR	0x90	Tx Buffer Ready Request Register	0x0
IETRS	0x94	Interrupt Enable Tx Buffer Ready Request Served/ Cleared Register	0x0
TCR	0x98	Tx Buffer Cancel Request Register	0x0
IETCS	0x9C	Interrupt Enable Tx Buffer Cancellation Request Served/Cleared Register	0x0
TxE_FSR	0xA0	Tx Event FIFO Status Register	0x0

Table 13-18 CAN FD register map (continued)

Register	Offset	Description	Default Value
TxE_WMR	0xA4	Tx Event FIFO Watermark Register	0xF
AFR	0xE0	Acceptance Filter (Control) Register	0x0
FSR	0xE8	Rx FIFO Status Register	0x0
WMR	0xEC	Rx FIFO Watermark Register	0x1F0F0F
Tx Message Space			
TBiID	0x100 - 0x9B8	TBi Message ID Register (for i = 0; i <=31)	
TBiDLC	0x104 - 0x9BC	TBi Data Length Code Register (for i = 0; i <=31)	
TBiDWn	0x108 - 0x9FC	TBi Data Byte n Register (for i = 0; i <=31 and for n = 0; n <=15)	
AFMRi	0xA00 - 0xAF8	Acceptance Filter i Mask Register (for i = 0; i <=31)	
AFIRi	0x0A04 - 0xAFC	Acceptance Filter i ID Registers (for i = 0; i <=31)	
Tx Event FIFO Status Register Space			
TXE_FIFO_TBi_ID	0x2000 - 0x20F8	TXE FIFO TBi Message ID Register (for i = 0; i <=31)	
TXE_FIFO_TBi_DLC	0x2004 - 0x20FC	TXE FIFO TBi Data Length Code Register (for i = 0; i <=31)	
Rx Message Space			
RBiID	0x2100 - 0x32B8	RBi Message ID Register (for i = 0; i <=63)	
RBiDLC	0x2104 - 0x32BC	RBi Data Length Code Register (for i = 0; i <=63)	
RBiDWn	0x2108 - ox32FC	RBi Data Byte n Register (for i = 0; i <=63 and for n = 0; n <=15)	

13.2.2.2. Register Descriptions

In the tables below, the **R/W** column specifies how the application and the core can access the register fields.

13.2.2.2.1. CAN FD Core Register Space

13.2.2.2.1.1. Software Reset Register (SRR)

The SRR register is at offset 0x00.

Writing to the Software Reset register (SRR) places the CAN FD in Configuration mode. In Configuration mode, the CAN FD drives recessive on the bus line and does not transmit or receive

messages. During power-up, the CEN and SRST bits are 0 and the CONFIG bit in the Status register (SR) is 1. The Transfer Layer Configuration registers can be changed only when the CEN bit in the SRR is 0. Mode Select register bits (except SLEEP and SBR in MSR register) can be changed only when the CEN bit is 0. If the CEN bit is changed while the CAN FD controller is operating, it is strongly recommended to reset the CAN FD controller to ensure reliable restart.

The following table shows the register bit assignments.

Table 13-19 Fields for register: SRR

Bits	Name	R/W	Description
[31:2]			Reserved
[1]	CEN	R/W	CAN Enable. This is the Enable bit for the CAN FD. **Values:** **0x0:** The CAN FD is in Configuration mode. **0x1:** The CAN FD is in Loopback, Sleep, Snoop, or Normal mode, depending on the LBACK, SLEEP, and SNOOP bits in the MSR. **Value After Reset:** 0x0
[0]	SRST	W	Reset. This is the software reset bit for the CAN FD. **Value:** **0x1:** CAN FD is reset. If a 1 is written to this bit, all CAN FD configuration registers (including the SRR) are reset. Reads to this bit always return 0. After performing a soft or hard reset, wait for 16 Register Interface clock cycles before initiating next Register Interface transaction. **Value After Reset:** 0x0

13.2.2.2.1.2. Mode Select Register (MSR)

The MSR register is at offset 0x04.

Writing to the Mode Select register (MSR) enables the CAN FD to enter Snoop, Sleep, Loopback, or Normal modes.

LBACK, SLEEP, and SNOOP bits should never be set to 1 at the same time. At any given point, the CAN FD can either be in Loopback, Sleep, or Snoop mode. When all three bits are set to 0, the CAN FD can enter Normal mode subject to other conditions.

The following table shows the register bit assignments.

Table 13-20 Fields for register: MSR

Bits	Name	R/W	Description
[31:16]			Reserved
[15:8]	ITO	R/W	Internal Timing Optimization.

Table 13-20 Fields for register: MSR (continued)

Bits	Name	R/W	Description
			Bit [11] of this register is used for Internal Timing Optimization Enable. Program this field if you intend to configure the nominal and data phase baud rate prescaler as 1 (register value 0). Only change this field when CEN = 0 in SRR register. For CAN FD operation with BRPR.BRP=1 (register value 0), CAN FD implementation makes it mandatory to choose *Baud Rate Prescaler* (**BRP**) for nominal and data phase as 1 (register value 0). Additionally, you need to program the MSR. **Values:** **0x00:** BRPR.BRP != 1 **0x08:** BRPR.BRP=1 **Value After Reset:** 0x0
[7]	ABR	R/W	Auto Bus-Off Recovery Request. **Values:** **0x0:** No such request **0x1:** Auto Bus-Off Recovery request If this bit is set, the node does auto Bus-Off Recovery irrespective of the SBR bit setting in this register. This bit can be written only when the CEN bit in SRR is 0. **Value After Reset:** 0x0
[6]	SBR	R/W	Start Bus-Off Recovery Request. **Values:** **0x0:** No such request **0x1:** Start Bus-Off Recovery request Node stays in Bus-Off state until the SBR bit is set to 1 (providing that the ABR bit in this register is not set). This bit can be written only when node is in Bus-Off state. This bit auto clears after node completes the Bus-Off Recovery or leaves Bus-Off state due to hard/soft reset or SRR.CEN deassertion. **Value After Reset:** 0x0
[5]	DPEE	R/W	Disable Protocol Exception Event Detection/Generation. **Values:** **0x0:** *Protocol Exception Event* (**PEE**) detection/ generation is enabled. If the CAN FD receiver detects the res bit in CAN FD frame as 1, it goes to Bus Integration state (SR.PEE_CONFIG) and waits for Bus Idle condition (SR.BIDLE). The error counter remains unchanged. **0x1:** Disable Protocol Exception Event detection/ generation by CAN FD receiver if res bit in CAN FD frame is detected as 1. In this case, CAN FD receiver generates Form error.

Table 13-20 Fields for register: MSR (continued)

Bits	Name	R/W	Description
			This bit can be written only when the CEN bit in SRR is 0. **Value After Reset:** 0x0
[4]	DAR	R/W	Disable Auto-Retransmission. **Values:** **0x0:** Auto retransmission enabled **0x1:** Disable auto retransmission on the CAN bus to provide single shot transmission This bit can be written only when the CEN bit in SRR is 0. **Value After Reset:** 0x0
[3]	BRSD	R/W	CAN FD Bit Rate Switch Disable Override. **Values:** **0x0:** Makes the CAN FD transmit frames as per BRS bit in the TB* Data Length Code Register **0x1:** Makes the CAN FD transmit frames only in nominal bit rate (by overriding the TB* Data Length Code Register BRS bit setting) This bit can be written only when the CEN bit in SRR is 0. **Value After Reset:** 0x0
[2]	SNOOP	R/W	SNOOP Mode Select/Request. **Values:** **0x0:** No such request **0x1:** Request CAN FD to be in Snoop mode This bit can be written only when the CEN bit in SRR is 0. Make sure that Snoop mode is programmed only after system reset or software reset. For the CAN FD to enter Snoop mode, LBACK and SLEEP bits in this register should be set to 0. **Value After Reset:** 0x0
[1]	LBACK	R/W	Loopback Mode Select/Request. **Values:** **0x0:** No such request **0x1:** Request CAN FD to be in Loopback mode This bit can be written only when the CEN bit in SRR is 0. For the CAN FD to enter Loopback mode, SLEEP and SNOOP bits in this register should be set to 0. **Value After Reset:** 0x0
[0]	SLEEP	R/W	Sleep Mode Select/Request. **Values:** **0x0:** No such request **0x1:** Request CAN FD to be in Sleep mode This bit is cleared when the CAN FD wakes up from Sleep mode. For the CAN FD to enter Sleep mode, LBACK and SNOOP bits in this register should be set to 0. **Value After Reset:** 0x0

13.2.2.2.1.3. Arbitration Phase (Nominal) Baud Rate Prescaler Register (BRPR)

The BRPR register is at offset 0x08.

The CAN clock (can_clk) for the core is divided by (programmed prescaler value + 1) to generate the quantum clock needed for sampling and synchronization

The following table shows the register bit assignments.

Table 13-21 Fields for register: BRPR

Bits	Name	R/W	Description
[31:8]			Reserved
[7:0]	BRP	R/W	Arbitration Phase (Nominal) Baud Rate Prescaler. The actual value is one more than the value written to the register. These bits can be written only when the CEN bit in SRR is 0. **Value After Reset:** 0x0

13.2.2.2.1.4. Arbitration Phase (Nominal) Bit Timing Register (BTR)

The BTR register is at offset 0x0C.

The following table shows the register bit assignments.

Table 13-22 Fields for register: BTR

Bits	Name	R/W	Description
[31:23]			Reserved
[22:16]	SJW	R/W	Synchronization Jump Width. Indicates the Synchronization Jump Width as specified in the standard for Nominal Bit Timing. The actual value is one more than the value written to the register. These bits can be written only when the CEN bit in SRR is 0. **Value After Reset:** 0x0
[15]			Reserved
[14:8]	TS2	R/W	Time Segment 2 Indicates the Phase Segment 2 as specified in the standard for Nominal Bit Timing. The actual value is one more than the value written to the register. These bits can be written only when the CEN bit in SRR is 0. **Value After Reset:** 0x0
[7:0]	TS1	R/W	Time Segment 1. Indicates the Sum of Propagation Segment and Phase Segment 1 as specified in the standard for Nominal Bit Timing. The actual value is one more than the value written to the register. These bits can be written only when the CEN bit in SRR is 0. **Value After Reset:** 0x0

13.2.2.2.1.5. Error Count Register (ECR)

The ECR register is at offset 0x10.

The following table shows the register bit assignments.

Table 13-23 Fields for register: ECR

Bits	Name	R/W	Description
[31:16]			Reserved
[15:8]	REC	R	Receive Error Count. Indicates the value of Receive Error Counter. **Value After Reset:** 0x0
[7:0]	TEC	R	Transmit Error Count. Indicates the value of Transmit Error Counter. **Value After Reset:** 0x0

13.2.2.2.1.6. Error Status Register (ESR)

The ESR register is at offset 0x14.

The Error Status register (ESR) indicates the type of error that has occurred on the bus. If more than one error occurs, all relevant error flag bits are set in this register. The ESR is a write 1 to clear register. Writes to this register do not set any bits, but clear the bits that are set.

The following table shows the register bit assignments.

Table 13-24 Fields for register: ESR

Bits	Name	R/W	Description
[31:12]			Reserved
[11]	F_BERR	R/W	Bit Error in CAN FD Data Phase. **Values:** **0x0:** Indicates a bit error has not occurred in Data Phase (Fast) data rate after the last write to this bit. **0x1:** Indicates a bit error occurred in Data Phase (Fast) data rate. If this bit is set, writing a 1 clears it. In transmitter delay compensation phase, any error is reported as fast bit error (by the transmitter). **Value After Reset:** 0x0
[10]	F_STER	R/W	Stuff Error in CAN FD Data Phase. **Values:** **0x0:** Indicates stuff error has not occurred in Data Phase (Fast) data rate. after the last write to this bit. **0x1:** Indicates stuff error occurred in Data Phase (Fast) data rate. If this bit is set, writing a 1 clears it. **Value After Reset:** 0x0

Table 13-24 Fields for register: ESR (continued)

Bits	Name	R/W	Description
[9]	F_FMER	R/W	Form Error in CAN FD Data Phase. **Values:** **0x0:** Indicates form error has not occurred in Data Phase (Fast) data rate. after the last write to this bit. **0x1:** Indicates form error occurred in Data Phase (Fast) data rate. If this bit is set, writing a 1 clears it. **Value After Reset:** 0x0
[8]	F_CRCER	R/W	CRC Error in CAN FD Data Phase. **Values:** **0x0:** Indicates CRC error has not occurred in Data Phase (Fast) data rate after the last write to this bit. **0x1:** Indicates CRC error occurred in Data Phase (Fast) data rate. If this bit is set, writing a 1 clears it. **Value After Reset:** 0x0
[7:5]			Reserved
[4]	ACKER	R/W	ACK Error. Indicates an acknowledgment error. **Values:** **0x0:** Indicates an acknowledgment error has not occurred on the bus after the last write to this bit. **0x1:** Indicates an acknowledgment error has occurred. If this bit is set, writing a 1 clears it. **Value After Reset:** 0x0
[3]	BERR	R/W	Bit Error. Indicates the received bit is not the same as the transmitted bit during bus communication. **Values:** **0x0:** Indicates a bit error has not occurred on the bus after the last write to this bit. **0x1:** Indicates a bit error has occurred. If this bit is set, writing a 1 clears it. **Value After Reset:** 0x0
[2]	STER	R/W	Stuff Error. Indicates an error if there is a stuffing violation. **Values:** **0x0:** Indicates a stuff error has not occurred on the bus after the last write to this bit. **0x1:** Indicates a stuff error has occurred. If this bit is set, writing a 1 clears it. **Value After Reset:** 0x0
[1]	FMER	R/W	Form Error.

Table 13-24 Fields for register: ESR (continued)

Bits	Name	R/W	Description
			Indicates an error in one of the fixed form fields in the message frame. **Values:** **0x0:** Indicates a form error has not occurred on the bus after the last write to this bit. **0x1:** Indicates a form error has occurred. If this bit is set, writing a 1 clears it. **Value After Reset:** 0x0
[0]	CRCER	R/W	CRC Error. Indicates a CRC error has occurred. **Values:** **0x0:** Indicates a CRC error has not occurred on the bus after the last write to this bit. **0x1:** Indicates a CRC error has occurred. If this bit is set, writing a 1 clears it. In case of a CRC Error and a CRC delimiter corruption, only the FMER bit is set. **Value After Reset:** 0x0

13.2.2.2.1.7. Status Register (SR)

The SR register is at offset 0x18.

The following table shows the register bit assignments.

Table 13-25 Fields for register: SR

Bits	Name	R/W	Description
[31:21]			Reserved
[22:16]	TDCV	R	Transmitter Delay Compensation Value. This field gives the position of secondary sample point (defined as sum of DP_BRPR.TDCOFF and measured delay for TBiDLC.FDF to res bit falling edge from TX to RX in CAN FD frame) in CAN clocks. This field is for status purposes. **Value After Reset:** 0x0
[15:13]			Reserved
[12]	SNOOP	R	Snoop Mode. **Value:** **0x1:** Indicates controller is in Snoop mode provided Normal mode bit is also set in this register. **Value After Reset:** 0x0
[11]			Reserved
[10]	BSFR_CONFIG	R	Bus-Off Recovery Mode Indicator. **Value:**

Table 13-25 Fields for register: SR (continued)

Bits	Name	R/W	Description
			0x1: Indicates the CAN FD is in Bus-Off Recovery mode (Bus Integration State). When this bit is set, the BBSY and NORMAL status bits in this register do not mean anything. **Value After Reset:** 0x0
[9]	PEE_CONFIG	R	PEE Mode Indicator. **Value:** **0x1:** Indicates the CAN FD is in PEE mode (Bus Integration State). When this bit is set, the BBSY and NORMAL status bits in this register do not mean anything. **Value After Reset:** 0x0
[8:7]	ESTAT	R	Error Status. Indicates the error status of the CAN FD. **Values:** **0x0:** Indicates Configuration mode (CONFIG = 1). Error state is undefined. **0x1:** Indicates error active state. **0x2:** Indicates Bus-Off state. **0x3:** Indicates error passive state. **Value After Reset:** 0x0
[6]	ERRWRN	R	Error Warning. Indicates that either the Transmit Error counter or the Receive Error counter has exceeded a value of 96. **Values:** **0x0:** Neither of the error counters has a value of 96. **0x1:** One or more error counters have a value of 96. **Value After Reset:** 0x0
[5]	BBSY	R	Indicates the CAN bus status. **Values:** **0x0:** Indicates that the CAN FD is either in Configuration mode or the bus is idle. **0x1:** Indicates that the CAN FD is either receiving a message or transmitting a message. **Value After Reset:** 0x0
[4]	BIDLE	R	Bus Idle. Indicates the CAN bus status. **Values:** **0x0:** Indicates the CAN FD is either in Configuration mode or the bus is busy. **0x1:** Indicates no bus communication is taking place. **Value After Reset:** 0x0
[3]	NORMAL	R	Normal Mode.

Table 13-25 Fields for register: SR (continued)

Bits	Name	R/W	Description
			Indicates that the CAN FD is in Normal mode. **Values:** **0x0:** Indicates that the CAN FD is not in Normal mode. **0x1:** Indicates that the CAN FD is in Normal mode. **Value After Reset:** 0x0
[2]	SLEEP	R	Sleep Mode. Indicates that the CAN FD is in Sleep mode. **Values:** **0x0:** Indicates that the CAN FD is not in Sleep mode. **0x1:** Indicates that the CAN FD is in Sleep mode. **Value After Reset:** 0x0
[1]	LBACK	R	Loopback Mode. Indicates that the CAN FD is in Loopback mode. **Values:** **0x0:** Indicates that the CAN FD is not in Loopback mode. **0x1:** Indicates that the CAN FD is in Loopback mode. **Value After Reset:** 0x0
[0]	CONFIG	R	Configuration Mode Indicator. Indicates that the CAN FD is in Configuration mode. **Values:** **0x0:** Indicates that the CAN FD is not in Configuration mode. **0x1:** Indicates that the CAN FD is in Configuration mode. **Value After Reset:** 0x1

13.2.2.2.1.8. Interrupt Status Register (ISR)

The ISR register is at offset 0x1C.

Interrupt status bits in the ISR can be cleared by writing to the ICR. For all bits in the ISR, a set condition takes priority over the clear condition and the bit continues to remain 1.

The following table shows the register bit assignments.

Table 13-26 Fields for register: ISR

Bits	Name	R/W	Description
[31]	TXEWMFLL	R	Tx Event FIFO Watermark Full Interrupt. **Value:** **0x1:** Indicates that Tx Event FIFO is full based on watermark programming.

Table 13-26 Fields for register: ISR (continued)

Bits	Name	R/W	Description
			The interrupt continues to assert as long as the Tx Event FIFO Fill Level is above Tx Event FIFO Full watermark. This bit can be cleared by writing to the respective bit in the ICR. **Value After Reset:** 0x0
[30]	TXEOFLW	R	Tx Event FIFO Overflow Interrupt. **Value:** **0x1:** Indicates that a message has been lost. This condition occurs when the CAN FD has successfully transmitted a message for which an event store is requested but the Tx Event FIFO is full. This bit can be cleared by writing to the respective bit in the ICR.This bit is also cleared when a 0 is written to the CEN bit in SRR. **Value After Reset:** 0x0
[29:18]			Reserved
[17]	RXMNF	R	Rx Match Not Finished. **Value:** **0x1:** Indicates that Match process did not finish until the start of sixth bit in EOF field and frame was discarded. This bit can be cleared by writing to the respective bit in the ICR. **Value After Reset:** 0x0
[16:15]			Reserved
[14]	TXCRS	R	Tx Cancellation Request Served Interrupt. **Value:** **0x1:** Indicates that a cancellation request was cleared. This bit can be cleared by writing to the respective bit in the ICR. **Value After Reset:** 0x0
[13]	TXRRS	R	Tx Buffer Ready Request Served Interrupt. **Value:** **0x1:** Indicates that a Buffer Ready request was cleared. This bit can be cleared by writing to the respective bit in the ICR. **Value After Reset:** 0x0
[12]	RXFWMFLL	R	Rx FIFO-0 Watermark Full Interrupt. **Value:** **0x1:** Indicates that Rx FIFO-0 is full based on watermark programming. The interrupt continues to assert as long as the Rx FIFO-0 Fill Level is above Rx FIFO-0 Full watermark.

Table 13-26 Fields for register: ISR (continued)

Bits	Name	R/W	Description
			This bit can be cleared by writing to the respective bit in the ICR. **Value After Reset:** 0x0
[11]	WKUP	R	Wake-Up Interrupt. **Value:** **0x1:** Indicates that the core entered Normal mode from Sleep mode. This bit can be cleared by writing to the respective bit in the ICR. This bit is also cleared when a 0 is written to the CEN bit in SRR. **Value After Reset:** 0x0
[10]	SLP	R	Sleep Interrupt. **Value:** **0x1:** Indicates that the CAN core entered Sleep mode. This bit can be cleared by writing to the respective bit in the ICR. This bit is also cleared when a 0 is written to the CEN bit in SRR. **Value After Reset:** 0x0
[9]	BSOFF	R	Bus-Off Interrupt. **Value:** **0x1:** Indicates that the CAN core entered the Bus-Off state. This bit can be cleared by writing to the respective bit in the ICR. This bit is also cleared when a 0 is written to the CEN bit in SRR.
[8]	ERROR	R	Error Interrupt. **Value:** **0x1:** Indicates that an error occurred during message transmission or reception. This bit can be cleared by writing to the respective bit in the ICR. This bit is also cleared when a 0 is written to the CEN bit in SRR. **Value After Reset:** 0x0
[7]			Reserved
[6]	RXFOFLW	R	Rx FIFO-0 Overflow Interrupt. **Value:** **0x1:** Indicates that a message has been lost. This condition occurs when a new message with ID matching to Rx FIFO-0 is received and the Rx FIFO-0 is full. This bit can be cleared by writing to the respective bit in the ICR.This bit is also cleared when a 0 is written to the CEN bit in SRR. **Value After Reset:** 0x0

Table 13-26 Fields for register: ISR (continued)

Bits	Name	R/W	Description
[5]	TSCNT_OFLW	R	Timestamp Counter Overflow Interrupt. **Values:** **0x1:** Indicates that Timestamp counter rolled over (from 0xFFFF to 0x0). This bit can be cleared by writing to the respective bit in the ICR. This bit is also cleared when a 0 is written to the CEN bit in SRR. **Value After Reset:** 0x0
[4]	RXOK	R	New Message Received Interrupt. **Value:** **0x1:** Indicates that a message was received successfully and stored into the Rx FIFO-0. This bit can be cleared by writing to the respective bit in the ICR.This bit is also cleared when a 0 is written to the CEN bit in SRR. In Loopback mode, both TXOK and RXOK bits are set. The RXOK bit is set before the TXOK bit. **Value After Reset:** 0x0
[3]	BSFRD	R	Bus-Off Recovery Done Interrupt. **Value:** **0x1:** Indicates that the CAN FD recovered from Bus-Off state. This bit can be cleared by writing to the respective bit in the ICR.This bit is also cleared when a 0 is written to the CEN bit in SRR. **Value After Reset:** 0x0
[2]	PEE	R	Protocol Exception Event Interrupt. **Value:** **0x1:** Indicates that the CAN FD receiver has detected PEE event. This bit can be cleared by writing to the respective bit in the ICR. This bit is also cleared when a 0 is written to the CEN bit in SRR. **Value After Reset:** 0x0
[1]	TXOK	R	Transmission Successful Interrupt. **Value:** **0x1:** Indicates that a message was transmitted successfully. This bit can be cleared by writing to the respective bit in the ICR. This bit is also cleared when a 0 is written to the CEN bit in SRR. In Loopback mode, both TXOK and RXOK bits are set. The RXOK bit is set before the TXOK bit.

Table 13-26 Fields for register: `ISR` (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0
[0]	ARBLST	R	Arbitration Lost Interrupt. **Values:** **0x1:** Indicates that arbitration was lost during message transmission. This bit can be cleared by writing to the respective bit in the ICR. This bit is also cleared when a 0 is written to the CEN bit in SRR. **Value After Reset:** 0x0

13.2.2.2.1.9. Interrupt Enable Register (`IER`)

The `IER` register is at offset 0x20.

The Interrupt Enable register (`IER`) bits are used to enable interrupt generation when respective event happens.

The following table shows the register bit assignments.

Table 13-27 Fields for register: `IER`

Bits	Name	R/W	Description
[31]	ETXEWMFLL	R/W	Tx Event FIFO Watermark Full Interrupt Enable. **Values:** **0x0:** Disables interrupt generation if TXEWMFLL bit in the ISR is set. **0x1:** Enables interrupt generation if TXEWMFLL bit in the ISR is set. **Value After Reset:** 0x0
[30]	ETXEOFLW	R/W	Tx Event FIFO Overflow Interrupt Enable. **Values:** **0x0:** Disables interrupt generation if TXEOFLW bit in the ISR is set. **0x1:** Enables interrupt generation if TXEOFLW bit in the ISR is set. **Value After Reset:** 0x0
[29:18]			Reserved
[17]	ERXMNF	R/W	Rx Match Not Finished interrupt Enable. **Values:** **0x0:** Disables interrupt generation if RXMNF bit in the ISR is set. **0x1:** Enables interrupt generation if RXMNF bit in the ISR is set. **Value After Reset:** 0x0
[16:15]			Reserved

Table 13-27 Fields for register: IER (continued)

Bits	Name	R/W	Description
[14]	ETXCRS	R/W	Tx Cancellation Request Served Interrupt Enable. **Values:** **0x0:** Disables interrupt generation if TXCRS bit in the ISR is set. **0x1:** Enables interrupt generation if TXCRS bit in the ISR is set. **Value After Reset:** 0x0
[13]	ETXRRS	R/W	Tx Buffer Ready Request Served Interrupt Enable. **Values:** **0x0:** Disables interrupt generation if TXRRS bit in the ISR is set. **0x1:** Enables interrupt generation if TXRRS bit in the ISR is set. **Value After Reset:** 0x0
[12]	ERXFWMFLL	R/W	Rx FIFO-0 Watermark Full Interrupt Enable (Sequential/ FIFO Mode). **Values:** **0x0:** Disables interrupt generation if RXFWMFLL bit in the ISR is set. **0x1:** Enables interrupt generation if RXFWMFLL bit in the ISR is set. **Value After Reset:** 0x0
[11]	EWKUP	R/W	Wake-Up Interrupt Enable. **Values:** **0x0:** Disables interrupt generation if WKUP bit in the ISR is set. **0x1:** Enables interrupt generation if WKUP bit in the ISR is set. **Value After Reset:** 0x0
[10]	ESLP	R/W	Sleep Interrupt Enable. **Values:** **0x0:** Disables interrupt generation if SLP bit in the ISR is set. **0x1:** Enables interrupt generation if SLP bit in the ISR is set. **Value After Reset:** 0x0
[9]	EBSOFF	R/W	Bus-Off Interrupt Enable. **Values:** **0x0:** Disables interrupt generation if BSOFF bit in the ISR is set. **0x1:** Enables interrupt generation if BSOFF bit in the ISR is set. **Value After Reset:** 0x0

Table 13-27 Fields for register: IER (continued)

Bits	Name	R/W	Description
[8]	EERROR	R/W	Error Interrupt Enable. **Values:** **0x0:** Disables interrupt generation if ERROR bit in the ISR is set. **0x1:** Enables interrupt generation if ERROR bit in the ISR is set. **Value After Reset:** 0x0
[7]			Reserved
[6]	ERFXOFLW	R/W	Rx FIFO-0 Overflow Interrupt Enable (Sequential/FIFO Mode). **Values:** **0x0:** Disables interrupt generation if RFXOFLW bit in the ISR is set. **0x1:** Enables interrupt generation if RFXOFLW bit in the ISR is set. **Value After Reset:** 0x0
[5]	ETSCNT_OFLW	R/W	Timestamp Counter Overflow Interrupt Enable. **Values:** **0x0:** Disables interrupt generation if TSCNT_OFLW bit in the ISR is set. **0x1:** Enables interrupt generation if TSCNT_OFLW bit in the ISR is set. **Value After Reset:** 0x0
[4]	ERXOK	R/W	New Message Received Interrupt Enable. **Values:** **0x0:** Disables interrupt generation if RXOK bit in the ISR is set. **0x1:** Enables interrupt generation if RXOK bit in the ISR is set. **Value After Reset:** 0x0
[3]	EBSFRD	R/W	Bus-Off Recovery Done Interrupt Enable. **Values:** **0x0:** Disables interrupt generation if BSFRD bit in the ISR is set. **0x1:** Enables interrupt generation if BSFRD bit in the ISR is set. **Value After Reset:** 0x0
[2]	EPEE	R/W	Protocol Exception Event Interrupt Enable. **Values:** **0x0:** Disables interrupt generation if PEE bit in the ISR is set. **0x1:** Enables interrupt generation if PEE bit in the ISR is set.

Table 13-27 Fields for register: IER (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0
[1]	ETXOK	R/W	Transmission Successful Interrupt Enable. **Values:** **0x0:** Disables interrupt generation if TXOK bit in the ISR is set. **0x1:** Enables interrupt generation if TXOK bit in the ISR is set. **Value After Reset:** 0x0
[0]	EARBLOST	R/W	Arbitration Lost Interrupt Enable. **Values:** **0x0:** Disables interrupt generation if ARBLST bit in the ISR is set. **0x1:** Enables interrupt generation if ARBLST bit in the ISR is set. **Value After Reset:** 0x0

13.2.2.2.1.10. Interrupt Clear Register (ICR)

The ICR register is at offset 0x24.

The Interrupt Clear register (ICR) is used to clear interrupt status bits in the ISR register.

The following table shows the register bit assignments.

Table 13-28 Fields for register: ICR

Bits	Name	R/W	Description
[31]	CTXEWMFLL	W	Clear Tx Event FIFO Watermark Full Interrupt. **Value:** **0x1:**Clears Tx Event FIFO Watermark Full interrupt status bit. Writing a 1 to this bit clears the respective bit in the ISR. Reads always 0. **Value After Reset:** 0x0
[30]	CTXEOFLW	W	Clear Tx Event FIFO Overflow Interrupt. **Value:** **0x1:** Clears Tx Event FIFO Overflow interrupt status bit. Writing a 1 to this bit clears the respective bit in the ISR. Reads always 0. **Value After Reset:** 0x0
[29:18]			Reserved
[17]	CRXMNF	W	Clear Rx Match Not Finished Interrupt. **Value:** **0x1:** Clears Rx Match Not Finished interrupt status bit.

Table 13-28 Fields for register: ICR (continued)

Bits	Name	R/W	Description
			Writing a 1 to this bit clears the respective bit in the ISR. Reads always 0. **Value After Reset:** 0x0
[16:15]			Reserved
[14]	CTXCRS	W	Clear Tx Cancellation Request Served Interrupt. **Value:** **0x1:** Clears Tx Cancellation Request Served Interrupt status bit. Writing a 1 to this bit clears the respective bit in the ISR. Reads always 0. **Value After Reset:** 0x0
[13]	CTXRRS	W	Clear Tx Buffer Ready Request Served Interrupt. **Value:** **0x1:** Clears Tx Buffer Ready Request Served Interrupt status bit. Writing a 1 to this bit clears the respective bit in the ISR. Reads always 0. **Value After Reset:** 0x0
[12]	CRXFWMFLL	W	Clear Rx FIFO-0 Watermark Full Interrupt. **Value:** **0x1:** Clears Rx FIFO-0 Watermark Full interrupt status bit. Writing a 1 to this bit clears the respective bit in the ISR. Reads always 0. **Value After Reset:** 0x0
[11]	CWKUP	W	Clear Wake-Up Interrupt. **Value:** **0x1:** Clears Wake-Up interrupt status bit. Writing a 1 to this bit clears the respective bit in the ISR. Reads always 0. **Value After Reset:** 0x0
[10]	CSLP	W	Clear Sleep Interrupt. **Value:** **0x1:** Clears Sleep interrupt status bit. Writing a 1 to this bit clears the respective bit in the ISR. Reads always 0. **Value After Reset:** 0x0
[9]	CBSOFF	W	Clear Bus-Off Interrupt. **Value:** **0x1:** Clears Bus-Off interrupt status bit. Writing a 1 to this bit clears the respective bit in the ISR. Reads always 0. **Value After Reset:** 0x0

Table 13-28 Fields for register: ICR (continued)

Bits	Name	R/W	Description
[8]	CERROR	W	Clear Error Interrupt. **Value:** **0x1:** Clears Error interrupt status bit. Writing a 1 to this bit clears the respective bit in the ISR. Reads always 0. **Value After Reset:** 0x0
[7]			Reserved
[6]	CRFXOFLW	W	Clear Rx FIFO-0 Overflow Interrupt. **Value:** **0x1:** Clears RX FIFO-0 Overflow interrupt status bit. Writing a 1 to this bit clears the respective bit in the ISR. Reads always 0. **Value After Reset:** 0x0
[5]	ETSCNT_OFLW	W	Clear Timestamp Counter Overflow Interrupt. **Value:** **0x1:** Clears Timestamp Counter Overflow Interrupt status bit. Writing a 1 to this bit clears the respective bit in the ISR. Reads always 0. **Value After Reset:** 0x0
[4]	CRXOK	W	Clear New Message Received Interrupt. **Value:** **0x1:** Clears New Message Received interrupt status bit. Writing a 1 to this bit clears the respective bit in the ISR. Reads always 0. **Value After Reset:** 0x0
[3]	CBSFRD	W	Clear Bus-Off Recovery Done Interrupt. **Value:** **0x1:** Clears Bus-Off Recovery Done interrupt status bit. Writing a 1 to this bit clears the respective bit in the ISR. Reads always 0. **Value After Reset:** 0x0
[2]	CPEE	W	Clear Protocol Exception Event Interrupt. **Value:** **0x1:** Clears Protocol Exception Event interrupt status bit. Writing a 1 to this bit clears the respective bit in the ISR. Reads always 0. **Value After Reset:** 0x0
[1]	CTXOK	W	Clear Transmission Successful Interrupt. **Value:**

Table 13-28 Fields for register: ICR (continued)

Bits	Name	R/W	Description
			0x1: Clears Transmission Successful interrupt status bit. **Value After Reset:** 0x0
[0]	CARBLOST	W	Clear Arbitration Lost Interrupt. **Value:** **0x1:** Clears Arbitration Lost interrupt status bit.. Writing a 1 to this bit clears the respective bit in the ISR. Reads always 0. **Value After Reset:** 0x0

13.2.2.2.1.11. Timestamp Register (TSR)

The TSR register is at offset 0x28.

The following table shows the register bit assignments.

Table 13-29 Fields for register: TSR

Bits	Name	R/W	Description
[31:16]	TIMESTAMP_CNT	R	Timestamp Counter Value. This Status field gives running value of the timestamp counter. This field is cleared when a 0 is written to the CEN bit in the SRR. **Value After Reset:** 0x0
[15:1]			Reserved
[0]	CTS	W	Clear Timestamp Counter. Internal free running counter is cleared to 0 when CTS = 1. This bit only needs to be written once with a 1 to clear the counter. The bit always reads as 0. **Value After Reset:** 0x0

13.2.2.2.1.12. Data Phase Baud Rate Prescaler Register (DP_BRPR)

The DP_BRPR register is at offset 0x88.

The following table shows the register bit assignments.

Table 13-30 Fields for register: DP_BRPR

Bits	Name	R/W	Description
[31:17]			Reserved
[16]	TDC	R/W	*Transmitter Delay Compensation* (**TDC**) Enable. **Values:** **0x0:** TDC is disabled. **0x1:** Enables TDC function as specified in the CAN FD standard. This bit can be written only when CEN bit in SRR is 0.

Table 13-30 Fields for register: DP_BRPR (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0
[15:14]			Reserved
[13:8]	TDCOFF	R/W	Transmitter Delay Compensation Offset. This offset is specified in CAN clock cycles and is added to the measured transmitter delay to place the *Secondary Sample Point* (**SSP**) at appropriate position (for example, set this to half data bit time in terms of CAN clock cycles to place SSP in the middle of the data bit). This bit can be written only when CEN bit in SRR is 0. **Value After Reset:** 0x0
[7:0]	DP_BRP	R/W	Data Phase Baud Rate Prescaler. These bits indicate the prescaler value for Data Bit Timing as specified in the CAN FD standard. The actual value is one more than the value written to the register. This bit can be written only when CEN bit in SRR is 0. **Value After Reset:** 0x0

The following boundary conditions are imposed on sum of measured loop delay and TDC Offset: `Measured loop delay + TDCOFF < 3 bit times in the data phase`.

Ensure that the boundary condition is respected while programming the offset and data phase bit rate. In case this sum exceeds 127 CAN clock periods, the maximum value of 127 CAN clock periods is used by the core for transmitter delay compensation.

If loop delay is < 1 data phase bit time, then TDC/SSP method is not needed.

13.2.2.2.1.13. Data Phase Bit Timing Register (DP_BTR)

The DP_BTR register is at offset 0x8C.

The following table shows the register bit assignments.

Table 13-31 Fields for register: DP_BTR

Bits	Name	R/W	Description
[31:20]			Reserved
[19:16]	DP_SJW	R/W	Data Phase Synchronization Jump Width. Indicates the Synchronization Jump Width as specified in the CAN FD standard for Data Bit Timing. The actual value is one more than the value written to the register. This bit can be written only when CEN bit in SRR is 0. **Value After Reset:** 0x0
[15:12]			Reserved
[11:8]	DP_TS2	R/W	Data Phase Time Segment 2. Indicates the Phase Segment 2 as specified in the CAN FD standard for Data Bit Timing.

Table 13-31 Fields for register: DP_BTR (continued)

Bits	Name	R/W	Description
			The actual value is one more than the value written to the register. This bit can be written only when CEN bit in SRR is 0. **Value After Reset:** 0x0
[7:5]			Reserved
[4:0]	DP_TS1	R/W	Data Phase Time Segment 1. Indicates the Sum of Propagation Segment and Phase Segment 1 as specified in the CAN FD standard for Data Bit Timing. The actual value is one more than the value written to the register. This bit can be written only when CEN bit in SRR is 0. **Value After Reset:** 0x0

13.2.2.2.1.14. Tx Buffer Ready Request Register (TRR)

The TRR register is at offset 0x90.

The following table shows the register bit assignments.

Table 13-32 Fields for register: TRR

Bits	Name	R/W	Description
[31:0]	RRi	R/W	Tx Buffer_i Ready Request (for i = 0; i <=31). ⚠ RRi corresponds to the bit with number [i], i.e. bit [31] corresponds to the RR31, [30] to the RR30, [29] to the RR29 and etc. This is control bit corresponds to TBi (for i = 0; i <=31) message in Tx block RAM. Host writes 1 to indicate buffer is ready for transmission. Core clears this bit when: • Buffer transmission is completed on CAN Bus • If core is in DAR mode, then after one transmission attempt on the CAN bus [either successful or unsuccessful (that is, arbitration lost or error)] • If message is cancelled due to cancellation request • Any combination of the above three. Host writes to this bit are ignored when this bit is 1. 📝 This register remains in reset when SNOOP mode is enabled. **Value After Reset:** 0x0

📝
- Host can set transmission requests for multiple buffers in one write to this register.
- Write with any value to this register triggers buffer scheduler to redo the scheduling round to find winning buffer (exceptions: when Transfer Layer is in 3-bit Intermission space without locked buffer or if previous scheduling round is already running. In those situation, buffer scheduler trigger is postponed till the event is over).

Unnecessary writes to this register might reduce core throughput on the CAN bus. Ensure this register is written only when it is required.

13.2.2.2.1.15. Interrupt Enable Tx Buffer Ready Request Served/Cleared (IETRS)

The IETRS register is at offset 0x94.

The following table shows the register bit assignments.

Table 13-33 Fields for register: IETRS

Bits	Name	R/W	Description
[31:0]	ERRSi	R/W	Tx Buffer_i (for i = 0; i <=31) Ready Req Served/Cleared Interrupt Enable. ERRSi corresponds to the bit with number [i], i.e. bit [31] corresponds to the ERRS31, [30] to the ERRS30, [29] to the ERRS29 and etc. **Values:** **0x0:** TXRRS bit in the ISR does not set if RRi (for i = 0; i <=31) bit in TRR register clears. **0x1:** Enables setting TXRRS bit in the ISR when RRi (for i = 0; i <=31) bit in TRR register clears. **Value After Reset:** 0x0

13.2.2.2.1.16. Tx Buffer Cancel Request (TCR)

The TCR register is at offset 0x98.

The following table shows the register bit assignments.

Table 13-34 Fields for register: TCR

Bits	Name	R/W	Description
[31:0]	CRi	R/W	Tx Buffer_i (for i = 0; i <=31) Cancel Request. CRi corresponds to the bit with number [i], i.e. bit [31] corresponds to the CR31, [30] to the CR30, [29] to the CR29 and etc This is cancellation request bit corresponds to RRi (for i = 0; i <=31) bit in TRR register. Host writes 1 to indicate cancellation request of corresponding buffer ready request (that is, RRi bit in TRR register). The core clears this bit when cancellation request is completed. Host writes to this bit are ignored if CRi is 1 or RRi bit of TRR register is 0. If the buffer is already locked for transmission then cancellation is performed at the end of transmission cycle irrespective whether frame transmitted successfully or failed. That is, if message is failed due to arbitration loss or any error, then the message is cancelled (no retransmission attempt)

Table 13-34 Fields for register: TCR (continued)

Bits	Name	R/W	Description
			and cancellation request is cleared. Along with RRi bit this is cleared. If message is transmitted successfully, then RRi bit in TRR clears and cancellation request is cleared anyway. If internal buffer scheduling round is in progress, then cancellation consideration is postponed till it is over. **Value After Reset:** 0x0

Host can set cancellation requests for multiple buffers in one write to this register.

Performing unnecessary cancellation of Tx buffers might reduce core throughput on the CAN bus. Ensure that buffer cancellation is requested only when it is required.

13.2.2.2.1.17. Interrupt Enable Tx Buffer Cancellation Served/Cleared (IETCS)

The IETRS register is at offset 0x9C.

The following table shows the register bit assignments.

Table 13-35 Fields for register: IETCS

Bits	Name	R/W	Description
[31:0]	ECRSi	R/W	Tx Buffer_i (for i = 0; i <=31) Transmission Served/Cleared Interrupt Enable. ECRSi corresponds to the bit with number [i], i.e. bit [31] corresponds to the ECRS31, [30] to the ECRS30, [29] to the ECRS29 and etc **Values:** **0x0:** TXCRS bit in the ISR does not set if CRi (for i = 0; i <=31) bit in TCR register clears. **0x1:** Enables setting TXCRS bit in the ISR when CRi (for i = 0; i <=31) bit in TCR register clears. **Value After Reset:** 0x0

13.2.2.2.1.18. Tx Event FIFO Status Register (TxE_FSR)

The TxE_FSR register is at offset 0xA0.

The following table shows the register bit assignments.

Table 13-36 Fields for register: TxE_FSR

Bits	Name	R/W	Description
[31:14]			Reserved
[13:8]	TXE_FL	R	Fill Level (0-31).

Table 13-36 Fields for register: TxE_FSR (continued)

Bits	Name	R/W	Description
			Number of stored message in Tx Event FIFO starting from Read Index given in this register. For example, if TXE_FL = 0x5 and TXE_RI = 0x3 then Tx Event FIFO has five messages starting from Read Index 3 (Start address 0x2018). TXE_FL is maintained if CEN bit in SRR is cleared. TXE_FL gets reset to 0 if soft or hard reset is asserted. **Value After Reset:** 0x0
[7]	TXE_IRI	W	Increment Read Index by 1. With each Host writes setting this bit as 1, CAN FD increments Read index (TXE_RI) by 1 and update fill level (that is, decrement by 1). If FILL level is 0, setting this bit has no effect. The FILL level might remain unchanged when TXE_IRI is written if CAN FD is just finishing a successful transmission and incrementing internal write index. This bit always read as 0. **Value After Reset:** 0x0
[6:5]			Reserved
[4:0]	TXE_RI	R	Read Index (0 to 63). Each time IRI bit is set, CAN FD increments read index by + 1 (provided FILL level is not 0) and maintains it for Host to access next available message. **Values:** **0x0:** Next message read starts from location = 0x2000. **0x1:** Next message read starts from location = 0x2008. TXE_RI is maintained if CEN in SRR bit is cleared. TXE_RI gets reset to 0 if soft or hard reset is asserted. **Value After Reset:** 0x0

13.2.2.2.1.19. Tx Event FIFO Watermark Register (TxE_WMR)

The TxE_WMR register is at offset 0xA4.

The following table shows the register bit assignments.

Table 13-37 Fields for register: TxE_WMR

Bits	Name	R/W	Description
[31:5]			Reserved
[4:0]	TXE_FWM	R/W	Tx Event FIFO Full Watermark. Tx Event FIFO generates FULL interrupt based on the value programmed in this field. Set it within (1-31) range. The Tx FIFO Full Watermark interrupt in the ISR register continues to assert as long as the Tx Event FIFO Fill Level is above Tx Event FIFO Full watermark. This field can be written to only when CEN bit in SRR is 0. **Value After Reset:** 0xF

13.2.2.2.1.20. Acceptance Filter (Control) Register (AFR)

The AFR register is at offset 0xE0.

There are 32 acceptance filters. Each acceptance filter has the Acceptance Filter i Mask Register (AFMRi) and Acceptance Filter i ID registers Register (AFIRi) (which is controlled by the Acceptance Filter (Control) Register bits).

Each AFIRi and AFMRi pair is associated with a UAF bit.

- When the UAF bit is 1, the corresponding acceptance filter pair is used for acceptance filtering. When the UAF bit is 0, the corresponding acceptance filter pair is not used for acceptance filtering.
- To modify an acceptance filter pair in Normal mode, the corresponding UAF bit in this register must be set to 0.
- After the acceptance filter is modified, the corresponding UAF bit must be set to 1.
- If all UAF bits are set to 0, the received messages are not stored in the Rx Sequential/ FIFO buffers.
- If the UAF bits are changed from a 1 to 0 during reception of a message, then that message might or might not be stored.

The following table shows the register bit assignments.

Table 13-38 Fields for register: AFR

Bits	Name	R/W	Description
[31:0]	UAFi	R/W	Use Acceptance Filter Mask Pairi (for i = 0; i <=31). UAFi corresponds to the bit with number [i], i.e. bit [31] corresponds to the UAF31, [30] to the UAF30, [29] to the UAF29 and etc Enables the use of acceptance filter mask pair i. **Values:** **0x0:** Indicates AFMRi and AFIRi pair is not used for acceptance filtering. **0x1:** Indicates AFMRi and AFIRi pair is used for acceptance filtering. **Value After Reset:** 0x0

13.2.2.2.1.21. Rx FIFO Status Register (FSR)

The FSR register is at offset 0xE8 .

The following table shows the register bit assignments.

Table 13-39 Fields for register: FSR

Bits	Name	R/W	Description
[31:15]			Reserved
[14:8]	FL	R	Rx FIFO-0 Fill Level (0 to 31). The number of stored messages in Receive FIFO 0 starting from the Read Index is given in this register.

Table 13-39 Fields for register: FSR (continued)

Bits	Name	R/W	Description
			For example, if FL = 0x5 and RI = 0x2, then Rx FIFO-0 has five messages starting from Read Index 2 (Start address 0x4190). FL is maintained if CEN bit in SRR is cleared. FL is reset to 0 if a soft or hard reset is asserted. **Value After Reset:** 0x0
[7]	IRI	W	Rx FIFO-0 Increment Read Index by 1. With each Host writes setting this bit as 1, the core increments the Read Index by 1 and updates fill level (that is, decrement by 1). If FILL level is 0, setting this bit has no effect. The FILL level might remain unchanged when IRI is written if core is just finishing a successful receive and incrementing internal write index. This bit always read as 0. **Value After Reset:** 0x0
[6]			Reserved
[5:0]	RI	R	Rx FIFO-0 Read Index (0 to 63). Each time IRI bit is set, CAN FD increments read index by + 1 (provided FILL level is not 0) and maintains it for Host to access next available message. **Values:** **0x0:** Next message read starts from location = 0x2100. **0x1:** Next message read starts from location = 0x2148. RI is maintained if CEN in SRR is cleared. RI gets reset to 0 if soft or hard reset is asserted. **Value After Reset:** 0x0

13.2.2.2.1.22. Rx FIFO Watermark Register (WMR)

The WMR register is at offset 0xEC.

The following table shows the register bit assignments.

Table 13-40 Fields for register: WMR

Bits	Name	R/W	Description
[31:6]			Reserved **Value After Reset:** 0x1F0F00
[5:0]	RXFWM	R/W	Rx FIFO-0 Full Watermark. Rx FIFO-0 generates FULL interrupt based on the value programmed in this field. Set it within (1-63) range. The Rx FIFO-0 Full Watermark interrupt in the ISR register continues to assert as long as the Rx FIFO-0 Fill Level is above Rx FIFO-0 Full watermark. This field can be written to only when CEN bit in SRR is 0. **Value After Reset:** 0xF

13.2.2.2.2. CAN FD Tx Message Space

13.2.2.2.2.1. TBi Message ID Register (`TBiID`)

The `TBiID` (for i = 0; i <=31) register is offset as 0x100 + (i*0x48).

The following table shows the register bit assignments.

Table 13-41 Fields for register: `TBiID`

Bits	Name	R/W	Description
[31:21]	`ID[28:18]`	R/W	Standard Message ID. The Identifier portion for a Standard Frame is 11 bits. These bits indicate the Standard Frame ID. This field is valid for both CAN and CAN FD Standard and Extended Frames.
[20]	`SRR/RTR/RRS`	R/W	Substitute Remote Request. For Extended CAN frames and Extended CAN FD frame this bit is transmitted at `SRR` position of the respective frame and must be set as 1. For Standard CAN FD frame this bit is transmitted at `RRS` position of the frame and must be set as 0. This bit differentiates between standard CAN data frames and standard CAN remote frames as the following: **0x0:** Indicates that the message frame is a Standard Data CAN Frame. **0x1:** Indicates that the message frame is a Standard Remote CAN Frame. There are no CAN FD remote frames.
[19]	`IDE`	R/W	Identifier Extension. This bit differentiates between frames using the Standard Identifier and those using the Extended Identifier. Valid for both CAN and CAN FD Standard and Extended Frames. **Values:** **0x0:** Indicates the use of a Standard Message Identifier. **0x1:** Indicates the use of an Extended Message Identifier.
[18:1]	`ID[17:0]`	R/W	Extended Message ID. This field Indicates the Extended Identifier. Valid only for Extended CAN and CAN FD Frames. For Standard CAN and CAN FD Frames, writes to this field should be 0.
[0]	`RTR/RRS`	R/W	Remote Transmission Request. This bit differentiates between CAN extended data frames and CAN extended remote frames. **Values:**

Table 13-41 Fields for register: `TBiID` (continued)

Bits	Name	R/W	Description
			0x0: Indicates the message frame is a CAN Data Frame. For Extended CAN FD frame this bit is transmitted at `RRS` position of the frame and must be set as 0. **0x1:** Indicates the message frame is a CAN Remote Frame. For Standard CAN and CAN FD Frames, writes to this bit should be 0. There are no CAN FD remote frames.

13.2.2.2.2.2. TBi Data Length Code Register (`TBiDLC`)

The `TBiDLC` (for i = 0; i <=31) register is offset as 0x104 + (i*0x48)

The following table shows the register bit assignments.

Table 13-42 Fields for register: `TBiDLC`

Bits	Name	R/W	Description
[31:28]	`DLC`	R/W	Data Length Code. This is the data length code of the R/W field of the CAN and CAN FD frame.
[27]	`EDL/FDF`	R/W	Extended Data Length/FD Frame Format. This bit distinguishes between CAN format and CAN FD format frames. **Values:** **0x0:** CAN format frame **0x1:** CAN FD format frame
[26]	`BRS`	R/W	Bit Rate Switch. The `BRS` bit decides whether the bit rate is switched inside a CAN FD format frame or not (provided `BRSD` bit is not set in `MSR` register). **Values:** **0x0:** Bit rate is not switched inside a CAN FD frame. **0x1:** Bit rate is switched from the standard bit rate of the Arbitration phase to the preconfigured alternate bit rate of the Data phase inside a CAN FD frame. `BRS` does not exist in CAN format frames and should be set to 0.
[25]			Reserved
[24]	`EFC`	R/W	Event FIFO R/W. **Values:** **0x0:** Doesn't store Tx events **0x1:** Stores Tx Events

Table 13-42 Fields for register: `TBiDLC` (continued)

Bits	Name	R/W	Description
[23:16]	`MM`	R/W	Written by software during Tx Buffer configuration. Copied into Tx Event FIFO element for identification of Tx message status.
[15:0]			Reserved

13.2.2.2.2.3. TBi Data Byte n Register (`TBiDWn`)

The `TBiDWn` (for i = 0; i <=31 and for n = 0; n <=15) register is offset as 0x108 + (i*0x48) + (n*0x4)

The following table shows the register bit assignments.

Table 13-43 Fields for register: `TBiDWn`

Bits	Name	R/W	Description
[31:24]	`Data bytes0`	R/W	Data Byte 0. Data byte needs to be transmitted with CAN or CAN FD frame based on the `TBiDLC`.`DLC` control field.
[23:16]	`Data bytes1`	R/W	Data Byte 1. Data byte needs to be transmitted with CAN or CAN FD frame based on the `TBiDLC`.`DLC` control field.
[15:8]	`Data bytes2`	R/W	Data Byte 2. Data byte needs to be transmitted with CAN or CAN FD frame based on the `TBiDLC`.`DLC` control field.
[7:0]	`Data bytes3`	R/W	Data Byte 3. Data byte needs to be transmitted with CAN or CAN FD frame based on the `TBiDLC`.`DLC` control field.

13.2.2.2.2.4. Acceptance Filter i Mask Register (`AFMRi`)

The `AFMRi` (for i = 0; i <=31) register is offset as 0xA00 + (i*0x8)

There are 32 acceptance filters. Each acceptance filter has the Acceptance Filter Mask Register and the `AFIRi` register (which is controlled by the `AFR` register bits). The Acceptance Filter Mask registers contain mask bits used for acceptance filtering. The incoming message identifier portion of a message frame is compared with the message identifier stored in the acceptance filter ID register. The mask bits define which identifier bits stored in the Acceptance Filter ID register are compared to the incoming message identifier for CAN or CAN FD frames.

Acceptance filtering is performed in this sequence:

1. The incoming Identifier is masked with the bits in the Acceptance Filter Mask register.
2. The Acceptance Filter i ID Register is also masked with the bits in the Acceptance Filter Mask register.
3. Both resulting values are compared.
4. If both these values are equal, the message is stored in Rx FIFO-0.
5. Acceptance filtering is processed by each of the defined filters. If the incoming identifier passes through any acceptance filter, the message is stored in Rx FIFO-0.

All bit fields (AMID[28:18], AMSRR, AMIDE, AMID[17:0], and AMRTR) need to be defined for Extended frames. Only AMID[28:18], AMSRR and AMIDE need to be defined for Standard frames. AMID[17:0], and AMRTR should be written as 0 for Standard frames.

The following table shows the register bit assignments.

Table 13-44 Fields for register: AFMRi

Bits	Name	R/W	Description
[31:21]	AMID[28:18]	R/W	Standard Message ID Mask. These bits are used for masking the TBiID.ID[28:18] in a Standard Frame. **Values:** **0x0:** Indicates the corresponding bit in AFIRi is not used when comparing the incoming message identifier. **0x1:** Indicates the corresponding bit in AFIRi is used when comparing the incoming message identifier. **Value After Reset:** 0x0
[20]	AMSRR	R/W	Substitute Remote Transmission Request Mask. This bit is used for masking the TBiID.RTR bit in a Standard Frame. **Values:** **0x0:** Indicates the corresponding bit in AFIRi is not used when comparing the incoming message identifier. **0x1:** Indicates the corresponding bit in AFIRi is used when comparing the incoming meessage identifier. **Value After Reset:** 0x0
[19]	AMIDE	R/W	Identifier Extension Mask. Used for masking the TBiID.IDE bit. **Values:** **0x0:** Indicates the corresponding bit in AFIRi is not used when comparing the incoming message identifier. **0x1:** Indicates the corresponding bit in AFIRi is used when comparing the incoming meessage identifier. If AMIDE = 0x1 and the AIIDE bit in the corresponding AFIRi is 0x0, this mask is applicable to only Standard frames. If AMIDE = 0x1 and the AIIDE bit in the corresponding AFIRi is 0x1, this mask is applicable to only extended frames. If AMIDE = 0x0, this mask is applicable to both Standard and Extended frames. **Value After Reset:** 0x0
[18:1]	AMID[17:0]	R/W	Extended Message ID Mask . These bits are used for masking the TBiID.ID[17:0] in an Extended Frame. **Values:**

Table 13-44 Fields for register: AFMRi (continued)

Bits	Name	R/W	Description
			0x0: Indicates the corresponding bit in AFIRi is not used when comparing the incoming message identifier. **0x1:** Indicates the corresponding bit in AFIRi is used when comparing the incoming meessage identifier. **Value After Reset:** 0x0
[0]	AMRTR	R/W	Remote Transmission Request Mask . This bit is used for masking the TBiID.RTR bit in an Extended Frame. **Values:** **0x0:** Indicates the corresponding bit in AFIRi is not used when comparing the incoming message identifier. **0x1:** Indicates the corresponding bit in AFIRi is used when comparing the incoming meessage identifier. **Value After Reset:** 0x0

13.2.2.2.2.5. Acceptance Filter i ID registers Register (AFIRi)

The AFIRi(for i = 0; i <=31) register is offset as 0xA04 + (i*0x8).

The Acceptance Filter i ID registers (AFIRi) contain Identifier bits, which are used for acceptance filtering. All bit fields (AIID[28:18], AISRR, AIIDE, AIID[17:0] and AIRTR) need to be defined for Extended frames.

Only AIID[28:18], AISRR and AIIDE need to be defined for Standard frames. AIID[17:0] and AIRTR should be written as 0 for Standard frames.

The following table shows the register bit assignments.

Table 13-45 Fields for register: AFIRi

Bits	Name	R/W	Description
[31:21]	AIID [28:18]	R/W	Standard Message ID. Standard Identifier.
[20]	AISRR	R/W	Substitute Remote Transmission Request. Indicates the Remote Transmission Request bit for Standard frames.
[19]	AIIDE	R/W	Identifier Extension. Differentiates between Standard and Extended frames.
[18:1]	AIID[17:0]	R/W	Extended Message ID. Extended Identifier.
[0]	AIRTR	R/W	Remote Transmission Request. TBiID.RTR bit for Extended frames.

13.2.2.2.3. CAN FD Tx Event FIFO Status Register Space

13.2.2.2.3.1. TXE FIFO TBi Message ID Register (TXE_FIFO_TBi_ID)

The TXE_FIFO_TBi_ID (for i = 0; i <=31) register is offset as 0x2000 + (i*0x48).

The following table shows the register bit assignments.

Table 13-46 Fields for register: `TXE_FIFO_TBi_ID`

Bits	Name	R/W	Description
[31:21]	`ID`	R	Standard Message ID. The Identifier portion for a Standard Frame is 11 bits. These bits indicate the Standard Frame ID. This field is valid for both CAN and CAN FD Standard and Extended Frames.
[20]	`SRR/RTR/RRS`	R	Substitute Remote Transmission Request. For Extended CAN frames and Extended CAN FD frame this bit is transmitted at `SRR` position of the respective frame and must be set as 1. For Standard CAN FD frames, this bit is transmitted at the `RRS` position of the frame and must be set as 0. This bit differentiates between standard CAN data frames and standard CAN remote frames as the following: **0x0:** Indicates that the message frame is a Standard Data CAN Frame. **0x1:** Indicates that the message frame is a Standard Remote CAN Frame. There are no CAN FD remote frames.
[19]	`IDE`	R	Identifier Extension. This bit differentiates between frames using the Standard Identifier and those using the Extended Identifier. Valid for both CAN and CAN FD Standard and Extended Frames. **Values:** **0x0:** Indicates the use of a Standard Message Identifier. **0x1:** Indicates the use of an Extended Message Identifier.
[18:1]	`ID[17:0]`	R	Extended Message ID. This field indicates the Extended Identifier. Valid only for Extended CAN and CAN FD frames.
[0]	`RTR/RRS`	R	Remote Transmission Request. This bit differentiates between CAN extended data frames and CAN extended remote frames. **Values:** **0x0:** Indicates the message frame is a CAN Data Frame. For Extended CAN FD frame this bit is transmitted at `RRS` position of the frame and is always equal to 0. **0x1:** Indicates the message frame is a CAN Remote Frame. There are no CAN FD remote frames.

13.2.2.2.3.2. TXE FIFO TBi Data Length Code Register (TXE_FIFO_TBi_DLC)

The TXE_FIFO_TBi_DLC (for i = 0; i <=31) register is offset as 0x104 + (i*0x48)

The following table shows the register bit assignments.

Table 13-47 Fields for register: TXE_FIFO_TBi_DLC

Bits	Name	R/W	Description
[31:28]	DLC	R	Data Length Code. This is the data length code of the control field of the CAN and CAN FD frame.
[27]	FDF	R	Extended Data Length/FD Frame Format. This bit distinguishes between CAN format and CAN FD format frames. **Values:** **0x0:** CAN format frame **0x1:** CAN FD format frame
[26]	BRS	R	Bit Rate Switch. The BRS bit decides whether the bit rate is switched inside a CAN FD format frame or not (provided the BRSD bit is not set in MSR register). **Values:** **0x0:** Bit rate is not switched inside a CAN FD frame. **0x1:** Bit rate is switched from the standard bit rate of the Arbitration phase to the preconfigured alternate bit rate of the Data phase inside a CAN FD frame. BRS does not exist in CAN format frames and is always equal to 0
[25:24]	ET	R	Event Type. **Values:** **0x0:** Reserved **0x1:** Transmitted in spite of cancellation request or DAR mode transmissions **0x2:** Reserved **0x3:** Transmitted
[23:16]	MM	R	Message Marker. Written by software during Tx Buffer configuration. Copied into Tx Event FIFO element for identification of Tx message status.
[15:0]	Timestamp	R	Timestamp captured after SOF bit.

13.2.2.2.4. CAN FD Rx Message Space

13.2.2.2.4.1. RBi Message ID Register (RBiID)

The RBiID (for i = 0; i <=63) register is offset as 0x2100 + (i*0x48) .

The following table shows the register bit assignments.

Table 13-48 Fields for register: `RBiID`

Bits	Name	R/W	Description
[31:21]	`ID[28:18]`	R	Standard Message ID. The Identifier portion for a Standard Frame is 11 bits. These bits indicate the Standard Frame ID. This field is valid for both CAN and CAN FD Standard and Extended Frames.
[20]	`SRR/RTR/ RRS`	R	Substitute Remote Transmission Request. For Extended CAN frames and Extended CAN FD frame this bit was received at `SRR` position of the respective frame. For Standard CAN FD frame this bit was received at `RRS` position of the frame. This bit differentiates between received standard CAN data frames and standard CAN remote frames as following: **0x0:** Indicates that the message frame is a Standard Data CAN Frame. **0x1:** Indicates that the message frame is a Standard Remote CAN Frame.
[19]	`IDE`	R	Identifier Extension. This bit differentiates between frames using the Standard Identifier and those using the Extended Identifier. Valid for both CAN and CAN FD Standard and Extended Frames. **Values:** **0x0:** Indicates the use of a Standard Message Identifier. **0x1:** Indicates the use of an Extended Message Identifier.
[18:1]	`ID[17:0]`	R	Extended Message ID. This field indicates the Extended Identifier. Valid only for Extended CAN and CAN FD Frames. For Standard CAN and CAN FD Frames, this field is reserved and has no meaning.
[0]	`RTR/RRS`	R	Remote Transmission Request. This bit differentiates between CAN extended data frames and CAN extended remote frames. **Values:** **0x0:** Indicates the message frame is a CAN Data Frame. For Extended CAN FD frame this bit is transmitted at `RRS` position of the frame and is always equal to 0. **0x1:** Indicates the message frame is a CAN Remote Frame. For Standard CAN and CAN FD frames, this field is reserved and has no meaning.

13.2.2.2.4.2. RBi Data Length Code Register (RBiDLC)

The RBiDLC (for i = 0; i <=63) register is offset as 0x2104 + (i*0x48).

The following table shows the register bit assignments.

Table 13-49 Fields for register: RBiDLC

Bits	Name	R/W	Description
[31:28]	DLC[3:0]	R	Data Length Code. Received data length code of the control field of the CAN and CAN FD frame.
[27]	EDL/FDF	R	Extended Data Length/FD Frame Format. This bit distinguishes between CAN format and CAN FD format frames. **Values:** **0x0:** CAN format frame **0x1:** CAN FD format frame
[26]	BRS	R	Bit Rate Switch. The BRS bit provides R whether the bit rate was switched inside the received CAN FD format frame or not. **Values:** **0x0:** Bit rate is not switched inside a CAN FD frame. **0x1:** Bit rate is switched from the standard bit rate of the Arbitration phase to the preconfigured alternate bit rate of the Data phase inside a CAN FD frame. This bit has no meaning if received frame is CAN frame.
[25]	ESI	R	Error State Indicator. The ESI bit provides the error R of the sender/ transmitter of the message. **Values:** **0x0:** Received frame was sent by error active transmitter. This bit has no meaning if the received frame is a CAN frame. **0x1:** Received frame was sent by error passive transmitter
[24:21]			Reserved
[20:16]	MFI	R	This R field is written by the core in RX FIFO mode to provide the matched filter index for received message.
[15:0]	Timestamp	R	Timestamp captured after SOF bit.

13.2.2.2.4.3. RBi Data Byte n Register (RBiDWn)

The RBiDWn (for i = 0; i <=63 and for n = 0; n <=15) register is offset as 0x2108 + (i*0x48) + (n*0x4).

The following table shows the register bit assignments.

Table 13-50 Fields for register: `RBiDWn`

Bits	Name	R/W	Description
[31:24]	`Data bytes0 [7:0]`	R	Data Byte 0. Data byte that was received with CAN or CAN FD frame based on `RBiDLC.DLC` control field.
[23:16]	`Data bytes1 [7:0]`	R	Data Byte 1. Data byte that was received with CAN or CAN FD frame based on `RBiDLC.DLC` control field.
[15:8]	`Data bytes2 [7:0]`	R	Data Byte 2. Data byte that was received with CAN or CAN FD frame based on `RBiDLC.DLC` control field.
[7:0]	`Data bytes3 [7:0]`	R	Data Byte 3. Data byte that was received with CAN or CAN FD frame based on `RBiDLC.DLC` control field.

13.3. UART

The *Universal Asynchronous Receiver/Transmitter* (**UART**) is a programmable component used for serial communication with peripherals and data sets.

The BE-U1000 microcontroller integrates eight UARTs.

A simplified block diagram of the UART is shown in the following figure.

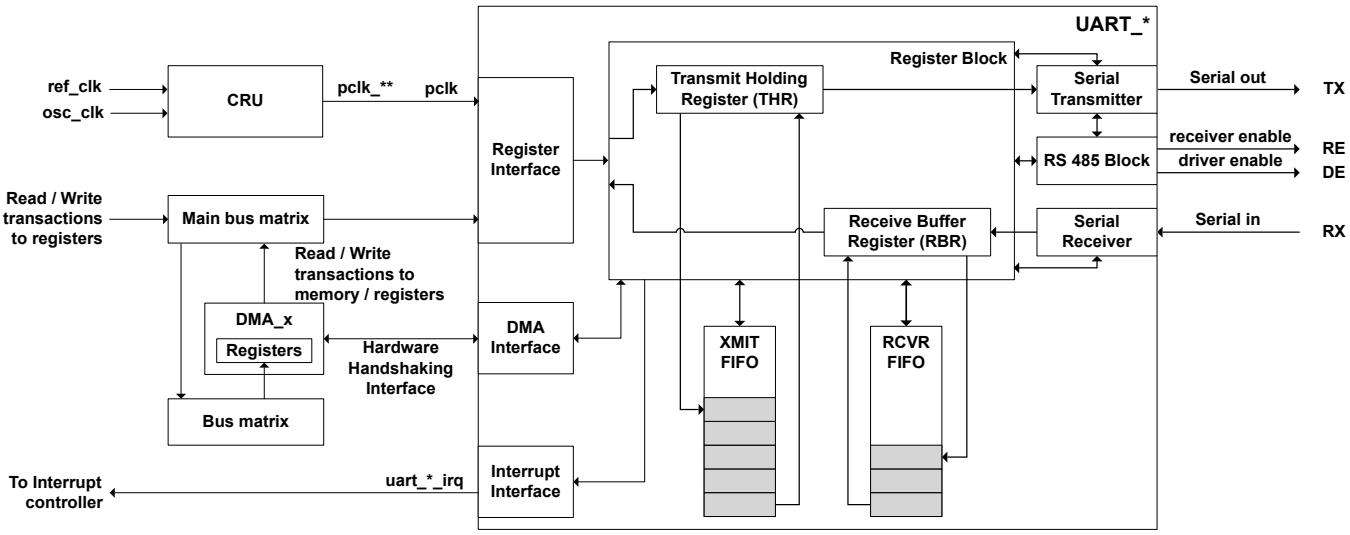

Figure 13-3 UART block diagram

In the figure above:

- `x` indicates 0 for DMA_0 and 1 for DMA_1. For more information, refer to the DMA Controller.
- An asterisk symbol (*) indicates 0 for UART_0, 1 for UART_1, 2 for UART_2, 3 for UART_3, 4 for UART_4, 5 for UART_5, 6 for UART_6 and 7 for UART_7.
- A double asterisk (**) indicates 0 for UART_0, UART_1, UART_2, 1 for UART_3, UART_4, UART_5 and 2 for UART_6, UART_7.

RS-485 Block is supported only by UART_6 and UART_7.

UART features:

- Transmit FIFO and receive FIFO depths of 16
- IrDA 1.0 SIR mode support with up to 115.2 Kbaud data rate
- Programmable *Transmit Holding Register Empty* (**THRE**) Interrupt mode
- Programmable serial data baud rate (up to 6.25 Mb/s)
- Programmable fractional baud rate

13.3.1. UART Functional Description

This section covers the following topics:

- Programmable THRE Interrupt
- UART Serial Protocol
- 9-bit Data Transfer
- RS485 Support (for UART_6 and UART_7 Only)
- Fractional Baud Rate Support
- IrDA 1.0 SIR Support

13.3.1.1. Programmable THRE Interrupt

The UART supports Programmable *Transmit Holding Register Empty* (**THRE**) Interrupt mode in order to increase system performance.

Programmable THRE Interrupt mode can be enabled using the Interrupt Enable Register (`IER[7]`).

When FIFOs and THRE mode are enabled, the THRE Interrupts are active at, and below, a programmed transmitter FIFO empty threshold level, as opposed to empty.The threshold level is programmed into `FCR[5:4]`.

In addition to the interrupt change, the Line Status Register (`LSR[5]`) also switches from indicating that the transmitter FIFO is empty to the FIFO being full. This allows software to fill the FIFO for each transmit sequence by polling `LSR[5]` before writing another character. The flow then allows the transmitter FIFO to be filled whenever an interrupt occurs and there is data to transmit, rather than waiting until the FIFO is completely empty. Waiting until the FIFO is empty causes a reduction in performance whenever the system is too busy to respond immediately.

Even if everything else is enabled, if the FIFOs are disabled using the `FCR[0]` bit, the Programmable THRE Interrupt mode is also disabled. When not selected or disabled, THRE interrupts and the `LSR[5]` bit function normally, signifying an empty `THR` or FIFO.

13.3.1.2. UART Serial Protocol

Because the serial communication between the UART and a selected device is asynchronous, additional bits (start and stop) are added to the serial data to indicate the beginning and end. Utilizing these bits allows two devices to be synchronized. This structure of serial data — accompanied by start and stop bits — is referred to as a character, as shown in the following figure.

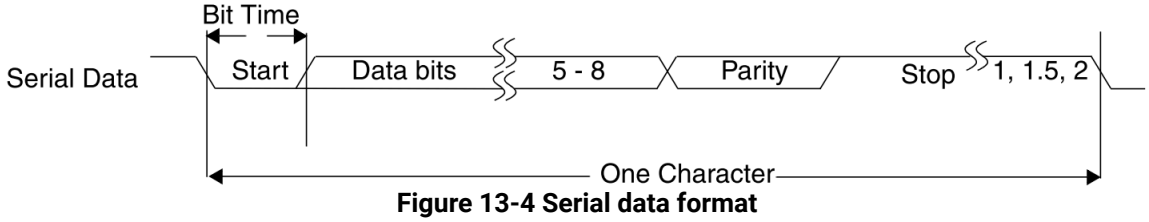

Figure 13-4 Serial data format

An additional parity bit can be added to the serial character. This bit appears after the last data bit and before the stop bit(s) in the character structure in order to provide the UART with the ability to perform simple error checking on the received data.

The UART Line Control Register (LCR) is used to control the serial character characteristics. The individual bits of the data word are sent after the start bit, starting with the *Least Significant Bit* (**LSB**). These are followed by the optional parity bit, followed by the stop bit(s), which can be 1, 1.5, or 2.

13.3.1.3. 9-bit Data Transfer

The UART can be configured to have 9-bit data transfer in both transmit and receive mode.

By enabling 9-bit data transfer mode, the UART can be used in multi-drop systems where one master is connected to multiple slaves in a system. The master communicates with one of the slaves. When the master wants to transfer a block of data to a slave, it first sends an address byte to identify the target slave.

The differentiation between the address/data byte is done based on the 9th bit in the incoming character. If the 9th bit is set to 0, then the character represents a data byte. If the 9th bit is set to 1, then the character represents address byte. All the slave systems compare the address byte with their own address and only the target slave (in which the address has matched) is enabled to receive data from the master. The master then starts transmitting data bytes to the target slave. The non-addressed slave systems ignore the incoming data until a new address byte is received.

13.3.1.3.1. Transmit Mode

The UART supports two types of transmit modes:

- Transmit Mode 0 (when LCR_EXT[3] is set to 0)
- Transmit Mode 1 (when LCR_EXT[3] is set to 1)

13.3.1.3.1.1. Transmit Mode 0

In Transmit Mode 0 the address is programmed in the Transmit Address Register (TAR) and data is written into the Transmit Holding Register (THR) or the Shadow Transmit Holding Register (STHR). The 9th bit of the THR and STHR register is not applicable in this mode.

The following figure illustrates the transmission of address and data based on SEND_ADDR bit of LCR_EXT register, HTx, and Tx FIFO/THR empty conditions.

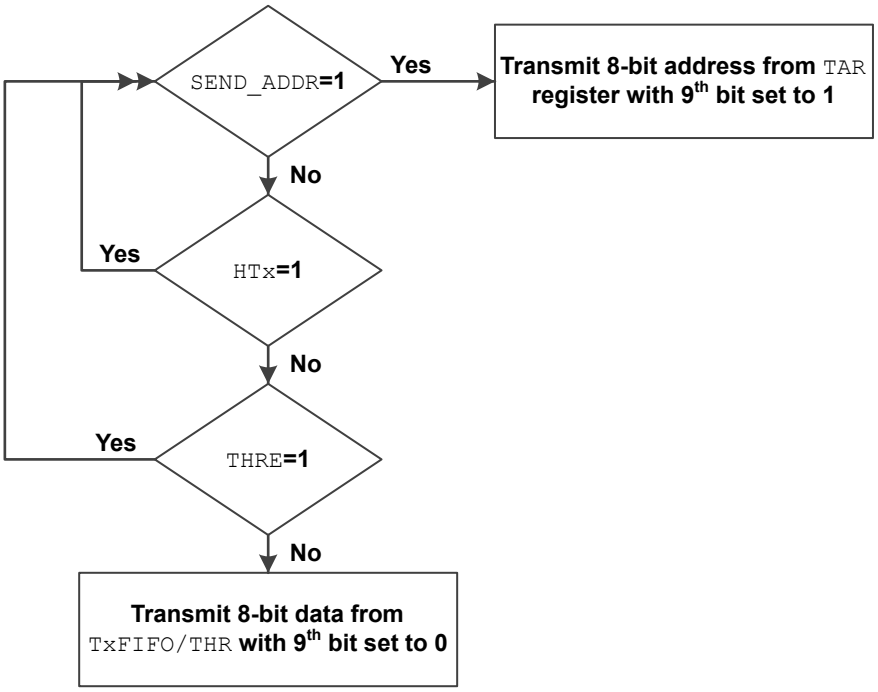

Figure 13-5 Auto address transmit flow chart

The address of the target slave to which the data is to be transmitted is programmed in the TAR register. You must enable the SEND_ADDR bit of LCR_EXT register to transmit the target slave address present in the TAR register on the serial UART line with the 9th data bit set to 1 to indicate that the address is being sent to the slave. The UART clears the SEND_ADDR bit after the address character starts transmitting on the UART line. The data required to transmit to the target slave is programmed through Transmit Holding Register (THR). The data is transmitted on the UART line with 9th data bit set to 0 to indicate data is being sent to the slave.

If the application is required to fill the data bytes in the Tx FIFO before sending the address on the UART line (before setting LCR_EXT[2]=1), then it is recommended to set the HTx to 1 such that the UART does not start sending out the data in the Tx FIFO as data byte. Once the Tx FIFO is filled, then program SEND_ADDR bit of LCR_EXT register to 1 and then set HTx to 0.

13.3.1.3.1.2. Transmit Mode 1

In Transmit Mode 1 THR and STHR registers are of 9-bit wide and both address and data are programmed through the THR and STHR registers. The UART does not differentiate between address and data, and both are taken from the Tx FIFO. The SEND_ADDR bit of LCR_EXT register and Transmit Address Register (TAR) are not applicable in this mode. The software must pack the 9th bit with 1/0 depending on whether address/data has to be sent.

13.3.1.3.2. Receive Mode

The UART supports two receive modes:

- Hardware Address Match Receive Mode (when ADDR_MATCH bit of LCR_EXT register is set to 1)
- Software Address Match Receive Mode (when ADDR_MATCH bit of LCR_EXT register is set to 0)

13.3.1.3.2.1. Hardware Address Match Receive Mode

In the Hardware Address Match Receive Mode the UART matches the received character with the address programmed in the Receive Address register (RAR), if the 9th bit of the received character is set

to 1. If the received address is matched with the programmed address in RAR register, then subsequent data bytes (with 9th bit set to 0) are pushed into the Rx FIFO. If the address matching fails, then the UART Controller discards further data characters until a matching address is received.

The following figure shows the flow chart for the reception of data bytes based on the address matching feature.

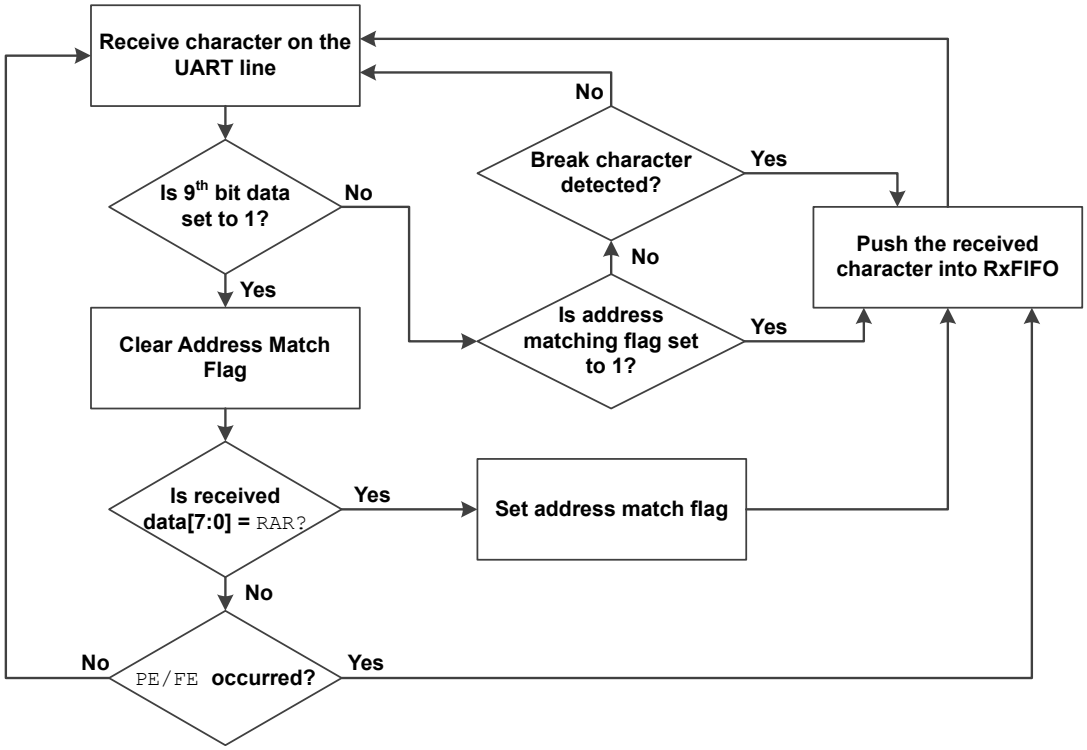

Figure 13-6 Hardware address match receive mode

The UART receives the character irrespective of whether the 9th bit data is set to 1. If 9th bit of the received character is set to 1, then it clears internal address match flag and then compares the received 8-bit character information with the address programmed in the RAR register. If the received address character matches with the address programmed in the RAR register, then the address match flag is set to 1 and the received character is pushed to the Rx FIFO in FIFO-mode or to RBR register in non-FIFO mode and the ADDR_RCVD bit in LSR register is set to indicate that the address has been received. In case of parity or if a framing error is found in the received address character and if the address is not matched with the RAR register, then the received address character is still pushed to Rx FIFO or RBR register with ADDR_RCVD and PE/FE error bit in LSR set to 1. The subsequent data bytes (9th bit of received character is set to 0) are pushed to the Rx FIFO in FIFO mode or to the RBR register in non-FIFO mode until the new address character is received. If any break character is received, the UART treats it as a special character and pushes to the Rx FIFO or RBR register based on the FIFO mode irrespective of address match flag.

The break character can be used to alert the complete system in case all slaves are in sleep mode (entered in to the low power mode). Therefore, the break character is treated as special character.

13.3.1.3.2.2. Software Address Match Receive Mode

In this mode of operation the UART does not perform the address matching for the received address character (9th bit data set to 1) with the RAR register. The UART always receives the 9-bit data and pushes in to Rx FIFO in FIFO mode or to the RBR register in non-FIFO mode. The user must compare

the address whenever address byte is received and indicated through ADDR_RCVD bit in the LSR. The user can flush/reset the Rx FIFO in case of address not matched through RFIFOR bit in FIFO Control Register (FCR).

13.3.1.4. RS485 Support (for UART_6 and UART_7 Only)

The UART supports the RS485 serial protocol that enables transfer of serial data using the RS485 interface. The Driver Enable (de) and Receiver Enable (re) signals are generated for enabling the RS485 interface support. The de and re signals are hardware generated and the assertion/de-assertion times for these signals are programmable. The active level of these signals is configurable.

Configuration of the UART for the RS485 interface does the following:

- Bit [0] of the Transceiver Control Register (TCR) enables or disables the RS485 mode.
- Bit [1] and bit [2] of TCR are used to select the polarity of the re and de signals.
- Bit [4:3] of the TCR selects the type of transfer in RS485 mode.
- Driver output enable (DE_EN) and Receiver output enable (RE_EN) registers are used for software control of the de and re signals.
- Driver output Enable Timing (DET) register is used to program the assertion and de-assertion timings of the de signal.
- Turn Around Timing (TAT) register is used to program the turnaround time from de to re and re to de.

13.3.1.4.1. de Assertion and De-assertion Timing

The assertion and de-assertion timings of the de signal are controlled through the DET register:

- de assertion time (DET[7:0]): The assertion time is the time between the activation of the de signal and the beginning of the START bit. The value represented is in terms of serial clock cycles.
- de de-assertion time (DET[23:16]): The de-assertion time is the time between the end of the last stop bit, in a transmitted character, and the de-activation of the de signal. The value represented is in terms of serial clock cycles.

Hardware ensures that these values are met for de assertion and de de-assertion before/after active data transmission.

13.3.1.4.2. RS485 Modes

The UART consists of the following RS485 modes based on the XFER_MODE field in the Transceiver Control Register (TCR):

Full Duplex Mode

In this mode, XFER_MODE of TCR is set to 0.

The Full Duplex Mode supports both transmit and receive transfers simultaneously.

In Full Duplex Mode, the de signal:

- Goes active if both these conditions are satisfied:
  - When the DE_Enable field of DE_EN is set to 1.
  - THR is not empty in non-FIFO mode or transmitter FIFO is not empty in FIFO mode.

- Goes inactive if both these conditions are satisfied:
  - When the current ongoing transmitting serial transfer is completed.
  - Either DE_Enable field of the DE_EN is set to 0, transmitter FIFO is empty in FIFO mode or THR is empty in non-FIFO mode.

In Full Duplex Mode, the re signal:

- Goes active when RE_Enable field of RE_EN is set to 1.
- Goes inactive when RE_Enable field of RE_EN is set to 0.

The user can choose when to transmit or when to receive. Both re and de can be simultaneously asserted or de-asserted at any time. The UART does not impose any turnaround time between transmit and receive (TAT.DE_to_Re) or receive to transmit (TAT.Re_to_De) in this mode. This mode can directly be used in full duplex operation where separate differential pair of wires is present for transmit and receive.

Software-Controlled Half Duplex Mode

In this mode, XFER_MODE of TCR is set to 1

The Software-Controlled Half Duplex mode supports either transmit or receive transfers at a time but not both simultaneously. The switching between transmit to receive or receive to transmit is through programming the Driver output enable (DE_EN) and Receiver output enable (RE_EN) registers.

In Software-Controlled Half Duplex mode, the de signal:

- Goes active if the following conditions are satisfied:
  - The DE_Enable field of the DE_EN is set to 1.
  - THR is not empty in non-FIFO mode or transmitter FIFO is not empty in FIFO mode.
  - If any receive transfer is ongoing, then the signal waits until receive has finished, and after the turnaround time counter (TAT.Re_to_De) has elapsed.
- Goes inactive if the following conditions are satisfied:
  - The current ongoing transmitting serial transfer is completed.
  - The DE_Enable field of the DE_EN is set to 0.
  - Either transmitter FIFO is empty in FIFO mode or THR is empty in non-FIFO mode.

In Software-Controlled Half Duplex mode, the re signal:

- Goes active if the following conditions are satisfied:
  - When RE_Enable field of RE_EN is set to 1.
  - If any transmit transfer is ongoing, then the signal waits until transmit has finished and after the turnaround time counter (TAT.DE_to_Re) has elapsed.
- Goes in-active under the following conditions:
  - The current ongoing receive serial transfer is completed.
  - When DE_Enable field of the DE_EN is set to 0.

The user must enable either de or re but not both at any point of time. As re and de signals are mutually exclusive, the user must ensure that both of them are not programmed to be active at any point of time. In this mode, the hardware ensures that a proper turnaround time is maintained while

switching from (TAT.Re_to_De) or from (TAT.DE_to_Re) (value of turnaround is obtained from the TAT register, in terms of serial clock cycles).

Hardware-Controlled Half Duplex Mode

In this mode, XFER_MODE of TCR is set to 2

The Hardware-Controlled Half Duplex mode supports either transmit or receive transfers at a time but not both simultaneously. If both DE_Enable and RE_Enable bits of Driver output enable (DE_EN) and Receiver output enable (RE_EN) registers are enabled, the switching between transmit to receive or receive to transmit is automatically done by the hardware based on the empty condition of Tx FIFO.

In Hardware-Controlled Half Duplex mode, the de signal:

- Goes active if the following conditions are satisfied:
  - The DE_Enable field of the DE_EN is set to 1.
  - Transmit Holding Register is not empty in non-FIFO mode or transmitter FIFO is not empty in FIFO mode.
  - If any receive transfer is ongoing, then the signal waits until receive has finished, and after the turnaround time counter (TAT.Re_to_De) has elapsed.
- Goes inactive if the following conditions are satisfied:
  - The current ongoing transmitting serial transfer is completed.
  - The DE_Enable field of the DE_EN is set to 0.
  - Either transmitter FIFO is empty in FIFO mode or THR is empty in non-FIFO mode or the DE_Enable field of the DE_EN is set to 0.

In Hardware-Controlled Half Duplex mode, the re signal:

- Goes active if the following conditions are satisfied:
  - When RE_Enable field of RE_EN is set to 1.
  - Either transmitter FIFO is empty in FIFO mode or THR is empty in non-FIFO mode.
  - If any transmit transfer is ongoing, then the signal waits until transmit has finished and after the turnaround time counter (TAT.DE_to_Re) has elapsed.
- Goes in-active under the following conditions:
  - The current ongoing receive serial transfer is completed.
  - Either transmitter FIFO is non-empty in FIFO mode or THR is non empty in non-FIFO mode or the RE_Enable field of RE_EN is set to 0.

In this mode, the hardware ensures that a proper turnaround time is maintained while switching from (TAT.Re_to_De) or from (TAT.DE_to_Re) (value of turnaround is obtained from the TAT register, in terms of serial clock cycles).

13.3.1.5. Fractional Baud Rate Support

The programmable fractional baud rate divisor enables a finer resolution of the baud clock than the conventional integer divider. The programmable fractional baud clock divider allows for the programmability of both an integer divisor as well as fractional component. The equation to calculate the baud rate divisor is as follows:

```
Baud Rate Divisor = Serial Clock Frequency / (16 x Required Baud Rate) = BRDI + BRDF,
```

where:

- `BRDI` — Integer part of the divisor composed of `DLH` and `DLL` registers
- `BRDF` — Fractional part of the divisor that is programmed through the Divisor Latch Fraction Register (`DLF`). The fractional value is computed by using the (Divisor Latch Fractional Value)/(2^4) formula and shows in the following table.

Table 13-51 Divisor latch fractional values

`DLF` Value	Fraction	Fractional Value
0000	0/16	0.0000
0001	1/16	0.0625
0010	2/16	0.125
0011	3/16	0.1875
0100	4/16	0.25
0101	5/16	0.3125
0110	6/16	0.375
0111	7/16	0.4375
1000	8/16	0.5
1001	9/16	0.5625
1010	10/16	0.625
1011	11/16	0.6875
1100	12/16	0.75
1101	13/16	0.8125
1110	14/16	0.875
1111	15/16	0.9375

13.3.1.6. IrDA 1.0 SIR Support

To enable SIR mode, write 0x1 to the `MCR`.`SIRE` bit before writing to the `LCR` register.

The *Infrared Data Association* (**IrDA**) 1.0 *Serial Infrared* (**SIR**) mode supports bi-directional data communications with remote devices using infrared radiation as the transmission medium. IrDA 1.0 SIR mode specifies a maximum baud rate of 115.2 Kbaud.

In this mode, each data character is sent serially in this order:

- Begins with a start bit
- Followed by 8 data bits
- Ends with at least one stop bit

Thus, the number of data bits that can be sent is fixed. No parity information can be supplied, and only one stop bit is used in this mode. Trying to adjust the number of data bits sent or enable parity with the Line Control Register (`LCR`) has no effect.

13.3.2. UART Registers

The register regions of the UART Controllers are mapped in the BE-U1000 microcontroller memory map as the following table shows.

Table 13-52 Memory address regions for UART Controller registers

Region	Block Name	Size	Start Address	End Address
Periphery 0	UART_0	4 KB	0x1100_1000	0x1000_1FFF
	UART_1	4 KB	0x1100_2000	0x1100_2FFF
	UART_2	4 KB	0x1100_3000	0x1100_3FFF
Periphery 1	UART_3	4 KB	0x1300_1000	0x1300_1FFF
	UART_4	4 KB	0x1300_2000	0x1300_2FFF
	UART_5	4 KB	0x1300_3000	0x1300_3FFF
Periphery 2	UART_6	4 KB	0x1400_9000	0x1400_9FFF
	UART_7	4 KB	0x1400_A000	0x1400_AFFF

For details, refer to Memory Map chapter.

This section covers the following topics:

- Register Map
- Register Descriptions

13.3.2.1. Register Map

This section contains information about the register memory map.

The following table provides a high-level summary of each register. To view the detailed description of a register, click the register name in the **Description** column.

Table 13-53 UART register map

Name	Offset	Description	Reset Value
`RBR`	0x0	Receive Buffer Register **Dependencies:** `LCR[7]` bit = 0	0x0
`THR`	0x0	Transmit Holding Register **Dependencies:** `LCR[7]` bit = 0	0x0
`DLL`	0x0	Divisor Latch Low **Dependencies:** `LCR[7]` bit = 1	0x0
`DLH`	0x4	Divisor Latch High **Dependencies:** `LCR[7]` bit = 1	0x0
`IER`	0x4	Interrupt Enable Register **Dependencies:** `LCR[7]` bit = 0	0x0
`IIR`	0x8	Interrupt Identity Register	0x1
`FCR`	0x8	FIFO Control Register	0x0

Table 13-53 UART register map (continued)

Name	Offset	Description	Reset Value
LCR	0x0C	Line Control Register	0x0
MCR	0x10	Mode Control Register	0x0
LSR	0x14	Line Status Register	0x60
SCR	0x1C	Scratchpad Register	0x0
SRBRi (for i=0; i<=15)	0x30 +(i *0x4)	Shadow Receive Buffer Register **Dependencies:** LCR[7] bit = 0	0x0
STHRi (for i=0; i<=15)	0x30 +(i *0x4)	Shadow Transmit Holding Register **Dependencies:** LCR[7] bit = 0	0x0
FAR	0x70	FIFO Access Register	0x0
TFR	0x74	Transmit FIFO Read	0x0
RFW	0x78	Receive FIFO Write	0x0
USR	0x7C	UART Status Register	0x6
TFL	0x80	Transmit FIFO Level	0x0
RFL	0x84	Receive FIFO Level	0x0
SRR	0x88	Software Reset Register	0x0
SBCR	0x90	Shadow Break Control Register	0x0
SFE	0x98	Shadow FIFO Enable	0x0
SRT	0x9C	Shadow RCVR Trigger	0x0
STET	0xA0	Shadow Tx Empty Trigger	0x0
HTX	0xA4	Halt Tx	0x0
DMASA	0xA8	DMA Software Acknowledge Register	0x0
TCR	0xAC	Transceiver Control Register **Exists:** UART6 and UART7	0x6
DE_EN	0xB0	Driver Output Enable Register **Exists:** UART6 and UART7	0x0
RE_EN	0xB4	Receiver Output Enable Register **Exists:** UART6 and UART7	0x0
DET	0xB8	Driver Output Enable Timing Register **Exists:** UART6 and UART7	0x0
TAT	0xBC	TurnAround Timing Register **Exists:** UART6 and UART7	0x0

Table 13-53 UART register map (continued)

Name	Offset	Description	Reset Value
DLF	0xC0	Divisor Latch Fraction Register	0x0
RAR	0xC4	Receive Address Register	0x0
TAR	0xC8	Transmit Address Register	0x0
LCR_EXT	0xCC	Line Extended Control Register	0x0
REG_TIMEOUT_RST	0xD4	Register Timeout Counter Reset Value	0x80

13.3.2.2. Register Descriptions

In the tables below, the **R/W** column specifies how the application and the core can access the register fields.

13.3.2.2.1. Receive Buffer Register (RBR)

The RBR register is at offset 0x0.

This register can be accessed only when the DLAB bit (LCR[7]) is cleared.

The following table shows the register bit assignments.

Table 13-54 Fields for register: RBR

Bits	Name	R/W	Description
[31:9]			Reserved and read as 0
[8:0]	RBR	R	Data byte received on the serial in port Rx. The data in this register is valid only if the Data Ready (DR) bit in the Line Status Register (LSR) is set. If FIFOs are disabled (FCR[0] set to 0), the data in the RBR must be read before the next data arrives, otherwise it is overwritten, resulting in an over-run error. If FIFOs are enabled (FCR[0] set to 1), this register accesses the head of the receive FIFO. If the receive FIFO is full and this register is not read before the next data character arrives, then the data already in the FIFO is preserved, but any incoming data are lost and an over-run error occurs. **Value After Reset:** 0x0

13.3.2.2.2. Transmit Holding Register (THR)

The THR register is at offset 0x0.

This register can be accessed only when the DLAB bit (LCR[7]) is cleared.

The following table shows the register bit assignments.

Table 13-55 Fields for register: THR

Bits	Name	R/W	Description
[31:9]			Reserved and read as 0
[8:0]	THR	W	Data to be transmitted on the serial out port Tx.

Table 13-55 Fields for register: THR (continued)

Bits	Name	R/W	Description
			Data should only be written to the THR when the THR Empty (THRE) bit (LSR[5]) is set. If FIFOs are disabled (FCR[0]=0) and THRE is set, writing a single character to the THR clears the THRE. Any additional writes to the THR before the THRE is set again causes the THR data to be overwritten. If FIFOs are enabled (FCR[0]=1) and THRE is set, 16 number of characters of data may be written to the THR before the FIFO is full. Any attempt to write data when the FIFO is full results in the write data being lost. **Value After Reset:** 0x0

13.3.2.2.3. Divisor Latch Low Register (DLL)

The DLL register is at offset 0x0.

This register can be accessed only when the DLAB bit (LCR[7]) is set and the UART is not busy — that is, USR[0] is 0.

The following table shows the register bit assignments.

Table 13-56 Fields for register: DLL

Bits	Name	R/W	Description
[31:8]			Reserved and read as 0
[7:0]	DLL	R/W	Lower 8 bits of a 16-bit, read/write, Divisor Latch register that contains the baud rate divisor for the UART. The output baud rate is equal to the serial clock (pclk) frequency divided by sixteen times the value of the baud rate divisor, as follows: `baud rate = (serial clock freq) / (16 * divisor)` With the Divisor Latch Registers (DLL and DLH) set to 0, the baud clock is disabled and no serial communications occur. Also, once the DLL is set, at least 8 clock cycles of the slowest UART clock should be allowed to pass before transmitting or receiving data. **Value After Reset:** 0x0

13.3.2.2.4. Divisor Latch High Register (DLH)

The DLH register is at offset 0x4.

This register can be accessed only when the DLAB bit (LCR[7]) is set and the UART is not busy — that is, USR[0] is 0.

The following table shows the register bit assignments.

Table 13-57 Fields for register: DLH

Bits	Name	R/W	Description
[31:8]			Reserved and read as 0
[7:0]	DLH	R/W	Upper 8-bits of a 16-bit, read/write, Divisor Latch register that contains the baud rate divisor for the UART.

Table 13-57 Fields for register: DLH (continued)

Bits	Name	R/W	Description
			The output baud rate is equal to the serial clock (pclk) frequency divided by sixteen times the value of the baud rate divisor, as follows: `baud rate = (serial clock freq) / (16 * divisor)` With the Divisor Latch Registers (DLL and DLH) set to 0, the baud clock is disabled and no serial communications occur. Also, once the DLH is set, at least 8 clock cycles of the slowest UART clock should be allowed to pass before transmitting or receiving data. **Value After Reset:** 0x0

13.3.2.2.5. Interrupt Enable Register (IER)

The IER register is at offset 0x4.

This register can be accessed only when the DLAB bit (LCR[7]) is cleared.

The following table shows the register bit assignments.

Table 13-58 Fields for register: IER

Bits	Name	R/W	Description
[31:8]			Reserved and read as 0
[7]	PTIME	R/W	Programmable THRE Interrupt Mode Enable This bit is used to enable/disable the generation of THRE Interrupt. **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x0
[6:5]			Reserved and read as 0
[4]	ELCOLR	R/W	Method for clearing the status in the LSR register This is applicable only for Overrun Error, Parity Error, Framing Error, and Break Interrupt status bits. **0x0:** Disabled LSR status bits are cleared either on reading Rx FIFO (RBR Read) or On reading LSR register. **0x1:** Enabled LSR status bits are cleared only on reading LSR register. **Value After Reset:** 0x0
[3]			Reserved
[2]	ELSI	R/W	Enable Receiver Line Status Interrupt This bit used to enable/disable the generation of Receiver Line Status Interrupt. This is the highest priority interrupt. **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x0
[1]	ETBEI	R/W	Enable Transmit Holding Register Empty Interrupt

Table 13-58 Fields for register: `IER` (continued)

Bits	Name	R/W	Description
			This bit used to enable/disable the generation of Transmitter Holding Register Empty Interrupt. This is the third highest priority interrupt. **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x0
[0]	`ERBFI`	R/W	Enable Received Data Available Interrupt This bit used to enable/disable the generation of Received Data Available Interrupt and the Character Timeout Interrupt. These are the second highest priority interrupts. **0x0:** Disabled **0x1:** Enabled **Value After Reset:** 0x0

13.3.2.2.6. Interrupt Identity Register (`IIR`)

The `IIR` register is at offset 0x8.

The following table shows the register bit assignments.

Table 13-59 Fields for register: `IIR`

Bits	Name	R/W	Description
[31:8]			Reserved and read as 0. **Value After Reset:** 0x0
[7:6]	`FIFOSE`	R	FIFOs Enabled This bit used to indicate whether the FIFOs are enabled or disabled. **0x0:** Disabled **0x3:** Enabled **Value After Reset:** 0x0
[5:4]			Reserved and read as 0 **Value After Reset:** 0x0
[3:0]	`IID`	R	Interrupt ID This bit indicates the highest priority pending interrupt, which can be one of the following types: **0x1:** No interrupt pending **0x2:** `THR` empty **0x4:** Received data available **0x6:** Receiver line status **0x7:** Busy detect **0xC:** Character timeout Bit 3 indicates an interrupt that can occur only when the FIFOs are enabled and used to distinguish a Character Timeout condition interrupt. **Value After Reset:** 0x1

The following table shows the priority level for interrupt types used in the UART.

Table 13-60 Interrupt control functions

Interrupt ID				Interrupt Set and Reset Functions			
Bit 3	Bit 2	Bit 1	Bit 0	Priority Level	Interrupt Type	Interrupt Source	Interrupt Reset Control
0	0	0	1	–	None	None	–
0	1	1	0	Highest	Receiver line status	Overrun/parity/framing errors or break interrupt.	Reading the LSR
0	1	0	0	Second	Received data available	Receiver data available (FIFOs disabled, FCR[0]=0) or RCVR FIFO trigger level reached (FIFOs enabled, FCR[0]=1).	Reading the RBR (FIFOs disabled, FCR[0]=0) or the FIFO drops below the trigger level (FIFOs enabled, FCR[0]=1).
1	1	0	0	Second	Character timeout indication	No characters in or out of the RCVR FIFO during the last 4 character times and there is at least 1 character in it during this time.	Reading the RBR
0	0	1	0	Third	Transmit holding register empty	THR empty (Programmable THRE Interrupt Mode disabled, IER[7]=0) or xMIT FIFO at or below threshold (Programmable THRE Interrupt Mode enabled, IER[7]=1).	Reading the IIR register or writing into THR (FIFOs or Programmable THRE Interrupt Mode disabled) or xMIT FIFO above threshold (FIFOs and THRE Mode enabled).
0	1	1	1	Fourth	Busy detect indication	Master has tried to write to the LCR while the UART is busy (USR[0]is set to 1).	Reading the USR

13.3.2.2.7. FIFO Control Register (FCR)

The FCR register is at offset 0x8.

The following table shows the register bit assignments.

Table 13-61 Fields for register: FCR

Bits	Name	R/W	Description
[31:8]			Reserved and read as 0
[7:6]	RT	W	RCVR Trigger This bit used to select the trigger level in the receiver FIFO at which the Received Data Available Interrupt is generated. The following trigger levels are supported: **0x0:** 1 character in the FIFO **0x1:** FIFO ¼ full **0x2:** FIFO ½ full **0x3:** FIFO 2 less than full **Value After Reset:** 0x0
[5:4]	TET	W	Tx Empty Trigger This bit used to select the empty threshold level at which the THRE Interrupts are generated when the mode is active. The following trigger levels are supported:

Table 13-61 Fields for register: FCR (continued)

Bits	Name	R/W	Description
			0x0: FIFO empty **0x1:** 2 characters in the FIFO **0x2:** FIFO ¼ full **0x3:** FIFO ½ full **Value After Reset:** 0x0
[3]			Reserved and read as 0
[2]	FIFOR	W	xMIT FIFO Reset This resets the control portion of the transmit FIFO and treats the FIFO as empty. This bit is 'self-clearing'. It is not necessary to clear this bit. **Value After Reset:** 0x0
[1]	RFIFOR	W	RCVR FIFO Reset This resets the control portion of the receive FIFO and treats the FIFO as empty. This bit is 'self-clearing'. It is not necessary to clear this bit. **Value After Reset:** 0x0
[0]	FIFOE	W	FIFO Enable This bit enables/disables the transmit (xMIT) and receive (RCVR) FIFOs. Whenever the value of this bit is changed both the xMIT and RCVR controller portion of FIFOs is reset. **0x0:** FIFO disabled **0x1:** FIFO enabled **Value After Reset:** 0x0

13.3.2.2.8. Line Control Register (LCR)

The LCR register is at offset 0x0C.

The following table shows the register bit assignments.

Table 13-62 Fields for register: LCR

Bits	Name	R/W	Description
[31:8]			Reserved and read as 0
[7]	DLAB	R/W	Divisor Latch Access Bit Writeable only when UART is not busy (USR[0] is 0). This bit is used to enable reading and writing of the Divisor Latch register (DLL and DLH) to set the baud rate of the UART. This bit must be cleared after initial baud rate setup in order to access other registers. **Value After Reset:** 0x0
[6]	BC	R/W	Break Control Bit This is used to cause a break condition to be transmitted to the receiving device. If set to 1, the serial output is forced to the spacing (logic 0) state. When not in Loopback Mode, as determined by MCR[4], the sout line is forced low until the

Table 13-62 Fields for register: LCR (continued)

Bits	Name	R/W	Description
			Break bit is cleared. When in Loopback Mode, the break condition is internally looped back to the receiver. **Value After Reset:** 0x0
[5]	SP	R/W	Stick Parity Writeable only when UART is not busy (USR[0] is 0). This bit is used to force parity value. When PEN, EPS, and SP are set to 1, the parity bit is transmitted and checked as logic 0. If PEN and SP are set to 1 and EPS is logic 0, then parity bit is transmitted and checked as logic 1. If this bit is set to 0, Stick Parity is disabled. **Value After Reset:** 0x0
[4]	EPS	R/W	Even Parity Select Writeable only when UART is not busy (USR[0] is 0). This is used to select between even and odd parity, when parity is enabled (PEN set to 1). If set to 1, an even number of logic 1s is transmitted or checked. If set to 0, an odd number of logic 1s is transmitted or checked. **Value After Reset:** 0x0
[3]	PEN	R/W	Parity Enable Writeable only when UART is not busy (USR[0] is 0). This bit is used to enable and disable parity generation and detection in transmitted and received serial character respectively. **0x0:** Parity disabled **0x1:** Parity enabled **Value After Reset:** 0x0
[2]	STOP	R/W	Number of stop bits Writeable only when UART is not busy (USR[0] is 0). This is used to select the number of stop bits per character that the peripheral transmits and receives. If set to 0, one stop bit is transmitted in the serial data. If set to 1 and the data bits are set to 5 (LCR[1:0] set to 0) one and a half stop bits are transmitted. Otherwise, two stop bits are transmitted. Regardless of the number of stop bits selected the receiver checks only the first stop bit. **0x0:** 1 stop bit **0x1:** 1.5 stop bits when DLS (LCR[1:0]) is 0, otherwise 2 stop bits **Value After Reset:** 0x0
[1:0]	DLS	R/W	Data Length Select Writeable only when UART is not busy (USR[0] is 0); otherwise always writable, always readable. This is used to select the number of data bits per character that the peripheral transmits and receives. The number of bits that may be selected are as follows: **0x0:** 5 bits **0x1:** 6 bits **0x2:** 7 bits **0x3:** 8 bits **Value After Reset:** 0x0

13.3.2.2.9. Mode Control Register (MCR)

The MCR register is at offset 0x10.

The following table shows the register bit assignments.

Table 13-63 Fields for register: MCR

Bits	Name	R/W	Description
[31:7]			Reserved and read as 0
[6]	SIRE	R/W	SIR Mode Enable To enable SIR mode, write the appropriate value to the MCR register before writing to the LCR register. **Values:** **0x0:** IrDA SIR Mode disabled **0x1:** IrDA SIR Mode enabled **Value After Reset:** 0x0
[5]	AFCE	R	Auto Flow Control Enable **Values:** **0x0:** Auto Flow Control Mode disabled **0x1:** Auto Flow Control Mode enabled **Value After Reset:** 0x0
[4:0]			Reserved

13.3.2.2.10. Line Status Register (LSR)

The LSR register is at offset 0x14.

The following table shows the register bit assignments.

Table 13-64 Fields for register: LSR

Bits	Name	R/W	Description
[31:9]			Reserved and read as 0
[8]	ADDR_RCVD	R	**Address Received Bit** If 9-bit data mode (LCR_EXT[0]=1) is enabled, this bit is used to indicate the 9th bit of the receive data is set to 1. This bit can also be used to indicate whether the incoming character is address or data. **0x0:** Indicates the character is data **0x1:** Indicates the character is address In the FIFO mode, since the 9th bit is associated with a character received, it is revealed when the character with the 9th bit set to 1 is at the top of the FIFO. Reading the LSR clears the 9bit. User needs to ensure that interrupt gets cleared (reading LSR register) before the next address byte arrives. If there is a delay in clearing the interrupt, then software will not be able to distinguish between multiple address related interrupt.

Table 13-64 Fields for register: `LSR` (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0
[7]	`RFE`	R	Receiver FIFO Error bit This bit is only relevant when FIFOs are enabled (`FCR[0]` set to 1). This is used to indicate if there is at least one parity error, framing error, or break indication in the FIFO. **0x0:** No error in Rx FIFO **0x1:** Error in Rx FIFO This bit is cleared when the `LSR` is read and the character with the error is at the top of the receiver FIFO and there are no subsequent errors in the FIFO. **Value After Reset:** 0x0
[6]	`TEMT`	R	Transmitter Empty bit If FIFOs enabled (`FCR[0]` set to 1), this bit is set whenever the Transmitter Shift Register and the FIFO are both empty. If FIFOs are disabled, this bit is set whenever the `THR` and the Transmitter Shift Register are both empty. **Value After Reset:** 0x1
[5]	`THRE`	R	Transmit Holding Register Empty bit If THRE mode is disabled (`IER[7]` set to 0) and regardless of FIFO's being enabled or not, this bit indicates that the `THR` or Tx FIFO is empty. This bit is set whenever data is transferred from the `THR` or Tx FIFO to the Transmitter Shift Register and no new data has been written to the `THR` or Tx FIFO. This also causes a `THRE` Interrupt to occur, if the `THRE` Interrupt is enabled. If both modes are active (`IER[7]` set to 1 and `FCR[0]` set to 1 respectively), the functionality is switched to indicate the transmitter FIFO is full, and no longer controls `THRE` interrupts, which are then controlled by the `FCR[5:4]` threshold setting. For more details, see Programmable THRE Interrupt. **Value After Reset:** 0x1
[4]	`BI`	R	Break Interrupt bit This bit used to indicate the detection of a break sequence on the serial input data. It is set whenever the serial input is held in logic '0' state for longer than the sum of `start time + data bits + parity + stop bits`. In FIFO mode, the character associated with the break condition is carried through the FIFO and is revealed when the character is at the top of the FIFO. Reading the `LSR` clears the `BI` bit. In non-FIFO mode, the `BI` indication occurs immediately and persists until the `LSR` is read. If a FIFO is full when a break condition is received, a FIFO overrun occurs. The break condition and all the information associated with it — parity and framing errors — is discarded; any information that a break character was received is lost. **Value After Reset:** 0x0
[3]	`FE`	R	Framing Error bit This bit used to indicate the occurrence of a framing error in the receiver. A framing error occurs when the receiver does not detect a valid STOP bit in the received data.

Table 13-64 Fields for register: LSR (continued)

Bits	Name	R/W	Description
			In the FIFO mode, since the framing error is associated with a character received, it is revealed when the character with the framing error is at the top of the FIFO. When a framing error occurs, the UART tries to resynchronize. It does this by assuming that the error was due to the start bit of the next character and then continues receiving the other bit; that is, data, and/or parity and stop. It should be noted that the *Framing Error* (**FE**) bit (LSR[3]) is set if a break interrupt has occurred, as indicated by *Break Interrupt* (**BI**) bit (LSR[4]). This happens because the break character implicitly generates a framing error by holding the serial input to logic 0 for longer than the duration of a character. **0x0:** No framing error **0x1:** Framing error Reading the LSR clears the FE bit. **Value After Reset:** 0x0
[2]	PE	R	Parity Error bit This bit used to indicate the occurrence of a parity error in the receiver if the Parity Enable (PEN) bit (LCR[3]) is set. In the FIFO mode, since the parity error is associated with a character received, it is revealed when the character with the parity error arrives at the top of the FIFO. It should be noted that the *Parity Error* (**PE**) bit (LSR[2]) can be set if a break interrupt has occurred, as indicated by *Break Interrupt* (**BI**) bit (LSR[4]). In this situation, the Parity Error bit is set if parity generation and detection is enabled (LCR[3]=1) and the parity is set to odd (LCR[4]=0). **0x0:** No parity error **0x1:** Parity error Reading the LSR clears the PE bit. **Value After Reset:** 0x0
[1]	OE	R	Overrun Error bit This bit used to indicate the occurrence of an overrun error. This occurs if a new data character was received before the previous data was read. An overrun error occurs when the FIFO is full and a new character arrives at the receiver. The data in the FIFO is retained and the data in the receive shift register is lost. **0x0:** No overrun error **0x1:** Overrun error Reading the LSR clears the OE bit. **Value After Reset:** 0x0
[0]	DR	R	Data Ready bit This bit used to indicate that the receiver contains at least one character in the RBR or the receiver FIFO. **0x0:** No data ready **0x1:** Data ready This bit is cleared when the RBR is read in non-FIFO mode, or when the receiver FIFO is empty, in FIFO mode.

Table 13-64 Fields for register: LSR (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0

13.3.2.2.11. Scratchpad Register (SCR)

The SCR register is at offset 0x1C.

The following table shows the register bit assignments.

Table 13-65 Fields for register: SCR

Bits	Name	R/W	Description
[31:8]			Reserved and read as 0
[7:0]	SR	R/W	This register is for programmers to use as a temporary storage space. It has no defined purpose in the UART. **Value After Reset:** 0x0

13.3.2.2.12. Shadow Receive Buffer Register (SRBRi)

The SRBRi register is at offset 0x30+ (i * 0x4), where i is from 0 to 15.

This register can be accessed only when the DLAB bit (LCR[7]) is cleared.

The following table shows the register bit assignments.

Table 13-66 Fields for register: SRBRi

Bits	Name	R/W	Description
[31:9]			Reserved and read as 0
[8:0]	SRBRn	R	This is a shadow register for the RBR and has been allocated sixteen 32-bit locations so as to accommodate burst accesses from the master. This register contains the data byte received on the serial in port (Rx) in UART mode. The data in this register is valid only if the Data Ready (DR) bit in the Line Status Register (LSR) is set. If FIFOs are disabled (FCR[0] set to 0), the data in the RBR must be read before the next data arrives; otherwise it is overwritten, resulting in an overrun error. If FIFOs are enabled (FCR[0] set to 1), this register accesses the head of the receive FIFO. If the receive FIFO is full and this register is not read before the next data character arrives, then the data already in the FIFO are preserved, but any incoming data is lost. An overrun error also occurs. **Value After Reset:** 0x0

13.3.2.2.13. Shadow Transmit Holding Register (STHRi)

The STHRi register is at offset 0x30+ (i * 0x4), where i is from 0 to 15.

This register can be accessed only when the DLAB bit (LCR[7]) is cleared.

The following table shows the register bit assignments.

Table 13-67 Fields for register: STHRi

Bits	Name	R/W	Description
[31:9]			Reserved and read as 0
[8:0]	STHR	W	This is a shadow register for the THR and has been allocated sixteen 32-bit locations so as to accommodate burst accesses from the master. This register contains data to be transmitted on the serial out (Tx) in UART mode. Data should only be written to the THR when the THR Empty (THRE) bit (LSR[5]) is set. If FIFOs are disabled (FCR[0] set to 0) and THRE is set, writing a single character to the THR clears the THRE. Any additional writes to the THR before the THRE is set again causes the THR data to be overwritten. If FIFOs are enabled (FCR[0] set to 1) and THRE is set, 16 characters of data may be written to the THR before the FIFO is full. **Value After Reset:** 0x0

13.3.2.2.14. FIFO Access Register (FAR)

The FAR register is at offset 0x70.

The following table shows the register bit assignments.

Table 13-68 Fields for register: FAR

Bits	Name	R/W	Description
[31:1]			Reserved and read as 0
[0]	FAR	R/W	This register is use to enable FIFO access mode for testing, so that the receive FIFO can be written by the master and the transmit FIFO can be read by the master when FIFOs are enabled. When FIFOs are not enabled it allows the RBR to be written by the master and the THR to be read by the master. **0x0:** FIFO access mode disabled **0x1:** FIFO access mode enabled When the FIFO access mode is enabled/disabled, the control portion of the receive FIFO and transmit FIFO is reset and the FIFOs are treated as empty. **Value After Reset:** 0x0

13.3.2.2.15. Transmit FIFO Read Register (TFR)

The TFR register is at offset 0x74.

The following table shows the register bit assignments.

Table 13-69 Fields for register: TFR

Bits	Name	R/W	Description
[31:8]			Reserved and read as 0
[7:0]	TFR	R	Transmit FIFO Read These bits are only valid when FIFO access mode is enabled (FAR[0] is set to 1). When FIFOs are enabled, reading this register gives the data at the top of the transmit FIFO. Each consecutive read pops the transmit FIFO and gives the next data value that is currently at the top of the FIFO. When FIFOs are not enabled, reading this register gives the data in the THR.

Table 13-69 Fields for register: TFR (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0

13.3.2.2.16. Receive FIFO Write Register (RFW)

The RFW register is at offset 0x78.

The following table shows the register bit assignments.

Table 13-70 Fields for register: RFW

Bits	Name	R/W	Description
[31:10]			Reserved and read as 0
[9]	RFFE	W	Receive FIFO Framing Error This bit is only valid when FIFO access mode is enabled (FAR[0] is set to 1). When FIFOs are enabled, this bit is used to write framing error detection information to the receive FIFO. When FIFOs are not enabled, this bit is used to write framing error detection information to the RBR. **Value After Reset:** 0x0
[8]	RFPE	W	Receive FIFO Parity Error This bit is only valid when FIFO access mode is enabled (FAR[0] is set to 1). When FIFOs are enabled, this bit is used to write parity error detection information to the receive FIFO. When FIFOs are not enabled, this bit is used to write parity error detection information to the RBR. **Value After Reset:** 0x0
[7:0]	RFWD	W	Receive FIFO Write Data These bits are only valid when FIFO access mode is enabled (FAR[0] is set to 1). When FIFOs are enabled, the data that is written to the RFWD is pushed into the receive FIFO. Each consecutive write pushes the new data to the next write location in the receive FIFO. When FIFOs are not enabled, the data that is written to the RFWD is pushed into the RBR. **Value After Reset:** 0x0

13.3.2.2.17. UART Status Register (USR)

The USR register is at offset 0x7C.

The following table shows the register bit assignments.

Table 13-71 Fields for register: USR

Bits	Name	R/W	Description
[31:5]			Reserved and read as 0
[4]	RFF	R	Receive FIFO Full This bit indicates that the receive FIFO is full. **0x0:** Receive FIFO is not full **0x1:** Receive FIFO is full This bit is cleared when the Rx FIFO is no longer full. **Value After Reset:** 0x0

Table 13-71 Fields for register: USR (continued)

Bits	Name	R/W	Description
[3]	RFNE	R	Receive FIFO Not Empty This bit indicates that the receive FIFO contains one or more entries. **0x0:** Receive FIFO is empty **0x1:** Receive FIFO is not empty This bit is cleared when the Rx FIFO is empty. **Value After Reset:** 0x0
[2]	TFE	R	Transmit FIFO Empty This bit indicates that the transmit FIFO is empty. **0x0:** Transmit FIFO is not empty **0x1:** Transmit FIFO is empty This bit is cleared when the Tx FIFO is no longer empty. **Value After Reset:** 0x1
[1]	TFNF	R	Transmit FIFO Not Full This bit indicates that the transmit FIFO is not full. **0x0:** Transmit FIFO is full **0x1:** Transmit FIFO is not full This bit is cleared when the Tx FIFO is full. **Value After Reset:** 0x1
[0]	BUSY	R	UART Busy This bit indicates that a serial transfer is in progress; when cleared, it indicates that the UART is idle or inactive. **0x0:** UART is idle or inactive **0x1:** UART is busy (actively transferring data) This bit will be set to 1 (busy) under any of the following conditions: • Transmission in progress on serial interface. • Transmit data present in THR, when FIFO access mode is not being used (FAR = 0) and the baud divisor is non-zero ({DLH, DLL} does not equal 0) when the divisor latch access bit is 0 (LCR.DLAB= 0). • Reception in progress on the interface. • Receive data present in RBR, when FIFO access mode is not being used (FAR = 0). It is possible for the UART Busy bit to be cleared even though a new character may have been sent from another device. That is, if the UART has no data in THR and RBR and there is no transmission in progress and a start bit of a new character has just reached the UART. This is due to the fact that a valid start is not seen until the middle of the bit period and this duration is dependent on the baud divisor that has been programmed. **Value After Reset:** 0x0

13.3.2.2.18. Transmit FIFO Level Register (TFL)

The TFL register is at offset 0x80.

The following table shows the register bit assignments.

Table 13-72 Fields for register: TFL

Bits	Name	R/W	Description
[31:5]			Reserved and read as 0
[4:0]	TFL	R	Transmit FIFO Level This bit field indicates the number of data entries in the transmit FIFO. **Value After Reset:** 0x0

13.3.2.2.19. Receive FIFO Level Register (RFL)

The RFL register is at offset 0x84.

The following table shows the register bit assignments.

Table 13-73 Fields for register: RFL

Bits	Name	R/W	Description
[31:5]			Reserved and read as 0
[4:0]	RFL	R	Receive FIFO Level. This bit field indicates the number of data entries in the receive FIFO. **Value After Reset:** 0x0

13.3.2.2.20. Software Reset Register (SRR)

The SRR register is at offset 0x88.

The following table shows the register bit assignments.

Table 13-74 Fields for register: SRR

Bits	Name	R/W	Description
[31:3]			Reserved and read as 0
[2]	xFR	W	xMIT FIFO Reset This is a shadow bit for the xMIT FIFO Reset bit (FCR[2]). This can be used to remove the burden on software having to store previously written FCR values (which are pretty static) just to reset the transmit FIFO. This resets the control portion of the transmit FIFO and treats the FIFO as empty. This will also de-assert the DMA Tx request and single signals. This bit is 'self-clearing'. It is not necessary to clear this bit. **Value After Reset:** 0x0
[1]	RFR	W	RCVR FIFO Reset This is a shadow bit for the RCVR FIFO Reset bit (FCR[1]). This can be used to remove the burden on software having to store previously written FCR values (which are pretty static) just to reset the receive FIFO. This resets the control portion of the receive FIFO and treats the FIFO as empty. This will also de-assert the DMA Tx request and single signals. This bit is 'self-clearing'. It is not necessary to clear this bit. **Value After Reset:** 0x0

Table 13-74 Fields for register: SRR (continued)

Bits	Name	R/W	Description
[0]	UR	W	UART Reset This bit asynchronously resets the UART and synchronously removes the reset assertion. **Value After Reset:** 0x0

13.3.2.2.21. Shadow Break Control Register (SBCR)

The SBCR register is at offset 0x90.

The following table shows the register bit assignments.

Table 13-75 Fields for register: SBCR

Bits	Name	R/W	Description
[31:1]			Reserved and read as 0
[0]	SBCR	R/W	Shadow Break Control Bit This is a shadow register for the Break Control Bit bit (LCR[6]), this can be used to remove the burden of having to performing a read modify write on the LCR. This is used to cause a break condition to be transmitted to the receiving device. If set to 1, the serial out is forced to the spacing (logic 0) state. When not in Loopback Mode, as determined by MCR[4], the sout line is forced low until the Break bit is cleared. If MCR[6] set to 1, the serial out line is continuously pulsed. When in Loopback Mode, the break condition is internally looped back to the receiver. **Value After Reset:** 0x0

13.3.2.2.22. FIFO Enable Register (SFE)

The SFE register is at offset 0x98.

The following table shows the register bit assignments.

Table 13-76 Fields for register: SFE

Bits	Name	R/W	Description
[31:1]			Reserved and read as 0
[0]	SFE	R/W	Shadow FIFO Enable This is a shadow register for the FIFO enable bit (FCR[0]). This can be used to remove the burden of having to store the previously written value to the FCR in memory and having to mask this value so that only the FIFO enable bit gets updated. This enables/disables the transmit (xMIT) and receive (RCVR) FIFOs. If this bit is set to 0 (disabled) after being enabled then both the xMIT and RCVR controller portion of FIFOs are reset. **Value After Reset:** 0x0

13.3.2.2.23. Shadow RCVR Trigger Register (SRT)

The SRT register is at offset 0x9C.

The following table shows the register bit assignments.

Table 13-77 Fields for register: SRT

Bits	Name	R/W	Description
[31:2]			Reserved and read as 0
[1:0]	SRT	R/W	Shadow RCVR Trigger This is a shadow register for the RCVR Trigger bits (FCR[7:6]). This can be used to remove the burden of having to store the previously written value to the FCR in memory and having to mask this value so that only the RCVR Trigger bits get updated. This is used to select the trigger level in the receiver FIFO at which the Received Data Available Interrupt is generated. The following trigger levels are supported: **0x0**: 1 character in FIFO **0x1**: FIFO ¼ full **0x2**: FIFO ½ full **0x3**: FIFO 2 less than full **Value After Reset:** 0x0

13.3.2.2.24. Shadow Tx Empty Trigger Register (STET)

The STET register is at offset 0xA0.

The following table shows the register bit assignments.

Table 13-78 Fields for register: STET

Bits	Name	R/W	Description
[31:2]			Reserved and read as 0
[1:0]	STET	R/W	Shadow Tx Empty Trigger This is a shadow register for the Tx Empty Trigger bits (FCR[5:4]). This can be used to remove the burden of having to store the previously written value to the FCR in memory and having to mask this value so that only the Tx Empty Trigger bit gets updated. This is used to select the empty threshold level at which the THRE Interrupts are generated when THRE mode is active. The following trigger levels are supported: **0x0**: FIFO empty **0x1**: 2 characters in FIFO **0x2**: FIFO ¼ full **0x3**: FIFO ½ full **Value After Reset:** 0x0

13.3.2.2.25. Halt Tx Register (HTx)

The HTx register is at offset 0xA4.

The following table shows the register bit assignments.

Table 13-79 Fields for register: HTx

Bits	Name	R/W	Description
[31:1]			Reserved and read as 0
[0]	HTx	R/W	This register is use to halt transmissions for testing, so that the transmit FIFO can be filled by the master when FIFOs are implemented and enabled. **0x0:** Halt Tx disabled **0x1:** Halt Tx enabled If FIFOs are not enabled, the setting of the Halt Tx register has no effect on operation. **Value After Reset:** 0x0

13.3.2.2.26. DMA Software Acknowledge Register (DMASA)

The DMASA register is at offset 0xA8.

The following table shows the register bit assignments.

Table 13-80 Fields for register: DMASA

Bits	Name	R/W	Description
[31:1]			Reserved and read as 0
[0]	DMASA	W	DMA Software Acknowledge This register is use to perform DMA software acknowledge if a transfer needs to be terminated due to an error condition. This bit is 'self-clearing' and it is not necessary to clear this bit. **Values:** **0x1**: DMA software acknowledge **Value After Reset:** 0x0

13.3.2.2.27. Transceiver Control Register (TCR)

The TCR register is at offset 0xAC.

Exists: UART6 and UART7.

This register is used to enable or disable RS485 mode and also control the polarity values for Driven enable (de) and Receiver Enable (re) signals.

Driven Enable (de) and Receiver Enable (re) signals correspond to the appropriate UART*_DE and UART*_RE BE-U1000 microcontroller I/O pins, where * is the UART number. For more information, refer the ***BE-U1000 Datasheet***.

The following table shows the register bit assignments.

Table 13-81 Fields for register: TCR

Bits	Name	R/W	Description
[31:5]			Reserved and read as 0.

Table 13-81 Fields for register: TCR (continued)

Bits	Name	R/W	Description
[4:3]	XFER_MODE	R/W	Transfer Mode. **Values:** **0x0:** In this mode, transmit and receive can happen simultaneously. The user can enable DE_EN, RE_EN at any point of time. Turn around timing as programmed in the TAT register is not applicable in this mode. **0x1:** In this mode, de and re are mutually exclusive. Either de or re only one of them is expected to be enabled through programming. Hardware will consider the Turn Around timings which are programmed in the TAT register while switching from Receiver Enable to Driver Enable or Driver Enable to Receiver Enable.For transmission Hardware will wait if it is in middle of receiving any transfer, before it starts transmitting. **0x2:** In this mode, de and re are mutually exclusive. Once DE_EN / RE_EN is programed - by default re will be enabled and UART Controller will be ready to receive. If the user programs the Tx FIFO with the data then UART, after ensuring no receive is in progress, disable re and enable de signal. Once the Tx FIFO becomes empty, re signal gets enabled and de signal will be disabled.In this mode of operation hardware will consider the Turn Around timings which are programmed in the TAT register while switching from Receiver Enable to Driver Enable or Driver Enable to Receiver Enable. In this mode, de and re signals are strictly complementary to each other. **Value After Reset:** 0x0
[2]	DE_POL	R/W	Driver Enable Polarity. **Values:** **0x0**: de signal is active low **0x1**: de signal is active high **Value After Reset:** 0x1
[1]	RE_POL	R/W	Receiver Enable Polarity. **Values:** **0x0**: re signal is active low **0x1**: re signal is active high **Value After Reset:** 0x1
[0]	RS485_EN	R/W	RS485 Transfer Enable. **Values:** **0x0**: UART Mode. All other fields in this register are reserved and registers DE_EN / RE_EN / TAT are also reserved. **0x1**: RS485 Mode. In this mode, the transfers will happen in RS485 mode. All other fields of this register are applicable. **Value After Reset:** 0x0

13.3.2.2.28. Driver Output Enable Register (DE_EN)

The DE_EN register is at offset 0xB0.

Exists: UART6 and UART7.

The following table shows the register bit assignments.

Table 13-82 Fields for register: DE_EN

Bits	Name	R/W	Description
[31:1]			Reserved and read as 0.
[0]	DE_Enable	R/W	DE Enable control. The register bit is used to control assertion and de-assertion of Driven enable (de) signal. **Values:** **0x0**: De-assert de signal **0x1**: Assert de signal **Value After Reset:** 0x0

13.3.2.2.29. Receiver Output Enable Register (RE_EN)

The RE_EN register is at offset 0xB4.

Exists: UART6 and UART7.

The following table shows the register bit assignments.

Table 13-83 Fields for register: RE_EN

Bits	Name	R/W	Description
[31:1]			Reserved and read as 0.
[0]	RE_Enable	R/W	RE Enable control. The register bit is used to control assertion and de-assertion of re signal. **Values:** **0x0:** De-assert re signal **0x1:** Assert re signal **Value After Reset:** 0x0

13.3.2.2.30. Driver Output Enable Timing Register (DET)

The DET register is at offset 0xB8.

Exists: UART6 and UART7.

The following table shows the register bit assignments.

Table 13-84 Fields for register: DET

Bits	Name	R/W	Description
[31:24]			Reserved and read as 0.
[23:16]	DE_De-assertion_Time	R/W	Driver Enable de-assertion time.

Table 13-84 Fields for register: DET (continued)

Bits	Name	R/W	Description
			This field controls the amount of time (in terms of number of serial clock periods) between the end of stop bit on the serial out to the falling edge of Driver Enable (de) signal. **Value After Reset:** 0x0
[15:8]			Reserved and read as 0.
[7:0]	DE_Assertion_Time	R/W	Driver Enable assertion time. This field controls the amount of time (in terms of number of serial clock periods) between the assertion of rising edge of Driver Enable (de) signal to serial transmit enable. Any data in transmit buffer, will start on serial output after the transmit enable. **Value After Reset:** 0x0

13.3.2.2.31. TurnAround Timing Register (TAT)

The TAT register is at offset 0xBC.

Exists: UART6 and UART7.

The following table shows the register bit assignments.

Table 13-85 Fields for register: TAT

Bits	Name	R/W	Description
[31:16]	RE_to_DE	R/W	Receiver Enable to Driver Enable TurnAround time. Turnaround time (in terms of serial clock) for re De-assertion to de assertion. • If the de assertion time in the DET register is 0, then the actual value is the programmed value + 3. • If the de assertion time in the DET register is 1, then the actual value is the programmed value + 2. • If the de assertion time in the DET register is greater than 1, then the actual value is the programmed value + 1. **Value After Reset:** 0x0
[15:0]	DE_to_RE	R/W	Driver Enable to Receiver Enable TurnAround time. Turnaround time (in terms of serial clock) for de De-assertion to re assertion. The actual time is the programmed value + 1. **Value After Reset:** 0x0

13.3.2.2.32. Divisor Latch Fraction Register (DLF)

The DLF register is at offset 0xC0.

The following table shows the register bit assignments.

Table 13-86 Fields for register: DLF

Bits	Name	R/W	Description
[31:4]			Reserved and read as 0
[3:0]	DLF	R/W	Fractional part of divisor. The fractional value is added to integer value set by DLH, DLL. For information on DLF values, see the Fractional Baud Rate Support. **Value After Reset:** 0x0

13.3.2.2.33. Receive Address Register (RAR)

The RAR register is at offset 0xC4.

The following table shows the register bit assignments.

Table 13-87 Fields for register: RAR

Bits	Name	R/W	Description
[31:8]			Reserved and read as 0
[7:0]	RAR	R/W	This is an address matching register during receive mode. If the 9-th bit is set in the incoming character then the remaining 8-bits will be checked against this register value. If the match happens then sub-sequent characters with 9-th bit set to 0 will be treated as data byte until the next address byte is received. • This register is applicable only when LCR_EXT[1] and LCR_EXT[0] bits are set to 1. • RAR should be programmed only when UART is not busy, at any point of the time. However, user must not change this register value when any receive is in progress. **Value After Reset:** 0x0

13.3.2.2.34. Transmit Address Register (TAR)

The TAR register is at offset 0xC8.

The following table shows the register bit assignments.

Table 13-88 Fields for register: TAR

Bits	Name	R/W	Description
[31:8]			Reserved and read as 0
[7:0]	TAR	R/W	Address matching register during transmit mode If LCR_EXT[0] bit is enabled, then UART will send the 9-bit character with 9-th bit set to 1 and remaining 8-bit address will be sent from this register provided SEND_ADDR bit of LCR_EXT register is set to 1. • This register is used only to send the address. The normal data should be sent by programming THR register. • Once the address is started to send on the UART serial line, then SEND_ADDR bit will be auto-cleared by the hardware. **Value After Reset:** 0x0

13.3.2.2.35. Line Extended Control Register (LCR_EXT)

The LCR_EXT register is at offset 0xCC.

The following table shows the register bit assignments.

Table 13-89 Fields for register: LCR_EXT

Bits	Name	R/W	Description
[31:4]			Reserved and read as 0
[3]	TRANSMIT_MODE	R/W	Transmit mode control bit This bit is used to control the type of transmit mode during 9-bit data transfers. **0x0:** In this mode of operation, Transmit Holding Register (THR) and Shadow Transmit Holding Register (STHR) are 8-bit wide. The user needs to program the address into Transmit Address Register (TAR) and data into the THR/STHR register. SEND_ADDR bit is used as a control knob to indicate the UART on when to send the address. **0x1:** In this mode of operation, Transmit Holding Register (THR) and Shadow Transmit Holding Register (STHR) are 9-bit wide. The user needs to ensure that the THR/STHR register is written correctly for address/data.Address: 9th bit is set to 1 Data : 9th bit is set to 0. Transmit Address Register (TAR) is not applicable in this mode of operation. **Value After Reset:** 0x0
[2]	SEND_ADDR	R/W	Send address control bit This bit is used as a control knob for the user to determine when to send the address during transmit mode. **0x0:** 9-bit character will be transmitted with 9-th bit set to 0 and the remaining 8-bits will be taken from the Tx FIFO which is programmed through 8-bit wide THR/STHR register. **0x1:** 9-bit character will be transmitted with 9-th bit set to 1 and the remaining 8-bits will match to what is being programmed in Transmit Address Register (TAR). This bit is auto-cleared by the hardware, after sending out the address character. User is not expected to program this bit to 0. This field is applicable only when DLS_E bit is set to 1 and TRANSMIT_MODE is set to 0. **Value After Reset:** 0x0
[1]	ADDR_MATCH	R/W	Address Match Mode This bit is used to enable the address match feature during receive. **0x0:** Normal mode UART will start to receive the data and 9-bit character will be formed and written into the Rx FIFO. User is responsible to read the data and differentiate b/n address and data. **0x1:** Address match mode UART will wait until the incoming character with 9-th bit set to 1. And further checks to see if the address matches with what is

Table 13-89 Fields for register: `LCR_EXT` (continued)

Bits	Name	R/W	Description
			programmed in Receive Address Match Register. If match is found, then sub-sequent characters will be treated as valid data and UART starts receiving data. This field is applicable only when `DLS_E` is set to 1. **Value After Reset:** 0x0
[0]	`DLS_E`	R/W	Extension for `DLS`. This bit is used to enable 9-bit data for transmit and receive transfers. **Value After Reset:** 0x0

13.3.2.2.36. Register Timeout Counter Reset Value Register (`REG_TIMEOUT_RST`)

The `REG_TIMEOUT_RST` register is at offset 0xD4.

The following table shows the register bit assignments.

Table 13-90 Fields for register: `REG_TIMEOUT_RST`

Bits	Name	R/W	Description
[31:8]			Reserved and read as 0
[7:0]	`TIMEOUT_RST`	R/W	This field holds reset value of the Register timeout counter. **Value After Reset:** 0x80

13.4. SPI & QSPI

This section describes the *Serial Peripheral Interface* (**SPI**) and *Quad Serial Peripheral Interface* (**QSPI**) controllers of the BE-U1000.

In this section, both SPI and QSPI controllers are collectively referred to as *SPI Controllers.

The *SPI Controller is a programmable component used for serial communication with peripheral devices.

The BE-U1000 integrates six *SPI Controllers:

- Two controllers configured as serial masters (referred to as SPI Master)
- Two controllers configured as serial slaves (referred to as SPI Slave)
- Two *Quad SPI* (**QSPI**) controllers

A simplified block diagram of the *SPI Controller is shown in the following figure.

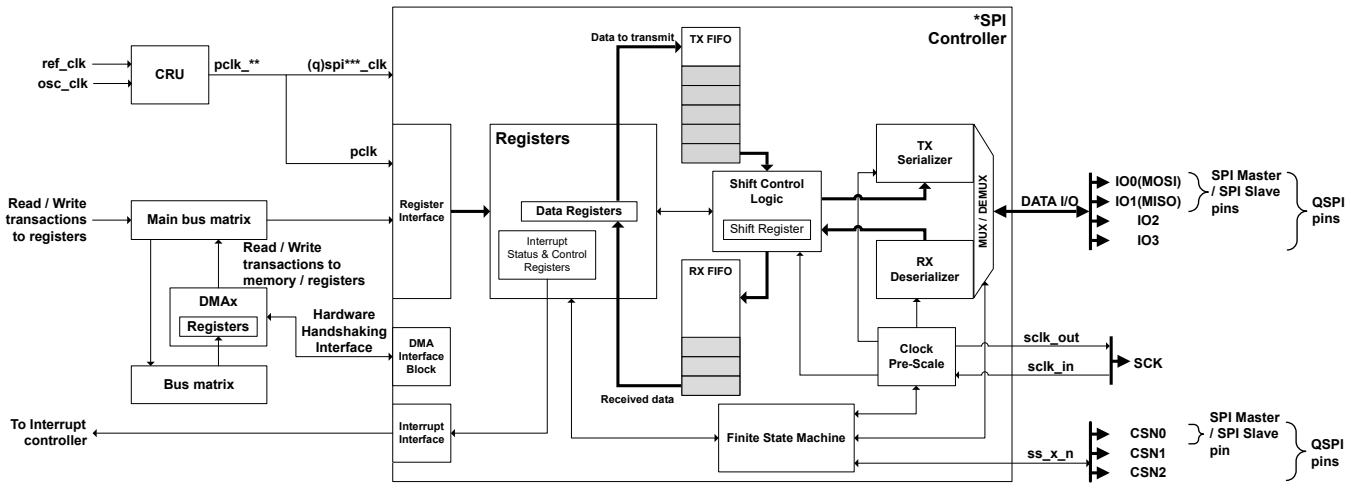

Figure 13-7 *SPI Controller block diagram

In the figure above:

- A double asterisk(**) indicates 0 for SPI_0, SPI_1, QSPI_0 and 1 for SPI_2, SPI_3, QSPI_1
- A triple asterisk (***) indicates 0 for SPI_0, QSPI_0, 1 for SPI_1, QSPI_1, 2 for SPI_2 and 3 for SPI_3
- x indicates 0 for DMA_0 and 1 for DMA_1 controllers

SPI_0 and SPI_2 are SPI Slave controllers. SPI_1 and SPI_3 are SPI Master controllers.

For more detailed information about I/O pins for QSPI, SPI Slave and SPI Master respectively, refer to the **4 Pinout and pin description** section of the ***BE-U1000 Datasheet.***

The *SPI Controller has the features listed in the following table:

Table 13-91 *SPI feature table

Feature	QSPI	SPI Master	SPI Slave
Serial master or slave configuration	Serial Master	Serial Master	Serial Slave
Maximum transfer size	32 bits	32 bits	32 bits
Receive FIFO buffer depth	8	8	8
Transmit FIFO buffer depth	8	8	8
Slave select lines	3	1	1
Programmable RXD sample logic	Yes	Yes	No
Maximum RXD sample delay	8	4	4
Endian conversion for XIP and data register reads	Use register programming	Use register programming	No

13.4.1. *SPI Functional Description

This section covers the following topics:

- *SPI Clock
- Endian Cinversion Support
- Transmit and Receive FIFO Buffers
- Transfer Modes
- Dual Data-Rate Support
- Serial Master Operation Mode
- Serial-Slave Operation Mode
- Interrupts

13.4.1.1. *SPI Clock

The *SPI clock (`spi*_clk`) is provided by the CRU. When the *SPI Controller is configured as a master device, the maximum frequency of the bit-rate clock (`sclk_out`) is one-half the frequency of `spi*_clk`. This allows the shift control logic to capture data on one clock edge of `sclk_out` and propagate data on the opposite edge.

The `sclk_out` line toggles only when an active transfer is in progress. At all other times it is held in an inactive state, as defined by the serial protocol under which it operates. The frequency of `sclk_out` can be derived from the following equation:

```
Fsclk_out = Fspi*_clk / SCKDV
```

`SCKDV` is a bit field in the programmable register `BAUDR`, holding any even value in the range 0 to 65534. If `SCKDV` is 0, then `sclk_out` is disabled.

When the *SPI Controller is configured as a slave device, the minimum frequency of `spi*_clk` is 12 times the maximum expected frequency of the bit-rate clock from the master device (`sclk_in`). This minimum frequency is to ensure that data on the master's data input line is stable before the master's shift control logic captures the data. The 12:1 ratio ensures that the slave has driven data onto the master's data input line three `spi*_clk` cycles before the data is captured.

13.4.1.2. Endian Conversion Support

The endian conversion support is available for QSPI and SPI Master controllers.

QSPI/SPI Master controller supports endian conversion feature for Data register and XIP read transfers. The endianness can be changed using the register programming bit (`CTRLR0.SECONV` bit). The following figure shows the endianness conversion when the `CTRLR0.SECONV` register bit is set to 1 for different values of data frame size (controlled via the `CTRLR0.DFS_32` bit settings).

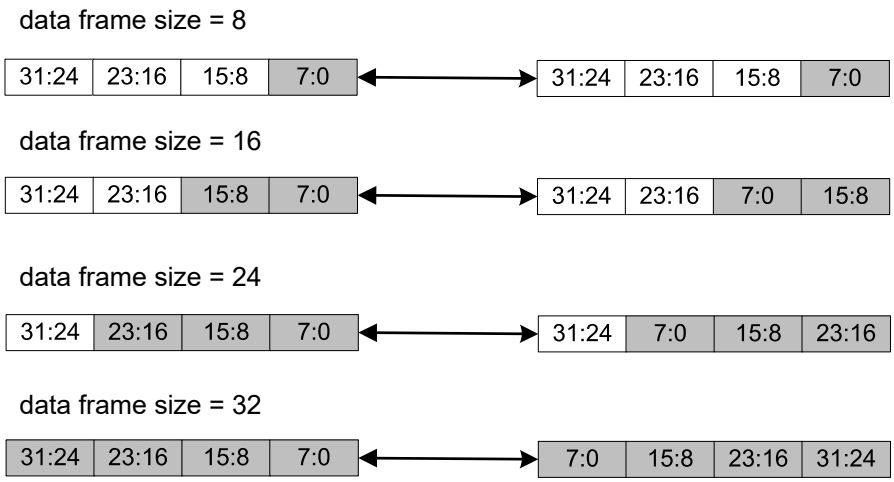

Figure 13-8 Endian conversion for data register and XIP reads

- Endian conversion is supported only for Data Frame Sizes of 16, 24 and 32.
- The Endianness conversion is only applicable for data register or XIP reads, all the other register reads are not affected by this feature

13.4.1.3. Transmit and Receive FIFO Buffers

The FIFO buffers used by *SPI Controllers have a depth of 8. The width of the transmit and receive FIFO buffers is fixed at 32 bits.

Each data entry in the FIFO buffers contains a single data frame. It is impossible to store multiple data frames in a single FIFO location.

The transmit and receive FIFO buffers are cleared when the *SPI Controller is disabled (`SPI_EN = 0` in `SPIENR`) or when it is reset (`presetn`).

The transmit FIFO is loaded by Register Interface write commands to the *SPI Controller Data Register (`DRi`). Data is popped (removed) from the transmit FIFO by the shift control logic into the transmit shift register. The transmit FIFO generates a FIFO empty interrupt request (`spi*_txe_intr`) when the number of entries in the FIFO is less than or equal to the FIFO threshold value. The threshold value, set through the programmable register `TXFTLR`, determines the level of FIFO entries at which an interrupt is generated. The threshold value allows you to provide early indication to the processor that the transmit FIFO is nearly empty. A transmit FIFO overflow interrupt (`spi*_txo_intr`) is generated if you attempt to write data into an already full transmit FIFO.

Data is popped from the receive FIFO by Register Interface read commands to the *SPI Controller Data Register (`DRi`). The receive FIFO is loaded from the receive shift register by the shift control logic. The receive FIFO generates a FIFO-full interrupt request (`spi*_rxf_intr`) when the number of entries in the FIFO is greater than or equal to the FIFO threshold value plus 1. The threshold value, set through programmable register `RXFTLR`, determines the level of FIFO entries at which an interrupt is generated.

The threshold value allows you to provide early indication to the processor that the receive FIFO is nearly full. A receive FIFO overrun interrupt (`spi*_rxo_intr`) is generated when the receive shift logic attempts to load data into a completely full receive FIFO. However, this newly received data are lost. A receive FIFO underflow interrupt (`spi*_rxu_intr`) is generated if you attempt to read from an empty receive FIFO. This alerts the processor that the read data is invalid.

The following tables provide description for different Transmit/Receive FIFO Threshold values for *SPI controllers respectively.

Table 13-92 Transmit FIFO Threshold (TFT) decode values

TFT Value	Description
000	`spi*_txe_intr` is asserted when 0 data entries are present in transmit FIFO
001	`spi*_txe_intr` is asserted when 1 data entries are present in transmit FIFO
010	`spi*_txe_intr` is asserted when 2 data entries are present in transmit FIFO
011	`spi*_txe_intr` is asserted when 3 data entries are present in transmit FIFO
100	`spi*_txe_intr` is asserted when 4 data entries are present in transmit FIFO
101	`spi*_txe_intr` is asserted when 5 data entries are present in transmit FIFO
110	`spi*_txe_intr` is asserted when 6 data entries are present in transmit FIFO
111	`spi*_txe_intr` is asserted when 7 data entries are present in transmit FIFO

Table 13-93 Receive FIFO Threshold (RFT) decode values

RFT Value	Description
000	`spi*_rxf_intr` is asserted when 1 data entries are present in receive FIFO
001	`spi*_rxf_intr` is asserted when 2 data entries are present in receive FIFO
010	`spi*_rxf_intr` is asserted when 3 data entries are present in receive FIFO
011	`spi*_rxf_intr` is asserted when 4 data entries are present in receive FIFO
100	`spi*_rxf_intr` is asserted when 5 data entries are present in receive FIFO
101	`spi*_rxf_intr` is asserted when 6 data entries are present in receive FIFO
110	`spi*_rxf_intr` is asserted when 7 data entries are present in receive FIFO
111	`spi*_rxf_intr` is asserted when 8 data entries are present in receive FIFO

For the description of the interrupt signals shown above and how to use them, see the Interrupts section.

13.4.1.4. Transfer Modes

When transferring data on the serial bus, the *SPI Controller operates in the modes discussed in this section. The transfer mode is set by writing to the `TMOD` bit of Control Register 0 (`CTRLR0`).

The transfer mode setting does not affect the duplex of the serial transfer.

Transmit And Receive

When `CTRLR0.TMOD = 0`, both transmit and receive logic are valid. Transmit data is popped from the transmit FIFO and sent through the data output line to the target device, which replies with data on the data input line. The receive data from the target device is moved from the Rx Shift block into the receive FIFO at the end of each data frame.

Transmit Only

When `CTRLR0.TMOD = 1`, the receive data are invalid and should not be stored in the receive FIFO. Transmit data are popped from the transmit FIFO and sent through the data output line to the target device, which replies with data on the data input line. At the end of the data frame, the Rx Shift block does not load its newly received data into the receive FIFO. The data in the Rx Shift block is overwritten by the next transfer.

You should mask interrupts originating from the receive logic when this mode is entered.

Receive Only

When `CTRLR0.TMOD = 2`, the transmit data are invalid. When configured as a slave, the transmit FIFO is never popped in Receive Only mode. The data output output remains at a constant logic level during the transmission. The receive data from the target device is moved from the Rx Shift block into the receive FIFO at the end of each data frame.

You should mask interrupts originating from the transmit logic when this mode is entered.

EEPROM Read

When `CTRLR0.TMOD = 3`, the transmit data is used to transmit an opcode and/or an address to the *Electrically Erasable Programmable Read-Only Memory* (**EEPROM**) device. Typically this takes three data frames (8-bit opcode followed by 8-bit upper address and 8-bit lower address). During the transmission of the opcode and address, no data is captured by the receive logic (as long as the *SPI master is transmitting data on its data output line, data on the data input line is ignored). The *SPI master continues to transmit data until the transmit FIFO is empty. Therefore, you should only have enough data frames in the transmit FIFO to supply the opcode and address to the EEPROM. If more data frames are in the transmit FIFO than are needed, then read data is lost.

When the transmit FIFO becomes empty (all control information has been sent), data on the data input is valid and is stored in the receive FIFO; the data output is held at a constant logic level. The serial transfer continues until the number of data frames received by the *SPI master matches the value of the `NDF` field in the `CTRLR1` register + 1.

13.4.1.5. Dual Data-Rate Support (QSPI only)

Only QSPI controller supports a dual date-rate support.

The *Dual Data-Rate* (**DDR**) mode supports the following modes of SPI protocol:

- `CTRLR0.SCPH=0 & CTRLR0.SCPOL=0` (Mode 0)
- `CTRLR0.SCPH=1 & CTRLR0.SCPOL=1` (Mode 3)

DDR commands enable data to be transferred on both edges of clock. Following are the different types of DDR commands:

- Address and data are transmitted (or received in case of data) in DDR format, while instruction is transmitted in standard format.
- Instruction, address, and data are all transmitted or received in DDR format.

The `SPI_DDR_EN` bit of `SPI_CTRLR0` register bit is used to determine if the Address and data have to be transferred in DDR mode and `INST_DDR_EN` bit of `SPI_CTRLR0` register bit is used to determine if Instruction must be transferred in DDR format. These bits are only valid when the `SPI_FRF` bit of `CTRLR0` register is set to be in Dual or Quad mode.

13.4.1.6. Serial Master Operation Mode (QSPI/SPI Master only)

Only QSPI and SPI Master controllers support serial master operation mode.

This mode enables serial communication with serial-slave peripheral devices. Since the QSPI/SPI Master Controller is configured as a serial-master device, it initiates and controls all serial transfers.

The serial bit-rate clock, generated and controlled by the QSPI/SPI Master Controller, is driven out on the `sclk_out` line. When the QSPI/SPI Master controller is disabled (`SPIENR.SPI_EN = 0`), no serial transfers can occur and `sclk_out` is held in “inactive” state.

Slave Selection

QSPI/SPI Master controller allows for serial slaves to be selected or addressed using either hardware or software. When implemented in hardware, serial slaves are selected under the control of dedicated hardware slave select (`ss_x_n`) lines (one slave select line for SPI Master and three slave select lines for QSPI). The number of slave select lines generated from the serial master is equal to the number of serial slaves present on the bus. The serial-master device asserts the select line of the target serial slave before data transfer begins. For details, see the description of `SER` register.

When implemented in software, the input select line for all serial slave devices should originate from a single slave select output on the serial master. In this mode it is assumed that the serial master has only a single slave select output. If there are multiple serial masters in the system, the slave select output from all masters can be logically ANDed to generate a single slave select input for all serial slave devices.

The main program in the software domain controls selection of the target slave device. The software would use the `SPIENR` register in all slaves in order to control which slave is to respond to the serial transfer request from the master device.

Master Contention Input

The QSPI/SPI Master controller master configuration has a serial slave select input, `ss_in_n`, which can be used to inform the QSPI/SPI Master that another serial master is active on the bus. When this input is active, the QSPI/SPI Master remains in an IDLE state and holds off any pending serial transfer until the `ss_in_n` input is returned to an in-active level. The state of the `ss_in_n` signal can be read using the `COL` bit of the `SR` register.

Data Transfers

Data transfers are started by the serial-master device. When the *SPI controller is enabled (`SPI_EN=1` in `SPIENR` register), at least one valid data entry is present in the transmit FIFO and a serial-slave device is selected. When actively transferring data, the busy flag (BUSY) in the Status Register (`SR`) is set. You must wait until the busy flag is cleared before attempting a new serial transfer.

The BUSY status is not set when the data are written into the transmit FIFO. This bit gets set only when the target slave has been selected and the transfer is underway. After writing data into the transmit FIFO, the shift logic does not begin the serial transfer until a positive edge of the `sclk_out` signal is present. The delay in waiting for this positive edge depends on the baud rate of the serial transfer. Before polling the `SR.BUSY` status, you should first poll the `SR.TFE` status (waiting for 1) or wait for `BAUDR.SCKDV * spi*_clk` clock cycles.

Master SPI Serial Transfers

When the transfer mode is **Transmit And Receive** or **Transmit Only** (`TMOD = 0` or `TMOD = 1`, respectively), transfers are terminated by the shift control logic when the transmit FIFO is empty. For continuous data transfers, you must ensure that the transmit FIFO buffer does not become empty

before all the data have been transmitted. The Transmit FIFO Threshold Level Register (`TXFTLR`) can be used to early interrupt (`spi*_txe_intr`) the processor indicating that the transmit FIFO buffer is nearly empty. When a DMA is used for the Register Interface accesses, the `DMATDLR.DMATDL` can be used to early request the DMA Controller, indicating that the transmit FIFO is nearly empty. The FIFO can then be refilled with data to continue the serial transfer. You may also write a block of data (at least two FIFO entries) into the transmit FIFO before enabling a serial slave. This ensures that serial transmission does not begin until the number of data-frames that make up the continuous transfer are present in the transmit FIFO.

In order to make sure the data is present before the transfer starts, the data register (`DRi`) could be programmed earlier and then `SER` should be programmed. This ensures that all the required data is present in the FIFO before the transfer starts on the SPI interface.

When the transfer mode is **Receive Only** (`TMOD = 2`), a serial transfer is started by writing one "dummy" data word into the transmit FIFO when a serial slave is selected. The data output from the QSPI Controller is held at a constant logic level for the duration of the serial transfer. The transmit FIFO is popped only once at the beginning and may remain empty for the duration of the serial transfer. The end of the serial transfer is controlled by the `NDF` field in Control Register 1 (`CTRLR1`).

This transfer mode increases the bandwidth of the Register Interface bus as the transmit FIFO never needs to be serviced during the transfer. The receive FIFO buffer should be read each time the receive FIFO generates a FIFO full interrupt request to prevent an overflow.

When the transfer mode is **EEPROM Read** (`TMOD = 3`), a serial transfer is started by writing the opcode and/or address into the transmit FIFO when a serial slave (EEPROM) is selected. The opcode and address are transmitted to the EEPROM device, after which read data is received from the EEPROM device and stored in the receive FIFO. The end of the serial transfer is controlled by the `NDF` field in the Control Register 1 (`CTRLR1`).

The Receive FIFO Threshold Level Register (`RXFTLR`) can be used to give early indication that the receive FIFO is nearly full.

For details on the mentioned above transfer modes, see Transfer Modes section.

A typical software flow for completing an SPI serial transfer from the QSPI/SPI Master controller is outlined as follows:

1. If the QSPI/SPI Master controller is enabled, disable it by writing 0 to the SPI Enable Register (`SPIENR`).
2. Set up the QSPI/SPI Master controller control registers for the transfer; these registers can be set in any order.
    - Write Control Register 0 (`CTRLR0`). For SPI transfers, the serial clock polarity and serial clock phase parameters must be set identical to target slave device.
    - If the transfer mode is **Receive Only**, write Control Register 1 (`CTRLR1`) with the number of frames in the transfer minus 1; for example, if you want to receive four data frames, write this register with 3.
    - Write the Baud Rate Select Register (`BAUDR`) to set the baud rate for the transfer.
    - Write the Transmit FIFO Threshold Level Register (`TXFTLR`) and Receive FIFO Threshold Level Register (`RXFTLR`) to set FIFO threshold levels.
    - Write the `IMR` register to set up interrupt masks.

- The Slave Enable Register (SER) register can be written here to enable the target slave for selection. If a slave is enabled here, the transfer begins as soon as one valid data entry is present in the transmit FIFO. If no slaves are enabled prior to writing to the Data Registers (DRi), the transfer does not begin until a slave is enabled.

3. Enable the QSPI/SPI Master controller by writing 1 to the SPIENR register.
4. Write data for transmission to the target slave into the transmit FIFO (write DRi).

   If no slaves were enabled in the SER register at this point, enable it now to begin the transfer.
5. Poll the BUSY status to wait for completion of the transfer. The BUSY status cannot be polled immediately; for more information, see the note in **Data Transfers**.

   If a transmit FIFO empty interrupt request is made, write the transmit FIFO (write DRi). If a receive FIFO full interrupt request is made, read the receive FIFO (read DRi).
6. The transfer is stopped by the shift control logic when the transmit FIFO is empty. If the transfer mode is **Receive Only** (TMOD = 2), the transfer is stopped by the shift control logic when the specified number of frames have been received. When the transfer is done, the BUSY status is reset to 0.
7. If the transfer mode is not **Transmit Only** (TMOD != 1), read the receive FIFO until it is empty.
8. Disable the QSPI/SPI Master controller by writing 0 to SPIENR.

13.4.1.7. Serial-Slave Operation Mode (SPI Slave only)

This mode is available only for the SPI Slave controller.

This mode enables serial communication with master peripheral devices. Since the SPI Slave Controller is configured as a serial-slave device, all serial transfers are initiated and controlled by the serial bus master.

All data transfers to and from the serial-slave are regulated on the serial clock line (sclk_in), driven from the serial-master device. Data are propagated from the serial-slave on one edge of the serial clock line and sampled on the opposite edge.

When the SPI Slave is not selected, it must not interfere with data transfers between the serial-master and other serial-slave devices. When the SPI Slave is not selected, its data output is buffered, resulting in a high impedance drive onto the serial master data input line.

The serial clock that regulates the data transfer is generated by the serial-master device and input to the SPI Slave on sclk_in. The slave remains in an idle state until selected by the bus master. When not actively transmitting data, the slave must hold its data output line in a high impedance state to avoid interference with serial transfers to other slave devices. The slave output enable line is available for use to control the data output buffer. The slave continues to transfer data to and from the master device as long as it is selected. If the master transmits to all serial slaves, a Slave Output Enable bit (SLV_OE) in the CTRLR0 register can be programmed to inform the slave if it should respond with data from the its data output line.

Slave SPI Serial Transfers

If the SPI Slave is **Receive Only** (TMOD=2), the transmit FIFO need not contain valid data because the data currently in the transmit shift register is resent each time the slave device is selected. The Transmission Error flag (TXE bit) in the SR register is not set when transfer mode is **Transmit Only** (TMOD=1). You should mask the transmit FIFO empty interrupt (spi*_txe_intr) when this mode is used.

If the SPI Slave transmits data to the master, you must ensure that data exists in the transmit FIFO before a transfer is initiated by the serial-master device. If the master initiates a transfer to the SPI Slave when no data exists in the transmit FIFO, the Transmission Error flag (`TXE` bit) in the `SR` register is set, and the previously transmitted data frame is resent on data output. For continuous data transfers, you must ensure that the transmit FIFO buffer does not become empty before all the data have been transmitted. The `TXFTLR` can be used to early interrupt (`spi*_txe_intr`) the processor, indicating that the transmit FIFO buffer is nearly empty. When a DMA Controller is used for the Register Interface accesses, the `DMATDLR.DMATDL` can be used to early request the DMA Controller, indicating that the transmit FIFO is nearly empty. The FIFO can then be refilled with data to continue the serial transfer. The receive FIFO buffer should be read each time the receive FIFO full interrupt (`spi*_rxf_intr`) request is generated to prevent an overflow. The `RXFTLR` register can be used to give early indication that the receive FIFO is nearly full. When a DMA Controller is used for the Register Interface accesses, the `DMARDRL.DMARDL` can be used to early request the DMA controller, indicating that the receive FIFO is nearly full.

A typical software flow for completing an SPI serial transfer from a serial-master to the SPI Slave controller is outlined as follows:

1. If the SPI Slave controller is enabled, disable it by writing 0 to the SPI Enable Register (`SPIENR`).
2. Set up the SPI Slave controller control registers for the transfer; these registers can be set in any order.
    - Write Control Register 0 (`CTRLR0`). For SPI transfers, the serial clock polarity and serial clock phase parameters must be set identical to the master device.
    - Write the Transmit FIFO Threshold Level Register (`TXFTLR`) and Receive FIFO Threshold Level Register (`RXFTLR`) to set FIFO threshold levels.
    - Write the `IMR` register to set up interrupt masks.
3. Enable the SPI Slave controller by writing 1 to the `SPIENR` register.
4. If the transfer mode is **Transmit and Receive** (`TMOD=0`) or **Transmit Only** (`TMOD=1`), write data for transmission to the master into the transmit FIFO (write `DRi`).

   If the transfer mode is **Receive Only** (`TMOD=2`), there is no need to write data into the transmit FIFO; the current value in the transmit shift register is retransmitted.
5. The SPI Slave slave is now ready for the serial transfer. The transfer begins when the SPI Slave controller is selected by a serial-master device.
6. When the transfer is underway, the BUSY status can be polled to return the transfer status. If a transmit FIFO empty interrupt request is made, write the transmit FIFO (write `DRi`). If a receive FIFO full interrupt request is made, read the receive FIFO (read `DRi`).
7. The transfer ends when the serial master removes the select input to the SPI Slave controller. When the transfer is completed, the BUSY status is reset to 0.
8. If the transfer mode is not **Transmit Only** (`TMOD != 1`), read the receive FIFO until empty.
9. Disable the SPI Slave controller by writing 0 to `SPIENR`.

13.4.1.8. Interrupts

Each of the *SPI Controllers generates a combined interrupt that is the logical OR of the following individual interrupts:

- Transmit FIFO empty interrupt (`spi*_txe_intr`)
- Transmit FIFO overflow interrupt (`spi*_txo_intr`)

- Receive FIFO full interrupt (`spi*_rxf_intr`)
- Receive FIFO overflow interrupt (`spi*_rxo_intr`)
- Receive FIFO underflow interrupt (`spi*_rxu_intr`)
- Multi-master contention interrupt (`spi*_mst_intr`)

The *SPI Controller contains the following registers to manage the listed individual interrupts:

- Interrupt Mask Register (`IMR`)
- Interrupt Status Register (`ISR`)
- Raw Interrupt Status Register (`RISR`)
- Transmit FIFO Overflow Interrupt Clear Register (`TXOICR`)
- Receive FIFO Overflow Interrupt Clear Register (`RXOICR`)
- Receive FIFO Underflow Interrupt Clear Register (`RXUICR`)
- Multi-Master Interrupt Clear Register (`MSTICR`)
- Interrupt Clear Register (`ICR`)

The *SPI Controller interrupts are described as follows:

- **Transmit FIFO Empty Interrupt** (`spi*_txe_intr`) – Set when the transmit FIFO is equal to or below its threshold value and requires service to prevent an under-run. The threshold value, set through the Transmit FIFO Threshold Level Register (`TXFTLR`), determines the level of transmit FIFO entries at which an interrupt is generated. This interrupt is cleared by hardware when data are written into the transmit FIFO buffer, bringing it over the threshold level.
- **Transmit FIFO Overflow Interrupt** (`spi*_txo_intr`) – Set when a Register Interface access attempts to write into the transmit FIFO after it has been completely filled. When set, data written from the Register Interface is discarded. This interrupt remains set until you read the Transmit FIFO Overflow Interrupt Clear Register (`TXOICR`).
- **Receive FIFO Full Interrupt** (`spi*_rxf_intr`) – Set when the receive FIFO is equal to or above its threshold value plus 1 and requires service to prevent an overflow. The threshold value, set through the Receive FIFO Threshold Level Register (`RXFTLR`), determines the level of receive FIFO entries at which an interrupt is generated. This interrupt is cleared by hardware when data are read from the receive FIFO buffer, bringing it below the threshold level.
- **Receive FIFO Overflow Interrupt** (`spi*_rxo_intr`) – Set when the receive logic attempts to place data into the receive FIFO after it has been completely filled. When set, newly received data is discarded. This interrupt remains set until you read the Receive FIFO Overflow Interrupt Clear Register (`RXOICR`).
- **Receive FIFO Underflow Interrupt** (`spi*_rxu_intr`) – Set when a Register Interface access attempts to read from the receive FIFO when it is empty. When set, zeros are read back from the receive FIFO. This interrupt remains set until you read the Receive FIFO Underflow Interrupt Clear Register (`RXUICR`).
- **Multi-Master Contention Interrupt** (`spi*_mst_intr`) – Present only when the QSPI/SPI Master controller component is configured as a serial-master device. The interrupt is set when another serial master on the serial bus selects the QSPI/SPI Master Controller master as a serial-slave device and is actively transferring data. This informs the processor of possible contention on the serial bus. This interrupt remains set until you read the Multi-Master Interrupt Clear Register (`MSTICR`).
- **Combined Interrupt Request** (`spi*_intr`) – OR'ed result of all the above interrupt requests after masking. To mask this interrupt signal, you must mask all other *SPI Controller interrupt requests.

13.4.2. *SPI Registers

The register regions of the *SPI Controllers are mapped in the BE-U1000 microcontroller memory map as the following table shows.

Table 13-94 Memory address regions for *SPI Controller registers

Region	Block Name	Size	Start Address	End Address
Periphery 0	QSPI_0	4 KB	0x1100_0000	0x1100_0FFF
	SPI_0	4 KB	0x1100_B000	0x1100_BFFF
	SPI_1	4 KB	0x1100_C000	0x1100_CFFF
Periphery 1	QSPI_1	4 KB	0x1300_0000	0x1300_0FFF
	SPI_2	4 KB	0x1300_B000	0x1300_BFFF
	SPI_3	4 KB	0x1300_C000	0x1300_CFFF

For details, refer to Memory Map chapter.

This section covers the following topics:

- Register Map
- Register Descriptions

13.4.2.1. Register Map

In the tables below, the offset of a register is relative to the *SPI controller memory region start address. The following table provides a top-level summary of each *SPI Controller register. To view the detailed description of a register, click the register name in the **Description** column.

Table 13-95 *SPI register map

Register	Offset	Description	Default Value
CTRLR0	0x00	Control Register 0	0x1070000
CTRLR1	0x04	Control Register 1 **Exists**:QSPI/SPI Master	0x0
SPIENR	0x08	SPI Enable Register	0x0
MWCR	0xC	Microwire Control Register	0x0
SER	0x10	Slave Enable Register **Exists**:QSPI/SPI Master	0x0
BAUDR	0x14	Baud Rate Select Register **Exists**:QSPI/SPI Master	0x0
TXFTLR	0x18	Transmit FIFO Threshold Level Register	0x0
RXFTLR	0x1C	Receive FIFO Threshold Level Register	0x0
TXFLR	0x20	Transmit FIFO Level Register	0x0

Table 13-95 *SPI register map (continued)

Register	Offset	Description	Default Value
RXFLR	0x24	Receive FIFO Level Register	0x0
SR	0x28	Status Register	0x6
IMR	0x2C	Interrupt Mask Register	QSPI: 0x3F SPI Master: 0x3F SPI Slave: 0x1F
ISR	0x30	Interrupt Status Register	0x0
RISR	0x34	Raw Interrupt Status Register	0x0
TXOICR	0x38	Transmit FIFO Overflow Interrupt Clear Register	0x0
RXOICR	0x3C	Receive FIFO Overflow Interrupt Clear Register	0x0
RXUICR	0x40	Receive FIFO Underflow Interrupt Clear Register	0x0
MSTICR	0x44	Multi-Master Interrupt Clear Register	0x0
ICR	0x48	Interrupt Clear Register	0x0
DMACR	0x4C	DMA Control Register	0x0
DMATDLR	0x50	DMA Transmit Data Level	0x0
DMARDLR	0x54	DMA Receive Data Level	0x0
DRi (for i=0; i<=35)	0x60 +(i *0x4)	Data Registers	0x0
RX_SAMPLE_DLY	0xF0	RX Sample Delay Register	0x0
SPI_CTRLR0	0xF4	SPI Control Register **Exists**: QSPI	0x200
TXD_DRIVE_EDGE	0xF8	Transmit Drive Edge Register **Exists**: QSPI	0x0
XIP_CMD	0xFC	XIP Command Register **Exists**: QSPI	0x0

13.4.2.2. Register Descriptions

In the tables below, the **R/W** column of a register description specifies how the application and the core can access the register fields.

13.4.2.2.1. Control Register 0 (CTRLR0)

The CTRLR0 register is at offset 0x0.

This register controls the serial data transfer. It is impossible to write to this register when the *SPI Controller is enabled. The *SPI Controller is enabled and disabled by writing to the SPIENR register.

The following table shows the register bit assignments.

Table 13-96 Fields for register: `CTRLR0`

Bits	Name	R/W	Description
[31:26]			Reserved
[25]	`SECONV`	R/W	This bit field is only available for SPI Master and QSPI controllers. Set the Endianness for XIP and data register reads. **Values:** **0x0**: Endian Conversion disabled **0x1**: Endian Conversion enabled **Value After Reset:** 0x0
[24]	`SSTE`	R/W	**Slave Select Toggle Enable** When operating in SPI mode with clock phase (`SCPH`) set to 0, this register controls the behavior of the slave select line (`ss_*_n`) between data frames. If this register field is set to 1 the `ss_*_n` line will toggle between consecutive data frames, with the serial clock being held to its default value while `ss_*_n` is high; if this register field is set to 0 the `ss_*_n` will stay low and serial clock will run continuously for the duration of the transfer. **Value After Reset:** 0x1
[23]			Reserved
[22:21]	`SPI_FRF`	*Varies	**SPI Frame Format** Selects data frame format for Transmitting/Receiving the data Bits. **Values:** **0x0:** (`STD_SPI_FRF`): Standard SPI Frame Format **0x1:** (`DUAL_SPI_FRF`): Dual SPI Frame Format **0x2:** (`QUAD_SPI_FRF`): Quad SPI Frame Format **0x3:** Reserved The access attributes of this field are the following: **QSPI**: R/W **SPI Slave/SPI Master**: R **Value After Reset:** 0x0
[20:16]	`DFS_32`	R/W	**Data Frame Size** in 32-bit transfer size mode. **Values**: **0x7:** 8-bit serial data transfer **Value After Reset:** 0x7
[15:12]	`CFS`	R/W	**Control Frame Size** Selects the length of the control word for the Microwire frame format. **Values:** **0x0:** 1-bit Control Word **0x1:** 2-bit Control Word **0x2:** 3-bit Control Word **0x3:** 4-bit Control Word **0x4:** 5-bit Control Word

Table 13-96 Fields for register: `CTRLR0` (continued)

Bits	Name	R/W	Description
			0x5: 6-bit Control Word **0x6:** 7-bit Control Word **0x7:** 8-bit Control Word **0x8:** 9-bit Control Word **0x9:** 10-bit Control Word **0xA:** 11-bit Control Word **0xB:** 12-bit Control Word **0xC:** 13-bit Control Word **0xD:** 14-bit Control Word **0xE:** 15-bit Control Word **0xF:** 16-bit Control Word **Value After Reset:** 0x0
[11]	`SRL`	R/W	**Shift Register Loop** Used for testing purposes only. When internally active, connects the transmit shift register output to the receive shift register input. Can be used in both serial-slave and serial-master modes. When the *SPI controller is configured as a slave in loopback mode, the `ss_in_n` and `spi*_clk` signals must be provided by an external source. In this mode, the slave cannot generate these signals because there is nothing to which to loop back. **Values:** **0x0**: Normal mode operation **0x1**: Test mode: Tx and Rx shift reg connected **Value After Reset:** 0x0
[10]	`SLV_OE`	*Varies	**Slave Output Enable** This bit enables or disables the setting of the slave output enable line from the SPI Slave. When `SLV_OE` = 0x1, the slave output enable line can never be active. When slave output enable line controls the tri-state buffer on the data output from the slave, a high impedance state is always present on the slave data output when `SLV_OE` = 0x1. This is useful when the master transmits in broadcast mode (master transmits data to all slave devices). Only one slave may respond with data on the master data input line. This bit is enabled (set to 0x0) after reset and must be disabled (set to 0x1) by software, if you do not want this device to respond with data. **Values:** **0x0**: Slave Output is enabled **0x1**: Slave Output is disabled The access attributes of this field are the following: **QSPI/SPI Master**: R **SPI Slave**: R/W **Value After Reset:** 0x0
[9:8]	`TMOD`	R/W	**Transfer Mode**

Table 13-96 Fields for register: `CTRLR0` (continued)

Bits	Name	R/W	Description
			Selects the mode of transfer for serial communication. This field does not affect the transfer duplicity. Only indicates if the receive or transmit data is valid. In transmit-only mode, data received from the external device is not valid and is not stored in the receive FIFO memory; it is overwritten on the next transfer. In receive-only mode, transmitted data are not valid. After the first write to the transmit FIFO, the same word is retransmitted for the duration of the transfer. In transmit-and-receive mode, both transmit and receive data are valid. The transfer continues until the transmit FIFO is empty. Data received from the external device are stored into the receive FIFO memory, where it can be accessed by the host processor. In EEPROM-read mode, receive data is not valid while control data is being transmitted. When all control data is sent to the EEPROM, receive data becomes valid and transmit data becomes invalid. All data in the transmit FIFO is considered control data in this mode. This transfer mode is only valid when the QSPI controller is configured as a master device. **Values for `SPI_FRF=0`:** **0x0** (`TX_AND_RX`): Transmit and receive **0x1** (`TX_ONLY`): Transmit only mode or Write (`SPI_FRF != 0`) **0x2** (`RX_ONLY`): Receive only mode or Read (`SPI_FRF != 0`) **0x3** (`EEPROM_READ`): EEPROM Read mode **Values for `SPI_FRF != 0`:** **0x1** (`TX_ONLY`): Write **0x2** (`RX_ONLY`): Read **Value After Reset:** 0x0
[7]	`SCPOL`	R/W	**Serial Clock Polarity** Valid when the frame format (`FRF`) is set to 0x0 (SPI Frame Format). Used to select the polarity of the inactive serial clock, which is held inactive when the *SPI controller master is not actively transferring data on the serial bus. **Values:** **0x0** (`SCLK_LOW`): Inactive state of serial clock is low **0x1** (`SCLK_HIGH`): Inactive state of serial clock is high **Value After Reset:** 0x0
[6]	`SCPH`	R/W	**Serial Clock Phase** Valid when the frame format (`FRF`) is set to 0x0 (SPI Frame Format). The serial clock phase selects the relationship of the serial clock with the slave select signal. When `SCPH = 0`, data are captured on the first edge of the serial clock. When `SCPH = 1`, the serial clock starts toggling one cycle after the slave select line is activated, and data are captured on the second edge of the serial clock. **Values:** **0x0:** (`SCPH_MIDDLE`): Serial clock toggles in the middle of the first data bit **0x1:** (`SCPH_START`): Serial clock toggles at the start of the first data bit **Value After Reset:** 0x0

Table 13-96 Fields for register: CTRLR0 (continued)

Bits	Name	R/W	Description
[5:4]	FRF	R/W	**Frame Format** Selects which serial protocol transfers the data. **Values:** **0x0**: SPI Frame Format **0x1**: SSP Frame Format **0x2**: Microwire Frame Format **0x3**: Reserved **Value After Reset:** 0x0
[3:0]			Reserved

13.4.2.2.2. Control Register 1 (CTRLR1)

This register is available only for QSPI and SPI Master.

The CTRLR1 register is at offset 0x4.

This register exists only when the *SPI Controller is configured as a master device. Control register 1 controls the end of serial transfers when in receive-only mode. It is impossible to write to this register when the *SPI Controller is enabled. The *SPI Controller is enabled and disabled by writing to the SPIENR register.

The following table shows the register bit assignments.

Table 13-97 Fields for register: CTRLR1

Bits	Name	R/W	Description
[31:16]			Reserved
[15:0]	NDF	R/W	**Number of Data Frames** When TMOD bit of CTRLR0 register is 0x2 or TMOD bit of CTRLR0 register is 0x3 , this register field sets the number of data frames to be continuously received by the *SPI Controller. The *SPI Controller continues to receive serial data until the number of data frames received is equal to this register value plus 1, which enables you to receive up to 64 KB of data in a continuous transfer. **Value After Reset:** 0x0

13.4.2.2.3. SPI Enable Register (SPIENR)

The SPIENR register is at offset 0x8.

This register enables and disables the *SPI Controller.

The following table shows the register bit assignments.

Table 13-98 Fields for register: SPIENR

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	SPI_EN	R/W	***SPI Enable**

Table 13-98 Fields for register: `SPIENR` (continued)

Bits	Name	R/W	Description
			Enables and disables all *SPI Controller operations. When disabled, all serial transfers are halted immediately. Transmit and receive FIFO buffers are cleared when the device is disabled. It is impossible to program some of the *SPI Controller control registers when enabled. **Values:** **0x0**: Disables Serial Transfer **0x1**: Enables Serial Transfer **Value After Reset:** 0x0

13.4.2.2.4. Microwire Control Register (`MWCR`)

The `MWCR` register is at offset 0xC.

This register controls the direction of the data word for the half-duplex Microwire serial protocol. It is impossible to write to this register when the *SPI is enabled. The *SPI is enabled and disabled by writing to the `SPIENR` register.

Table 13-99 Fields for register: `MWCR`

Bits	Name	R/W	Description
[31:3]			Reserved
[2]	`MHS`	R/W	**Microwire Handshaking.** This bit field is only available for QSPI and SPI Master controllers. Relevant only when the *SPI is configured as a serial-master device. When configured as a serial slave, this bit field has no functionality. Used to enable and disable the busy/ready handshaking interface for the Microwire protocol. When enabled, the *SPI checks for a ready status from the target slave, after the transfer of the last data/control bit, before clearing the BUSY status in the `SR` register. **Values:** **0x0:** Handshaking interface is disabled **0x1:** Handshaking interface is enabled **Value After Reset:** 0x0
[1]	`MDD`	R/W	**Microwire Control.** Defines the direction of the data word when the Microwire serial protocol is used. When this bit is set to 0, the data word is received by the *SPI MacroCell from the external serial device. When this bit is set to 1, the data word is transmitted from the *SPI MacroCell to the external serial device. **Values:** **0x0:** *SPI receives data **0x1:** *SPI transmits data **Value After Reset:** 0x0
[0]	`MWMOD`	R/W	**Microwire Transfer Mode.** Defines whether the Microwire transfer is sequential or non-sequential. When sequential mode is used, only one control word is needed to transmit or receive

Table 13-99 Fields for register: MWCR (continued)

Bits	Name	R/W	Description
			a block of data words. When non-sequential mode is used, there must be a control word for each data word that is transmitted or received. **Values:** **0x0:** Non-Sequential Microwire Transfer **0x1:** Sequential Microwire Transfer **Value After Reset:** 0x0

13.4.2.2.5. Slave Enable Register (SER)

This register is available only for QSPI and SPI Master controllers.

The SER register is at offset 0x10.

The register enables the individual slave select output lines from the *SPI Controller master. Three slave-select output pins are available on the QSPI Controller and one slave-select output pin is available on the SPI Master controller. Register bits can be set or cleared when SPIENR.SPI_EN=0.

If SPIENR.SPI_EN=1, then register bits can be set (to delay the slave select assertion while Tx FIFO is getting filled) but cannot be cleared.

The following table shows the register bit assignments.

Table 13-100 Fields for register: SER

Bits	Name	R/W	Description
[31:3]			Reserved
[2:0]	SER	R/W	**Slave Select Enable Flag** Each bit in this field corresponds to a slave select line (ss_x_n) from the *SPI Controller master, where x is the bit number. When a bit in this register is set (1), the corresponding slave select line from the master is activated when a serial transfer begins. It should be noted that setting or clearing bits in this register has no effect on the corresponding slave select outputs until a transfer is started. Before beginning a transfer, you should enable the bit in this field that corresponds to the slave device with which the master wants to communicate. When not operating in broadcast mode, only one bit in this field should be set. **Values:** **0x0**: No slave selected **0x1**: Slave is selected Since the SPI Master controller has only one slave select line (ss_0_n), only the SER[0] bit is available to it. **Value After Reset:** 0x0

13.4.2.2.6. Baud Rate Select Register (BAUDR)

This register is available only for QSPI and SPI Master controllers.

The BAUDR register is at offset 0x14.

The register derives the frequency of the serial clock that regulates the data transfer. The 16-bit field in this register defines the spi*_clk divider value. It is impossible to write to this register when the *SPI Controller is enabled. The *SPI Controller is enabled and disabled by writing to the SPIENR register.

The following table shows the register bit assignments.

Table 13-101 Fields for register: BAUDR

Bits	Name	R/W	Description
[31:16]			Reserved
[15:0]	SCKDV	R/W	***SPI Clock Divider** The LSB for this field is always set to 0 and is unaffected by a write operation, which ensures an even value is held in this register. If the value is 0, the serial output clock (sclk_out) is disabled. The frequency of the sclk_out is derived from the following equation: $F_{sclk\_out} = F_{spi*\_clk} / SCKDV$ where SCKDV is any even value between 2 and 65534. For example: for F_{spi*_clk} = 3.6864 MHz and SCKDV =2 F_{sclk_out} = 3.6864/2 = 1.8432 MHz. **Value After Reset:** 0x0

13.4.2.2.7. Transmit FIFO Threshold Level Register (TXFTLR)

The TXFTLR register is at offset 0x18.

This register controls the threshold value for the transmit FIFO memory. The *SPI Controller is enabled and disabled by writing to the SPIENR register.

The following table shows the TXFTLR register bit assignments.

Table 13-102 Fields for register: TXFTLR

Bits	Name	R/W	Description
[31:3]			Reserved
[2:0]	TFT	R/W	**Transmit FIFO Threshold** Controls the level of entries (or below) at which the transmit FIFO controller triggers an interrupt. The FIFO buffer depth is 8; this register is sized to the number of address bits needed to access the FIFO. If you attempt to set this value greater than or equal to the depth of the FIFO, this field is not written and retains its current value. When the number of transmit FIFO entries is less than or equal to this value, the transmit FIFO empty interrupt is triggered. spi*_txe_intr is asserted when TFT or less data entries are present in transmit FIFO. **Value After Reset:** 0x0

13.4.2.2.8. Receive FIFO Threshold Level Register (RXFTLR)

The RXFTLR register is at offset 0x1C.

This register controls the threshold value for the receive FIFO memory. The *SPI Controller is enabled and disabled by writing to the SPIENR register.

The following table shows the register bit assignments.

Table 13-103 Fields for register: RXFTLR

Bits	Name	R/W	Description
[31:3]			Reserved
[2:0]	RFT	R/W	**Receive FIFO Threshold** Controls the level of entries (or above) at which the receive FIFO controller triggers an interrupt. The FIFO buffer depth is 8. This register is sized to the number of address bits needed to access the FIFO. If you attempt to set this value greater than the depth of the FIFO, this field is not written and retains its current value. When the number of receive FIFO entries is greater than or equal to this value + 1, the receive FIFO full interrupt is triggered. spi*_rxf_intr is asserted when RFT or more data entries are present in receive FIFO. **Value After Reset:** 0x0

13.4.2.2.9. Transmit FIFO Level Register (TXFLR)

The TXFLR register is at offset 0x20.

This register contains the number of valid data entries in the transmit FIFO memory.

The following table shows the register bit assignments.

Table 13-104 Fields for register: TXFLR

Bits	Name	R/W	Description
[31:4]			Reserved
[3:0]	TXTFL	R	**Transmit FIFO Level** Contains the number of valid data entries in the transmit FIFO. **Value After Reset:** 0x0 **Volatile:** true

13.4.2.2.10. Receive FIFO Level Register (RXFLR)

The RXFLR register is at offset 0x24.

This register contains the number of valid data entries in the receive FIFO memory. This register can be ready at any time.

The following table shows the register bit assignments.

Table 13-105 Fields for register: RXFLR

Bits	Name	R/W	Description
[31:4]			Reserved
[3:0]	RXTFL	R	**Receive FIFO Level** Contains the number of valid data entries in the receive FIFO. **Value After Reset:** 0x0 **Volatile:** true

13.4.2.2.11. Status Register (SR)

The SR register is at offset 0x28.

This is a read-only register used to indicate the current transfer status, FIFO status, and any transmission/reception errors that may have occurred. The status register may be read at any time. None of the bits in this register request an interrupt.

The following table shows the register bit assignments.

Table 13-106 Fields for register: SR

Bits	Name	R/W	Description
[31:7]			Reserved
[6]	DCOL	R	**Data Collision Error** This bit field is available only for QSPI and SPI Master controllers. This bit will be set if ss_in_n input is asserted by other master, when the *SPI controller master is in the middle of the transfer. This informs the processor that the last data transfer was halted before completion. This bit is cleared when read. **Values:** **0x0**: No Error **0x1**: Transmit Data Collision Error **Value After Reset:** 0x0 **Volatile:** true
[5]	TXE	R	**Transmission Error** This bit field is available only for SPI Slave controller. Set if the transmit FIFO is empty when a transfer is started. This bit can be set only when the *SPI is configured as a slave device. Data from the previous transmission is resent on the data output line. This bit is cleared when read. **Values:** **0x0**: No Error **0x1**: Transmission Error **Value After Reset:** 0x0 **Volatile:** true
[4]	RFF	R	**Receive FIFO Full** When the receive FIFO is completely full, this bit is set. When the receive FIFO contains one or more empty location, this bit is cleared. **Values:** **0x0**: Receive FIFO is not full **0x1**: Receive FIFO is full **Value After Reset:** 0x0 **Volatile:** true
[3]	RFNE	R	**Receive FIFO Not Empty** Set when the receive FIFO contains one or more entries and is cleared when the receive FIFO is empty. This bit can be polled by software to completely empty the receive FIFO. **Values:** **0x0**: Receive FIFO is empty **0x1**: Receive FIFO is not empty

Table 13-106 Fields for register: `SR` (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0 **Volatile:** true
[2]	`TFE`	R	**Transmit FIFO Empty** When the transmit FIFO is completely empty, this bit is set. When the transmit FIFO contains one or more valid entries, this bit is cleared. This bit field does not request an interrupt. **Values:** **0x0**: Transmit FIFO is not empty **0x1**: Transmit FIFO is empty **Value After Reset:** 0x1 **Volatile:** true
[1]	`TFNF`	R	**Transmit FIFO Not Full** Set when the transmit FIFO contains one or more empty locations, and is cleared when the FIFO is full. **Values:** **0x0**: Transmit FIFO is full **0x1**: Transmit FIFO is not full **Value After Reset:** 0x1 **Volatile:** true
[0]	`BUSY`	R	***SPI Busy Flag** When set, indicates that a serial transfer is in progress; when cleared indicates that the *SPI controller is idle or disabled. **Values:** **0x0**: *SPI controller is idle or disabled **0x1**: *SPI controller is actively transferring data **Value After Reset:** 0x0 **Volatile:** true

13.4.2.2.12. Interrupt Mask Register (`IMR`)

The `IMR` register is at offset 0x2C.

This read/write reigster masks or enables all interrupts generated by the *SPI controller.

The following table shows the register bit assignments.

Table 13-107 Fields for register: `IMR`

Bits	Name	R/W	Description
[31:6]			Reserved
[5]	`MSTIM`	R/W	This bit is available only for QSPI and SPI Master controllers. Multi-Master Contention Interrupt Mask **Values:** **0x0**: `spi*_mst_intr` interrupt is masked **0x1**: `spi*_mst_intr` interrupt is not masked

Table 13-107 Fields for register: `IMR` (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x1
[4]	`RXFIM`	R/W	Receive FIFO Full Interrupt Mask **Values:** **0x0**: `spi*_rxf_intr` interrupt is masked **0x1**: `spi*_rxf_intr` interrupt is not masked **Value After Reset:** 0x1
[3]	`RXOIM`	R/W	Receive FIFO Overflow Interrupt Mask **Values:** **0x0**: `spi*_rxo_intr` interrupt is masked **0x1**: `spi*_rxo_intr` interrupt is not masked **Value After Reset:** 0x1
[2]	`RXUIM`	R/W	Receive FIFO Underflow Interrupt Mask **Values:** **0x0**: `spi*_rxu_intr` interrupt is masked **0x1**: `spi*_rxu_intr` interrupt is not masked **Value After Reset:** 0x1
[1]	`TXOIM`	R/W	Transmit FIFO Overflow Interrupt Mask **Values:** **0x0**: `spi_*_txo_intr` interrupt is masked **0x1**: `spi_*_txo_intr` interrupt is not masked **Value After Reset:** 0x1
[0]	`TXEIM`	R/W	Transmit FIFO Empty Interrupt Mask **Values:** **0x0**: `spi*_txe_intr` interrupt is masked **0x1**: `spi*_txe_intr` interrupt is not masked **Value After Reset:** 0x1

13.4.2.2.13. Interrupt Status Register (`ISR`)

The `ISR` register is at offset 0x30.

This register reports the status of the *SPI controller interrupts after they have been masked.

The following table shows the register bit assignments.

Table 13-108 Fields for register: `ISR`

Bits	Name	R/W	Description
[31:6]			Reserved
[5]	`MSTIS`	R	This bit is available only for QSPI and SPI Master controllers. Multi-Master Contention Interrupt Status **Values:**

Table 13-108 Fields for register: `ISR` (continued)

Bits	Name	R/W	Description
			0x0: `spi*_mst_intr` interrupt is not active after masking **0x1**: `spi*_mst_intr` interrupt is active after masking **Value After Reset:** 0x0 **Volatile:** true
[4]	`RXFIS`	R	Receive FIFO Full Interrupt Status **Values:** **0x0**: `spi*_rxf_intr` interrupt is not active after masking **0x1**: `spi*_rxf_intr` interrupt is active after masking **Value After Reset:** 0x0 **Volatile:** true
[3]	`RXOIS`	R	Receive FIFO Overflow Interrupt Status **Values:** **0x0**: `spi*_rxo_intr` interrupt is not active after masking **0x1**: `spi*_rxo_intr` interrupt is active after masking **Value After Reset:** 0x0 **Volatile:** true
[2]	`RXUIS`	R	Receive FIFO Underflow Interrupt Status **Values:** **0x0**: `spi*_rxu_intr` interrupt is not active after masking **0x1**: `spi*_rxu_intr` interrupt is active after masking **Value After Reset:** 0x0 **Volatile:** true
[1]	`TXOIS`	R	Transmit FIFO Overflow Interrupt Status **Values:** **0x0**: `spi*_txo_intr` interrupt is not active after masking **0x1**: `spi*_txo_intr` interrupt is active after masking **Value After Reset:** 0x0 **Volatile:** true
[0]	`TXEIS`	R	Transmit FIFO Empty Interrupt Status **Values:** **0x0**: `spi*_txe_intr` interrupt is not active after masking **0x1**: `spi*_txe_intr` interrupt is active after masking **Value After Reset:** 0x0 **Volatile:** true

13.4.2.2.14. Raw Interrupt Status Register (`RISR`)

The `RISR` register is at offset 0x34.

This read-only register reports the status of the *SPI Controller interrupts prior to masking.

The following table shows the register bit assignments.

Table 13-109 Fields for register: `RISR`

Bits	Name	R/W	Description
[31:6]			Reserved
[5]	`MSTIR`	R	This bit is available only for QSPI and SPI Master controllers. Multi-Master Contention Raw Interrupt Status **Values:** **0x0**: `spi*_mst_intr` interrupt is not active prior to masking **0x1**: `spi*_mst_intr` interrupt is active prior to masking **Value After Reset:** 0x0 **Volatile:** true
[4]	`RXFIR`	R	Receive FIFO Full Raw Interrupt Status **Values:** **0x0**: `spi*_rxf_intr` interrupt is not active prior to masking **0x1**: `spi*_rxf_intr` interrupt is active prior to masking **Value After Reset:** 0x0 **Volatile:** true
[3]	`RXOIR`	R	Receive FIFO Overflow Raw Interrupt Status **Values:** **0x1**: `spi*_rxo_intr` interrupt is not active prior to masking **0x0**: `spi*_rxo_intr` interrupt is active prior to masking **Value After Reset:** 0x0 **Volatile:** true
[2]	`RXUIR`	R	Receive FIFO Underflow Raw Interrupt Status **Values:** **0x0**: `spi*_rxu_intr` interrupt is not active prior to masking **0x1**: `spi*_rxu_intr` interrupt is active prior to masking **Value After Reset:** 0x0 **Volatile:** true
[1]	`TXOIR`	R	Transmit FIFO Overflow Raw Interrupt Status **Values:** **0x0**: `spi*_txo_intr` interrupt is not active prior to masking **0x1**: `spi*_txo_intr` interrupt is active prior to masking **Value After Reset:** 0x0 **Volatile:** true
[0]	`TXEIR`	R	Transmit FIFO Empty Raw Interrupt Status **Values:** **0x0**: `spi*_txe_intr` interrupt is not active prior to masking **0x1**: `spi*_txe_intr` interrupt is active prior to masking **Value After Reset:**0x0 **Volatile:** true

13.4.2.2.15. Transmit FIFO Overflow Interrupt Clear Register (TXOICR)

The TXOICR register is at offset 0x38.

The following table shows the register bit assignments.

Table 13-110 Fields for register: TXOICR

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	TXOICR	R	**Clear Transmit FIFO Overflow Interrupt** This register reflects the status of the interrupt. A read from this register clears the spi*_txo_intr interrupt; writing has no effect. **Value After Reset:** 0x0 **Volatile:** true

13.4.2.2.16. Receive FIFO Overflow Interrupt Clear Register (RXOICR)

The RXOICR register is at offset 0x3C.

The following table shows the register bit assignments.

Table 13-111 Fields for register: RXOICR

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	RXOICR	R	**Clear Receive FIFO Overflow Interrupt** This register reflects the status of the interrupt. A read from this register clears the spi*_rxo_intr interrupt; writing has no effect. **Value After Reset:** 0x0 **Volatile:** true

13.4.2.2.17. Receive FIFO Underflow Interrupt Clear Register (RXUICR)

The RXUICR register is at offset 0x40.

The following table shows the register bit assignments.

Table 13-112 Fields for register: RXUICR

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	RXUICR	R	**Clear Receive FIFO Underflow Interrupt** This register reflects the status of the interrupt. A read from this register clears the spi*_rxu_intr interrupt; writing has no effect. **Value After Reset:** 0x0 **Volatile:** true

13.4.2.2.18. Multi-Master Interrupt Clear Register (MSTICR)

The MSTICR register is at offset 0x44.

The following table shows the register bit assignments.

Table 13-113 Fields for register: `MSTICR`

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	`MSTICR`	R	**Clear Multi-Master Contention Interrupt** This register reflects the status of the interrupt. A read from this register clears the `spi*_mst_intr interrupt`; writing has no effect. **Value After Reset:** 0x0 **Volatile:** true

13.4.2.2.19. Interrupt Clear Register (`ICR`)

The `ICR` register is at offset 0x48.

The following table shows the register bit assignments.

Table 13-114 Fields for register: `ICR`

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	`ICR`	R	**Clear Interrupts** This register is set if any of the interrupts below is active. A read clears the `spi*_txo_intr`, `spi*_rxu_intr`, `spi*_rxo_intr`, and the `spi*_mst_intr` interrupts. Writing to this register has no effect. **Value After Reset:** 0x0 **Volatile:** true

13.4.2.2.20. DMA Control Register (`DMACR`)

The `DMACR` register is at offset 0x4C.

The register is used to enable the DMA Controller interface operation.

The following table shows the register bit assignments.

Table 13-115 Fields for register: `DMACR`

Bits	Name	R/W	Description
[31:2]			Reserved
[1]	`TDMAE`	R/W	**Transmit DMA Enable** This bit enables/disables the transmit FIFO DMA channel. **Values:** **0x0**: Transmit DMA disabled **0x1**: Transmit DMA enabled **Value After Reset:** 0x0
[0]	`RDMAE`	R/W	**Receive DMA Enable** This bit enables/disables the receive FIFO DMA channel **Values:** **0x0**: Receive DMA disabled **0x1**: Receive DMA enabled

Table 13-115 Fields for register: DMACR (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0

13.4.2.2.21. DMA Transmit Data Level Register (DMATDLR)

The DMATDLR register is at offset 0x50.

The following table shows the register bit assignments.

Table 13-116 Fields for register: DMATDLR

Bits	Name	R/W	Description
[31:3]			Reserved
[2:0]	DMATDL	R/W	**Transmit Data Level** This bit field controls the level at which a DMA request is made by the transmit logic. It is equal to the watermark level; that is, the Transmit Request is made when the number of valid data entries in the transmit FIFO is equal to or below this field value, and DMACR.TDMAE = 1. **Value After Reset:** 0x0

13.4.2.2.22. DMA Receive Data Level Register (DMARDLR)

The DMARDLR register is at offset 0x54.

The following table shows the register bit assignments.

Table 13-117 Fields for register: DMARDLR

Bits	Name	R/W	Description
[31:3]			Reserved
[2:0]	DMARDL	R/W	**Receive Data Level.** This bit field controls the level at which a DMA request is made by the receive logic. The watermark level = DMARDL+1; that is, Receive request is made when the number of valid data entries in the receive FIFO is equal to or above this field value + 1, and DMACR.RDMAE=1. **Value After Reset:** 0x0

13.4.2.2.23. Data Register (DRi)

The DRi register is at offset 0x60+ (i * 0x4), where i is from 0 to 35.

The *SPI Controller data register is a 32-bit read/write buffer for the transmit/receive FIFOs. All 32 bits ([31:0]) of the register are valid. When the register is read, data in the receive FIFO buffer is accessed. When it is written to, data are moved into the transmit FIFO buffer; a write can occur only when SPIENR.SPI_EN = 1. FIFOs are reset when SPIENR.SPI_EN = 0.

The DRi register in the *SPI Controller occupies thirty-six 32-bit address locations of the memory map to facilitate burst transfers. Writing to any of these address locations has the same effect as pushing the data from the pwdata bus into the transmit FIFO. Reading from any of these locations has the same effect as popping data from the receive FIFO onto the prdata bus. The FIFO buffers on the *SPI Controller are not addressable.

The following table shows the bit assignments of one DRi register.

Table 13-118 Fields for register: DRi

Bits	Name	R/W	Description
[31:0]	DR	R/W	Data Register When writing to this register, you must right-justify the data. Read data are automatically right-justified. All 32 bits are valid. • Read = Receive FIFO buffer. • Write = Transmit FIFO buffer. **Value After Reset:** 0x0 **Volatile:** true

13.4.2.2.24. Rx Sample Delay Register (RX_SAMPLE_DLY)

This register exists only for QSPI and SPI Master controllers.

The RX_SAMPLE_DLY register is at offset 0xF0.

This register control the number of spi*_clk cycles that are delayed (from the default sample time) before the actual sample of the data input occurs. It is impossible to write to this register when the *SPI Controller is enabled. The *SPI Controller is enabled and disabled by writing to the SPIENR register.

The following table shows the register bit assignments.

Table 13-119 Fields for register: RX_SAMPLE_DLY

Bits	Name	R/W	Description
[31:8]			Reserved
[7:0]	RSD	R/W	**Data Input Sample Delay** This register is used to delay the sample of the data input port. Each value represents a single spi*_clk delay on the sample of data input. If this register is programmed with a value that exceeds the depth of the internal shift registers (4) zero delay will be applied to the data input sample. **Value After Reset:** 0x0

13.4.2.2.25. SPI Control Register (SPI_CTRLR0)

This register exists only for QSPI controller.

The SPI_CTRLR0 register is at offset 0xF4.

This register is used to control the serial data transfer in SPI mode of operation. The register is only relevant when CTRLR0.SPI_FRF is set to either 1 or 2 or 3. It is not possible to write to this register when the QSPI controller is enabled (SPIENR.SPI_EN=1).

The following table shows the register bit assignments.

Table 13-120 Fields for register: `SPI_CTRLR0`

Bits	Name	R/W	Description
[31:19]			Reserved
[18]	`SPI_RXDS_EN`	R	**Read data strobe enable bit** Once this bit is set to 1 QSPI controller will use Read data strobe to capture read data in DDR mode. **Value After Reset:** 0x0
[17]	`INST_DDR_EN`	R/W	**Instruction DDR Enable bit** This will enable dual-data rate transfer for instruction phase. **Value After Reset:** 0x0
[16]	`SPI_DDR_EN`	R/W	**SPI DDR Enable bit** This will enable dual-data rate transfers in Dual/Quad frame formats of SPI. **Value After Reset:** 0x0
[15:11]	`WAIT_CYCLES`	R/W	**Wait cycles** Number of wait cycles in Dual/Quad mode between control frames transmit and data reception. This value is specified as number of SPI clock cycles. **Value After Reset:** 0x0
[10]			Reserved
[9:8]	`INST_L`	R/W	**Instruction Length** Dual/Quad mode instruction length in bits. **Values:** **0x0** (`INST_L_0`): 0-bit (No Instruction) **0x1** (`INST_L_1`): 4-bit Instruction **0x2** (`INST_L_2`): 8-bit Instruction **0x3** (`INST_L_3`): 16-bit Instruction **Value After Reset:** 0x2
[7:6]			Reserved
[5:2]	`ADDR_L`	R/W	**Address Length** This bit defines Length of Address to be transmitted. Only after this much bits are programmed in to the FIFO the transfer can begin. **Values:** **0x0** (`ADDR_L_0`): 0-bit Address Width **0x1** (`ADDR_L_1`): 4-bit Address Width **0x2** (`ADDR_L_2`): 8-bit Address Width **0x3** (`ADDR_L_3`): 12-bit Address Width **0x4** (`ADDR_L_4`): 16-bit Address Width **0x5** (`ADDR_L_5`): 20-bit Address Width **0x6** (`ADDR_L_6`): 24-bit Address Width **0x7** (`ADDR_L_7`): 28-bit Address Width **0x8** (`ADDR_L_8`): 32-bit Address Width **0x9** (`ADDR_L_9`): 36-bit Address Width **0xA** (`ADDR_L_10`): 40-bit Address Width **0xB** (`ADDR_L_11`): 44-bit Address Width

Table 13-120 Fields for register: `SPI_CTRLR0` (continued)

Bits	Name	R/W	Description
			0xC (`ADDR_L_12`): 48-bit Address Width **0xD** (`ADDR_L_13`): 52-bit Address Width **0xE** (`ADDR_L_14`): 56-bit Address Width **0xF** (`ADDR_L_15`): 60-bit Address Width **Value After Reset:** 0x0
[1:0]	`TRANS_TYPE`	R/W	**Address and instruction transfer format** Selects whether QSPI controller will transmit instruction/address either in Standard SPI mode or the SPI mode selected in `CTRLR0.SPI_FRF` field. **Values:** **0x0**: Instruction and Address will be sent in Standard SPI Mode **0x1**: Instruction will be sent in Standard SPI Mode and Address will be sent in the mode specified by `CTRLR0.SPI_FRF` **0x2**: Both Instruction and Address will be sent in the mode specified by `CTRLR0.SPI_FRF` **0x3**: Reserved **Value After Reset:** 0x0

13.4.2.2.26. Transmit Drive Edge Register (`TXD_DRIVE_EDGE`)

This register exists only for QSPI controller.

The `TXD_DRIVE_EDGE` register is at offset 0xF8.

This register is used to control the driving edge of transmit data in DDR mode. It is not possible to write to this register when the QSPI controller is enabled (`SPIENR.SPI_EN=1`).

The following table shows the register bit assignments.

Table 13-121 Fields for register: `TXD_DRIVE_EDGE`

Bits	Name	R/W	Description
[31:8]			Reserved
[7:0]	`TDE`	R/W	*Transmit Data Drive Edge* (**TDE**) — value of which decides the driving edge of transmit data. The maximum value of this register is = `(BAUDR/2) -1`. **Value After Reset:** 0x0

13.4.2.2.27. XIP Command Register (`XIP_CMD`)

This register exists only for QSPI controller.

The `XIP_CMD` register is at offset 0xFC.

It is necessary to add the XIP mode continuation phase after the address phase to operate the XIP mode with different flash memories.

The following table shows the register bit assignments. To use this mode, the microcontroller in XIP mode must be programmed for 4-byte address phases. Flash, in turn, must be programmed for a 3-byte address message. When `XIP_CMD.CMD_EN` = 0x1 and `SYSCR0.XIP*_EN` = 0x1 (where * is the QSPI number):

- Transmit data = `{addr[23:0], cmd_byte_w[7:0]}` (when `XIP_CMD.S2R` = 0x0)
- Transmit data = `{cmd_byte_w[7:0], addr[23:0],}` (when `XIP_CMD.S2R` = 0x1)

Table 13-122 Fields for register: `XIP_CMD`

Bits	Name	R/W	Description
[31:24]	`CMD_BYTE_2`	R/W	Byte 2 **Value After Reset:** 0x0
[23:16]	`CMD_BYTE_1`	R/W	Byte 1 **Value After Reset:** 0x0
[15:8]	`CMD_BYTE_0`	R/W	Byte 0 **Value After Reset:** 0x0
[7:4]			Reserved
[3]	`S2R`	R/W	Place to insert extra byte **Values:** **0x0**: Address phase then extra byte **0x1**: Extra byte then address phase **Value After Reset:** 0x0
[2:1]	`CMD_SEL`	R/W	Selects the contents of `cmd_byte_w[7:0]` **Values:** **0x0**: All zeros **0x1**: From the `CMD_BYTE_0` field **0x2**: From the `CMD_BYTE_1` field **0x3**: From the `CMD_BYTE_2` field **Value After Reset:** 0x0
[0]	`CMD_EN`	R/W	The bit is responsible for changing the address space to the required one with the addition of the required bits to the end of the address phase **Values:** **0x0**: Additions are ignored **0x1**: Additions are used **Value After Reset:** 0x0

13.5. I2C

This section describes the *Inter-Integrated Circuit* (**I2C**) Controllers of the BE-U1000 microcontroller.

The BE-U1000 microcontroller contains four I2C Controllers.

Each I2C Controller consists of a Register Interface, an I2C interface, and FIFO logic that buffers data between the two interfaces to ensure reliable transfer and synchronization. A simplified block diagram of the I2C is shown in the following figure.

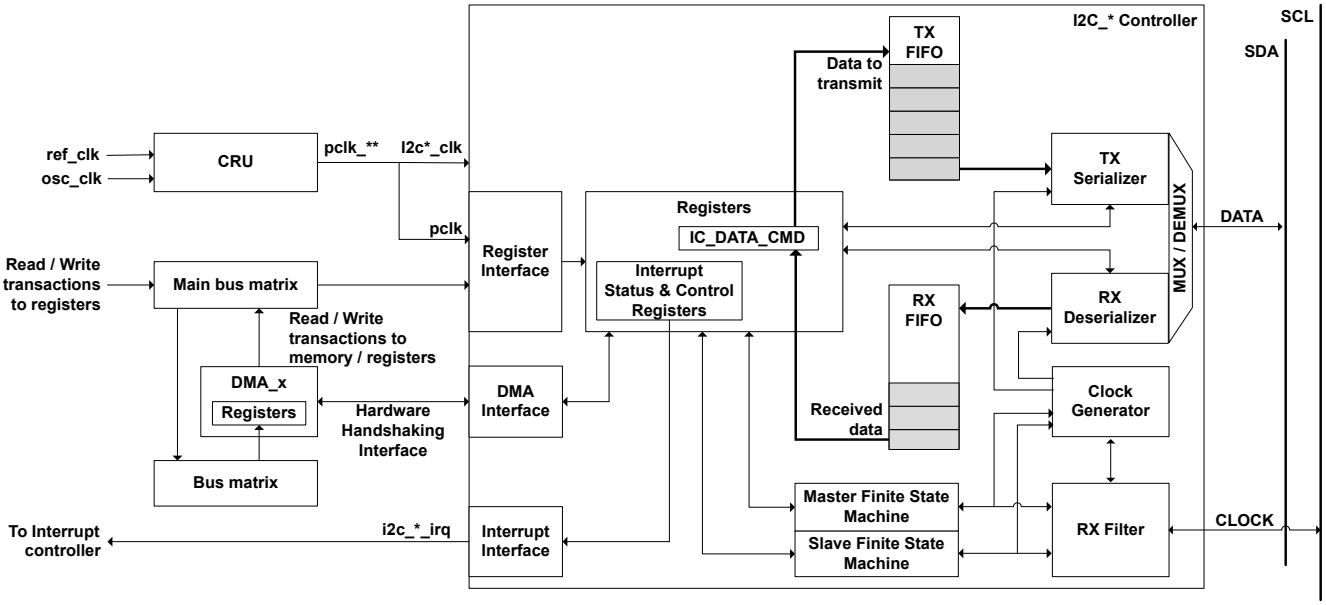

Figure 13-9 I2C block diagram

In the figure above:

- x indicates 0 for DMA_0 and 1 for DMA_1. For more information, refer to DMA Controller chapter.
- An asterisk symbol (*) indicates 0 for I2C_0, 1 for I2C_1, 2 for I2C_2 and 3 for I2C_3.
- A double asterisk (**) indicates 0 for I2C_0 an I2C_1, 1 for I2C_2 and I2C_3.

13.5.1. I2C Functional Description

This section covers the following topics:

- Addressing Slave Protocol
- Operation Modes
- Fast Mode Plus Operation
- SDA Hold Time
- Interrupts

13.5.1.1. Addressing Slave Protocol

There are two address formats: the 7-bit address format and the 10-bit address format.

7-bit Address Format

During the 7-bit address format, the first seven bits (bits [7:1]) of the first byte set the Slave address and the LSB bit (bit [0]) is the R/W bit as shown in the following figure. When bit [0] (R/W) is set to 0, the Master writes to the Slave. When bit [0] (R/W) is set to 1, the Master reads from the Slave.

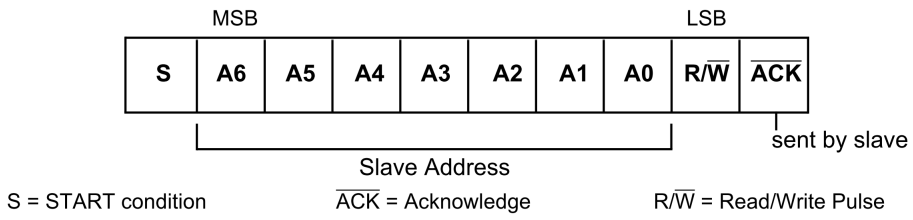

Figure 13-10 7-bit address format

10-bit Address Format

During 10-bit addressing, two bytes are transferred to set the 10-bit address. The transfer of the first byte contains the following bit definition. The first five bits (bits [7:3]) notify the Slaves that this is a 10-bit transfer followed by the next two bits (bits [2:1]), which set the Slaves address bits [9:8], and the LSB bit (bit 0) is the R/W bit. The second byte transferred sets bits [7:0] of the Slave address. The following figure shows the 10-bit address format.

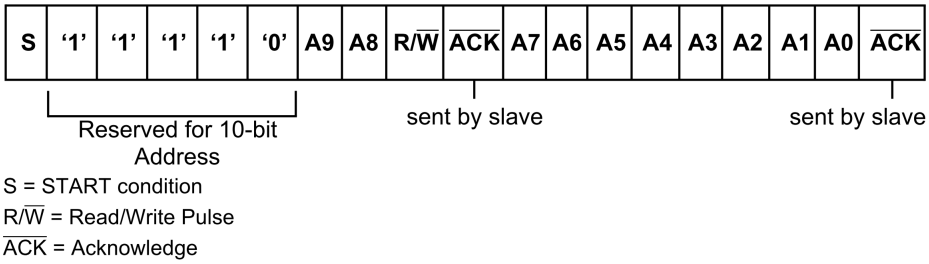

Figure 13-11 10-bit address format

The following table defines the special purpose and reserved first byte addresses.

Table 13-123 I2C definition of bits in the first byte

Slave Address	R/W Bit	Description
0000 000	0	General Call Address. The I2C Controller places the data in the receive buffer and issues a General Call interrupt.
0000 000	1	START byte
0000 001	X	CBUS address I2C Controller ignores these accesses
0000 010	X	Reserved
0000 011	X	Reserved
0000 1XX	X	High-speed Master code
1111 1XX	X	Reserved
1111 0XX	X	10-bit Slave addressing

The I2C Controller does not restrict you from using these reserved addresses. However, if you use these reserved addresses, you may run into incompatibilities with other I2C components.

13.5.1.2. Operation Modes

This section provides information on operation modes.

It is important to note that the I2C should only be set to operate as an I2C Master, or I2C Slave, but not both simultaneously. This is achieved by ensuring that `IC_CON.IC_SLAVE_DISABLE` and `IC_CON.IC_MASTER_MODE` bits are never set to 0 and 1, respectively.

13.5.1.2.1. Slave Mode Operation

Initial Configuration

To use the I2C Controller as a Slave, perform the following steps:

1. Disable the I2C Controller by writing a 0 to IC_ENABLE.ENABLE bit.
2. Write to the IC_SAR.IC_SAR to set the Slave address. This is the address to which the I2C Controller responds.
3. Write to the IC_CON.IC_10BITADDR_SLAVE to specify which type of addressing is supported (7- or 10-bit). Enable the I2C Controller in Slave-only mode by writing a 0 into IC_CON.IC_SLAVE_DISABLE and a 0 to IC_CON.IC_MASTER_MODE.

Slaves and Masters do not have to be programmed with the same type of addressing 7- or 10-bit address. For instance, a Slave can be programmed with 7-bit addressing and a Master with 10-bit addressing, and vice versa.

4. Enable the I2C Controller by writing a 1 to IC_CON.IC_MASTER_MODE.

Depending on the reset values chosen, steps 2 and 3 may not be necessary because the reset values can be configured. For instance, if the device is only going to be a Master, there would be no need to set the Slave address because you can configure the I2C Controller to have the Slave disabled after reset and to enable the Master after reset. The values stored are static and do not need to be reprogrammed if the I2C Controller is disabled.

Slave-Transmitter Operation for a Single Byte

When another I2C Master device on the bus addresses the I2C Controller and requests data, the I2C Controller acts as a Slave-transmitter and the following steps occur:

1. The other I2C Master device initiates an I2C transfer with an address that matches the Slave address in the IC_SAR register of the I2C Controller.
2. The I2C Controller acknowledges the sent address and recognizes the direction of the transfer to indicate that it is acting as a Slave-transmitter.
3. The I2C Controller asserts the RD_REQ interrupt (IC_RAW_INTR_STAT[5] register) and holds the SCL line low. It is in a wait state until software responds.

   If the RD_REQ interrupt has been masked, due to IC_INTR_MASK[5] register being set to 0, then it is recommended that a hardware and/or software timing routine be used to instruct the CPU to perform periodic reads of the IC_RAW_INTR_STAT register.

   a. Reads that indicate IC_RAW_INTR_STAT[5] being set to 1 must be treated as the equivalent of the RD_REQ interrupt being asserted.
   b. Software must then act to satisfy the I2C transfer.
   c. The timing interval used should be in the order of 10 times the fastest SCL clock period the I2C Controller can handle. For example, for 400 kb/s, the timing interval is 25us.

The value of 10 is recommended here because this is approximately the amount of time required for a single byte of data transferred on the I2C bus.

4. If there is any data remaining in the Tx FIFO before receiving the read request, then the I2C Controller asserts a TX_ABRT interrupt (IC_RAW_INTR_STAT[6] register) to flush the old data from the Tx FIFO.

Because the I2C Controller's Tx FIFO is forced into a flushed/reset state whenever a TX_ABRT event occurs, it is necessary for software to release the I2C Controller from this state by reading the IC_CLR_TX_ABRT register before attempting to write into the Tx FIFO. See register IC_RAW_INTR_STAT for more details.

If the TX_ABRT interrupt has been masked, due to of IC_INTR_MASK[6] register being set to 0, then it is recommended that re-using the timing routine (described in the previous step), or a similar one, be used to read the IC_RAW_INTR_STAT register.

a. Reads that indicate IC_INTR_STAT.R_TX_ABRT being set to 1 must be treated as the equivalent of the TX_ABRT interrupt being asserted.
b. There is no further action required from software.
c. The timing interval used should be similar to that described in the previous step for the IC_RAW_INTR_STAT[5] register.

5. Software writes to the IC_DATA_CMD register with the data to be written (by writing a 0 in IC_DATA_CMD.CMD bit).
6. Software must clear the RD_REQ and TX_ABRT interrupts (RD_REQ and TX_ABRT bits , respectively) of the IC_RAW_INTR_STAT register before proceeding.

   If the RD_REQ and/or TX_ABRT interrupts have been masked, then clearing of the IC_RAW_INTR_STAT register will have already been performed when either the IC_RAW_INTR_STAT.R_RD_REQ or IC_RAW_INTR_STAT.R_TX_ABRT bit has been read as 1.
7. The I2C Controller releases the SCL and transmits the byte.
8. The Master may hold the I2C bus by issuing a RESTART condition or release the bus by issuing a STOP condition.

Slave-Receiver Operation for a Single Byte

When another I2C Master device on the bus addresses the I2C Controller and is sending data, the I2C Controller acts as a Slave-receiver and the following steps occur:

1. The other I2C Master device initiates an I2C transfer with an address that matches the I2C Controller's Slave address in the IC_SAR register.
2. The I2C Controller acknowledges the sent address and recognizes the direction of the transfer to indicate that the I2C Controller is acting as a Slave-receiver.
3. The I2C Controller receives the transmitted byte and places it in the receive buffer.

If the Rx FIFO is completely filled with data when a byte is pushed then an overflow occurs and the I2C Controller continues with subsequent I2C transfers. Because a NACK is not generated, software must recognize the overflow when indicated by the I2C Controller (by the IC_INTR_STAT.R_RX_OVER bit) and take appropriate actions to recover from lost data. Hence, there is a real time constraint on software to service the Rx FIFO before the latter overflows, as there is no way to re-apply pressure to the remote transmitting Master. You must select a deep enough Rx FIFO depth to satisfy the interrupt service interval of the system.

4. The I2C Controller asserts the RX_FULL interrupt (IC_RAW_INTR_STAT[2] register).

   If the RX_FULL interrupt has been masked, due to setting IC_INTR_MASK[2] register to 0 or setting IC_TX_TL to a value larger than 0, then it is recommended that a timing routine be implemented for periodic reads of the IC_STATUS register. Reads of the IC_STATUS register, with RFNE bit set at 1, must then be treated by software as the equivalent of the RX_FULL interrupt being asserted.
5. Software may read the byte from the IC_DATA_CMD.DAT register .
6. The other Master device may hold the I2C bus by issuing a RESTART condition, or release the bus by issuing a STOP condition.

Slave-Transfer Operation for Bulk Transfers

In the standard I2C protocol, all transactions are single byte transactions and the programmer responds to a remote Master read request by writing one byte into the Slave's Tx FIFO. When a Slave (Slave-

transmitter) is issued with a read request (RD_REQ) from the remote Master (Master-receiver), at a minimum there should be at least one entry placed into the Slave-transmitter's Tx FIFO. The I2C Controller is designed to handle more data in the Tx FIFO so that subsequent read requests can take that data without raising an interrupt to get more data. Ultimately, this eliminates the possibility of significant latencies being incurred between raising the interrupt for data each time had there been a restriction of having only one entry placed in the Tx FIFO.

This mode only occurs when the I2C Controller is acting as a Slave-transmitter. If the remote Master acknowledges the data sent by the Slave-transmitter and there is no data in the Slave's Tx FIFO, the I2C Controller holds the I2C SCL line low while it raises the read request interrupt (RD_REQ) and waits for data to be written into the Tx FIFO before it can be sent to the remote Master.

If the RD_REQ interrupt is masked, due to IC_INTR_STAT.M_RD_REQ bit being set to 0, then it is recommended that a timing routine be used to activate periodic reads of the IC_RAW_INTR_STAT register. Reads of IC_RAW_INTR_STAT that return IC_INTR_STAT.R_RD_REQ set to 1 must be treated as the equivalent of the RD_REQ interrupt referred to in this section.

The RD_REQ interrupt is raised upon a read request, and like interrupts, must be cleared when exiting the *Interrupt Service Handling Routine* (**ISR**). The ISR allows you to either write 1 byte or more than 1 byte into the Tx FIFO. During the transmission of these bytes to the Master, if the Master acknowledges the last byte then the Slave must raise the RD_REQ again because the Master is requesting for more data.

If the programmer knows in advance that the remote Master is requesting a packet of n bytes, then when another Master addresses I2C Controller and requests data, the Tx FIFO could be written with n number bytes and the remote Master receives it as a continuous stream of data. For example, the I2C Slave continues to send data to the remote Master as long as the remote Master is acknowledging the data sent and there is data available in the Tx FIFO. There is no need to hold the SCL line low or to issue RD_REQ again.

If the remote Master is to receive n bytes from the I2C Controller but the programmer wrote a number of bytes larger than n to the Tx FIFO, then when the Slave finishes sending the requested n bytes, it clears the Tx FIFO and ignores any excess bytes.

The I2C Controller generates a transmit abort (TX_ABRT) event to indicate the clearing of the Tx FIFO in this example. At the time an ACK/NACK is expected, if a NACK is received, then the remote Master has all the data it wants. At this time, a flag is raised within the Slave's state machine to clear the leftover data in the Tx FIFO. This flag is transferred to the processor bus clock domain where the FIFO exists and the contents of the Tx FIFO is cleared at that time.

13.5.1.2.2. Master Mode Operation

Initial Configuration

The initial configuration procedure for Master Mode Operation proceeds as described below.

1. Disable the I2C Controller by writing 0 to the IC_ENABLE.ENABLE.
2. Write to the IC_CON.SPEED register to set the maximum speed mode supported for Slave operation and to specify whether the I2C Controller starts its transfers in 7/10 bit addressing mode when the device is a Slave (IC_CON.IC_10BITADDR_SLAVE).
3. Write to the IC_TAR register the address of the I2C device to be addressed. It also indicates whether a General Call or a START BYTE command is going to be performed by I2C. The addressing mode — 7-bit or 10-bit — is controlled by the IC_TAR.IC_10BITADDR_MASTER.

4. Enable the I2C Controller by writing 1 to the IC_ENABLE.ENABLE.
5. Now write the transfer direction and data to be sent to the IC_DATA_CMD register. If the IC_DATA_CMD register is written before the I2C Controller is enabled, the data and commands are lost as the buffers are kept cleared when the I2C Controller is not enabled.

Dynamic IC_TAR or IC_10BITADDR_MASTER Update

The I2C Controller supports dynamic updating of the IC_TAR and IC_10BITADDR_MASTER bit fields of the IC_TAR register. You can dynamically write to the IC_TAR register provided the software ensures that in the Tx FIFO there are no other commands that use the existing TAR address. If the software does not ensure this, then IC_TAR should be re-programmed only if the following conditions are met:

I2C Controller is not enabled (IC_ENABLE[0]=0)
OR
I2C Controller is enabled (IC_ENABLE[0]=1)
AND
I2C Controller is NOT engaged in any Master (Tx, Rx) operation (IC_STATUS[5]=0)
AND
I2C Controller is enabled to operate in Master mode (IC_CON[0]=1)
AND
there are NO entries in the Tx FIFO (IC_STATUS[2]=1).

If the software or application is aware that the I2C Controller is not using the TAR address for the pending commands in the Tx FIFO, then it is possible to update the TAR address even while the Tx FIFO has entries (IC_STATUS[2]=0).

The updated TAR address comes into effect only when the next START or RESTART occurs on the bus.

Master Transmit and Master Receive

The I2C Controller supports switching back and forth between reading and writing dynamically. To transmit data, write the data to be written to the lower byte of the I2C Rx/Tx Data Buffer and Command Register (IC_DATA_CMD). The IC_DATA_CMD.CMD should be written to 0 for I2C write operations. Subsequently, a read command may be issued by writing don't cares to the lower byte of the IC_DATA_CMD register, and a 1 should be written to the IC_DATA_CMD.CMD bit. The I2C Controller Master continues to initiate transfers as long as there are commands present in the transmit FIFO. If the transmit FIFO becomes empty the Master either inserts a STOP condition after completing the current transfers.

13.5.1.2.3. Disabling I2C Controller

The register IC_ENABLE_STATUS is added to allow software to unambiguously determine when the hardware has completely shut down in response to ENABLE bit of the IC_ENABLE register being set from 1 to 0.

To disable the I2C Controller, perform the following procedure:

1. Set the IC_ENABLE.ENABLE to 0.
2. Read the IC_ENABLE_STATUS register and test the IC_ENABLE_STATUS.IC_EN.
3. If IC_ENABLE_STATUS[0] is 1, then proceed to the previous step. Otherwise, exit with a relevant success code.

13.5.1.2.4. Aborting I2C Transfers

The ABORT control bit of the IC_ENABLE register allows the software to relinquish the I2C bus before completing the issued transfer commands from the Tx FIFO. In response to an ABORT request, the controller issues the STOP condition over the I2C bus, followed by Tx FIFO flush. Aborting the transfer is allowed only in Master mode of operation.

To abort I2C transfers, perform the following procedure:

1. Stop filling the Tx FIFO (IC_DATA_CMD) with new commands.
2. When operating in DMA mode, disable the transmit DMA by setting IC_DMA_CR.TDMAE to 0.
3. Set the IC_ENABLE.ABORT to 1.
4. Wait for the M_TX_ABRT interrupt.
5. Read the IC_TX_ABRT_SOURCE register to identify the source as ABRT_USER_ABRT.

13.5.1.3. Fast Mode Plus Operation

In Fast Mode Plus, the I2C Controller allows the Fast Mode operation to be extended to support speeds up to 1000 Kb/s. To enable the I2C Controller for Fast mode plus operation, perform the following steps before initiating any data transfer:

1. Set i2c*_clk frequency greater than or equal to 32 MHz.
2. Program the IC_CON.SPEED = 2 for Fast Mode or Fast Mode Plus.
3. Program IC_FS_SCL_LCNT and IC_FS_SCL_HCNT registers to meet the Fast Mode Plus SCL.
4. Program the IC_FS_SPKLEN register to suppress the maximum spike of 50 ns.
5. Program the IC_SDA_SETUP register to meet the minimum data setup time ($t_{SU;DAT}$).

13.5.1.4. SDA Hold Time

The I2C protocol specification requires 300 ns of hold time on the SDA signal ($t_{HD;DAT}$) in Standard Mode and Fast Mode, and a hold time long enough to bridge the undefined part between logic 1 and logic 0 of the falling edge of SCL in Fast Mode Plus.

Board delays on the SCL and SDA signals can mean that the hold time requirement is met at the I2C Master, but not at the I2C Slave (or vice-versa). As each application encounters differing board delays, the I2C Controller contains a software programmable register (IC_SDA_HOLD) to enable dynamic adjustment of the SDA hold time.

The IC_SDA_HOLD.IC_SDA_TX_HOLD bits are used to control the hold time of SDA during transmit in both Slave and Master mode (after SCL goes from HIGH to LOW).

The IC_SDA_HOLD.IC_SDA_RX_HOLD bits are used to extend the SDA transition (if any) whenever SCL is HIGH in the receiver (in either Master or Slave mode).

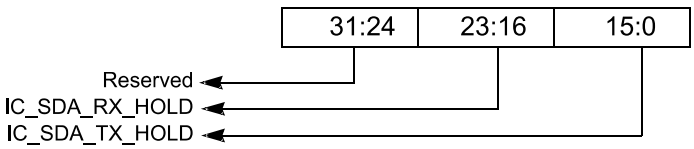

Figure 13-12 IC_SDA_HOLD register

If different SDA hold times are required for different speed modes, the IC_SDA_HOLD register must be reprogrammed when the speed mode is being changed. The IC_SDA_HOLD register can be programmed only when the I2C Controller is disabled (IC_ENABLE[0] = 0).

13.5.1.4.1. SDA Hold Timings in Receiver

When the I2C Controller acts as a receiver, according to the I2C protocol, the device should internally hold the SDA line to bridge undefined gap between logic 1 and logic 0 of SCL.

`IC_SDA_RX_HOLD` can be used to alter the internal hold time which the I2C Controller applies to the incoming SDA line. Each value in the `IC_SDA_RX_HOLD` bit of `IC_SDA_HOLD` register represents a unit of one `i2c*_clk` period. The minimum value of `IC_SDA_RX_HOLD` is 0. This hold time is applicable only when SCL is HIGH. The receiver does not extend the SDA after SCL goes LOW internally.

The maximum values of `IC_SDA_RX_HOLD` that can be programmed in the register for the respective speed modes are derived from the equations show in the table below.

Table 13-124 Maximum values for `IC_SDA_RX_HOLD`

Speed Mode	Maximum `IC_SDA_RX_HOLD` Value
Standard Mode	`IC_SS_SCL_HCNT - IC_FS_SPKLEN - 3`
Fast Mode or Fast Mode Plus	`IC_FS_SCL_HCNT - IC_FS_SPKLEN - 3`
High Speed	`IC_FS_SCL_HCNT - IC_FS_SPKLEN - 3` `IC_HS_SCL_LCNT - IC_HS_SPKLEN - 3`

13.5.1.4.2. SDA Hold Timings in Transmitter

The I2C Controller can use the `IC_SDA_TX_HOLD` bit of `IC_SDA_HOLD` register to alter the timing of the generated SDA signal. Each value in the `IC_SDA_TX_HOLD` register represents a unit of one `i2c*_clk` period.

When the I2C Controller is operating in Master Mode, the minimum $t_{HD:DAT}$ timing is one `i2c*_clk` period. Therefore even when `IC_SDA_TX_HOLD` has a value of zero, the I2C Controller drives SDA one `i2c*_clk` cycle after driving SCL to logic 0. For all other values of `IC_SDA_TX_HOLD`, the following is true:

- Drive on SDA occurs `IC_SDA_TX_HOLD` `i2c*_clk` cycles after driving SCL to logic 0.

When the I2C Controller is operating in Slave Mode, the minimum $t_{HD:DAT}$ timing is `SPKLEN + 7` `i2c*_clk` periods, where `SPKLEN` is `IC_FS_SPKLEN` if the component is operating in Standard Mode, Fast Mode, or Fast Mode Plus, `IC_HS_SPKLEN` if the component is operating in High Speed Mode.

This delay allows for synchronization and spike suppression on the SCL sample. Therefore, even when `IC_SDA_TX_HOLD` has a value less than `SPKLEN + 7`, the I2C Controller drives SDA `SPKLEN + 7` `i2c*_clk` cycles after SCL has transitioned to logic 0. For all other values of `IC_SDA_TX_HOLD`, the following is true:

- Drive on SDA occurs `IC_SDA_TX_HOLD` `i2c*_clk` cycles after SCL has transitioned to logic 0.

The programmed SDA hold time cannot exceed at any time the duration of the low part of SCL. Therefore the programmed value cannot be larger than `N_SCL_LOW-2`, where `N_SCL_LOW` is the duration of the low part of the SCL period measured in `i2c*_clk` cycles.

13.5.1.5. Interrupts

The combined I2C interrupt is asserted if any of the individual interrupts is asserted and enabled. You can clear the combined interrupt, all individual interrupts, and the I2C Transmit Abort Source

Register (IC_TX_ABRT_SOURCE) by reading the Clear Combined and Individual Interrupt Register (IC_CLR_INTR)

The following table lists the I2C events monitored and corresponding Interrupt IDs.

Table 13-125 I2C events monitored by individual interrupts

Event	Interrupt ID
SCL Stuck condition detect on I2C	SCL_STUCK_AT_LOW
General Call received	GEN_CALL
Start condition detect on I2C	START_DET
Stop condition detect on I2C	STOP_DET
I2C activity	ACTIVITY
Receive done	RX_DONE
Transmit abort	TX_ABRT
Slave read request	RD_REQ
Transmit buffer empty	TX_EMPTY
Transmit buffer overflow	TX_OVER
Receive buffer full	RX_FULL
Receive buffer overflow	RX_OVER
Receive buffer underflow	RX_UNDER

For detailed event descriptions, see I2C Raw Interrupt Status Register (IC_RAW_INTR_STAT).

The masked status of the individual interrupt sources can be read from I2C Interrupt Status Register (IC_INTR_STAT).

You can enable or disable the individual interrupts by changing the mask bits in the I2C Interrupt Mask Register (IC_INTR_MASK). This register is active low; a value of 0 masks the interrupt, whereas a value of 1 unmasks the interrupt.

The unmasked raw versions of these bits are available in the I2C Raw Interrupt Status Register (IC_RAW_INTR_STAT).

You can clear some of the listed individual I2C interrupts by reading the following registers:

- Clear Combined and Individual Interrupt Register (IC_CLR_INTR)
- Clear RX_UNDER Interrupt Register (IC_CLR_RX_UNDER)
- Clear RX_OVER Interrupt Register (IC_CLR_RX_OVER)
- Clear TX_OVER Interrupt Register (IC_CLR_TX_OVER)
- Clear RD_REQ Interrupt Register (IC_CLR_RD_REQ)
- Clear TX_ABRT Interrupt Register (IC_CLR_TX_ABRT)
- Clear RX_DONE Interrupt Register (IC_CLR_RX_DONE)
- Clear ACTIVITY Interrupt Register (IC_CLR_ACTIVITY)

- Clear `STOP_DET` Interrupt Register (`IC_CLR_STOP_DET`)
- Clear `START_DET` Interrupt Register (`IC_CLR_START_DET`)
- Clear `GEN_CALL` Interrupt Register (`IC_CLR_GEN_CALL`)

13.5.2. I2C Registers

The BE-U1000 microcontroller integrates four I2C Controllers. The following table describes address regions for the I2C Controllers.

Table 13-126 Memory address regions for I2C registers

Region	Block Name	Size	Start Address	End Address
Periphery 0	I2C0	4 KB	0x1100_4000	0x1100_4FFF
	I2C1	4 KB	0x1100_5000	0x1100_5FFF
Periphery 1	I2C2	4 KB	0x1300_4000	0x1300_4FFF
	I2C3	4 KB	0x1300_5000	0x1300_5FFF

 For details, refer to Memory Map chapter.

This section covers the following topics:

- Register Map
- Register Descriptions

13.5.2.1. Register Map

The following table provides a top-level summary of each register for the I2C controllers. To view the detailed description of a register, click the register name in the **Description** column.

Table 13-127 I2C register map

Name	Offset	Description	Default Value
`IC_CON`	0x00	I2C Control Register	0x67
`IC_TAR`	0x04	I2C Target Address Register	0x55
`IC_SAR`	0x08	I2C Slave Address Register	0x55
`IC_HS_MADDR`	0xC	I2C High Speed Master Mode Code Address Register	0x1
`IC_DATA_CMD`	0x10	I2C Rx/Tx Data Buffer and Command Register	0x0
`IC_SS_SCL_HCNT`	0x14	Standard Speed I2C Clock SCL High Count Register	0x190
`IC_SS_SCL_LCNT`	0x18	Standard Speed I2C Clock SCL Low Count Register	0x1D6
`IC_FS_SCL_HCNT`	0x1C	Fast Mode or Fast Mode Plus I2C Clock SCL High Count Register	0x3C

Table 13-127 I2C register map (continued)

Name	Offset	Description	Default Value
IC_FS_SCL_LCNT	0x20	Fast Mode or Fast Mode Plus I2C Clock SCL Low Count Register	0x82
IC_HS_SCL_HCNT	0x24	High Speed I2C Clock SCL High Count Register	0x6
IC_HS_SCL_LCNT	0x28	High Speed I2C Clock SCL Low Count Register	0x10
IC_INTR_STAT	0x2C	I2C Interrupt Status Register	0x0
IC_INTR_MASK	0x30	I2C Interrupt Mask Register	0x48FF
IC_RAW_INTR_STAT	0x34	I2C Raw Interrupt Status Register	0x0
IC_RX_TL	0x38	I2C Receive FIFO Threshold Register	0x0
IC_TX_TL	0x3C	I2C Transmit FIFO Threshold Register	0x0
IC_CLR_INTR	0x40	Clear Combined and Individual Interrupt Register	0x0
IC_CLR_RX_UNDER	0x44	Clear RX_UNDER Interrupt Register	0x0
IC_CLR_RX_OVER	0x48	Clear RX_OVER Interrupt Register	0x0
IC_CLR_TX_OVER	0x4C	Clear TX_OVER Interrupt Register	0x0
IC_CLR_RD_REQ	0x50	Clear RD_REQ Interrupt Register	0x0
IC_CLR_TX_ABRT	0x54	Clear TX_ABRT Interrupt Register	0x0
IC_CLR_RX_DONE	0x58	Clear RX_DONE Interrupt Register	0x0
IC_CLR_ACTIVITY	0x5C	Clear ACTIVITY Interrupt Register	0x0
IC_CLR_STOP_DET	0x60	Clear STOP_DET Interrupt Register	0x0
IC_CLR_START_DET	0x64	Clear START_DET Interrupt Register	0x0
IC_CLR_GEN_CALL	0x68	Clear GEN_CALL Interrupt Register	0x0
IC_ENABLE	0x6C	I2C ENABLE Register	0x4
IC_STATUS	0x70	I2C STATUS Register	0x6
IC_TXFLR	0x74	I2C Transmit FIFO Level Register	0x0
IC_RXFLR	0x78	I2C Receive FIFO Level Register	0x0
IC_SDA_HOLD	0x7C	I2C SDA Hold Time Length Register	0x1
IC_TX_ABRT_SOURCE	0x80	I2C Transmit Abort Source Register	0x0
IC_DMA_CR	0x88	DMA Control Register	0x0

Table 13-127 I2C register map (continued)

Name	Offset	Description	Default Value
IC_DMA_TDLR	0x8C	DMA Transmit Data Level Register	0x0
IC_DMA_RDLR	0x90	DMA Receive Data Level Register	0x0
IC_SDA_SETUP	0x94	I2C SDA Setup Register	0x64
IC_ACK_GENERAL_CALL	0x98	I2C ACK General Call Register	0x0
IC_ENABLE_STATUS	0x9C	I2C Enable Status Register	0x0
IC_FS_SPKLEN	0xA0	I2C SS, FS or FM+ Spike Suppression Limit	0x5
IC_HS_SPKLEN	0xA4	I2C HS Spike Suppression Limit Register	0x1
IC_SCL_STUCK_AT_LOW_TIMEOUT	0xAC	I2C SCL Stuck at Low Timeout Register	0xFFFFFFFF
IC_SDA_STUCK_AT_LOW_TIMEOUT	0xB0	I2C SDA Stuck at Low Timeout Register	0xFFFFFFFF
IC_CLR_SCL_STUCK_DET	0xB4	Clear SCL Stuck at Low Detect Interrupt Register	0x0
REG_TIMEOUT_RST	0xF0	Register Timeout Counter Reset Value	0x8

13.5.2.2. Register Descriptions

In the tables below, the **R/W** column specifies how the application and the core can access the register fields.

⚠ Reserved bits in the I2C registers cannot be used.

13.5.2.2.1. I2C Control Register (IC_CON)

The IC_CON register is at offset 0x00.

This register can be written only when the I2C Controller is disabled, which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect.

The following table shows the register bit assignments.

Table 13-128 Fields for register: IC_CON

Bits	Name	R/W	Description
[31:12]			Reserved
[11]	BUS_CLEAR_FEATURE_CTRL	R/W	In Master mode: **Values:** **0x0**: Bus Clear Feature is disabled **0x1**: Bus Clear Feature is enabled In Slave mode, this register bit is not applicable. **Value After Reset**: 0x0

Table 13-128 Fields for register: `IC_CON` (continued)

Bits	Name	R/W	Description
[10]	`STOP_DET_IF_MASTER_ACTIVE`	R/W	In Master mode: **Values:** **0x0**: Master issues the `STOP_DET` interrupt irrespective of whether Master is active or not **0x1**: Master issues the `STOP_DET` interrupt only when Master is active **Value After Reset**: 0x0
[9]	`RX_FIFO_FULL_HLD_CTRL`	R	Controls whether I2C Controller should hold the bus when the Rx FIFO is physically full to its `RX_BUFFER_DEPTH=8`. **Values:** **0x0**: Overflow when `RX_FIFO` is full **0x1**: Hold bus when `RX_FIFO` is full **Value After Reset**: 0x0
[8]	`TX_EMPTY_CTRL`	R/W	Controls the generation of the `TX_EMPTY` interrupt, as described in the `IC_RAW_INTR_STAT` register. **Values:** **0x0**: Default behavior of `TX_EMPTY` interrupt **0x1**: Controlled generation of `TX_EMPTY` interrupt **Value After Reset**: 0x0
[7]	`STOP_DET_IFADDRESSED`	R/W	In Slave mode: **Values:** **0x0**: Issues the `STOP_DET` interrupt irrespective of whether it is addressed or not **0x1**: Issues the `STOP_DET` irrespective only when it is addressed During a general call address, this Slave does not issue the `STOP_DET` interrupt if `STOP_DET_IFADDRESSED = 1`, even if the Slave responds to the general call address by generating ACK. The `STOP_DET` interrupt is generated only when the transmitted address matches the Slave address. **Value After Reset**: 0x0
[6]	`IC_SLAVE_DISABLE`	R/W	This bit controls whether I2C has its Slave disabled. That means once the `presetn` signal is applied, then this bit takes on the 1 value. You have the choice of having the Slave enabled or disabled after reset is applied, which means software does not have to configure the Slave. By default, the Slave is always enabled (in reset state as well). If you need to disable it after reset, set this bit to 1.

Table 13-128 Fields for register: IC_CON (continued)

Bits	Name	R/W	Description
			If this bit is set (Slave is disabled), I2C Controller functions only as a Master and does not perform any action that requires a Slave. Software should ensure that if this bit is written with 0, then MASTER_MODE should also be written with a 0. **Values:** **0x0**: Slave mode is enabled **0x1**: Slave mode is disabled **Value After Reset**: 0x1
[5]	IC_RESTART_EN	R/W	Determines whether RESTART conditions may be sent when acting as a Master. Some older Slaves do not support handling RESTART conditions; however, RESTART conditions are used in several I2C operations. When RESTART is disabled, the Master is prohibited from performing the following functions: • Sending a START BYTE • Performing any high-speed mode operation • High-speed mode operation • Performing direction changes in combined format mode • Performing a read operation with a 10-bit address By replacing RESTART condition followed by a STOP and a subsequent START condition, split operations are broken down into multiple I2C transfers. If the above operations are performed, it will result in setting the IC_RAW_INTR_STAT.TX_ABRT. **Values:** **0x0**: Master restart disabled **0x1**: Master restart enabled **Value After Reset**: 0x1
[4]	IC_10BITADDR_MASTER_RO	R	The function of this bit is handled by bit 12 of IC_TAR register, and becomes a read-only copy called IC_10BITADDR_MASTER_RO. **Values:** **0x0**: Master 7-bit addressing mode **0x1**: Master 10-bit addressing mode **Value After Reset**: 0x0
[3]	IC_10BITADDR_SLAVE	R/W	When acting as a Slave, this bit controls whether the I2C Controller responds to 7- or 10-bit addresses. **Values:**

Table 13-128 Fields for register: `IC_CON` (continued)

Bits	Name	R/W	Description
			0x0: Slave 7-bit addressing. The I2C Controller ignores transactions that involve 10-bit addressing; for 7-bit addressing, only the lower 7 bits of the `IC_SAR` register are compared. **0x1**: Slave 10-bit addressing. The I2C Controller responds to only 10-bit addressing transfers that match the full 10 bits of the `IC_SAR`. **Value After Reset**: 0x0
[2:1]	`SPEED`	R/W	These bits control at which speed the I2C Controller operates; its setting is relevant only if one is operating the I2C in Master mode. Hardware protects against illegal values programmed by software. These bits must be programmed appropriately for Slave mode also, as it is used to capture correct value of spike filter as per the speed mode. This register should be programmed only with a value in the range of 1 to 3; otherwise, hardware updates this register with the value of 3. **Values:** **0x1**: Standard Speed mode of operation (100 kbit/s) **0x2**: Fast (<=400 kbit/s) or Fast Plus (<=1000 kbit/s) mode of operation **0x3**: High Speed mode (3.4 Mbit/s) **Value After Reset**: 0x3
[0]	`MASTER_MODE`	R/W	This bit controls whether the I2C Master is enabled. Software should ensure that if this bit is written with '1' then `IC_SLAVE_DISABLE` should also be written with a '1'. **Values:** **0x0**: Master mode is disabled **0x1**: Master mode is enabled **Value After Reset**: 0x1

13.5.2.2.2. I2C Target Address Register (`IC_TAR`)

The `IC_TAR` register is at offset 0x4.

The register is 13 bits wide. In this case, writes to `IC_TAR` succeed when one of the following conditions are true:

- The I2C Controller is NOT enabled (`IC_ENABLE[0] = 0`);
- OR the following conditions are met:
  - The I2C Controller is enabled (`IC_ENABLE[0]=1`)
  - AND I2C is NOT engaged in any Master (Tx, Rx) operation (`IC_STATUS[5]=0`)

- AND I2C is enabled to operate in Master mode (`IC_CON[0]=1`)
- AND there are NO entries in the Tx FIFO (`IC_STATUS[2]=1`).

You can change the TAR address dynamically without losing the bus, only if the following conditions are met:

- The I2C Controller is enabled (`IC_ENABLE[0]=1`)
- AND I2C is enabled to operate in Master mode (`IC_CON[0]=1`)
- AND there are NO entries in the Tx FIFO and the Master is in HOLD state (`IC_INTR_STAT[13]=1`).

If the software or application is aware that the I2C Controller is not using the TAR address for the pending commands in the Tx FIFO, then it is possible to update the TAR address even while the Tx FIFO has entries (`IC_STATUS[2]= 0`).

- It is not necessary to perform any write to this register if the I2C Controller is enabled as an I2C Slave only.

The following table shows the register bit assignments.

Table 13-129 Fields for register: `IC_TAR`

Bits	Name	R/W	Description
[31:13]			Reserved
[12]	`IC_10BITADDR_MASTER`	R/W	This bit controls whether the I2C Controller starts its transfers in 7- or 10-bit addressing mode when acting as a Master. **Values:** **0x0**: Address 7-bit transmission format **0x1**: Address 10-bit transmission format **Value After Reset:** 0x0
[11]	`SPECIAL`	R/W	This bit indicates whether software performs a Device-ID or General Call or START BYTE command. **Values:** **0x0**: Disables programming of GENERAL_CALL or START_BYTE transmission to ignore bit 10 `GC_OR_START` and use `IC_TAR` normally **0x1**: Enables programming of GENERAL_CALL or START_BYTE transmission to perform special I2C command as specified in `GC_OR_START` bit **Value After Reset:** 0x0
[10]	`GC_OR_START`	R/W	If bit 11 (`SPECIAL`) is set to 1, then this bit indicates whether a General Call or START byte command is to be performed by the I2C Controller. **Values:** **0x0**: GENERAL_CALL byte transmission — after issuing a General Call, only writes may be performed. Attempting to issue a read command results in setting the

Table 13-129 Fields for register: IC_TAR (continued)

Bits	Name	R/W	Description
			IC_RAW_INTR_STAT.TX_ABRTr. The I2C Controller remains in General Call mode until the SPECIAL bit value is cleared. **0x1**: START byte transmission **Value After Reset:** 0x0
[9:0]	IC_TAR	R/W	This is the target address for any Master transaction. When transmitting a General Call, these bits are ignored. To generate a START BYTE, the CPU needs to write only once into these bits. If the IC_TAR and IC_SAR are the same, loopback exists but the FIFOs are shared between Master and Slave, so full loopback is not feasible. Only one direction loopback mode is supported (simplex), not duplex. A Master cannot transmit to itself; it can transmit to only a Slave. **Value After Reset:** 0x55

13.5.2.2.3. I2C Slave Address Register (IC_SAR)

The IC_SAR register is at offset 0x8.

The following table shows the register bit assignments.

Table 13-130 Fields for register: IC_SAR

Bits	Name	R/W	Description
[31:10]			Reserved
[9:0]	IC_SAR	R/W	The IC_SAR holds the Slave address when the I2C is operating as a Slave. For 7-bit addressing, only IC_SAR[6:0] is used. This register can be written only when the I2C interface is disabled, which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect. The default values cannot be any of the reserved address locations: that is, 0x00 to 0x07, or 0x78 to 0x7F. The correct operation of the device is not guaranteed if you program the IC_SAR or IC_TAR to a reserved value. Refer to Table I2C Definition of Bits in First Byte in section Addressing Slave Protocol for a complete list of these reserved values. **Value After Reset**: 0x55

13.5.2.2.4. I2C High Speed Master Mode Code Address Register (IC_HS_MADDR)

The IC_HS_MADDR register is at offset 0xC.

The following table shows the register bit assignments.

Table 13-131 Fields for register: IC_HS_MADDRR

Bits	Name	R/W	Description
[31:3]			Reserved

Table 13-131 Fields for register: IC_HS_MADDRR (continued)

Bits	Name	R/W	Description
[2:0]	IC_HS_MAR	R/W	This bit field holds the value of the I2C HS mode master code. HS mode master codes are reserved 8-bit codes (00001xxx) that are not used for slave addressing or other purposes. Each master has its unique master code; up to eight high-speed mode masters can be present on the same I2C bus system. Valid values are from 0 to 7. This register can be written only when the I2C Controller is disabled, which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect. **Value After Reset:** 0x1

13.5.2.2.5. I2C Rx/Tx Data Buffer and Command Register (IC_DATA_CMD)

The IC_DATA_CMD register is at offset 0x10.

The CPU writes to this register when filling the Tx FIFO and the CPU reads from it when retrieving bytes from Rx FIFO.

The size of the register changes as follows:

- Write: 9 bits
- Read: 8 bits

The following table shows the register bit assignments.

Table 13-132 Fields for register: IC_DATA_CMD

Bits	Name	R/W	Description
[31:12]			Reserved
[11]	FIRST_DATA_BYTE	R	Indicates the first data byte received after the address phase for receive transfer in Master receiver or Slave receiver mode. **Values:** **0x0**: Sequential data byte received **0x1**: Non sequential data byte received **Value After Reset:** 0x0 **Volatile:** true
[10:9]			Reserved
[8]	CMD	W	This bit controls whether a read or a write is performed. This bit does not control the direction when the I2C Controller acts as a Slave. It controls only the direction when it acts as a Master. When a command is entered in the Tx FIFO, this bit distinguishes the write and read commands. In Slave-transmitter mode, a 0 indicates that the data in IC_DATA_CMD is to be transmitted. When programming this bit, you should remember the following: attempting to perform a read operation after a General Call command has been sent results in a TX_ABRT interrupt (TX_ABRT bit of the IC_RAW_INTR_STAT register), unless the IC_TAR.SPECIAL has been

Table 13-132 Fields for register: `IC_DATA_CMD` (continued)

Bits	Name	R/W	Description
			cleared. If a 1 is written to this bit after receiving a `RD_REQ` interrupt, then a `TX_ABRT` interrupt occurs. **Values:** **0x0**: Master Write Command **0x1**: Master Read Command **Value After Reset**: 0x0 **Volatile:** true
[7:0]	`DAT`	R/W	This register contains the data to be transmitted or received on the I2C bus. If you are writing to this register and want to perform a read, `DAT` are ignored by the I2C Controller. However, when you read this register, these bits return the value of data received on the I2C interface. **Value After Reset**: 0x0 **Volatile:** true

13.5.2.2.6. Standard Speed I2C Clock SCL High Count Register (`IC_SS_SCL_HCNT`)

The `IC_SS_SCL_HCNT` register is at offset 0x14.

The following table shows the register bit assignments.

Table 13-133 Fields for register: `IC_SS_SCL_HCNT`

Bits	Name	R/W	Description
[31:16]			Reserved
[15:0]	`IC_SS_SCL_HCNT`	R/W	This register must be set before any I2C bus transaction can take place to ensure proper I/O timing. This register sets the SCL clock high-period count for standard speed. This register can be written only when the I2C interface is disabled which corresponds to the `IC_ENABLE[0]` register being set to 0. Writes at other times have no effect. The minimum valid value is 6; hardware prevents values less than this being written, and if attempted results in value 6 being written.  This register must not be programmed to a value higher than 65525, because the I2C Controller uses a 16-bit counter to flag an I2C bus idle condition when this counter reaches a value of `IC_SS_SCL_HCNT` + 10. **Value After Reset**: 0x190

13.5.2.2.7. Standard Speed I2C Clock SCL Low Count Register (`IC_SS_SCL_LCNT`)

The `IC_SS_SCL_LCNT` register is at offset 0x18.

The following table shows the register bit assignments.

Table 13-134 Fields for register: `IC_SS_SCL_LCNT`

Bits	Name	R/W	Description
[31:16]			Reserved
[15:0]	`IC_SS_SCL_LCNT`	R/W	This register must be set before any I2C bus transaction can take place to ensure proper I/O timing. This register sets the SCL clock low period count for standard speed. This register can be written only when the I2C interface is disabled which corresponds to the `IC_ENABLE[0]` register being set to 0. Writes at other times have no effect. The minimum valid value is 8; hardware prevents values less than this being written, and if attempted results in value 8 being written. **Value After Reset:** 0x1D6

13.5.2.2.8. Fast Mode or Fast Mode Plus I2C Clock SCL High Count Register (`IC_FS_SCL_HCNT`)

The `IC_FS_SCL_HCNT` register is at offset 0x1C.

The following table shows the register bit assignments.

Table 13-135 Fields for register: `IC_FS_SCL_HCNT`

Bits	Name	R/W	Description
[31:16]			Reserved
[15:0]	`IC_FS_SCL_HCNT`	R/W	This register must be set before any I2C bus transaction can take place to ensure proper I/O timing. This register sets the SCL clock high-period count for fast mode or fast mode plus. It is used in High Speed Mode to send the Master Code and START BYTE or General CALL. This register can be written only when the I2C interface is disabled, which corresponds to the `IC_ENABLE[0]` register being set to 0. Writes at other times have no effect. The minimum valid value is 6; hardware prevents values less than this being written, and if attempted results in value 6 being written. **Value After Reset:** 0x3C

13.5.2.2.9. Fast Mode or Fast Mode Plus I2C Clock SCL Low Count Register (`IC_FS_SCL_LCNT`)

The `IC_FS_SCL_LCNT` register is at offset 0x20.

The following table shows the register bit assignments.

Table 13-136 Fields for register: `IC_FS_SCL_LCNT`

Bits	Name	R/W	Description
[31:16]			Reserved
[15:0]	`IC_FS_SCL_LCNT`	R/W	This register must be set before any I2C bus transaction can take place to ensure proper I/O timing. This register sets the SCL clock low period

Table 13-136 Fields for register: `IC_FS_SCL_LCNT` (continued)

Bits	Name	R/W	Description
			count for Fast Mode or Fast Mode Plus. It is used in High Speed Mode to send the Master Code and START BYTE or General CALL. This register can be written only when the I2C interface is disabled, which corresponds to the `IC_ENABLE[0]` register being set to 0. Writes at other times have no effect. The minimum valid value is 8; hardware prevents values less than this being written, and if attempted results in value 8 being written. If the value is less than 8 then the count value gets changed to 8. **Value After Reset:** 0x82

13.5.2.2.10. High Speed I2C Clock SCL High Count Register (`IC_HS_SCL_HCNT`)

The `IC_HS_SCL_HCNT` register is at offset 0x24.

The following table shows the register bit assignments.

Table 13-137 Fields for register: `IC_HS_SCL_HCNT`

Bits	Name	R/W	Description
[31:16]			Reserved
[15:0]	`IC_HS_SCL_HCNT`	R/W	This register must be set before any I2C bus transaction can take place to ensure proper I/O timing. This register sets the SCL clock high period count for high speed. This register can be written only when the I2C interface is disabled, which corresponds to the `IC_ENABLE[0]` register being set to 0. Writes at other times have no effect. The minimum valid value is 6; hardware prevents values less than this being written, and if attempted results in value 6 being written. **Value After Reset:** 0x6

13.5.2.2.11. High Speed I2C Clock SCL Low Count Register (`IC_HS_SCL_LCNT`)

The `IC_HS_SCL_LCNT` register is at offset 0x28.

The following table shows the register bit assignments.

Table 13-138 Fields for register: `IC_HS_SCL_LCNT`

Bits	Name	R/W	Description
[31:16]			Reserved
[15:0]	`IC_HS_SCL_LCNT`	R/W	This register must be set before any I2C bus transaction can take place to ensure proper I/O timing. This register sets the SCL clock low period count for high speed. This register can be written only when the I2C interface is disabled, which corresponds to the `IC_ENABLE[0]` register being set to 0. Writes at other times have no effect. The minimum valid value is 8; hardware prevents values less than this being written, and if attempted results in value 8 being written.

Table 13-138 Fields for register: `IC_HS_SCL_LCNT` (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x10

13.5.2.2.12. I2C Interrupt Status Register (`IC_INTR_STAT`)

The `IC_INTR_STAT` register is at offset 0x2C.

Each bit in this register has a corresponding mask bit in the `IC_INTR_MASK` register. These bits are cleared by reading the matching interrupt clear register. The unmasked raw versions of these bits are available in the `IC_RAW_INTR_STAT` register.

The following table shows the register bit assignments.

Table 13-139 Fields for register: `IC_INTR_STAT`

Bits	Name	R/W	Description
[31:15]			Reserved
[14]	`R_SCL_STUCK_AT_LOW`	R	See `IC_RAW_INTR_STAT` for a detailed description of `R_SCL_STUCK_AT_LOW` bit. **Values:** **0x0**: `R_SCL_STUCK_AT_LOW` interrupt is inactive **0x1**: `R_SCL_STUCK_AT_LOW` interrupt is active **Value After Reset:** 0x0 **Volatile:** true
[13:12]			Reserved
[11]	`R_GEN_CALL`	R	See `IC_RAW_INTR_STAT` for a detailed description of `R_GEN_CALL` bit. **Values:** **0x0**: `R_GEN_CALL` interrupt is inactive **0x1**: `R_GEN_CALL` interrupt is active **Value After Reset:** 0x0 **Volatile:** true
[10]	`R_START_DET`	R	See `IC_RAW_INTR_STAT` for a detailed description of `R_START_DET` bit. **Values:** **0x0**: `R_START_DET` interrupt is inactive **0x1**: `R_START_DET` interrupt is active **Value After Reset:** 0x0 **Volatile:** true
[9]	`R_STOP_DET`	R	See `IC_RAW_INTR_STAT` for a detailed description of `R_STOP_DET` bit. **Values:** **0x0**: `R_STOP_DET` interrupt is inactive **0x1**: `R_STOP_DET` interrupt is active

Table 13-139 Fields for register: `IC_INTR_STAT` (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0 **Volatile:** true
[8]	`R_ACTIVITY`	R	See `IC_RAW_INTR_STAT` for a detailed description of `R_ACTIVITY` bit. **Values:** **0x0**: `R_ACTIVITY` interrupt is inactive **0x1**: `R_ACTIVITY` interrupt is active **Value After Reset:** 0x0 **Volatile:** true
[7]	`R_RX_DONE`	R	See `IC_RAW_INTR_STAT` for a detailed description of `R_RX_DONE` bit. **Values:** **0x0**: `R_RX_DONE` interrupt is inactive **0x1**: `R_RX_DONE` interrupt is active **Value After Reset:** 0x0 **Volatile:** true
[6]	`R_TX_ABRT`	R	See `IC_RAW_INTR_STAT` for a detailed description of `R_TX_ABRT` bit. **Values:** **0x0**: `R_TX_ABRT` interrupt is inactive **0x1**: `R_TX_ABRT` interrupt is active **Value After Reset:** 0x0 **Volatile:** true
[5]	`R_RD_REQ`	R	See `IC_RAW_INTR_STAT` for a detailed description of `R_RD_REQ` bit. **Values:** **0x0**: `R_RD_REQ` interrupt is inactive **0x1**: `R_RD_REQ` interrupt is active **Value After Reset:** 0x0 **Volatile:** true
[4]	`R_TX_EMPTY`	R	See `IC_RAW_INTR_STAT` for a detailed description of `R_TX_EMPTY` bit. **Values:** **0x0**: `R_TX_EMPTY` interrupt is inactive **0x1**: `R_TX_EMPTY` interrupt is active **Value After Reset:** 0x0 **Volatile:** true
[3]	`R_TX_OVER`	R	See `IC_RAW_INTR_STAT` for a detailed description of `R_TX_OVER` bit. **Values:** **0x0**: `R_TX_OVER` interrupt is inactive **0x1**: `R_TX_OVER` interrupt is active **Value After Reset:** 0x0

Table 13-139 Fields for register: `IC_INTR_STAT` (continued)

Bits	Name	R/W	Description
			Volatile: true
[2]	`R_RX_FULL`	R	See `IC_RAW_INTR_STAT` for a detailed description of `R_RX_FULL` bit. **Values:** **0x0**: `R_RX_FULL` interrupt is inactive **0x1**: `R_RX_FULL` interrupt is active **Value After Reset:** 0x0 **Volatile:** true
[1]	`R_RX_OVER`	R	See `IC_RAW_INTR_STAT` for a detailed description of `R_RX_OVER` bit. **Values:** **0x0**: `R_RX_OVER` interrupt is inactive **0x1**: `R_RX_OVER` interrupt is active **Value After Reset:** 0x0 **Volatile:** true
[0]	`R_RX_UNDER`	R	See `IC_RAW_INTR_STAT` for a detailed description of `R_RX_UNDER` bit. **Values:** **0x0**: `RX_UNDER` interrupt is inactive **0x1**: `RX_UNDER` interrupt is active **Value After Reset:** 0x0 **Volatile:** true

13.5.2.2.13. I2C Interrupt Mask Register (`IC_INTR_MASK`)

The `IC_INTR_MASK` register is at offset 0x30.

These bits mask their corresponding interrupt status bits. This register is active low; a value of 0 masks the interrupt, whereas a value of 1 unmasks the interrupt.

The following table shows the register bit assignments.

Table 13-140 Fields for register: `IC_INTR_MASK`

Bits	Name	R/W	Description
[31:15]			Reserved
[14]	`M_SCL_STUCK_AT_LOW`	R/W	This bit masks the `R_SCL_STUCK_AT_LOW` interrupt in `IC_INTR_STAT` register. **Values:** **0x0**: `SCL_STUCK_AT_LOW` interrupt is masked **0x1**: `SCL_STUCK_AT_LOW` interrupt is unmasked **Value After Reset:** 0x1
[13:12]			Reserved
[11]	`M_GEN_CALL`	R/W	This bit masks the `R_GEN_CALL` interrupt in `IC_INTR_STAT` register.

Table 13-140 Fields for register: `IC_INTR_MASK` (continued)

Bits	Name	R/W	Description
			Values: **0x0**: `GEN_CALL` interrupt is masked **0x1**: `GEN_CALL` interrupt is unmasked **Value After Reset:** 0x1
[10]	`M_START_DET`	R/W	This bit masks the `R_START_DET` interrupt in `IC_INTR_STAT` register. **Values:** **0x0**: `START_DET` interrupt is masked **0x1**: `START_DET` interrupt is unmasked **Value After Reset:** 0x0
[9]	`M_STOP_DET`	R/W	This bit masks the `R_STOP_DET` interrupt in `IC_INTR_STAT` register. **Values:** **0x0**: `STOP_DET` interrupt is masked **0x1**: `STOP_DET` interrupt is unmasked **Value After Reset:** 0x0
[8]	`M_ACTIVITY`	R/W	This bit masks the `R_ACTIVITY` interrupt in `IC_INTR_STAT` register. **Values:** **0x0**: `ACTIVITY` interrupt is masked **0x1**: `ACTIVITY` interrupt is unmasked **Value After Reset:** 0x0
[7]	`M_RX_DONE`	R/W	This bit masks the `R_RX_DONE` interrupt in `IC_INTR_STAT` register. **Values:** **0x0**: `RX_DONE` interrupt is masked **0x1**: `RX_DONE` interrupt is unmasked **Value After Reset:** 0x1
[6]	`M_TX_ABRT`	R/W	This bit masks the `R_TX_ABRT` interrupt in `IC_INTR_STAT` register. **Values:** **0x0**: `TX_ABORT` interrupt is masked **0x1**: `TX_ABORT` interrupt is unmasked **Value After Reset:** 0x1
[5]	`M_RD_REQ`	R/W	This bit masks the `R_RD_REQ` interrupt in `IC_INTR_STAT` register. **Values:** **0x0**: `RD_REQ` interrupt is masked **0x1**: `RD_REQ` interrupt is unmasked **Value After Reset:** 0x1
[4]	`M_TX_EMPTY`	R/W	This bit masks the `R_TX_EMPTY` interrupt in `IC_INTR_STAT` register. **Values:** **0x0**: `TX_EMPTY` interrupt is masked **0x1**: `TX_EMPTY` interrupt is unmasked

Table 13-140 Fields for register: `IC_INTR_MASK` (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x1
[3]	`M_TX_OVER`	R/W	This bit masks the `R_TX_OVER` interrupt in `IC_INTR_STAT` register. **Values:** **0x0**: `TX_OVER` interrupt is masked **0x1**: `TX_OVER` interrupt is unmasked **Value After Reset:** 0x1
[2]	`M_RX_FULL`	R/W	This bit masks the `R_RX_FULL` interrupt in `IC_INTR_STAT` register. **Values:** **0x0**: `RX_FULL` interrupt is masked **0x1**: `RX_FULL` interrupt is unmasked **Value After Reset:** 0x1
[1]	`M_RX_OVER`	R/W	This bit masks the `R_RX_OVER` interrupt in `IC_INTR_STAT` register. **Values:** **0x0**: `RX_OVER` interrupt is masked **0x1**: `RX_OVER` interrupt is unmasked **Value After Reset:** 0x1
[0]	`M_RX_UNDER`	R/W	This bit masks the `R_RX_UNDER` interrupt in `IC_INTR_STAT` register. **Values:** **0x0**: `RX_UNDER` interrupt is masked **0x1**: `RX_UNDER` interrupt is unmasked **Value After Reset:** 0x1

13.5.2.2.14. I2C Raw Interrupt Status Register (`IC_RAW_INTR_STAT`)

The `IC_RAW_INTR_STAT` register is at offset 0x34.

Unlike the `IC_INTR_STAT` register, these bits are not masked so they always show the true status of the I2C controller.

The following table shows the register bit assignments.

Table 13-141 Fields for register: `IC_RAW_INTR_STAT`

Bits	Name	R/W	Description
[31:15]			Reserved
[14]	`SCL_STUCK_AT_LOW`	R	Indicates whether the SCL Line is stuck at low for the `IC_SCL_STUCK_LOW_TIMEOUT` number of `i2c*_clk` periods. **Values:** **0x0**: `SCL_STUCK_AT_LOW` interrupt is inactive **0x1**: `SCL_STUCK_AT_LOW` interrupt is active **Value After Reset:** 0x0 **Volatile:** true

Table 13-141 Fields for register: `IC_RAW_INTR_STAT` (continued)

Bits	Name	R/W	Description
[13:12]			Reserved
[11]	`GEN_CALL`	R	Set only when a General Call address is received and it is acknowledged. It stays set until it is cleared either by disabling I2C or when the CPU reads bit 0 of the `IC_CLR_GEN_CALL` register. The I2C stores the received data in the Rx buffer. **Values:** **0x0**: `GEN_CALL` interrupt is inactive **0x1**: `GEN_CALL` interrupt is active **Value After Reset:** 0x0 **Volatile:** true
[10]	`START_DET`	R	Indicates whether a START or RESTART condition has occurred on the I2C interface regardless of whether the I2C is operating in Slave or Master mode. **Values:** **0x0**: `START_DET` interrupt is inactive **0x1**: `START_DET` interrupt is active **Value After Reset:** 0x0 **Volatile:** true
[9]	`STOP_DET`	R	Indicates whether a STOP condition has occurred on the I2C interface regardless of whether the I2C is operating in Slave or Master mode. **In Slave Mode:** If `IC_CON[7]=1`, the `STOP_DET` interrupt will be issued only if Slave is addressed. During a general call address, this Slave does not issue a `STOP_DET` interrupt if `IC_CON`.`STOP_DET_IFADDRESSED=1`, even if the Slave responds to the general call address by generating ACK. The `STOP_DET` interrupt is generated only when the transmitted address matches the *Slave Address* (**SAR**). If `IC_CON[7]=0`, the `STOP_DET` interrupt is issued irrespective of whether it is being addressed. **In Master Mode:** If `IC_CON[10]=1`, the `STOP_DET` interrupt will be issued only if Master is active. If `IC_CON[10]=0`, he `STOP_DET` interrupt will be issued irrespective of whether Master is active or not. **Values:** **0x0**: `STOP_DET` interrupt is inactive **0x1**: `STOP_DET` interrupt is active **Value After Reset:** 0x0 **Volatile:** true

Table 13-141 Fields for register: `IC_RAW_INTR_STAT` (continued)

Bits	Name	R/W	Description
[8]	`ACTIVITY`	R	This bit captures the I2C activity and stays set until it is cleared. There are four ways to clear it: • Disabling the I2C • Reading the `IC_CLR_ACTIVITY` register • Reading the `IC_CLR_INTR` register • System reset Once this bit is set, it stays set unless one of the four methods is used to clear it. Even if the I2C module is idle, this bit remains set until cleared, indicating that there was activity on the bus. **Values:** **0x0**: `RAW_INTR_ACTIVITY` interrupt is active **0x1**: `RAW_INTR_ACTIVITY` interrupt is inactive **Value After Reset:** 0x0 **Volatile:** true
[7]	`RX_DONE`	R	When the I2C is acting as a Slave-transmitter, this bit is set to 1 if the Master does not acknowledge a transmitted byte. This occurs on the last byte of the transmission, indicating that the transmission is done. **Values:** **0x0**: `RX_DONE` interrupt is active **0x1**: `RX_DONE` interrupt is inactive **Value After Reset:** 0x0 **Volatile:** true
[6]	`TX_ABRT`	R	This bit indicates if the I2C controller, as an I2C transmitter, is unable to complete the intended actions on the contents of the transmit FIFO. This situation can occur both on an I2C Master or on an I2C Slave, and is referred to as a 'transmit abort'. When this bit is set to 1, the `IC_TX_ABRT_SOURCE` register indicates the reason why the transmit abort takes places. The I2C flushes/resets/empties only the Tx FIFO whenever there is a transmit abort caused by any of the events tracked by the `IC_TX_ABRT_SOURCE` register. The Tx FIFO remains in this flushed state until the register `IC_CLR_TX_ABRT` is read. Once this read is performed, the Tx FIFO is then ready to accept more data bytes from the Register Interface. Rx FIFO is also flushed. **Values:** **0x0**: `TX_ABRT` interrupt is inactive **0x1**: `TX_ABRT` interrupt is active **Value After Reset:** 0x0 **Volatile:** true

Table 13-141 Fields for register: `IC_RAW_INTR_STAT` (continued)

Bits	Name	R/W	Description
[5]	`RD_REQ`	R	This bit is set to 1 when I2C Controller is acting as a Slave and another I2C Master is attempting to read data from I2C Controller. The I2C Controller holds the I2C bus in a wait state (SCL=0) until this interrupt is serviced, that means the Slave has been addressed by a remote Master asking for data to be transferred. The processor must respond to this interrupt and then write the requested data to the `IC_DATA_CMD` register. This bit is set to 0 just after the processor reads the `IC_CLR_RD_REQ` register. **Values:** **0x0**: `RD_REQ` interrupt is inactive **0x1**: `RD_REQ` interrupt is active **Value After Reset:** 0x0 **Volatile:** true
[4]	`TX_EMPTY`	R	The behavior of the `TX_EMPTY` interrupt status differs based on the `TX_EMPTY_CTRL` setting in the `IC_CON` register. If `TX_EMPTY_CTRL = 0` this bit is set to 1 when the transmit buffer is at the threshold value set in the `IC_TX_TL` register or below it. If `TX_EMPTY_CTRL = 1` this bit is set to 1 when the transmit buffer is at the threshold value set in the `IC_TX_TL` register or below it and the transmission of the address/data from the internal shift register for the most recently popped command is completed. It is automatically cleared by hardware when the buffer level goes above the threshold. When `IC_ENABLE[0]` is set to 0, the Tx FIFO is flushed and held in reset. There the Tx FIFO looks like it has no data within it, so this bit is set to 1, provided there is activity in the Master or Slave state machines. When there is no longer any activity, then with `IC_EN=0` in `IC_ENABLE_STATUS` register, this bit is set to 0. **Values:** **0x0**: `TX_EMPTY` interrupt is inactive **0x1**: `TX_EMPTY` interrupt is active **Value After Reset:** 0x0 **Volatile:** true
[3]	`TX_OVER`	R	Set during transmit if the transmit buffer is filled to 8 bits and the processor attempts to issue another I2C command by writing to the `IC_DATA_CMD` register. When the module is disabled, this bit keeps its level until the Master or Slave state machines go into idle, and when `IC_EN` in `IC_ENABLE_STATUS` register goes to 0, this interrupt is cleared. **Values:** **0x0**: `TX_OVER` interrupt is inactive **0x1**: `TX_OVER` interrupt is active **Value After Reset:** 0x0 **Volatile:** true

Table 13-141 Fields for register: `IC_RAW_INTR_STAT` (continued)

Bits	Name	R/W	Description
[2]	`RX_FULL`	R	Set when the receive buffer reaches or goes above the `RX_TL` threshold in the `IC_RX_TL` register. It is automatically cleared by hardware when buffer level goes below the threshold. If the module is disabled (`IC_ENABLE[0]=0`), the Rx FIFO is flushed and held in reset; therefore the Rx FIFO is not full. So this bit is cleared once the `IC_ENABLE[0]` is programmed with a zero, regardless of the activity that continues. **Values:** **0x0**: `RX_FULL` interrupt is inactive **0x1**: `RX_FULL` interrupt is active **Value After Reset:** 0x0 **Volatile:** true
[1]	`RX_OVER`	R	Set if the receive buffer is completely filled to 8 and an additional byte is received from an external I2C device. The I2C Controller acknowledges this, but any data bytes received after the FIFO is full are lost. If the module is disabled (`IC_ENABLE[0]=0`), this bit keeps its level until the Master or Slave state machines go into idle, and when `IC_EN` in `IC_ENABLE_STATUS` register goes to 0, this interrupt is cleared. **Values:** **0x0**: `RX_OVER` interrupt is inactive **0x1**: `RX_OVER` interrupt is active **Value After Reset:** 0x0 **Volatile:** true
[0]	`RX_UNDER`	R	Set if the processor attempts to read the receive buffer when it is empty by reading from the `IC_DATA_CMD` register. If the module is disabled (`IC_ENABLE[0]=0`), this bit keeps its level until the Master or Slave state machines go into idle, and when `IC_EN` in `IC_ENABLE_STATUS` register goes to 0, this interrupt is cleared. **Values:** **0x0**: `RX_UNDER` interrupt is inactive **0x1**: `RX_UNDER` interrupt is active **Value After Reset:** 0x0 **Volatile:** true

13.5.2.2.15. I2C Receive FIFO Threshold Register (`IC_RX_TL`)

The `IC_RX_TL` register is at offset 0x38.

Receive FIFO Threshold Level.

The following table shows the register bit assignments.

Table 13-142 Fields for register: IC_RX_TL

Bits	Name	R/W	Description
[31:8]			Reserved
[7:0]	RX_TL	R/W	Controls the level of entries (or above) that triggers the RX_FULL interrupt (RX_FULL bit in IC_RAW_INTR_STAT register). The valid range is 0-255, with the additional restriction that hardware does not allow this value to be set to a value larger than the depth of the buffer (8). If an attempt is made to do that, the actual value set will be the maximum depth of the buffer. A value of 0 sets the threshold for 1 entry, and a value of 255 sets the threshold for 256 entries. **Value After Reset:** 0x0

13.5.2.2.16. I2C Transmit FIFO Threshold Register (IC_TX_TL)

The IC_TX_TL register is at offset 0x3C.

Transmit FIFO Threshold Level.

The following table shows the register bit assignments.

Table 13-143 Fields for register: IC_TX_TL

Bits	Name	R/W	Description
[31:8]			Reserved
[7:0]	TX_TL	R/W	Controls the level of entries (or below) that trigger the TX_EMPTY interrupt (TX_EMPTY bit in IC_RAW_INTR_STAT register). The valid range is 0-255, with the additional restriction that it may not be set to value larger than the depth of the buffer (8). If an attempt is made to do that, the actual value set will be the maximum depth of the buffer. A value of 0 sets the threshold for 0 entries, and a value of 255 sets the threshold for 255 entries. **Value After Reset:** 0x0

13.5.2.2.17. Clear Combined and Individual Interrupt Register (IC_CLR_INTR)

The IC_CLR_INTR register is at offset 0x40.

The following table shows the register bit assignments.

Table 13-144 Fields for register: IC_CLR_INTR

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	CLR_INTR	R	Read this register to clear the combined interrupt, all individual interrupts, and the IC_TX_ABRT_SOURCE register. This bit does not clear hardware clearable interrupts but software clearable interrupts. Refer to the IC_TX_ABRT_SOURCE.ABRT_SBYTE_NORSTRT register for an exception to clearing IC_TX_ABRT_SOURCE. **Value After Reset:** 0x0 **Volatile:** true

13.5.2.2.18. Clear RX_UNDER Interrupt Register (IC_CLR_RX_UNDER)

The IC_CLR_RX_UNDER register is at offset 0x44.

The following table shows the register bit assignments.

Table 13-145 Fields for register: IC_CLR_RX_UNDER

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	CLR_RX_UNDER	R	Read this register to clear the RX_UNDER interrupt (RX_UNDER bit of the IC_RAW_INTR_STAT register). **Value After Reset:** 0x0 **Volatile:** true

13.5.2.2.19. Clear RX_OVER Interrupt Register (IC_CLR_RX_OVER)

The IC_CLR_RX_OVER register is at offset 0x48.

The following table shows the register bit assignments.

Table 13-146 Fields for register: IC_CLR_RX_OVER

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	CLR_RX_OVER	R	Read this register to clear the RX_OVER interrupt (RX_OVER bit of the IC_RAW_INTR_STAT register). **Value After Reset:** 0x0 **Volatile:** true

13.5.2.2.20. Clear TX_OVER Interrupt Register (IC_CLR_TX_OVER)

The IC_CLR_TX_OVER register is at offset 0x4C.

The following table shows the register bit assignments.

Table 13-147 Fields for register: IC_CLR_TX_OVER

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	CLR_TX_OVER	R	Read this register to clear the TX_OVER interrupt (TX_OVER bit of the IC_RAW_INTR_STAT register). **Value After Reset:** 0x0 **Volatile:** true

13.5.2.2.21. Clear RD_REQ Interrupt Register (IC_CLR_RD_REQ)

The IC_CLR_RD_REQ register is at offset 0x50.

The following table shows the register bit assignments.

Table 13-148 Fields for register: `IC_CLR_RD_REQ`

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	CLR_RD_REQ	R	Read this register to clear the RD_REQ interrupt (RD_REQ bit of the IC_RAW_INTR_STAT register). **Value After Reset:** 0x0 **Volatile:** true

13.5.2.2.22. Clear `TX_ABRT` Interrupt Register (`IC_CLR_TX_ABRT`)

The IC_CLR_TX_ABRT register is at offset 0x54.

The following table shows the register bit assignments.

Table 13-149 Fields for register: `IC_CLR_TX_ABRT`

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	CLR_TX_ABRT	R	Read this register to clear the TX_ABRT interrupt (TX_ABRT bit of the IC_RAW_INTR_STAT register), and the IC_TX_ABRT_SOURCE register. This also releases the Tx FIFO from the flushed/reset state, allowing more writes to the Tx FIFO. Refer to the IC_TX_ABRT_SOURCE.ABRT_SBYTE_NORSTRT register for an exception to clearing of IC_TX_ABRT_SOURCE. **Value After Reset:** 0x0 **Volatile:** true

13.5.2.2.23. Clear `RX_DONE` Interrupt Register (`IC_CLR_RX_DONE`)

The IC_CLR_RX_DONE register is at offset 0x58.

The following table shows the register bit assignments.

Table 13-150 Fields for register: `IC_CLR_RX_DONE`

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	CLR_RX_DONE	R	Read this register to clear the RX_DONE interrupt (RX_DONE bit of the IC_RAW_INTR_STAT register). **Value After Reset:** 0x0 **Volatile:** true

13.5.2.2.24. Clear `ACTIVITY` Interrupt Register (`IC_CLR_ACTIVITY`)

The IC_CLR_ACTIVITY register is at offset 0x5C.

The following table shows the register bit assignments.

Table 13-151 Fields for register: `IC_CLR_ACTIVITY`

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	`CLR_ACTIVITY`	R	Reading this register clears the `ACTIVITY` interrupt if the I2C Controller is not active anymore. If the I2C module is still active on the bus, the `ACTIVITY` interrupt bit continues to be set. It is automatically cleared by hardware if the module is disabled and if there is no further activity on the bus. The value read from this register to get status of the `ACTIVITY` interrupt (`ACTIVITY` bit of the `IC_RAW_INTR_STAT` register). **Value After Reset:** 0x0 **Volatile:** true

13.5.2.2.25. Clear `STOP_DET` Interrupt Register (`IC_CLR_STOP_DET`)

The `IC_CLR_STOP_DET` register is at offset 0x60.

The following table shows the register bit assignments.

Table 13-152 Fields for register: `IC_CLR_STOP_DET`

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	`CLR_STOP_DET`	R	Read this register to clear the `STOP_DET` interrupt (`STOP_DET` bit of the `IC_RAW_INTR_STAT` register). **Value After Reset:** 0x0 **Volatile:** true

13.5.2.2.26. Clear `START_DET` Interrupt Register (`IC_CLR_START_DET`)

The `IC_CLR_START_DET` register is at offset 0x64.

The following table shows the register bit assignments.

Table 13-153 Fields for register: `IC_CLR_START_DET`

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	`CLR_START_DET`	R	Read this register to clear the `START_DET` interrupt (`START_DET` bit of the `IC_RAW_INTR_STAT` register). **Value After Reset:** 0x0 **Volatile:** true

13.5.2.2.27. Clear `GEN_CALL` Interrupt Register (`IC_CLR_GEN_CALL`)

The `IC_CLR_GEN_CALL` register is at offset 0x68.

The following table shows the register bit assignments.

Table 13-154 Fields for register: `IC_CLR_GEN_CALL`

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	`CLR_GEN_CALL`	R	Read this register to clear the `GEN_CALL` interrupt (`GEN_CALL` bit of `IC_RAW_INTR_STAT` register). **Value After Reset:** 0x0 **Volatile:** true

13.5.2.2.28. I2C ENABLE Register (`IC_ENABLE`)

The `IC_ENABLE` register is at offset 0x6C.

The following table shows the register bit assignments.

Table 13-155 Fields for register: `IC_ENABLE`

Bits	Name	R/W	Description
[31:4]			Reserved
[3]	`SDA_STUCK_RECOVERY_ENABLE`	R/W	If SDA is stuck at low indicated through the `TX_ABORT` interrupt (`IC_TX_ABRT_SOURCE[17]`), then this bit is used as a control knob to initiate the SDA Recovery Mechanism (that is, send at most 9 SCL clocks and STOP to release the SDA line) and then this bit gets auto clear **Values:** **0x0**: Master disabled the SDA stuck at low recovery mechanism. **0x1**: Master initiates the SDA stuck at low recovery mechanism. **Value After Reset:** 0x0
[2]	`TX_CMD_BLOCK`	R/W	In Master mode: **Values:** **0x0**: The transmission of data starts on the I2C bus automatically, as soon as the first data is available in the Tx FIFO. **0x1**: Blocks the transmission of data on the I2C bus even if Tx FIFO has data to transmit. To block the execution of Master commands, set the `TX_CMD_BLOCK` bit only when Tx FIFO is empty (`IC_STATUS[2]==1`) and Master is in Idle state (`IC_STATUS[5]==0`). Any further commands put in the Tx FIFO are not executed until `TX_CMD_BLOCK` bit is unset. **Value After Reset:** 0x1
[1]	`ABORT`	R/W	When set, the controller initiates the transfer abort. The software can abort the I2C transfer in Master mode by setting this bit. The software can set this bit only when

Table 13-155 Fields for register: `IC_ENABLE` (continued)

Bits	Name	R/W	Description
			`ENABLE` is already set; otherwise, the controller ignores any write to `ABORT` bit. The software cannot clear the `ABORT` bit once set. In response to an `ABORT`, the controller issues a STOP and flushes the Tx FIFO after completing the current transfer, then sets the `TX_ABORT` interrupt after the abort operation. The `ABORT` bit is cleared automatically after the abort operation. For a detailed description on how to abort the I2C transfers, refer to Aborting I2C Transfers in the Operation Modes **Values:** **0x0**: ABORT operation is not in progress or done **0x1**: ABORT operation is in progress **Value After Reset:** 0x0
[0]	ENABLE	R/W	Controls whether the I2C Controller is enabled. Software can disable I2C while it is active. However, it is important that care be taken to ensure that I2C is disabled properly. A recommended procedure is described in Disabling I2C Controller in thr Operation Modes. When I2C Controller is disabled, the following occurs: • The Tx FIFO and Rx FIFO get flushed. • Status bits in the `IC_INTR_STAT` register are still active until I2C goes into IDLE state. If the module is transmitting, it stops as well as deletes the contents of the transmit buffer after the current transfer is complete. If the module is receiving, the I2C Controller stops the current transfer at the end of the current byte and does not acknowledge the transfer. **Values:** **0x0**: Disables the I2C Controller (Tx and Rx FIFOs are held in an erased state) **0x1**: Enables the I2C Controller **Value After Reset:** 0x0

13.5.2.2.29. I2C STATUS Register (`IC_STATUS`)

The `IC_STATUS` register is at offset 0x70.

This is a read-only register used to indicate the current transfer status and FIFO status. The status register may be read at any time. None of the bits in this register request an interrupt.

When the I2C Controller is disabled by writing 0 to the `IC_ENABLE`.`ENABLE`:

- Bits 1 and 2 are set to 1
- Bit 3 is set to 0

When the Master or Slave state machines go to idle and `IC_EN` = 0x0 (`IC_ENABLE_STATUS` register):

- Bits 5 and 6 are set to 0

The following table shows the register bit assignments.

Table 13-156 Fields for register: `IC_STATUS`

Bits	Name	R/W	Description
[31:12]			Reserved
[11]	`SDA_STUCK_NOT_RECOVERED`	R	This bit indicates that SDA stuck at low is not recovered after the recovery mechanism. In Slave mode, this register bit is not applicable. **Values:** **0x0**: SDA Stuck at low is not recovered after recovery mechanism **0x1**: SDA Stuck at low is recovered after recovery mechanism **Value After Reset:** 0x0 **Volatile:** true
[10]			Reserved
[9]	`SLV_HOLD_TX_FIFO_EMPTY`	R	This bit indicates the BUS Hold in Slave mode for the Read request when the Tx FIFO is empty. The Bus is in hold until the Tx FIFO has data to Transmit for the read request. **Values:** **0x0**: Slave is not holding the bus or Bus hold is not due to Tx FIFO is empty **0x1**: Slave holds the bus due to Tx FIFO is empty **Value After Reset:** 0x0 **Volatile:** true
[8:7]			Reserved
[6]	`SLV_ACTIVITY`	R	Slave FSM Activity Status When the Slave *Finite State Machine* (**FSM**) is not in the IDLE state, this bit is set. **Values:** **0x0**: Slave FSM is in IDLE state so the Slave part of the I2C Controller is not Active **0x1**: Slave FSM is not in IDLE state so the Slave part of the I2C Controller is Active **Value After Reset:** 0x0 **Volatile:** true
[5]	`MST_ACTIVITY`	R	Master FSM Activity Status When the Master *Finite State Machine* (**FSM**) is not in the IDLE state, this bit is set. **Values:**

Table 13-156 Fields for register: `IC_STATUS` (continued)

Bits	Name	R/W	Description
			0x0: Master FSM is in IDLE state so the Master part of the I2C Controller is not Active **0x1**: Master FSM is not in IDLE state so the Master part of I2C Controller is Active The `ACTIVITY` bit 0 in `IC_STATUS` register is the OR of `SLV_ACTIVITY` and `MST_ACTIVITY` bits. **Value After Reset:** 0x0 **Volatile:** true
[4]	`RFF`	R	Receive FIFO Completely Full When the receive FIFO is completely full, this bit is set. When the receive FIFO contains one or more empty locations, this bit is cleared. **Values:** **0x0**: Rx FIFO is not full **0x1**: Rx FIFO is full **Value After Reset:** 0x0 **Volatile:** true
[3]	`RFNE`	R	Receive FIFO Not Empty This bit is set when the receive FIFO contains one or more entries; it is cleared when the receive FIFO is empty. **Values:** **0x0**: Rx FIFO is empty **0x1**: Rx FIFO is not empty **Value After Reset:** 0x0 **Volatile:** true
[2]	`TFE`	R	Transmit FIFO Completely Empty When the transmit FIFO is completely empty, this bit is set. When it contains one or more valid entries, this bit is cleared. This bit field does not request an interrupt. **Values:** **0x0**: Tx FIFO is not empty **0x1**: Tx FIFO is empty **Value After Reset:** 0x1 **Volatile:** true
[1]	`TFNF`	R	Transmit FIFO Not Full Set when the transmit FIFO contains one or more empty locations, and is cleared when the FIFO is full. **Values:** **0x0**: Tx FIFO is full **0x1**: Tx FIFO is not full

Table 13-156 Fields for register: `IC_STATUS` (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x1 **Volatile:** true
[0]	`ACTIVITY`	R	I2C Activity Status **Values:** **0x0**: I2C is idle **0x1**: I2C is active **Value After Reset:** 0x0 **Volatile:** true

13.5.2.2.30. I2C Transmit FIFO Level Register (`IC_TXFLR`)

The `IC_TXFLR` register is at offset 0x74.

This register contains the number of valid data entries in the transmit FIFO buffer. It is cleared whenever:

- The I2C Controller is disabled
- There is a transmit abort — that is, `TX_ABRT` bit is set in the `IC_RAW_INTR_STAT` register
- The Slave bulk transmit mode is aborted

The register increments whenever data is placed into the transmit FIFO and decrements when data is taken from the transmit FIFO.

The following table shows the register bit assignments.

Table 13-157 Fields for register: `IC_TXFLR`

Bits	Name	R/W	Description
[31:4]			Reserved
[3:0]	`TXFLR`	R	Transmit FIFO Level. Contains the number of valid data entries in the transmit FIFO. **Value After Reset:** 0x0 **Volatile:** true

13.5.2.2.31. I2C Receive FIFO Level Register (`IC_RXFLR`)

The `IC_RXFLR` register is at offset 0x78.

This register contains the number of valid data entries in the receive FIFO buffer. It is cleared whenever:

- The I2C Controller is disabled
- Whenever there is a transmit abort caused by any of the events tracked in `IC_TX_ABRT_SOURCE`

The register increments whenever data is placed into the receive FIFO and decrements when data is taken from the receive FIFO.

The following table shows the register bit assignments.

Table 13-158 Fields for register: IC_RXFLR

Bits	Name	R/W	Description
[31:4]			Reserved
[3:0]	RXFLR	R	Receive FIFO Level. Contains the number of valid data entries in the receive FIFO. **Value After Reset:** 0x0 **Volatile:** true

13.5.2.2.32. I2C SDA Hold Time Length Register (IC_SDA_HOLD)

The IC_SDA_HOLD register is at offset 0x7C.

The IC_SDA_TX_HOLD bits of this register are used to control the hold time of SDA during transmit in both Slave and Master mode (after SCL goes from HIGH to LOW).

The IC_SDA_RX_HOLD bits of this register are used to extend the SDA transition (if any) whenever SCL is HIGH in the receiver in either Master or Slave mode.

Writes to this register succeed only when IC_ENABLE[0]=0.

The values in this register are in units of i2c*_clk period. The value programmed in IC_SDA_TX_HOLD must be greater than the minimum hold time in each mode one cycle in Master mode, seven cycles in Slave mode for the value to be implemented.

The programmed SDA hold time during transmit (IC_SDA_TX_HOLD) cannot exceed at any time the duration of the low part of SCL. Therefore the programmed value cannot be larger than N_SCL_LOW-2, where N_SCL_LOW is the duration of the low part of the SCL period measured in i2c*_clk cycles.

The following table shows the register bit assignments.

Table 13-159 Fields for register: IC_SDA_HOLD

Bits	Name	R/W	Description
[31:24]			Reserved
[23:16]	IC_SDA_RX_HOLD	R/W	Sets the required SDA hold time in units of i2c*_clk period, when the I2C Controller acts as a receiver. **Value After Reset:** 0x0
[15:0]	IC_SDA_TX_HOLD	R/W	Sets the required SDA hold time in units of i2c*_clk period, when the I2C Controller acts as a transmitter. **Value After Reset:** 0x1

13.5.2.2.33. I2C Transmit Abort Source Register (IC_TX_ABRT_SOURCE)

The IC_TX_ABRT_SOURCE register is at offset 0x80.

This register has 32 bits that indicate the source of the TX_ABRT bit. Except for ABRT_SBYTE_NORSTRT, this register is cleared whenever the IC_CLR_TX_ABRT register or the IC_CLR_INTR register is read. To clear ABRT_SBYTE_NORSTRT bit, the source of the ABRT_SBYTE_NORSTRT must be fixed first; RESTART must be enabled (IC_CON[5]=1), the SPECIAL bit must be cleared (IC_TAR[11]), or the GC_OR_START bit must be cleared (IC_TAR[10]).

Once the source of the ABRT_SBYTE_NORSTRT is fixed, then this bit can be cleared in the same manner as other bits in this register. If the source of the ABRT_SBYTE_NORSTRT is not fixed before attempting to clear this bit, ABRT_SBYTE_NORSTRT clears for one cycle and is then re-asserted.

The following table shows the register bit assignments.

Table 13-160 Fields for register: IC_TX_ABRT_SOURCE

Bits	Name	R/W	Description
[31:23]	TX_FLUSH_CNT	R	This field indicates the number of Tx FIFO Data Commands which are flushed due to TX_ABRT interrupt. It is cleared whenever I2C is disabled. **Role of the I2C Controller:** Master-Transmitter or Slave-Transmitter **Value After Reset:** 0x0 **Volatile:** true
[22:18]			Reserved
[17]	ABRT_SDA_STUCK_AT_LOW	R	This is a Master-mode-only bit Master detects the SDA Stuck at low for the IC_SDA_STUCK_AT_LOW_TIMEOUT value of i2c*_clk. **Role of the I2C Controller:** Master **Values:** **0x0**: This abort is not generated **0x1**: This abort is generated because of SDA stuck at low for IC_SDA_STUCK_AT_LOW_TIMEOUT value of i2c*_clk **Value After Reset:** 0x0 **Volatile:** true
[16]	ABRT_USER_ABRT	R	This is a Master-mode-only bit Master has detected the transfer abort (IC_ENABLE[1]) **Role of the I2C Controller:** Master-Transmitter **Values:** **0x0**: Transfer abort detected by Master — scenario not present **0x1**: Transfer abort detected by Master **Value After Reset:** 0x0 **Volatile:** true
[15]	ABRT_SLVRD_INTX	R	When the processor side responds to a Slave mode request for data to be transmitted to a remote Master and user writes a 1 in CMD of IC_DATA_CMD register. **Role of the I2C Controller:** Slave-Transmitter **Values:** **0x0**: Slave trying to transmit to remote Master in read mode — scenario not present **0x1**: Slave trying to transmit to remote Master in read mode

Table 13-160 Fields for register: `IC_TX_ABRT_SOURCE` (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0 **Volatile:** true
[14]	`ABRT_SLV_ARBLOST`	R	This field indicates that a Slave has lost the bus while transmitting data to a remote Master. `IC_TX_ABRT_SOURCE[12]` is set at the same time. Even though the Slave never 'owns' the bus, something could go wrong on the bus. This is a fail safe check. For instance, during a data transmission at the low-to-high transition of SCL, if what is on the data bus is not what is supposed to be transmitted, then the I2C Controller no longer own the bus. **Role of the I2C Controller:** Slave-Transmitter **Values:** **0x0**: Slave lost arbitration to remote Master — scenario not present **0x1**: Slave lost arbitration to remote Master **Value After Reset:** 0x0 **Volatile:** true
[13]	`ABRT_SLVFLUSH_TXFIFO`	R	This field specifies that the Slave has received a read command and some data exists in the Tx FIFO, so the Slave issues a `TX_ABRT` interrupt to flush old data in Tx FIFO. **Role of the I2C Controller:** Slave-Transmitter **Values:** **0x0**: Slave flushes existing data in Tx FIFO upon getting read command — scenario not present **0x1**: Slave flushes existing data in Tx FIFO upon getting read command **Value After Reset:** 0x0 **Volatile:** true
[12]	`ARB_LOST`	R	This field specifies that the Master has lost arbitration, or if `IC_TX_ABRT_SOURCE[14]` is also set, then the Slave transmitter has lost arbitration. **Role of the I2C Controller:** Master-Transmitter or Slave-Transmitter **Values:** **0x0**: Master or Slave-Transmitter lost arbitration — scenario not present **0x1**: Master or Slave-Transmitter lost arbitration **Value After Reset:** 0x0 **Volatile:** true
[11]	`ABRT_MASTER_DIS`	R	This field indicates that the User tries to initiate a Master operation with the Master mode disabled.

Table 13-160 Fields for register: `IC_TX_ABRT_SOURCE` (continued)

Bits	Name	R/W	Description
			Role of the I2C Controller: Master-Transmitter or Master-Receiver **Values:** **0x0**: User initiating Master operation when MASTER disabled — scenario not present **0x1**: User initiating Master operation when MASTER disabled **Value After Reset:** 0x0 **Volatile:** true
[10]	`ABRT_10B_RD_NORSTRT`	R	This field indicates that the restart is disabled (`IC_CON[5]=0`) and the Master sends a read command in 10-bit addressing mode. **Role of the I2C Controller:** Master-Receiver **Values:** **0x0**: Master is not attempting to read in 10-bit addressing mode with RESTART disabled **0x1**: Master is attempting to read in 10-bit addressing mode with RESTART disabled **Value After Reset:** 0x0 **Volatile:** true
[9]	`ABRT_SBYTE_NORSTRT`	R	To clear this bit, the source of the `ABRT_SBYTE_NORSTRT` must be fixed first; restart must be enabled (`IC_CON[5]=1`), the `SPECIAL` bit must be cleared (`IC_TAR[11]`), or the `GC_OR_START` bit must be cleared (`IC_TAR[10]`). Once the source of the `ABRT_SBYTE_NORSTRT` is fixed, then this bit can be cleared in the same manner as other bits in this register. If the source of the `ABRT_SBYTE_NORSTRT` is not fixed before attempting to clear this bit, `ABRT_SBYTE_NORSTRT` clears for one cycle and then gets reasserted. When this field is set to 1, the restart is disabled (`IC_CON[5] =0`) and the user is trying to send a START Byte. **Role of the I2C Controller:** Master **Values:** **0x0**: User is attempting to send START byte with RESTART disabled — scenario not present **0x1**: User is attempting to send START byte with RESTART disabled **Value After Reset:** 0x0 **Volatile:** true
[8]	`ABRT_HS_NORSTRT`	R	This field indicates that the restart is disabled (`IC_CON[5]=0`) and the user is trying to use the Master to transfer data in High Speed mode.

Table 13-160 Fields for register: `IC_TX_ABRT_SOURCE` (continued)

Bits	Name	R/W	Description
			Role of the I2C Controller: Master-Transmitter or Master-Receiver **Values:** **0x0**: User is attempting to switch Master to HS mode with RESTART disabled — scenario not present **0x1**: User is attempting to switch Master to HS mode with RESTART disabled **Value After Reset:** 0x0 **Volatile:** true
[7]	`ABRT_SBYTE_ACKDET`	R	This field indicates that the Master has sent a START Byte and the START Byte was acknowledged (wrong behavior). **Role of the I2C Controller:** Master **Values:** **0x0**: ACK detected for START byte — scenario not present **0x1**: ACK detected for START byte **Value After Reset:** 0x0 **Volatile:** true
[6]	`ABRT_HS_ACKDET`	R	This field indicates that the Master is in High Speed mode and the High Speed Master code was acknowledged (wrong behavior). **Role of the I2C Controller:** Master **Values:** **0x0**: HS Master code ACKed in HS Mode — scenario not present **0x1**: HS Master code ACKed in HS Mode **Value After Reset:** 0x0 **Volatile:** true
[5]	`ABRT_GCALL_READ`	R	This field indicates that I2C Controller in the Master mode has sent a General Call but the user programmed the byte following the General Call to be a read from the bus (`IC_DATA_CMD[9]` is set to 1). **Role of the I2C Controller:** Master-Transmitter **Values:** **0x0**: `GCALL` is followed by read from bus — scenario not present **0x1**: `GCALL` is followed by read from bus **Value After Reset:** 0x0 **Volatile:** true
[4]	`ABRT_GCALL_NOACK`	R	This field indicates that the I2C Controller in Master mode has sent a General Call and no Slave on the bus acknowledged the General Call.

Table 13-160 Fields for register: `IC_TX_ABRT_SOURCE` (continued)

Bits	Name	R/W	Description
			Role of the I2C Controller: Master-Transmitter **Values:** **0x0**: `GCALL` is not ACKed by any Slave — scenario not present **0x1**: `GCALL` is not ACKed by any Slave **Value After Reset:** 0x0 **Volatile:** true
[3]	`ABRT_TXDATA_NOACK`	R	This field indicates the Master-mode only bit. When the Master receives an acknowledgement for the address, but when it sends data byte(s) following the address, it did not receive the acknowledge from the remote Slave(s). **Role of the I2C Controller:** Master-Transmitter **Values:** **0x0**: Transmitted data is not ACKed by addressed Slave — scenario not present **0x1**: Transmitted data is not ACKed by addressed Slave **Value After Reset:** 0x0 **Volatile:** true
[2]	`ABRT_10ADDR2_NOACK`	R	This field indicates that the Master is in 10-bit address mode and that the second address byte of the 10-bit address was not acknowledged by any Slave. **Role of the I2C Controller:** Master-Transmitter or Master-Receiver **Values:** **0x0**: This abort is not generated **0x1**: Byte 2 of 10-bit address is not ACKed by any Slave **Value After Reset:** 0x0 **Volatile:** true
[1]	`ABRT_10ADDR1_NOACK`	R	This field indicates that the Master is in 10-bit address mode and the first 10-bit address byte was not acknowledged by any Slave. **Role of the I2C Controller:** Master-Transmitter or Master-Receiver **Values:** **0x0**: This abort is not generated **0x1**: Byte 1 of 10-bit address is not ACKed by any Slave **Value After Reset:** 0x0 **Volatile:** true

Table 13-160 Fields for register: `IC_TX_ABRT_SOURCE` (continued)

Bits	Name	R/W	Description
[0]	ABRT_7B_ADDR_NOACK	R	This field indicates that the Master is in 7-bit addressing mode and the address sent was not acknowledged by any Slave. **Role of the I2C Controller:** Master-Transmitter or Master-Receiver **Values:** **0x0**: This abort is not generated **0x1**: This abort is generated because of NOACK for 7-bit address **Value After Reset:** 0x0 **Volatile:** true

13.5.2.2.34. DMA Control Register (`IC_DMA_CR`)

The `IC_DMA_CR` register is at offset 0x88.

This register is only valid when the I2C Controller is configured with a set of DMA Controller interface signals. When the I2C Controller is not configured for DMA operation, this register does not exist and writing to the register's address has no effect and reading from this register address will return zero. The register is used to enable the DMA Controller interface operation. There is a separate bit for transmit and receive. This can be programmed regardless of the state of `IC_ENABLE`.

The following table shows the register bit assignments.

Table 13-161 Fields for register: `IC_DMA_CR`

Bits	Name	R/W	Description
[31:2]			Reserved
[1]	TDMAE	R/W	Transmit DMA Enable This bit enables/disables the transmit FIFO DMA channel. **Values**: **0x0**: Transmit FIFO DMA channel disabled **0x1**: Transmit FIFO DMA channel enabled **Value After Reset:** 0x0
[0]	RDMAE	R/W	Receive DMA Enable This bit enables/disables the receive FIFO DMA channel. **Values**: **0x0**: Receive FIFO DMA channel disabled **0x1**: Receive FIFO DMA channel enabled **Value After Reset:** 0x0

13.5.2.2.35. DMA Transmit Data Level Register (`IC_DMA_TDLR`)

The `IC_DMA_TDLR` register is at offset 0x8C.

This register is only valid when the I2C Controller is configured with a set of DMA interface signals. When the I2C Controller is not configured for DMA operation, this register does not exist; writing to its address has no effect; reading from its address returns zero.

The following table shows the register bit assignments.

Table 13-162 Fields for register: `IC_DMA_TDLR`

Bits	Name	R/W	Description
[31:8]			Reserved
[7:0]	`DMATDL`	R/W	Transmit Data Level This bit field controls the level at which a DMA request is made by the transmit logic. It is equal to the watermark level; that is, the Transmit Request is made when the number of valid data entries in the transmit FIFO is equal to or below this field value, and `IC_DMA_CR.TDMAE` = 1. **Value After Reset**: 0x0

13.5.2.2.36. DMA Receive Data Level Register (`IC_DMA_RDLR`)

The `IC_DMA_RDLR` register is at offset 0x90.

This register is only valid when the I2C Controller is configured with a set of DMA interface signals. When the I2C Controller is not configured for DMA operation, this register does not exist; writing to its address has no effect; reading from its address returns zero.

The following table shows the register bit assignments.

Table 13-163 Fields for register: `IC_DMA_RDLR`

Bits	Name	R/W	Description
[31:3]			Reserved
[2:0]	`DMARDL`	R/W	Receive Data Level This bit field controls the level at which a DMA request is made by the receive logic. The watermark level = `DMARDL+1`; that is, Receive request is made when the number of valid data entries in the receive FIFO is equal to or more than this field value + 1, and `IC_DMA_CR.RDMAE=1`. For instance, when `DMARDL` is 0, then Receive request is made when 1 or more data entries are present in the receive FIFO. **Value After Reset:** 0x0

13.5.2.2.37. I2C SDA Setup Register (`IC_SDA_SETUP`)

The `IC_SDA_SETUP` register is at offset 0x94.

This register controls the amount of time delay (in the number of `i2c*_clk` clock periods) introduced in the rising edge of SCL — relative to SDA changing — when I2C Controller services a read request in a Slave-transmitter operation. The relevant I2C requirement is $t_{SU:DAT}$ (Data setup time). This register must be programmed with a value equal to or greater than 2.

Writes to this register succeed only when `IC_ENABLE[0] = 0`.

The following table shows the register bit assignments.

Table 13-164 Fields for register: `IC_SDA_SETUP`

Bits	Name	R/W	Description
[31:8]			Reserved
[7:0]	SDA_SETUP	R/W	It is recommended that if the required delay is 1000 ns, then for an `i2c*_clk` frequency of 10 MHz, `IC_SDA_SETUP` should be programmed to a value of 11. `IC_SDA_SETUP` must be programmed with a minimum value of 2. **Value After Reset:** 0x64

The `IC_SDA_SETUP` register is only used by the I2C Controller when operating as a Slave transmitter.

13.5.2.2.38. I2C ACK General Call Register (`IC_ACK_GENERAL_CALL`)

The `IC_ACK_GENERAL_CALL` register is at offset 0x98

The register controls whether I2C Controller responds with an ACK or NACK when it receives an I2C General Call address.

This register is applicable only when the I2C Controller is in Slave mode.
The following table shows the register bit assignments.

Table 13-165 Fields for register: `IC_ACK_GENERAL_CALL`

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	ACK_GEN_CALL	R/W	ACK General Call When set to 1, I2C Controller responds with a ACK when it receives a General Call. Otherwise, I2C Controller responds with a NACK. **Values:** **0x0**: Generate NACK for General Call **0x1**: Generate ACK for a General Call **Value After Reset:** 0x0

13.5.2.2.39. I2C Enable Status Register (`IC_ENABLE_STATUS`)

The `IC_ENABLE_STATUS` register is at offset 0x9C.

The register is used to report the I2C Controller hardware status when the `IC_ENABLE[0]` register is changed from 1 to 0; that means, when the I2C Controller is disabled.

If `IC_ENABLE[0]` has been set to 1, bits [2:1] are forced to 0, and bit [0] is forced to 1.

If `IC_ENABLE[0]` has been set to 0, bits [2:1] can only be valid after bit [0] is read as 0.

The following table shows the register bit assignments.

Table 13-166 Fields for register: `IC_ENABLE_STATUS`

Bits	Name	R/W	Description
[31:3]			Reserved

Table 13-166 Fields for register: `IC_ENABLE_STATUS` (continued)

Bits	Name	R/W	Description
[2]	`SLV_RX_DATA_LOST`	R	Slave Received Data Lost This bit indicates if a Slave-Receiver operation has been aborted with at least one data byte received from an I2C transfer due to the setting of `IC_ENABLE[0]` from 1 to 0. When read as 1, the I2C Controller is deemed to have been actively engaged in an aborted the I2C transfer (with matching address) and the data phase of the I2C transfer has been entered, even though a data byte has been responded with a NACK. If the remote I2C Master terminates the transfer with a STOP condition before the I2C Controller has a chance to NACK a transfer, and `IC_ENABLE[0]` has been set to 0, then this bit is also set to 1. When read as 0, the I2C Controller is deemed to have been disabled without being actively involved in the data phase of a Slave-Receiver transfer. The CPU can safely read this bit when `IC_EN` is read as 0. **Values:** **0x0**: Slave Rx Data is not lost **0x1**: Slave Rx Data is lost **Value After Reset:** 0x0 **Volatile:** true
[1]	`SLV_DISABLED_WHILE_BUSY`	R	Slave Disabled While Busy (Transmit, Receive) This bit indicates if a potential or active Slave operation has been aborted due to the setting bit [0] of the `IC_ENABLE` register from 1 to 0. This bit is set when the CPU writes a 0 to the `IC_ENABLE` register while: • I2C Controller is receiving the address byte of the Slave-Transmitter operation from a remote Master. • OR, I2C Controller is receiving address and data bytes of the Slave-Receiver operation from a remote Master. When read as 1, the I2C Controller is deemed to have forced a NACK during any part of an I2C transfer, irrespective of whether the I2C address matches the Slave address set in the I2C Controller (`IC_SAR` register) OR if the transfer is completed before `IC_ENABLE` is set to 0 but has not taken effect. If the remote I2C Master terminates the transfer with a STOP condition before the I2C Controller has a chance to NACK a transfer, and `IC_ENABLE[0]` has been set to 0, then this bit will also be set to 1.

Table 13-166 Fields for register: `IC_ENABLE_STATUS` (continued)

Bits	Name	R/W	Description
			When read as 0, the I2C Controller is deemed to have been disabled when there is Master activity, or when the I2C bus is idle. The CPU can safely read this bit when `IC_EN` is read as 0. **Values:** **0x0**: Slave is disabled when it is idle **0x1**: Slave is disabled when it is active **Value After Reset:** 0x0 **Volatile:** true
[0]	`IC_EN`	R	I2C interface enable. This bit always reflects the value driven on the output port. When read as 1, the I2C Controller is deemed to be in an enabled state. When read as 0, the I2C Controller is deemed completely inactive. The CPU can safely read this bit anytime. When this bit is read as 0, the CPU can safely read `SLV_RX_DATA_LOST` and `SLV_DISABLED_WHILE_BUSY`. **Values:** **0x0**: I2C enabled **0x1**: I2C disabled **Value After Reset:** 0x0 **Volatile:** true

When `IC_ENABLE[0]` has been set to 0, a delay occurs for bit [0] to be read as 0 because disabling the I2C Controller depends on I2C bus activities.

13.5.2.2.40. I2C SS, FS or FM+ Spike Suppression Limit (`IC_FS_SPKLEN`)

The `IC_FS_SPKLEN` register is at offset 0xA0.

This register is used to store the duration, measured in `i2c*_clk` cycles, of the longest spike that is filtered out by the spike suppression logic when the component is operating in SS, FS or FM+ modes. The relevant I2C requirement is t_{SP} (Noise spike suppression time). This register must be programmed with a minimum value of 1.

The following table shows the register bit assignments.

Table 13-167 Fields for register: `IC_FS_SPKLEN`

Bits	Name	R/W	Description
[31:8]			Reserved

Table 13-167 Fields for register: IC_FS_SPKLEN (continued)

Bits	Name	R/W	Description
[7:0]	IC_FS_SPKLEN	R/W	This register must be set before any I2C bus transaction can take place to ensure stable operation. This register sets the duration, measured in i2c*_clk cycles, of the longest spike in the SCL or SDA lines that will be filtered out by the spike suppression logic. This register can be written only when the I2C interface is disabled which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect. The minimum valid value is 1; hardware prevents values less than this being written, and if attempted results in 1 being set. **Value After Reset:** 0x5

13.5.2.2.41. I2C HS Spike Suppression Limit Register (IC_HS_SPKLEN)

The IC_HS_SPKLEN register is at offset 0xA4.

This register is used to store the duration, measured in i2c*_clk cycles, of the longest spike that is filtered out by the spike suppression logic when the component is operating in HS modes.

The following table shows the register bit assignments.

Table 13-168 Fields for register: IC_HS_SPKLEN

Bits	Name	R/W	Description
[31:8]			Reserved
[7:0]	IC_HS_SPKLEN	R/W	This register must be set before any I2C bus transaction can take place to ensure stable operation. This register sets the duration, measured in i2c*_clk cycles, of the longest spike in the SCL or SDA lines that will be filtered out by the spike suppression logic. This register can be written only when the I2C interface is disabled which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect. The minimum valid value is 1; hardware prevents values less than this being written, and if attempted results in 1 being set. **Value After Reset**: 0x1

13.5.2.2.42. I2C SCL Stuck at Low Timeout Register (IC_SCL_STUCK_AT_LOW_TIMEOUT)

The IC_SCL_STUCK_AT_LOW_TIMEOUT register is at offset 0xAC.

This register is used to store the duration, measured in the number of i2c*_clk periods, used to generate an Interrupt (IC_RAW_INTR_STAT.SCL_STUCK_AT_LOW) if SCL is held low for the IC_SCL_STUCK_LOW_TIMEOUT duration.

The following table shows the register bit assignments.

Table 13-169 Fields for register: IC_SCL_STUCK_AT_LOW_TIMEOUT

Bits	Name	R/W	Description
[31:0]	IC_SCL_STUCK_LOW_TIMEOUT	R/W	The I2C Controller generates the interrupt to indicate SCL stuck at low (IC_RAW_INTR_STAT.SCL_STUCK_AT_LOW)

Table 13-169 Fields for register: `IC_SCL_STUCK_AT_LOW_TIMEOUT` (continued)

Bits	Name	R/W	Description
			if it detects the SCL stuck at low for the `IC_SCL_STUCK_LOW_TIMEOUT` in units of `i2c*_clk` period. This register can be written only when the I2C interface is disabled which corresponds to the `IC_ENABLE[0]` register being set to 0. Writes at other times have no effect. **Value After Reset:** 0xFFFFFFFF

13.5.2.2.43. I2C SDA Stuck at Low Timeout Register (`IC_SDA_STUCK_AT_LOW_TIMEOUT`)

The `IC_SDA_STUCK_AT_LOW_TIMEOUT` register is at offset 0xB0.

This register is used to store the duration, measured in the number of `i2c*_clk` periods, used to Recover the Data (SDA) line through sending SCL pulses while SDA is held low for the mentioned duration.

The following table shows the register bit assignments.

Table 13-170 Fields for register: `IC_SDA_STUCK_AT_LOW_TIMEOUT`

Bits	Name	R/W	Description
[31:0]	`IC_SDA_STUCK_LOW_TIMEOUT`	R/W	The I2C Controller initiates the recovery of SDA line through enabling the `SDA_STUCK_RECOVERY_EN` (`IC_ENABLE[3]`) register bit, if it detects the SDA stuck at low for the `IC_SDA_STUCK_LOW_TIMEOUT` in the number of `i2c*_clk` periods. **Value After Reset:** 0xFFFFFFFF

13.5.2.2.44. Clear SCL Stuck at Low Detect Interrupt Register (`IC_CLR_SCL_STUCK_DET`)

The `IC_CLR_SCL_STUCK_DET` register is at offset 0xB4.

The following table shows the register bit assignments.

Table 13-171 Fields for register: `IC_CLR_SCL_STUCK_DET`

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	`CLR_SCL_STUCK_DET`	R	Read this register to clear the `SCL_STUCK_AT_LOW` interrupt (`SCL_STUCK_AT_LOW` bit of the `IC_RAW_INTR_STAT` register). **Value After Reset:** 0x0 **Volatile:** true

13.5.2.2.45. Register Timeout Counter Reset Value (`REG_TIMEOUT_RST`)

The `REG_TIMEOUT_RST` register is at offset 0xF0.

The following table shows the register bit assignments.

Table 13-172 Fields for register: `REG_TIMEOUT_RST`

Bits	Name	R/W	Description
[31:8]			Reserved
[7:0]	`REG_TIMEOUT_RST`	R/W	This field holds reset value of `REG_TIMEOUT_RST` counter register. **Value After Reset:** 0x8 **Volatile:** true

13.6. I2S

This section describes the *Inter-IC Sound* (**I2S**) Controller of the BE-U1000 microcontroller.

The I2S Controller is a programmable component used for the serial communication with peripherals.

The following figure shows the simplified block diagram of the I2S_*, where asterisk symbol (*) indicates the I2S number and `x` indicates 0 for DMA_0 and 1 for DMA_1. For more information, refer to DMA Controller chapter.

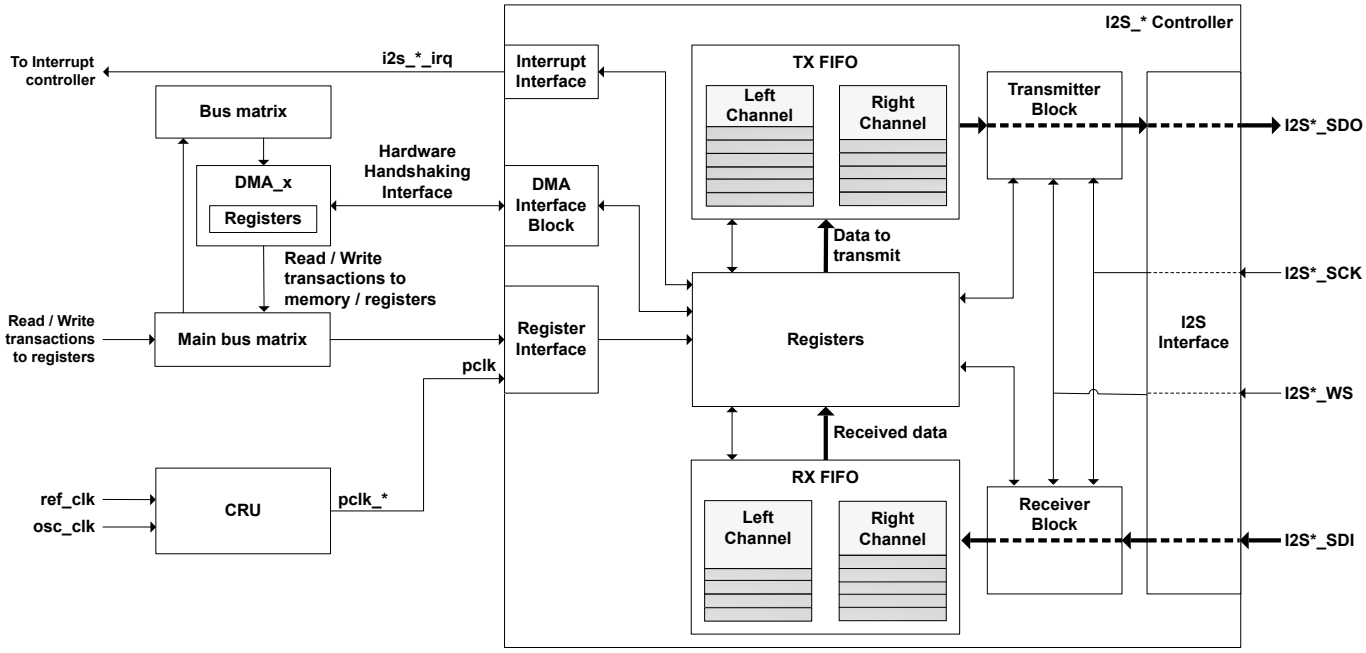

Figure 13-13 I2S block diagram

I2S Controller features:

- 1 stereo channel for transmitter and receiver respectively
- Full duplex communication due to the independence of transmitter and receiver
- The Controller operates in Slave mode
- Audio data resolutions: 12, 16, 20, 24, and 32 bits for Rx channel; 12 and 16 bits for Tx channel
- FIFO depth of 8 words, with the word size of 32 bits for Rx FIFO and 16 bits for Tx FIFO

13.6.1. I2S Functional Description

This section describes the functional operation of the I2S Controller.

This section covers the following topics:

- I2S Controller Enable
- I2S Controller as Transmitter
- I2S Controller as Receiver

13.6.1.1. I2S Controller Enable

You must enable the I2S Controller before any data is received or transmitted into the FIFOs. To enable the component, set the I2S Enable (`IEN`) bit of the I2S Enable Register (`IER`) to 1. When you disable the device, it acts as a global disable. To disable the I2S Controller, set `IER[0]` to 0.

After disable, the following events occur:

- Tx and Rx FIFOs are cleared, and read/write pointers are reset.
- Any data in the process of being transmitted or received is lost, all other programmable enables in the component are overridden.

On reset, the `IER[0]` is set to 0 (disable).

13.6.1.2. I2S Controller as Transmitter

The I2S Controller component supports 1 stereo I2S transmit (Tx) channel. This channel operates in slave mode. Stereo data pairs (such as, left and right audio data) written to the Tx channel via the Register Interface are shifted out serially on the serial data out line. The shifting is timed with respect to the serial clock (`i2s*_clk`) and the word select line (from I2S*_WS pin).

The diagram below illustrates the basic usage flow for the I2S Controller when it acts as a transmitter.

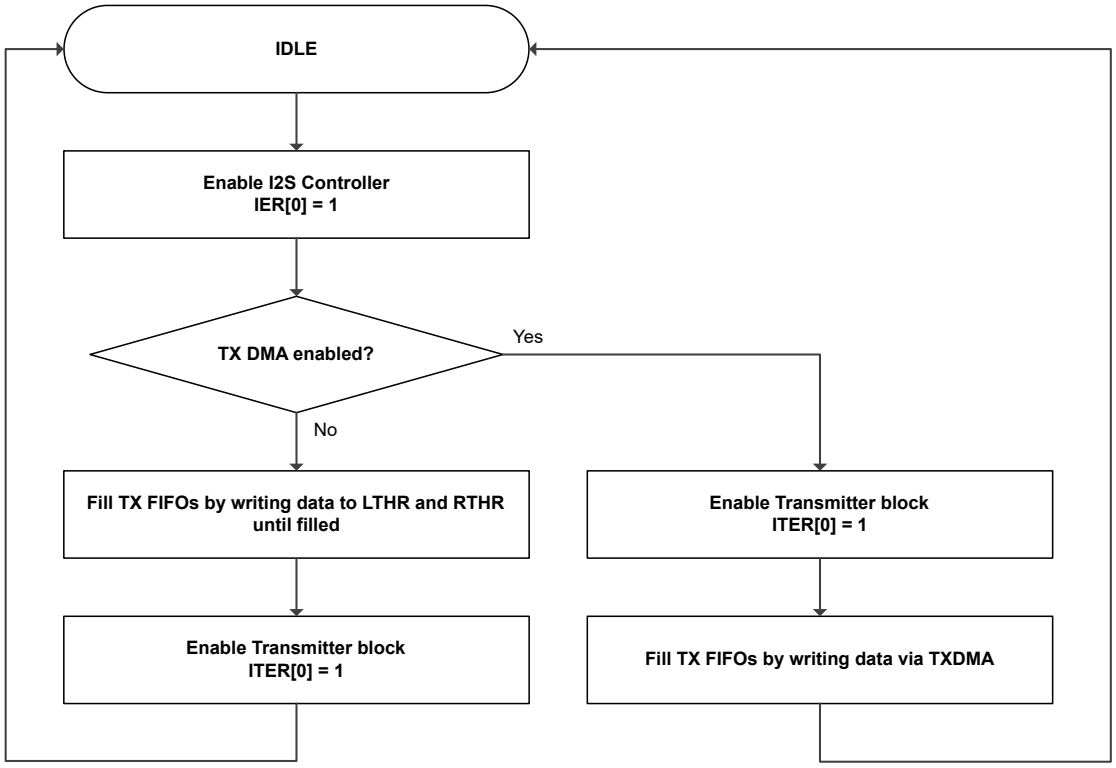

Figure 13-14 Basic usage flow - I2S Controller as Transmitter

13.6.1.2.1. Transmitter Block Enable

The Transmitter Block Enable (`TXEN`) bit of the I2S Transmitter Enable Register (`ITER`) globally turns on and off the Tx channel. To enable the transmitter block, set `ITER[0]` to 1. To disable the block, set `ITER[0]` to 0.

When the transmitter block is disabled, the following events occur:

- Outgoing data is lost and the channel output is held low; data in the Tx FIFO is preserved and the FIFO can be written to.
- Any previous programming (like changes in word size, threshold levels, and so on) of the Tx channel is preserved.
- When the transmitter block is enabled, if there is data in the Tx FIFO, the channel resumes transmission on the next left stereo data cycle

When the block is disabled, you can perform any of the following procedures:

- Program (or further program) the Tx channel register.
- Flush the Tx FIFO by programming the Transmitter FIFO Reset bit of the Transmitter FIFO Flush Register (`TXFFR[0]` = 1).
- Flush the channel's Tx FIFO by programming the Transmit Channel FIFO Reset (`TXCHFR`) bit of the Transmit FIFO Flush Register (`TFF0` = 1).

On reset, the `ITER[0]` is set to 0 (disable).

13.6.1.2.2. Transmit Channel Enable

The transmit channel has enable/disable allowing reprogramming of the channel and flushing the channel's Tx FIFOs. This enable/disable is controlled by bit 0 of the Transmitter Enable Register (`TER0`). To enable the Tx Channel, write 1 to `TER0[0]`. To disable it, write a 0 to `TER0[0]`.

When a Tx channel is disabled, the following occurs:

- Outgoing stereo data is lost
- Channel output is held low
- Data in the Tx FIFO is preserved, and the FIFO can be written to
- Any previous programming of the Tx channel's register is preserved, and the register can be further reprogrammed

When the Tx channel is disabled, you can flush the channel Tx FIFO by programming the Transmit Channel FIFO Reset (`TXCHFR`) bit of the Transmit FIFO Flush (`TFF0[0]` = 1). When the Tx channel is enabled, if there is data in the Tx FIFO, the channel resumes transmission on the next left stereo data cycle.

On reset, the `TFF0[0]` is set to 1 (enable).

13.6.1.2.3. Transmit Channel Audio Data Resolution

The Tx channel is initially configured with a maximum audio data resolution that is 16. The Tx channel can be reprogrammed during operation to any supported audio data resolution that is less than 16. I.e. it can be programmed to support a resolution of 12 or 16 bits. If the channel is programmed with an invalid audio resolution, the Tx channel defaults to 16.

Reprogramming of the audio resolution ensures that the MSB of the data is still transmitted first if the resolution of the data to be sent is reduced. Changes to the resolution are programmed via the

Word Length (WLEN) bits of the Transmitter Configuration Register (TCR0[2:0]). The channel must be disabled prior to any resolution changes.

On reset or if an invalid resolution is selected, the Tx channel's audio data resolution defaults back to 16.

13.6.1.2.4. Transmit Channel FIFOs

The Transmit Channel has two FIFO banks for left and right stereo data. The FIFOs depth is 8. The FIFO width is determined by the maximum audio resolution that is 16.

There are several ways to clear the Tx FIFOs and reset the read/write pointers as described as follows:

- On reset
- By disabling I2S Controller (IER[0] = 0)
- By flushing the transmitter block (TXFFR[0] = 1)
- By flushing the Tx channel (TFF0[0] = 1)

You must disable the transmitter block/channel before the transmitter block and the channel FIFO can be flushed.

The default low threshold level for the Tx FIFO is 3. This level though can be set to any value in the range of 0 to 7. When it is reached, a Transmit Channel Empty (Tx FIFO Empty) interrupt is generated. This level can be reprogrammed during operation by writing to the Transmit Channel Empty Trigger (TXCHET) bits of the Transmit FIFO Configuration Register (TFCR0[3:0]).

You must disable the Tx channel prior to changing the trigger level.

13.6.1.2.5. Transmit Channel Interrupts

All interrupts in I2S Controller are active high. The Tx channel generates two interrupts: Tx FIFO Empty and Data Overrun.

- **Tx FIFO Empty interrupt** – This interrupt is asserted when the low threshold level for the Tx FIFO is reached. A Tx FIFO Empty interrupt is cleared by writing data to the Tx FIFO to bring its level above the low threshold level that is 3.
- **Data Overrun interrupt** – This interrupt is asserted when an attempt is made to write to a full Tx FIFO (any data being written is lost while data in the FIFO is preserved). A Data Overrun interrupt is cleared by reading the TX_CHANN_OVERRUN bit of the Transmit Overrun Register (TOR0).

The interrupt status of the Tx channel can be determined by polling the Interrupt Status Register (ISR0). The ISR0.TXFE bit indicates the status of the 'Tx FIFO Empty' interrupt, while the ISR0.TXFO bit indicates the status of the Data Overrun interrupt.

Both the Tx FIFO Empty and Data Overrun interrupts can be masked off by writing 1 in the Transmit Empty Mask (TXFEM) and Transmit Overrun Mask (TXFOM) bits of the Interrupt Mask Register (IMR0), respectively. This prevents the interrupts from driving their output lines; however, the ISR0 always shows the current status of the interrupts regardless of any masking.

Constructively, both interrupts are included as separate output lines; each interrupt triggers an active state on a corresponding line.

13.6.1.2.6. Writing to the Transmit Channel

The stereo data pairs to be transmitted by the Tx channel must be written to the Tx FIFOs via the Left Transmit Holding Register (`LTHR0`) and the Right Transmit Holding Register (`RTHR0`). All stereo data pairs must be written using the following two-stage process:

1. Write left stereo data to `LTHR0`
2. Write right stereo data to `RTHR0`

You must write stereo data to the device in this order, otherwise, the interrupt and status lines values will be invalid, and the left / right stereo pairs might be transmitted out of sync.

The I2S Controller can use the DMA mechanism of data exchange. In this case, data to be transmitted by the Tx channel has to be written to the Tx FIFO via the `TXDMA` register rather than through `LTHR0` and `RTHR0`.

When the I2S Controller is enabled, if the Tx FIFO is empty and data is not written to the FIFOs before the next left cycle, the channel outputs zeros for a full frame (left and right cycle). Transmission only commences if there is data in the Tx FIFO prior to the transition to the left data cycle. In other words, if the start of the frame is missed, the channel output idles until the next available frame.

Data should only be written to the FIFO when it is not full. Any attempt to write to a full FIFO results in that data being lost and a Data Overrun interrupt being generated.

13.6.1.3. I2S Controller as Receiver

I2S Controller includes 1 stereo I2S receive (Rx) channel. This channel operates in slave mode. Stereo data pairs (such as, left and right audio data) are received serially from a data input line.

These data words are stored in Rx FIFOs until they are read via the Register Interface. The receiving is timed with respect to the serial clock (`i2s*_clk`) and the word select line (from I2S*_WS pin).

The following diagram illustrates the basic usage flow for I2S Controller when it acts as a receiver.

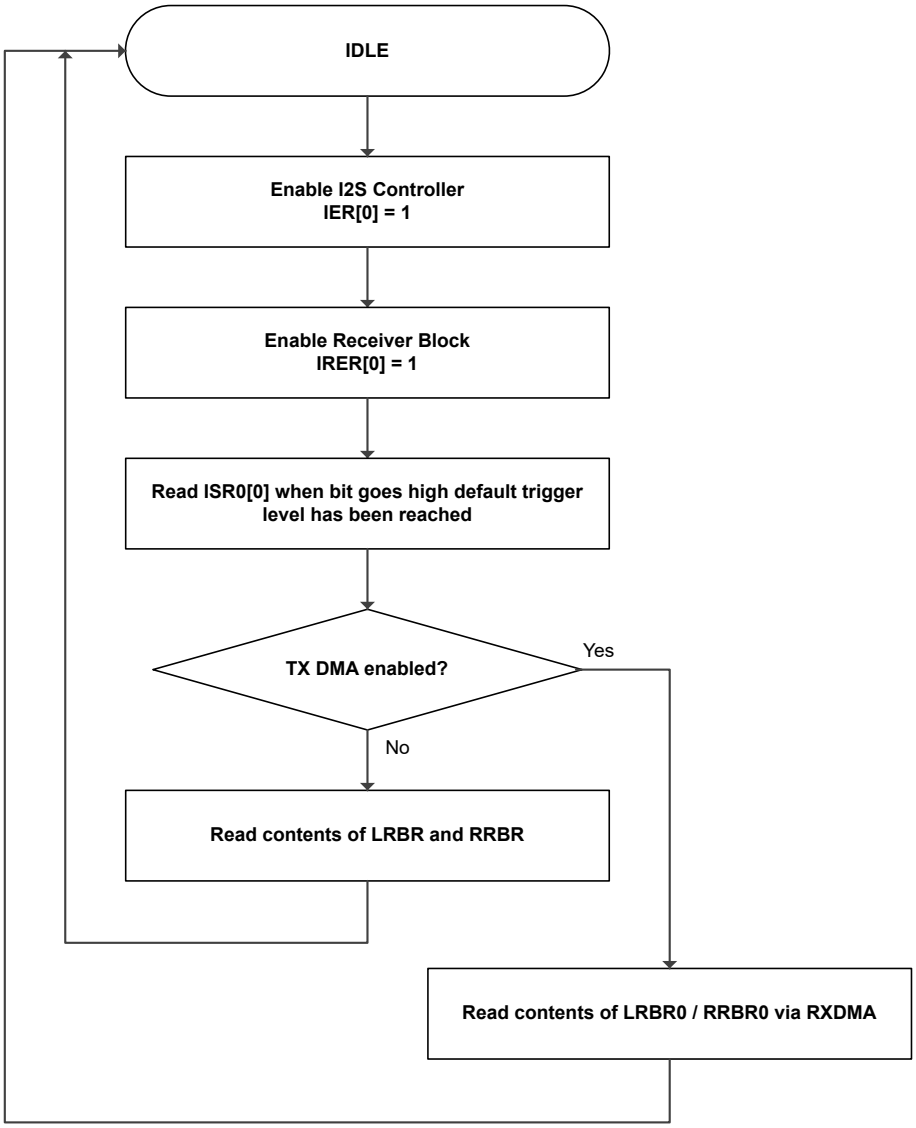

Figure 13-15 Basic usage flow - I2S Controller as Receiver

13.6.1.3.1. Receiver Block Enable

The Receiver Block Enable (`RXEN`) bit of the I2S Receiver Enable Register (`IRER`) enables / disables the Rx channel. To enable the receiver block, set `IRER[0]` to 1. To disable the block, set this bit to 0.

When the receiver block is disabled, the following events occur:

- Incoming data is lost.
- Data in the Rx FIFO is preserved and the FIFO can be read.
- Any previous programming (such as changes in word size, threshold levels, and so on) of the Rx channel is preserved.
- The Rx channel enable is overridden. Enabling the channel resumes receiving on the next left stereo data cycle.

When the block is disabled, you can perform any of the following procedures:

- Program (or further program) the Rx channel registers.
- Flush the RX FIFO by programming the Receiver FIFOs Reset (RXFFR) bit of the Receiver FIFO Flush Register (RXFFR[0] = 1).
- Flush the channel's Rx FIFO by programming the Receive Channel FIFO Reset (RXCHFR) bit of the Receive FIFO Flush Register (RFF0[0] = 1).

On reset, IRER is set to 0 (disable).

13.6.1.3.2. Receive Channel Enable

The Rx channel has its own enable/disable allowing programming of the channel clearing the channel's Rx FIFO. This enable/disable is controlled by bit 0 of the Receiver Enable Register (RER0[0]). To enable the Rx Channel, write a 1 to RER0[0]. To disable this channel, write a 0 to RER0[0].

When the Rx channel is disabled, the following occurs:

- Incoming data is lost
- Data in the Rx FIFO is preserved
- FIFO can be read
- Previous programming of the Rx channel is preserved
- The Rx channel can be further programmed

When the Rx channel or block is disabled, you can flush the channel's Rx FIFO by writing 1 in bit 0 of the Receive FIFO Flush Register (RFF0). When the channel is enabled it resumes receiving on the next left stereo data cycle.

On reset, the RFF0[0] is set to 1 (enable).

13.6.1.3.3. Receive Channel Audio Data Resolution

The Rx channel is initially configured with a maximum audio data resolution that is 32. The Rx channel can be programmed during operation to any supported audio data resolution that is equal or less than 32. In other words, it can be programmed to support resolutions of 12 16, 20, 24, or 32 bits. This programming ensures that the LSB of the received data is placed in the LSB position of the Rx FIFO if the resolution of the data being received is reduced. Changes to the resolution are programmed via the Word Length (WLEN) bits of the Receive Configuration registers (RCR0[3:0]). The channel must be disabled prior to any resolution changes.

The Rx channel also supports unknown data resolutions. If the received word is greater than the configured channel resolution, the least significant bits are ignored. If the received word is less than the configured/programmed channel resolution, the least significant bits are padded with zeros.

On reset or if an invalid resolution is selected, the Rx channel audio data resolution defaults back to the initial setting of 32.

13.6.1.3.4. Receive Channel FIFOs

The Receive Channel has two FIFO banks for left and right stereo data. The FIFOs have the depth of 8 words, 32 bits each. The Rx FIFOs can be cleared and the read/write pointers reset in a number of ways, as described as follows:

- On reset
- By disabling I2S Controller (IER[0] = 0)

- By flushing the receiver block (RXFFR[0] = 1)
- By flushing the Rx channel (RFF0[0] = 1)

Before you flush the receiver block or the channel, you must disable the receiver block or channel.

The default data available threshold for the Rx FIFO is 3. When this level is reached, the 'Rx channel data available' interrupt is generated. This level can be reprogrammed during operation via the Receive Channel Data Trigger (RXCHDT) bits of the Receive FIFO Configuration Register (RFCR0[3:0]). The Rx channel needs to be disabled prior to any changes in the trigger level.

13.6.1.3.5. Receive Channel Interrupts

All interrupts in I2S Controller are active high. The Rx channel generates two interrupts: Rx FIFO Data Available and Data Overrun.

- **Rx FIFO Data Available interrupt** — This interrupt is asserted when the trigger level for the Rx FIFO (that is 3 by default) is reached. This interrupt is cleared by reading data from the Rx FIFO until its level drops below the data available threshold level for the channel.
- **Data Overrun interrupt** — This interrupt is asserted when an attempt is made to write received data to a full Rx FIFO (any data being written is lost while data in the FIFO is preserved).This interrupt is cleared by reading the RXCH0 bit of the Receive Overrun Register (ROR0).

The interrupt status of the Rx channel can be determined by polling the Interrupt Status Register (ISR0). The ISR0.RXDA bit indicates the status of the Rx FIFO Data Available interrupt; the ISR0.RXFO bit indicates the status of the 'Rx FIFO Data Overrun' interrupt. Both the Rx FIFO Data Available interrupt and Data Overrun interrupt can be masked by writing a 1 in the Rx FIFO Data Available interrupt Mask (RXDAM) and Rx FIFO Overrun interrupt Mask (RXFOM) bits of the Interrupt Mask Register (IMR0), respectively. This prevents the interrupts from driving their output lines; however, the ISR0 always shows the current status of the interrupts regardless of any masking. Each interrupt uses an individual interrupt line switching into an active state when the corresponding interrupt is asserted.

13.6.1.3.6. Reading from the Receive Channel

The stereo data pairs received by the Rx channel are written to the left and right Rx FIFOs. These FIFOs can be read via the Left Receive Buffer Register (LRBR0) and the Right Receive Buffer Register (RRBR0). All stereo data pairs must be read using the following two-stage process:

1. Read the left stereo data from LRBR0
2. Read the right stereo data from RRBR0

You must read the stereo data in this order to avoid the status and interrupt lines becoming out of sync.

The I2S Controller can use the DMA mechanism of data exchange, and in this case, data can be read from Rx FIFOs via the RXDMA register rather than through LRBR0 and RRBR0.

13.6.2. I2S Registers

The BE-U1000 microcontroller integrates two I2S Controllers. The following table describes address regions for the I2S Controllers.

Table 13-173 Memory address regions for I2S Controllers

Region	Block Name	Size	Start Address	End Address
Periphery 0	I2S_0	4 KB	0x1100_6000	0x1100_6FFF

Table 13-173 Memory address regions for I2S Controllers (continued)

Region	Block Name	Size	Start Address	End Address
Periphery 1	I2S_1	4 KB	0x1300_6000	0x1300_6FFF

For details, refer to Memory Map chapter.

This section covers the following topics:

- Register Map
- Register Descriptions

13.6.2.1. Register Map

The following table provides a high-level summary of each I2S Controller register. To view the detailed description of a register, click the register name in the **Description** column.

Table 13-174 I2S register map

Name	Offset	Description	Default Value
IER	0x0	I2S Controller Enable Register	0x0
IRER	0x4	I2S Receiver Block Enable Register	0x0
ITER	0x8	I2S Transmitter Block Enable Register	0x0
RXFFR	0x14	Receiver Block FIFO Reset Register	0x0
TXFFR	0x18	Transmitter Block FIFO Reset Register	0x0
LRBR0	0x20	Left Receive Buffer Register	0x0
LTHR0	0x20	Left Transmit Holding Register	0x0
RRBR0	0x24	Right Receive Buffer Register	0x0
RTHR0	0x24	Right Transmit Holding Register	0x0
RER0	0x28	Receive Enable Register	0x1
TER0	0x2C	Transmit Enable Register	0x1
RCR0	0x30	Receive Configuration Register	0x5
TCR0	0x34	Transmit Configuration Register	0x2
ISR0	0x38	Interrupt Status Register	0x10
IMR0	0x3C	Interrupt Mask Register	0x33
ROR0	0x40	Receive Overrun Register	0x0

Table 13-174 I2S register map (continued)

Name	Offset	Description	Default Value
TOR0	0x44	Transmit Overrun Register	0x0
RFCR0	0x48	Receive FIFO Configuration Register	0x3
TFCR0	0x4C	Transmit FIFO Configuration Register	0x3
RFF0	0x50	Receive FIFO Flush Register	0x0
TFF0	0x54	Transmit FIFO Flush Register	0x0
RXDMA	0x1C0	Receiver Block DMA Register	0x0
RRXDMA	0x1C4	Reset Receiver Block DMA Register	0x0
TXDMA	0x1C8	Transmitter Block DMA Register	0x0
RTXDMA	0x1CC	Reset Transmitter Block DMA Register	0x0
DMACR	0x200	DMA Control Register	0x0

13.6.2.2. Register Descriptions

The following sections contain the register descriptions.

In the tables below, the **R/W** column specifies how the application and the core can access the register fields:

- **R:** Read Only Access
- **R/W:** Read/Write Access
- **W:** Write Access

13.6.2.2.1. I2S Controller Enable Register (IER)

The IER register is at offset 0x0

This register acts as a global enable/disable for I2S Controller.

The following table shows the register bit assignments.

Table 13-175 Fields for register: IER

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	IEN	R/W	I2S Controller enable A disable on this bit overrides any other block or channel enables and flushes all FIFOs. **Values:** **0x0:** Disable I2S Controller **0x1:** Enable I2S Controller **Value After Reset:** 0x0

13.6.2.2.2. I2S Receiver Block Enable Register (IRER)

The IRER register is at offset 0x4

This register acts as an enable/disable for the I2S Controller Receiver block.

The following table shows the register bit assignments.

Table 13-176 Fields for register: IRER

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	RXEN	R/W	Receiver block enable A disable on this bit overrides receive channel enable. **Values:** **0x0:** Disable receiver **0x1:** Enable receiver **Value After Reset:** 0x0

13.6.2.2.3. I2S Transmitter Block Enable Register (ITER)

The ITER register is at offset 0x8

This register acts as an enable/disable for the I2S Controller Transmitter block.

The following table shows the register bit assignments.

Table 13-177 Fields for register: ITER

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	TXEN	R/W	Transmitter block enable A disable on this bit overrides transmit channel enable. **Values:** **0x0:** Disable transmitter **0x1:** Enable transmitter **Value After Reset:** 0x0

13.6.2.2.4. Receiver Block FIFO Reset Register (RXFFR)

The RXFFR register is at offset 0x14

The following table shows the register bit assignments.

Table 13-178 Fields for register: RXFFR

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	RXFFR	W	Receiver FIFO Reset Writing a 1 to this register flushes the Rx FIFOs (this is a self-clearing bit). Receiver Block must be disabled prior to writing this bit. **Values:**

Table 13-178 Fields for register: RXFFR (continued)

Bits	Name	R/W	Description
			0x0: Does not flush the Rx FIFO **0x1:** Flushes the Rx FIFO **Value After Reset:** 0x0

13.6.2.2.5. Transmitter Block FIFO Reset Register (TXFFR)

The TXFFR register is at offset 0x18

The following table shows the register bit assignments.

Table 13-179 Fields for register: TXFFR

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	TXFFR	W	Transmitter FIFO Reset Writing a 1 to this register flushes the Tx FIFOs (this is a self-clearing bit). Transmitter Block must be disabled prior to writing this bit. **Values:** **0x0:** Does not flush the Tx FIFO **0x1:** Flushes the Tx FIFO **Value After Reset:** 0x0

13.6.2.2.6. Left Receive Buffer Register (LRBR0)

The LRBR0 register is at offset 0x20

The following table shows the register bit assignments.

Table 13-180 Fields for register: LRBR0

Bits	Name	R/W	Description
[31:0]	LRBR0	R	The left stereo data received serially from the receive channel input is read through this register. If the Rx FIFO is full and the two-stage read operation (for instance, a read from LRBR0 followed by a read from RRBR0) is not performed before the start of the next stereo pair, then the new data is lost and an overrun interrupt occurs. (Data already in the Rx FIFO is preserved.)  Before reading this register again, the right stereo data MUST be read from RRBR0, or the status/interrupts will not be valid. **Value After Reset:** 0x0

13.6.2.2.7. Left Transmit Holding Register (LTHR0)

The LTHR0 register is at offset 0x20

The following table shows the register bit assignments.

Table 13-181 Fields for register: `LTHR0`

Bits	Name	R/W	Description
[31:0]	`LTHR0`	W	The left stereo data to be transmitted serially through the transmit channel output is written through this register. Writing is a two-stage process: 1. A write to this register passes the left stereo sample to the transmitter. 2. This MUST be followed by writing the right stereo sample to the `RTHR0` register. Data should only be written to the FIFO when it is not full. Any attempt to write to a full FIFO results in that data being lost and an overrun interrupt being generated. **Value After Reset:** 0x0

13.6.2.2.8. Right Receive Buffer Register (`RRBR0`)

The `RRBR0` register is at offset 0x24

The following table shows the register bit assignments.

Table 13-182 Fields for register: `RRBR0`

Bits	Name	R/W	Description
[31:0]	`RRBR0`	R	The right stereo data received serially from the receive channel input is read through this register. If the Rx FIFO is full and the two-stage read operation (for instance, read from `LRBR0` followed by a read from `RRBR0`) is not performed before the start of the next stereo pair, then the new data is lost and an overrun interrupt occurs. (Data already in the Rx FIFO is preserved.)  Prior to reading this register, the left stereo data MUST be read from `LRBR0`, or the status/interrupts will not be valid. **Value After Reset:** 0x0

13.6.2.2.9. Right Transmit Holding Register (`RTHR0`)

The `RTHR0` register is at offset 0x24

The following table shows the register bit assignments.

Table 13-183 Fields for register: `RTHR0`

Bits	Name	R/W	Description
[31:0]	`RTHR0`	W	The right stereo data to be transmitted serially through the transmit channel output is written through this register. Writing is a two-stage process: 1. A left stereo sample MUST first be written to the `LTHR0` register. 2. A write to this register passes the right stereo sample to the transmitter. Data should only be written to the FIFO when it is not full. Any attempt to write to a full FIFO results in that data being lost and an overrun interrupt being generated. **Value After Reset:** 0x0

13.6.2.2.10. Receive Enable Register (RER0)

The RER0 register is at offset 0x28

The following table shows the register bit assignments.

Table 13-184 Fields for register: RER0

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	RXCHEN0	R/W	Receive channel enable This bit enables/disables the receive channel. On enable, the channel begins receiving on the next left stereo cycle. A global disable of I2S Controller (IER[0] = 0) or the Receiver block (IRER[0] = 0) overrides this value. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1

13.6.2.2.11. Transmit Enable Register (TER0)

The TER0 register is at offset 0x2C

The following table shows the register bit assignments.

Table 13-185 Fields for register: TER0

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	TXCHEN0	R/W	Transmit channel enable This bit enables/disables the transmit channel. On enable, the channel begins transmitting on the next left stereo cycle. A global disable of I2S Controller (IER[0] = 0) or Transmitter block (ITER[0] = 0) overrides this value. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x1

13.6.2.2.12. Receive Configuration Register (RCR0)

The RCR0 register is at offset 0x30

The following table shows the register bit assignments.

Table 13-186 Fields for register: RCR0

Bits	Name	R/W	Description
[31:3]			Reserved

Table 13-186 Fields for register: RCR0 (continued)

Bits	Name	R/W	Description
[2:0]	WLEN	R/W	These bits are used to program the desired data resolution of the receiver and enables the LSB of the incoming left (or right) word to be placed in the LSB of the LRBR0 (or RRBR0) register. **Values:** **0x0:** Ignore word length **0x1:** 12-bit resolution **0x2:** 16-bit resolution **0x3:** 20-bit resolution **0x4:** 24-bit resolution **0x5:** 32-bit resolution Programmed data resolution must be less than or equal to 32. If the selected resolution is greater than the 32, the receive channel defaults back to 32-bit resolution (0x5). The channel must be disabled prior to any changes in this value (RER0[0] = 0). **Value After Reset:** 0x5

13.6.2.2.13. Transmit Configuration Register (TCR0)

The TCR0 register is at offset 0x34

The following table shows the register bit assignments.

Table 13-187 Fields for register: TCR0

Bits	Name	R/W	Description
[31:3]			Reserved
[2:0]	WLEN	R/W	These bits are used to program the data resolution of the transmitter and ensure the MSB of the data is transmitted first. **Values:** **0x0:** Ignore word length **0x1:** 12-bit resolution **0x2:** 16-bit resolution **0x3:** Reserved **0x4:** Reserved **0x5:** Reserved Programmed resolution must be less than or equal to 16. If the selected resolution is greater than 16, the transmit channel defaults back 16-bit resolution (0x2). The channel must be disabled prior to any changes in this value (TER0[0] = 0). **Value After Reset:** 0x2

13.6.2.2.14. Interrupt Status Register (ISR0)

The ISR0 register is at offset 0x38

The following table shows the register bit assignments.

Table 13-188 Fields for register: `ISR0`

Bits	Name	R/W	Description
[31:6]			Reserved
[5]	TXFO	R	Status of Data Overrun interrupt for the Tx channel Attempt to write to full Tx FIFO. **Values:** **0x0:** Tx FIFO write valid **0x1:** Tx FIFO write overrun **Value After Reset:** 0x0
[4]	TXFE	R	Status of Transmit Empty Trigger interrupt Tx FIFO is empty. **Values:** **0x0:** Trigger level reached **0x1:** Trigger level not reached **Value After Reset:** 0x1
[3:2]		R	Reserved
[1]	RXFO	R	Status of Data Overrun interrupt for the Rx channel Incoming data lost due to a full Rx FIFO. **Values:** **0x0:** Rx FIFO write valid **0x1:** Rx FIFO write overrun **Value After Reset:** 0x0
[0]	RXDA	R	Status of Receive Data Available interrupt Rx FIFO data available. **Values:** **0x0:** Trigger level reached **0x1:** Trigger level not reached **Value After Reset:** 0x0

13.6.2.2.15. Interrupt Mask Register (`IMR0`)

The `IMR0` register is at offset 0x3C

The following table shows the register bit assignments.

Table 13-189 Fields for register: `IMR0`

Bits	Name	R/W	Description
[31:6]			Reserved
[5]	TXFOM	R/W	Masks Tx FIFO Overrun interrupt **Values:** **0x0:** Unmasks interrupt **0x1:** Masks interrupt **Value After Reset:** 0x1
[4]	TXFEM	R/W	Masks Tx FIFO Empty interrupt

Table 13-189 Fields for register: IMR0 (continued)

Bits	Name	R/W	Description
			Values: **0x0:** Unmasks interrupt **0x1:** Masks interrupt **Value After Reset:** 0x1
[3:2]			Reserved
[1]	RXFOM	R/W	Masks Rx FIFO Overrun interrupt **Values:** **0x0:** Unmasks interrupt **0x1:** Masks interrupt **Value After Reset:** 0x1
[0]	RXDAM	R/W	Masks Rx FIFO Data Available interrupt **Values:** **0x0:** Unmasks interrupt **0x1:** Masks interrupt **Value After Reset:** 0x1

13.6.2.2.16. Receive Overrun Register (ROR0)

The ROR0 register is at offset 0x40

The following table shows the register bit assignments.

Table 13-190 Fields for register: ROR0

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	RXCH0	R	Read this bit to clear the Rx FIFO Data Overrun interrupt. **Values:** **0x0:** Rx FIFO write valid **0x1:** Rx FIFO write overrun **Value After Reset:** 0x0

13.6.2.2.17. Transmit Overrun Register (TOR0)

The TOR0 register is at offset 0x44

The following table shows the register bit assignments.

Table 13-191 Fields for register: TOR0

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	TXCH0	R	Read this bit to clear the Tx FIFO Data Overrun interrupt **Values:**

Table 13-191 Fields for register: `TOR0` (continued)

Bits	Name	R/W	Description
			0x0: Tx FIFO write valid **0x1:** Tx FIFO write overrun **Value After Reset:** 0x0

13.6.2.2.18. Receive FIFO Configuration Register (`RFCR0`)

The `RFCR0` register is at offset 0x48

The following table shows the register bit assignments.

Table 13-192 Fields for register: `RFCR0`

Bits	Name	R/W	Description
[31:4]			Reserved
[3:0]	`RXCHDT`	R/W	These bits program the trigger level in the Rx FIFO at which the Received Data Available interrupt is generated. `Trigger Level = Programmed Value + 1` (for example, 1 to 8) Valid `RXCHDT` values: 0 to 7 If an illegal value is programmed, these bits saturate to 7. The channel must be disabled prior to any changes in this value (that is, `RER0[0]` = 0). **Value After Reset:** 0x3

13.6.2.2.19. Transmit FIFO Configuration Register (`TFCR0`)

The `TFCR0` register is at offset 0x4C

The following table shows the register bit assignments.

Table 13-193 Fields for register: `TFCR0`

Bits	Name	R/W	Description
[31:4]			Reserved
[3:0]	`TXCHET`	R/W	Transmit Channel Empty Trigger These bits program the trigger level in the Tx FIFO at which the Empty Threshold Reached Interrupt is generated. `Trigger Level = TXCHET` `TXCHET` values: 0 to 7. If an illegal value is programmed, these bits saturate to 7. The channel must be disabled prior to any changes in this value (that is, `TER0[0]` = 0). **Value After Reset:** 0x3

13.6.2.2.20. Receive FIFO Flush Register (`RFF0`)

The `RFF0` register is at offset 0x50

The following table shows the register bit assignments.

Table 13-194 Fields for register: RFF0

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	RXCHFR	W	Receive Channel FIFO Reset Writing a 1 to this register flushes the Rx FIFO (this is a self-clearing bit). Rx channel or block must be disabled prior to writing to this bit. **Values:** **0x0:** Does not flush an individual Rx FIFO **0x1:** Flushes an individual Rx FIFO **Value After Reset:** 0x0

13.6.2.2.21. Transmit FIFO Flush Register (TFF0)

The TFF0 register is at offset 0x54

The following table shows the register bit assignments.

Table 13-195 Fields for register: TFF0

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	TXCHFR	W	Transmit Channel FIFO Reset Writing a 1 to this register flushes channel's Tx FIFO (this is a self-clearing bit). Tx channel or block must be disabled prior to writing to this bit. **Values:** **0x0:** Does not flush Tx FIFO of the chanel **0x1:** Flushes Tx FIFO of the chanel **Value After Reset:** 0x0

13.6.2.2.22. Receiver Block DMA Register (RXDMA)

The RXDMA register is at offset 0x1C0

The RXDMA register allows access to all enabled Receive channels via a single point rather than through the the LRBR0 and RRBR0 registers. The Receive channels are targeted in a cyclical fashion (starting at the lowest numbered enabled channel) and takes two reads (left and right stereo data) before the component points to the next channel.

The following example describes the behavior of this register for a component that has been configured with four Receive channels, where Channels 0 and 3 are enabled:

Order of returned read data:

1. Ch0 - Left Data
2. Ch0 - Right Data
3. Ch3 - Left Data
4. Ch3 - Right Data
5. Ch0 - Left Data
6. Ch0 - Right Data, and so on

The channel can be enabled or disabled during the read cycles; however, I2S Controller does not support disabling the channel in the middle of a stereo pair.

The following table shows the register bit assignments.

Table 13-196 Fields for register: RXDMA

Bits	Name	R/W	Description
[31:0]	RXDMA	R	Receiver Block DMA Register Allows to read stereo data pairs from the Rx Channel. **Value After Reset:** 0x0

13.6.2.2.23. Reset Receiver Block DMA Register (RRXDMA)

The RRXDMA register is at offset 0x1C4

The RXDMA can be reset to the lowest enabled Channel via the RRXDMA register. The RRXDMA register can be written to at any stage of the RXDMA read cycle, however, it has no effect when the component is in the middle of a stereo pair read. The following example describes the operation of this register for a system with four Receive channels, where channels 0, 1, 2, and 3 are enabled.

Order of returned read data:

1. Ch0 - Left Data
2. Ch0 - Right Data
3. RRXDMA Reset
4. Ch0 - Left Data
5. Ch0 - Right Data
6. Ch1 - Left Data
7. RRXDMA Reset - No effect (read not complete)
8. Ch1 - Right Data, etc.
9. Ch2 - Left Data
10. Ch2 - Right Data
11. RRXDMA Reset
12. Ch0 - Left Data
13. Ch0 - Right Data

The following table shows the register bit assignments.

Table 13-197 Fields for register: RRXDMA

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	RRXDMA	W	Reset Receiver Block DMA Register Writing a 1 to this self-clearing register resets the RXDMA register mid-cycle to point to the lowest enabled Receive channel.  Writing to this register has no effect if the component is performing a stereo pair read (such as, when left stereo data has been read but not right stereo data). **Value After Reset:** 0x0

13.6.2.2.24. Transmitter Block DMA Register (TXDMA)

The TXDMA register is at offset 0x1C8

The TXDMA register functions similar to the RXDMA register and allows write access to the Transmit channel via a single point rather than through the LTHR0 and RTHR0 registers.

The I2S Controller does not support disabling the channel in the middle of a stereo pair.

The following table shows the register bit assignments.

Table 13-198 Fields for register: TXDMA

Bits	Name	R/W	Description
[31:0]	TXDMA	W	Transmitter Block DMA Register. The register bits can be used to cycle repeatedly through the enabled Transmit channels (from lowest numbered to highest) to allow writing of stereo data pairs. **Value After Reset:** 0x0

13.6.2.2.25. Reset Transmitter Block DMA Register (RTXDMA)

The RTXDMA register is at offset 0x1CC

This register provides the same functionality as the RRXDMA register but targets TXDMA instead.

The following table shows the register bit assignments.

Table 13-199 Fields for register: RTXDMA

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	RTXDMA	W	Reset Transmitter Block DMA Register Writing a 1 to this self-clearing register resets the TXDMA register mid-cycle to point to the lowest enabled Transmit channel.  This register has no effect in the middle of a stereo pair write (such as, when left stereo data has been written but not right stereo data). **Value After Reset:** 0x0

13.6.2.2.26. DMA Control Register (DMACR)

The DMACR register is at offset 0x200

The register is used to enable the DMA Controller interface operation.

The following table shows the register bit assignments.

Table 13-200 Fields for register: DMACR

Bits	Name	R/W	Description
[31:18]			Reserved
[17]	DMAEN_TXBLOCK	R/W	DMA Enable for transmit block The corresponding bits of this field enables/disables the DMA handshake logic for transmitter block.

Table 13-200 Fields for register: `DMACR` (continued)

Bits	Name	R/W	Description
			Values: **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x0
[16]	DMAEN_RXBLOCK	R/W	DMA Enable for receive block The corresponding bits of this field enables/disables the DMA handshake logic for receiver block. **Values:** **0x0:** Disable **0x1:** Enable **Value After Reset:** 0x0
[15:0]			Reserved

14. USB 2.0

This chapter covers the following topics:

- USB 2.0 Controller
- USB 2.0 PHY

The USB subsystem of BE-U1000 consists of two components — USB 2.0 *On-The-Go* (**OTG**) Controller (below referred as USB 2.0 Controller) and USB 2.0 PHY, therefore the sections below describe both mentioned above components.

14.1. USB 2.0 Controller

A simplified block diagram of the USB 2.0 Controller is shown in the following figure.

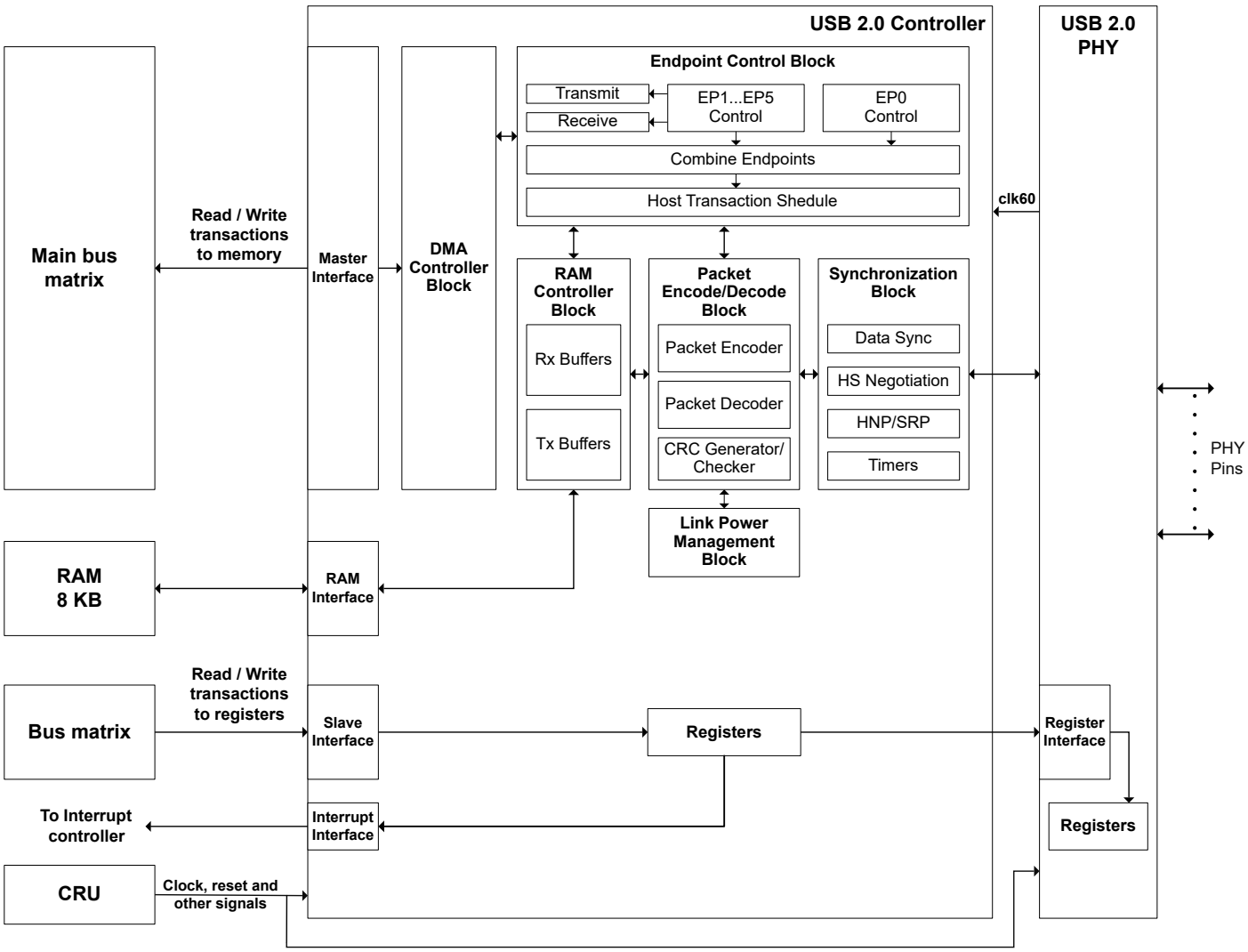

Figure 14-1 USB 2.0 Controller block diagram

USB 2.0 Controller features:

- Operates either as the function controller of a high- /full-speed USB peripheral or as the host/ peripheral in multi-point communications with other USB functions.
- Complies with the USB 2.0 standard for high-speed (480 Mb/s) functions and with the On-The-Go supplement to the USB 2.0 specification.
- Supports OTG communications with one or more high-, full- or low-speed device.

- Supports *Session Request Protocol* (**SRP**) and *Host Negotiation Protocol* (**HNP**).
- Supports Suspend and Resume signaling.
- Soft connect/disconnect.
- Supports 5 additional (to endpoint 0) transmit endpoints and 5 additional (to endpoint 0) receive endpoints.
- Supports Tx bulk packet splitting (refer to Bulk Transactions section for details).
- Supports Rx bulk packet combining (refer to Bulk Transactions section for details).
- Supports high bandwidth Tx ISO endpoints.
- Supports high bandwidth Rx ISO endpoints.
- Supports full multipoint capability (refer to Multiple Devices Support section for details).
- Offers dynamic allocation of endpoints, to maximize number of devices supported.
- Signle total RAM with size of 8 K.
- Supports dynamic FIFO sizing (allocation space) from the single total RAM.
- Synchronous 32 bits wide RAM interface for FIFOs.
- Supports DMA access to FIFOs through the Included DMA Controller.
- Included DMA Controller supports 5 DMA channels.
- Performs all transaction scheduling in hardware.
- Supports Link Power Management.
- Supports access to the USB 2.0 PHY registers via the USB 2.0 Controller registers space, by using `VCONTROL` and `VSTATUS`.

14.1.1. USB 2.0 Controller Functional Description

This section covers the following topics:

- Resets
- Modes of Operation
- Suspend/Resume
- Multiple Devices Support
- Connect/Disconnect
- Programming Scheme
- OTG Session Request
- Host Negotiation
- Fundamental DMA Support
- Included DMA Controller
- VBus Events
- Dynamic FIFO Sizing
- Control Transactions (via Endpoint 0)
- Bulk Transactions
- Full-Speed/Low-Bandwidth Interrupt Transactions
- Full-Speed/Low Bandwidth Isochronous Transactions
- High Bandwidth Isochronous/Interrupt Transactions

14.1.1.1. Resets

This section covers the following topics:

- In Peripheral Mode
- In Host Mode

14.1.1.1.1. In Peripheral Mode

When the USB 2.0 Controller is acting as a peripheral and a reset condition is detected on the USB, the device will perform the following actions:

- Sets `FADDR` to 0.
- Sets `INDEX` to 0.
- Flushes all endpoint FIFOs.
- Clears all control/status registers.
- Enables all endpoint interrupts.
- Generates a Reset interrupt.

If the `HS_ENAB` bit in the `POWER` was set, the USB 2.0 Controller also tries to negotiate for high-speed operation.

Whether high-speed operation is selected is indicated by `HS_MODE` bit of `POWER`.

When the application software driving the USB 2.0 Controller receives a Reset interrupt, it should close any open pipes and wait for bus enumeration to begin.

14.1.1.1.2. In Host Mode

If the `RESET` bit in the `POWER` is set while the USB 2.0 Controller is in Host mode, the USB 2.0 Controller will generate Reset signaling on the bus. If the `HS_ENAB` bit in the `POWER` was set, it will also try to negotiate for high-speed operation.

The CPU should keep the Reset bit set for at least 20 ms to ensure correct resetting of the target device.

After the CPU has cleared the bit, the USB 2.0 Controller will start its frame counter and transaction scheduler. Whether high-speed operation is selected will be indicated by `HS_MODE` bit of `POWER`.

14.1.1.2. Modes of Operation

The USB 2.0 Controller has two main modes of operation – peripheral mode and host mode.

In peripheral mode, the USB 2.0 Controller encodes, decodes, checks and directs all USB packets sent and received. IN transactions are handled through the device's Tx FIFOs, OUT transactions are handled through its Rx FIFOs. Control, Bulk, Isochronous and Interrupt transactions are supported.

In Host mode, the way in which the USB 2.0 Controller behaves depends on whether it is linked up for point-to-point communications with another USB function or whether it is attached to a hub:

- When attached to another USB function, the USB 2.0 Controller offers the range of capabilities needed in order to act as the host in point-to-point communications with this USB function.
- When attached to a hub, it provides the facilities required to act as the host to a number of devices, supported simultaneously.

When operating in Host mode and used for point-to-point communications with a single other USB device (which can be high-, full- or low-speed), the USB 2.0 Controller can support Control, Bulk, Isochronous or Interrupt transactions. IN transactions are handled through the Rx FIFOs, OUT transactions are handled through the Tx FIFOs. As well as encoding, decoding and checking the USB packets sent and received, the USB 2.0 Controller will also automatically schedule Isochronous endpoints and Interrupt endpoints to perform one transaction every `n` frames/microframes (or up to

three transactions if the high-bandwidth option is selected), where `n` represents the polling interval that has been programmed for the endpoint. The remaining bus bandwidth is shared equally amongst the Control and Bulk endpoints.

When attached to a hub, the USB 2.0 Controller continues to offer the above facilities but it further needs to be programmed with details of:

- The function address of the target device.
- The operating speed of the target device (so that the appropriate speed conversion can be carried out).
- If the target device is a full- or low-speed device that is accessed through a high-speed hub, the endpoint additionally needs to be programmed with the function address and port number of the hub.

The device may be required to power the VBus to 5 V as the 'A' device of the connection (source of power and default host) or, as the 'B' device (default peripheral), to be able to wake the 'A' device by charging the VBus to 2 V. Outputs from the USB 2.0 Controller indicate when these charging options are required.

Whether the USB 2.0 Controller initially operates in Host mode or in Peripheral mode depends on whether it is being used as an 'A' device or a 'B' device, which in turn depends on whether the `IDDIG` input is low or high.

When the USB 2.0 Controller is operating as an 'A' device, it is initially configured to operate in Host mode. When operating as a 'B' device, the USB 2.0 Controller is initially configured to operate in Peripheral mode.

However, a `HOST_REQ` bit is provided in the `DEVCTL` through which the CPU can request that the 'B' device becomes the Host the next time there is no activity on the USB bus.

The `IDDIG` input reflects the state of the `ID` pin of the device's mini-AB receptacle, with `IDDIG` being low indicating an 'A' plug i.e. operation as an 'A' device, and `IDDIG` being high indicating a 'B' plug and operation as a 'B' device.

Information on whether the USB 2.0 Controller is acting as an 'A' device or as a 'B' device and on whether the device it is connected to is high-, full- or low-speed is also recorded in the `DEVCTL`, along with information about the level of the VBus relative to the high and low voltage thresholds used to signal Session Start and Session End.

The procedures for session request and for transferring host/peripheral roles between the devices at either end of the connection are described in OTG Session Request and Host Negotiation sections, respectively. The transfers that are made are all subject to the standard USB data transfer protocols.

14.1.1.3. Suspend/Resume

This section covers the following topics:

- Control by Inactivity on USB Bus (L0 to L2 State)
- Control by LPM Transaction (L0 to L1 State)

With the introduction of Link Power Management, there are two basic methods for the USB 2.0 Controller to be suspended and resumed. These two methods are demonstrated in the basic LPM transaction diagram shown below.

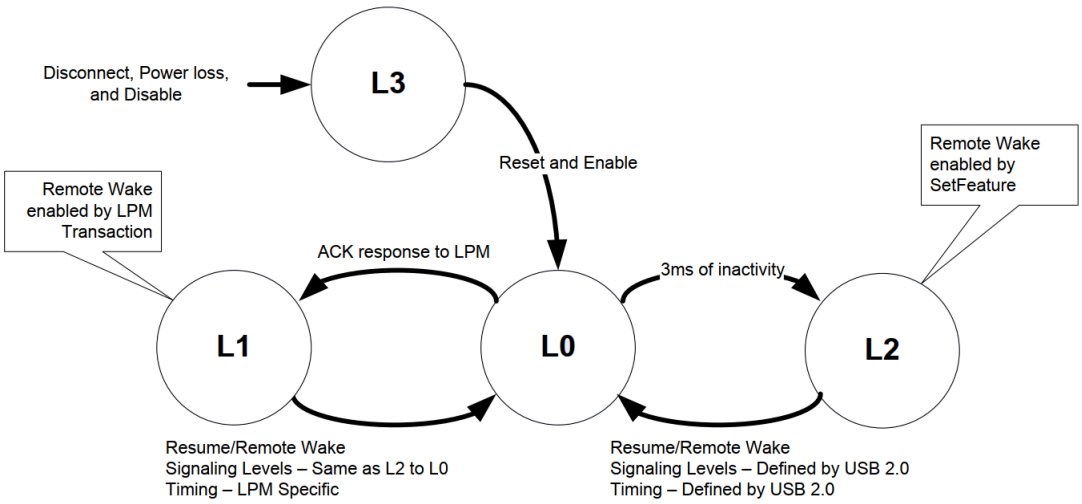

Figure 14-2 Link Power Management diagram

The procedure in which the USB 2.0 Controller is suspended and resumed depends on whether the core is operating as a device or a host and the method of suspend desired.

14.1.1.3.1. Control by Inactivity on USB Bus (L0 to L2 State)

This section covers the following topics:

- In Peripheral Mode
- In Host Mode

14.1.1.3.1.1. In Peripheral Mode

1. **Entry into Suspend mode.** When operating as a peripheral, the USB 2.0 Controller monitors activity on the USB and when no activity has occurred for 3 ms, it goes into Suspend mode. If the Suspend interrupt has been enabled, an interrupt will be generated at this time.
2. **When Resume signaling occurs on the bus,** the USB 2.0 Controller will automatically exit Suspend mode. If the Resume interrupt is enabled, an interrupt will be generated.
3. **Initiating a Remote Wakeup.** If the software wants to initiate a remote wakeup while the USB 2.0 Controller is in Suspend mode, it should write to the `POWER` to set the `RESUME` bit to '1'.

   The software should leave then this bit set for approximately 10 ms (minimum of 2 ms, a maximum of 15 ms) before resetting it to 0. By this time the hub should have taken over driving Resume signaling on the USB.

No Resume interrupt will be generated when the software initiates a remote wakeup.

14.1.1.3.1.2. In Host Mode

1. **Entry into Suspend mode.** When operating as a host, the USB 2.0 Controller can be prompted to go into Suspend mode by setting the `SUSPEND_MODE` bit in the `POWER`. When this bit is set, the USB 2.0 Controller will complete the current transaction then stop the transaction scheduler and frame counter. No further transactions will be started and no SOF packets will be generated.

   If the `ENABLE_SUSPENDM` bit of `POWER` is set, the PHY will go into low-power mode when the USB 2.0 Controller goes into Suspend mode.

2. **Sending Resume Signaling.** When the application requires the USB 2.0 Controller to leave Suspend mode, it needs to clear the `SUSPEND_MODE` bit in the `POWER`, set the `RESUME` bit in this

register, and leave it set for 20 ms. While the RESUME bit is high, the USB 2.0 Controller will generate Resume signaling on the bus. After 20 ms, the CPU should clear the RESUME bit, at which point the frame counter and transaction scheduler will be started.

3. **Responding to Remote Wake-up.** If Resume signaling is detected from the target while the USB 2.0 Controller is in Suspend mode, the PHY will be brought out of low-power mode. The USB 2.0 Controller will then exit Suspend mode and automatically set the RESUME bit in the POWER to '1' to take over generating the Resume signaling from the target. If the Resume interrupt is enabled, an interrupt will be generated.

14.1.1.3.2. Control by LPM Transaction (L0 to L1 State)

This section covers the following topics:

- In Peripheral Mode
- In Host Mode

14.1.1.3.2.1. In Peripheral Mode

Entry into Suspend mode.

When operating as a peripheral, the USB 2.0 Controller will never initiate an LPM Suspend (transition from the L0 state to the L1 state). Rather, the USB 2.0 Controller will only Suspend at the request of the host. However, for this to occur, the LPM feature must be enabled by setting up the LPM_CNTRL appropriately. The register field LPMEN (in the LPM_CNTRL) is used to enable and support extended and LPM transactions. The LPMXMT field is used instruct the hardware that it is ready to suspend and to respond to the next LPM transaction with an ACK. In this case, the USB 2.0 Controller will respond to the next LPM transaction with an ACK if all other conditions are met. The response to an LPM transaction by the USB 2.0 Controller is summarized below.

Table 14-1 Response to LPM transaction from USB 2.0 Controller

LPM_CNTRL.LPMXMT Field Value	LPM_CNTRL.LPMEN Register Value	Data Pending (Data resides in the Tx FIFOs)	Response to the next LPM transaction
0x0	0x0	Don't Care	Timeout
0x0	0x2		
0x1	0x0		
0x1	0x2		
0x0	0x1	Don't Care	STALL
0x1			
0x0	0x3	Don't Care	NYET
0x1	0x3	Yes	NYET
0x1	0x3	No	ACK

For all cases shown above in which the USB 2.0 Controller responds (No timeout occurs), an LPM interrupt will be generated in the LPM_INTR. Note that the USB 2.0 Controller will respond with an ACK only if there is no data pending in any of the Tx Endpoint FIFOs. If there is data pending, the USB 2.0 Controller will respond with a NYET.

Once an LPM transaction is successfully received three events will occur:

- The LPM_ATTR will be updated with values received in the LPM transaction just received.

  This register contains the following fields:

  - LINKSTATE — This field tells the USB 2.0 Controller what state to transition to. The only valid value for this field is 0x1 indicating that the core should suspend. For any other value, the USB 2.0 Controller will respond with a STALL and the appropriate interrupt will be generated. However, in this case the LPM_ATTR will be updated so that software can observe the non compliant LPM packet payload. In addition, the LPMERR interrupt will also be generated informing software of the non-compliant LPM transaction (see LPM_INTR).
  - HIRD — This field tells the USB 2.0 Controller the minimum duration that the host will drive resume signalling on the bus. This field represents a resume duration range from 50 µs to 1200 µs. This value may be different for subsequent LPM transactions.
  - RMTWAK — This is a 1 bit field indicating if the remote wakeup by the USB 2.0 Controller is allowed. This bit only applies to the current suspend/resume cycle only. This bit may be different for subsequent LPM transactions. This bit should not supersede the wakeup capability that was previously negotiated on enumeration of the USB 2.0 Controller.

- The USB 2.0 Controller will suspend 9 µs after transmitting the ACK. Resume signaling can be driven by the Host or the USB 2.0 Controller 50 µs after this event. During this 9 µs interval, the host may continue to transmit the LPM transaction.

  The USB 2.0 Controller will respond with an ACK in this case regardless of the LPMXMT value in the LPM_CNTRL register.

- An interrupt will be generated informing software of the response (an ACK in this case). An ACK response is the indication to software that the USB 2.0 Controller has suspended.

Since the primary purpose of LPM is to save power, the software will read the LPM_ATTR to determine the attributes of the Suspend. Software must make a determination based on these attributes whether additional power savings in the system can be exploited. In making this determination it is noted that if the host initiates the resume signalling, the USB 2.0 Controller will be required to respond to packet transmissions within the time specified by HIRD + 10 µs.

When resume signalling occurs on the Bus.

When the host resumes the bus, it will drive resume signalling for a minimum time specified by the HIRD field in the LPM_ATTR. The USB 2.0 Controller must be able to respond to traffic within the time HIRD + 10 µs. The USB 2.0 Controller will transition to a normal operating state automatically and a resume interrupt will be generated in LPMRES bit of LPM_INTR.

To facilitate the resume timing requirement, an additional feature, LPMNAK, is provided in the LPM_CNTRL. If LPMNAK is set to 0x1, all endpoints will respond to any transaction (other then an LPM) with a NAK.

This bit will only take effect after the USB 2.0 Controller has LPM suspended. Typically, this bit would be asserted when the LPMXMT field of the LPM_CNTRL is also asserted. Software can continue to restore the system to normal operation while the USB 2.0 Controller responds to all transactions with a NAK. After the system has been completely restored, software can then clear the LPMNAK field is the LPM_CNTRL.

Initiating remote wakeup.

If the software wants to initiate a remote wakeup while the USB 2.0 Controller is in Suspend mode, it should write a 0x1 to the LPMRES bit in the LPM_CNTRL. This bit is self clearing. Writing a 0x1 will cause resume signaling to be driven on the bus for 50 µs. The host will respond by driving resume for 60 µs to 990 µs. 10 us after the host stops driving resume, the USB 2.0 Controller will transition to its normal operational state and will be ready for packet transmission. A Resume interrupt will be generated in LPMRES bit of the LPM_INTR.

14.1.1.3.2.2. In Host Mode

Entry into Suspend mode.

When operating as a host, the USB 2.0 Controller will initiate an LPM Suspend (transition from the L0 state to the L1 state) by initiating an LPM transaction as follows:

- Software will set-up the desired attributes of the Suspend in the LPM_ATTR. Enabling remote wakeup and a large HIRD will give the peripheral more opportunity to conserve power.
- All LPM interrupts should be enabled in the LPM_INTREN.
- Software should initiate the transaction by writing a 0x1 to the LPM_CNTRL.
- An interrupt is generated to inform software of the response to the LPM transaction. If an ACK was received, then the USB 2.0 Controller will suspend automatically within 8 µs. This is the indication that the USB 2.0 Controller has suspended.

If the response from the device has a bit stuff error or a PID error, then an LPM_INTR.LPMERR interrupt is generated. The hardware will immediately attempt the LPM transaction two more times. The device will not suspend for 8 us after the initial LPM so it will be able to respond to either of these subsequent LPM transactions. If an LPM timeout has occurred three times, the LPM_INTR.LPMNC and the LPM_INTR.LPMERR interrupts will be set. At this time, software is unaware of the device state and must deduce it by other means.

Sending Resume Signaling.

Resume signaling should be generated by software as follows:

- All LPM interrupts should be enabled in the LPM_INTREN register.
- Software should write the LPMRES bit in the LPM_CNTRL. This bit is self clearing. This will cause resume signalling to be generated on the Bus for the time that is currently specified in the HIRD field in the LPM_ATTR. It is assumed by hardware that this value was used in the last LPM transaction that caused the Suspend.
- After HIRD + 10 µs, the USB 2.0 Controller will transition to its normal operational state and be ready for packet transmission. A resume interrupt will be generated in the LPM_INTR.LPMRES.

Responding to Remote Wake-up.

If the remote wakeup feature was enabled in the LPM transaction that caused the Suspend, then the device may drive resume signaling on the bus. When this occurs, the device will drive resume signaling BUS for 50 µs. The USB 2.0 Controller will immediately begin driving resume signaling on the BUS and will do so for 60 µs. 10 µs after completion of the resume signaling, the USB 2.0 Controller will transition to its normal operating state and will be ready for packet transmission. At this time, the resume interrupt is generated in the LPM_INTR.LPMRES.

14.1.1.4. Multiple Devices Support

This section covers the following topics:

- Allocating Devices to Endpoints
- Operation
- Bandwidth Issues

The USB 2.0 Controller has the facility, when operating in Host mode, to act as the host to a range of USB peripheral devices – high-speed, full-speed or low-speed – where these devices are connected to the USB 2.0 Controller via a USB hub.

The key feature of the core's support for multiple devices is its facility to allow the functions of the target devices to be individually allocated to the different Rx and Tx endpoints implemented in the USB 2.0 Controller core. Furthermore, this allocation can be made dynamically, allowing the devices from the targeted peripheral list to be used in different combinations. The combinations of peripheral devices that may be used together are however limited by the numbers of Tx and Rx endpoints implemented in the core. Further devices can only be added where the endpoints they require remain available.

14.1.1.4.1. Allocating Devices to Endpoints

The separate functions of the connected devices are allocated to the endpoints within the USB 2.0 Controller core through a group of three registers, which are associated with each Rx or Tx endpoint implemented in the core (including Endpoint 0).

The registers concerned are `TXFUNCADDR`/`RXFUNCADDR`, `TXHUBADDR`/`RXHUBADDR` and `TXHUBPORT`/`RXHUBPORT` (the location of these registers depends on which of the USB 2.0 Controller's endpoints is being addressed).

The information that needs to be recorded in the `TXFUNCADDR`/`RXFUNCADDR` is the address of the target function that is to be accessed through the selected endpoint. This information needs to be recorded separately for each Tx and Rx endpoint that is used. In particular, both `TXFUNCADDR` and `RXFUNCADDR` need to be set for Endpoint 0.

The `TXHUBADDR`/`RXHUBADDR` and `TXHUBPORT`/`RXHUBPORT` are provided for the case where a full- or low-speed device is connected to the USB 2.0 Controller via a high-speed USB 2.0 hub, which carries out the required transaction translation between high-speed transmission and low-/full-speed transmission. Where this is the case, the `TXHUBADDR`/`RXHUBADDR` and `TXHUBPORT`/`RXHUBPORT` need to record the address of the hub that carries out the transaction translation and the port of that hub through which the associated Tx/Rx endpoint needs to access the device.

If Endpoint 0 is connected to a hub, then both the Tx and the Rx versions of these registers need to be set for this endpoint.

The `TXHUBADDR`/`RXHUBADDR` is further used to record whether the hub offers multiple transaction translators or just a single transaction translator. This has a significant effect on the overall bandwidth that can be achieved.

In addition to recording the address of the target function through these three registers, the endpoint number and operating speed of the target device and the type of transaction that will be executed need to be recorded.

For a Tx endpoint, this information needs to be set in the `TXTYPE` when the `INDEX` is set to select the required endpoint. For an Rx endpoint, this information needs to be set in the `RXTYPE` when the `INDEX` is set to select the required endpoint. In both cases, the endpoint number is recorded in bits [3:0], the transaction type is selected through bits [5:4], and the operating speed is selected through bits [7:6].

In the case of Endpoint 0, just the speed needs to be set (this endpoint only having the facilities to handle Control transactions and therefore always being associated with a device Endpoint 0). This speed setting is made through bits [7:6] of the `TYPE0` when the `INDEX` is set to 0.

14.1.1.4.2. Operation

Once the allocation of functions to endpoints has been made and the operating speed of the target device recorded as described in Allocating Devices to Endpoints section, most operations in a Multi-point set-up are no different from those for the equivalent actions where the core is attached to a single other device. The details are given elsewhere in this Reference Manual.

However, additional steps are required:

- Where the option of dynamically switching the allocation of functions to endpoints is taken (e.g. to allow a wider range of devices to be supported).
- Where the control packets normally associated with Endpoint 0 are handled through a different endpoint.

If dynamic allocation is used, it becomes essential for the user to keep track of the current data toggle state associated with the endpoint and with each of the devices that are allocated to that endpoint. Knowledge of this state is necessary to allow the user to select the correct data toggle state when the switch is made between one device and other (this action is the user's responsibility because the core cannot determine what data toggle state is expected when a function is being switched in and out of use).

The data toggle state can be switched from its current state by writing to the appropriate Tx/Rx control and status register to set the `TXCSRH.DATA_TOGGLE_WRITE_ENABLE`, `RXCSRH.DATA_TOGGLE_WRITE_ENABLE` and `TXCSRH.DATA_TOGGLE`, `RXCSRH.DATA_TOGGLE` bits that are included in these registers when the core is in Host mode.

`DATA_TOGGLE_WRITE_ENABLE` and `DATA_TOGGLE` bits are also included in `CSR0H` when the core is operating in Host mode. However, Control operations carried out through the core's Endpoint 0 should normally always leave the data toggle in the expected state.

Where control packets are handled through an endpoint other than Endpoint 0, the user has additionally to prompt for each Setup token to be sent. This involves setting the `TXCSRL.SETUP_PKT` bit when the core is operating in Host mode, alongside the `TXPKTRDY` bit. If the `SETUP_PKT` bit is not set, an OUT token will be sent.

Overall, the recommendation is to use the core's Endpoint 0 to handle Control packets for all of the devices attached to the core, and to switch the allocation of this endpoint as appropriate. The issue of sending the correct token is then taken care of, as is the issue of ensuring that the data toggle is correctly set for this endpoint.

Using a different endpoint for this function is possible, as described above, but the further points to note are:

- The control function must be allocated to an Rx/Tx endpoint pair (i.e. with the same endpoint number).
- The chosen endpoints must each be associated with FIFOs that can accommodate the packet size associated with EP0 transactions at the chosen operating speed (i.e. a minimum of 8 bytes for Low- or Full-speed transactions but 64 bytes for High-speed transactions).

14.1.1.4.3. Bandwidth Issues

The ability of a multi-point system to cope with isochronous transactions (in particular) is determined by the available bandwidth.

Once an endpoint has been set up, all scheduling is handled in hardware. However, as with PC-based EHCI/OHCI/UHCI hosts, before opening a periodic pipe (for use by isochronous or interrupt traffic), software must determine that there is sufficient bandwidth available. Further, if the periodic pipe is opened to a full-speed device through a high-speed hub, software must confirm that sufficient bandwidth is available both on the local high-speed bus and the full-speed bus generated by the transaction translator in the hub.

The bandwidth required for different transactions can be determined using similar algorithms to those used in connection with PC-based hosts (detailed in ***Section 5.11.3*** of the ***Universal Serial Bus Specification, Revision 2.0***).

As would be expected, the bandwidth available will be greater where the hub used supports multiple transaction translators.

14.1.1.5. Connect/Disconnect

This section covers the following topics:

- In Host Mode
- In Peripheral Mode

The particular behavior related to connecting and disconnecting the USB 2.0 Controller concerns its use either in Host mode or Peripheral mode in peer-to-peer communications.

14.1.1.5.1. In Host Mode

Where the USB 2.0 Controller is operating in Host mode, the CPU starts the session by setting the `SESSION` bit of the `DEVCTL`. Power is then applied to VBus and the core waits for a device to be connected.

When a device is detected, a Connect interrupt is generated (i.e. `INTRUSB.CONN` goes high). The speed of the device that has been connected can be determined by reading the `DEVCTL` where the `FSDEV` bit will be high for a high-speed/full-speed device and the `LSDEV` bit will be high for a low-speed device. The CPU should then reset the device. If both `FSDEV` and `POWER.HS_ENAB` are set, the USB 2.0 Controller will try to negotiate for high-speed operation. Whether this is successful will be indicated by the `POWER.HS_MODE` bit.

The CPU should keep the Reset bit set for 20 ms to ensure that the target is reset. It can then begin device enumeration.

If the device is disconnected while a session is in progress, a Disconnect interrupt will be generated (i.e. `INTRUSB.DISCON` goes high).

14.1.1.5.2. In Peripheral Mode

Where the USB 2.0 Controller is operating in Peripheral Mode, no interrupt is generated when the device is connected to the host.

However a Disconnect interrupt (`INTRUSB.DISCON`) is generated when the host terminates a session.

14.1.1.6. Programming Scheme

This section covers the following topics:

- Soft Connect/Disconnect
- USB Interrupt Handle

This and the following sections look at the actions that the device controlling the USB 2.0 Controller core will need to perform and at the aspects of the operation of the core that affect this.

Throughout this discussion, the controlling device is assumed to be a microcontroller running some firmware but it could be a customized hard-wired logic block.

14.1.1.6.1. Soft Connect/Disconnect

If required, the USB 2.0 Controller will allow its connection to the USB bus to be controlled by software.

When the Soft Connect/Disconnect is selected, then when the USB 2.0 Controller is operating in Peripheral Mode, the PHY used alongside the USB 2.0 Controller can be switched between normal mode and non-driving mode by setting/clearing `POWER.SOFT_CONN` bit.

When `SOFT_CONN` bit is set to 1, the PHY is placed in its normal mode and the D+/D- lines of the USB bus are enabled. At the same time, the USB 2.0 Controller is placed in 'Powered' state, in which it will not respond to any USB signaling except a USB reset.

When this feature is enabled and the `SOFT_CONN` bit is zero, the PHY is put into non-driving mode, D+ and D- are tri-stated and the USB 2.0 Controller appears to other devices on the USB bus as if it has been disconnected.

After a hardware reset (`nrst = 0`), `POWER.SOFT_CONN` is cleared to 0. The USB 2.0 Controller will therefore appear disconnected until the software has set `SOFT_CONN` to 1. The application software can then choose when to set the PHY into its normal mode. Systems with a lengthy initialization procedure may use this to ensure that initialization is complete and the system is ready to perform enumeration before connecting to the USB.

Once the `SOFT_CONN` bit has been set to 1, the software can also simulate a disconnect by clearing this bit to 0.

14.1.1.6.2. USB Interrupt Handle

When the software is interrupted with a USB interrupt, it needs to read the interrupt status register to determine which endpoint(s) have caused the interrupt and jump to the appropriate routine. If multiple endpoints have caused the interrupt, Endpoint 0 should be serviced first, followed by the other endpoints.

A flowchart of the USB Interrupt Service Routine is shown below.

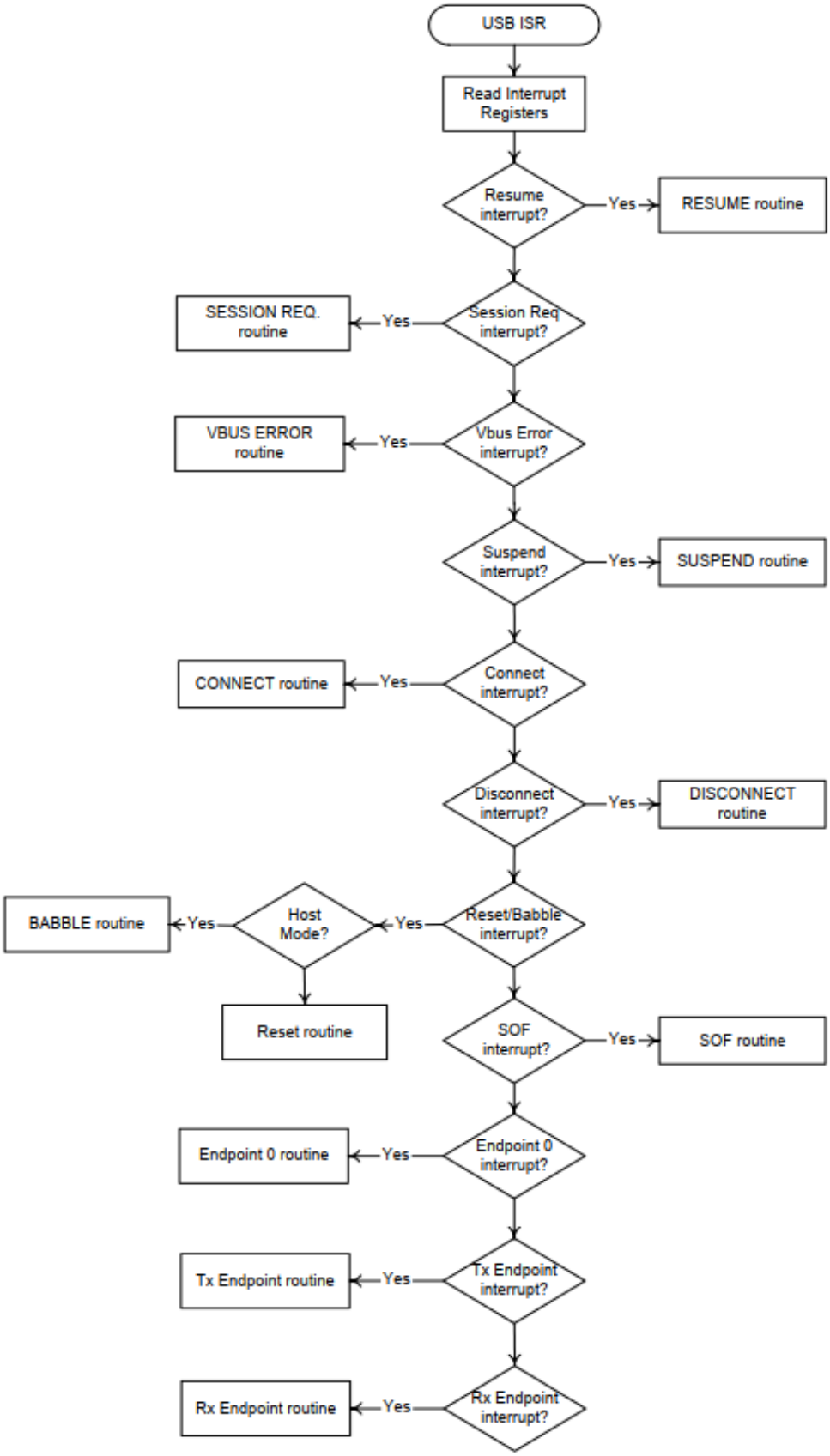

Figure 14-3 USB Interrupt Service Routine diagram

14.1.1.7. OTG Session Request

This section covers the following topics:

- Starting Session
- Detecting Activity

In order to conserve power, the USB On-The-Go supplement allows VBus to only be powered up when required and to be turned off when the bus is not in use.

VBus is always supplied by the 'A' device on the bus. The USB 2.0 Controller determines whether it is the 'A' device or the 'B' device by sampling the `iddig` input from the PHY. This signal is pulled low when an 'A-type' plug is sensed (signifying that the USB 2.0 Controller is the 'A' device) but taken high when a 'B-type' plug is sensed (signifying that the USB 2.0 Controller is the 'B' device).

14.1.1.7.1. Starting Session

When the device containing the USB 2.0 Controller requires starting a session, the software needs to set the `SESSION` bit in the `DEVCTL`. The USB 2.0 Controller will then enable ID pin sensing. This results in the `iddig` input either being taken low if an A-type connection is detected or high if a B-type connection is detected. The `B_DEVICE` bit in the `DEVCTL` is also set to indicate whether the USB 2.0 Controller has adopted the role of the 'A' device or the 'B' device.

If the USB 2.0 Controller is the 'A' device: The USB 2.0 Controller will then enter Host mode (the 'A' device is always the default host), turn on VBus and wait for VBus to go above the VBus Valid threshold, as indicated by the `VBUSVALID` input going high (this event also causes the `VBUS` bits in the `DEVCTL` [4:3] to go to 0x3).

The USB 2.0 Controller will then wait for a peripheral to be connected. When a peripheral is detected, a Connect interrupt (`INTRUSB.CONN`) will be generated (if enabled) and either the `FSDEV` or `LSDEV` bit in the `DEVCTL` (bit [6] or bit [5] respectively) will be set depending on whether a high-speed/full-speed peripheral or a low-speed peripheral was detected.

The software should then reset this peripheral. If both `FSDEV` and `HS_ENAB` bit of the `POWER` are set, the USB 2.0 Controller will monitor to see if a high-speed chirp is received from the peripheral. If a chirp is received, the USB 2.0 Controller will respond with high-speed chirps and enter High-Speed mode.

To end the session, the software should clear the `SESSION` bit of the `DEVCTL`. The USB 2.0 Controller will also automatically end the session if babble is detected.

If the USB 2.0 Controller is the 'B' device: The USB 2.0 Controller will request a session using the Session Request Protocol defined in the USB On-The-Go supplement, i.e. it will first discharge VBus.

Then when VBus has gone below the Session End threshold (as indicated by the `sessend` input going high and the `VBUS` bits (bits [4:3]) in the `DEVCTL` going to 0x0) — and the line state has been SE0 for > 2 ms — the USB 2.0 Controller will first pulse the data line, then pulse VBus.

At the end of the session, the `SESSION` bit of the `DEVCTL` is cleared — usually by the USB 2.0 Controller but it can also be cleared by the software if the application software wishes to perform a software disconnect. The USB 2.0 Controller will cause the PHY to switch out the pull-up resistor on D+. This signals the 'A' device to end the session.

14.1.1.7.2. Detecting Activity

When the other device of the OTG set-up wishes to a session to start, it will either raise VBus above the Session Valid threshold if it is the 'A' device (as indicated by state in the `VBUS` bits (bits [4:3]) in the `DEVCTL` going to 0x2) or, if it is the 'B' device, it will first pulse the data line, then pulse VBus. Depending on which of these actions happens, the USB 2.0 Controller can determine whether it is the 'A' device or the 'B' device in the current set-up and act accordingly as follows:

If VBus is raised above the Session Valid threshold: then the USB 2.0 Controller is the 'B' device. The USB 2.0 Controller will set the SESSION bit in the DEVCTL. When Reset signaling is detected on the bus, a Reset interrupt (INTRUSB.RESET_OR_BABBLE) will be generated (if enabled) which the software should interpret as the start of a session. The USB 2.0 Controller will be in Peripheral mode at this point as the 'B' device is the default peripheral.

At the end of the session, the 'A' device will turn off the power to VBus. When VBus drops below the Session Valid threshold (as indicated by state in the VBUS bits (bits [4:3]) in the DEVCTL going to 0x1), the USB 2.0 Controller will detect this and clear the SESSION bit to indicate that the session has ended. A Disconnect interrupt (INTRUSB.DISCON) will also be generated (if enabled).

If data line/VBus pulsing is detected: then the USB 2.0 Controller is the 'A' device. It will generate a Session Request interrupt (INTRUSB.SESS_REQ – if enabled) to indicate that the 'B' device is requesting a session. The software should then start a session by setting the SESSION bit of the DEVCTL.

14.1.1.8. Host Negotiation

- When the USB 2.0 Controller is the 'A' device (iddig low, B-Device (DEVCTL.B_DEVICE = 0)), it will automatically enter Host mode when a session starts.
- When the USB 2.0 Controller is the 'B' device (iddig high, B-Device (DEVCTL.B_DEVICE = 1)), it will automatically enter Peripheral mode when a session starts.

  The software can however request that the USB 2.0 Controller becomes the Host by setting the HOST_REQ bit in the DEVCTL. This bit can be set either at the same time as requesting a Session Start by setting the SESSION bit of the DEVCTL or at any time after a session has started. When the USB 2.0 Controller next enters Suspend mode (no activity on the bus for 3 ms), then assuming the HOST_REQ bit remains set, it will enter Host mode and begin host negotiation (as specified in the USB On-The-Go supplement), causing the PHY to disconnect the pull-up resistor on the D+ line. This should cause the 'A' device to switch to Peripheral mode and connect its own pull-up resistor. When the USB 2.0 Controller detects this, it will generate a Connect interrupt (INTRUSB.CONN) if this is enabled. It will also set the RESET bit in the POWER to begin resetting the 'A' device. (The USB 2.0 Controller begins this reset sequence automatically to ensure that reset is started as required within 1 ms of the 'A' device connecting its pull-up resistor). The software should wait at least 20 ms, then clear the RESET bit and enumerate the 'A' device.

When the USB 2.0 Controller-based 'B' device has finished using the bus, the software should put it into Suspend mode by setting the SUSPEND_MODE bit in the POWER. The 'A' device should detect this and either terminate the session or revert to Host mode. If the 'A' device is USB 2.0 Controller-based, it will generate a Disconnect interrupt (INTRUSB.DISCON) if this is enabled.

14.1.1.9. Fundamental DMA Support

The USB 2.0 Controller supports DMA access to the FIFOs for Tx Endpoints 1 – 5 and Rx Endpoints 1 – 5 by Included DMA Controller.

For total of 5 Tx Endpoints and 5 Rx Endpoints (in addition to Endpoint 0), DMA_REQ[0] ... DMA_REQ[4] are associated with Tx Endpoints 1 ... 5; DMA_REQ[5] ... DMA_REQ[9] are associated with Rx Endpoints 1 ... 5. These request lines are available at the top-level of the core to allow their use by Included DMA Controller.

The DMA request lines are individually enabled through the DMAREQENAB bit in the appropriate TXCSRH (bit 4) or RXCSRH (bit 5) and operate in two modes, referred to as DMA Request Mode 0 and DMA Request Mode 1. The choice of operating mode is made through the DMAREQMODE bit of TXCSRH for Tx endpoints and RXCSRH for Rx endpoints.

For Rx endpoints operating in Request Mode 0, the DMA request line goes high when a data packet is available in the endpoint FIFO and normally goes low at the end of the cycle in which the 8th from last byte starts to be processed. The request line will also go low if the software clears the RXPKTRDY bit of the CSR0L.

The behavior of the DMA request lines for Rx endpoints in Request Mode 1 is similar except the request line will only go high when the packet received is of the maximum packet size (as set in the RXMAXP register). If the packet received is of some other size, the DMA request line will stay low. Note, however, that if the Request Mode is switched from Request Mode 1 to Request Mode 0, the request line will be asserted if there is a packet in the FIFO in order to allow this 'pre-received' packet to be downloaded.

For Tx endpoints operating in either Request Mode 0 or Request Mode 1, the DMA request line will go high when the endpoint FIFO is able to accept a data packet. The request line will also go low if the software sets the TXPKTRDY bit of the CSR0L.

When operating in Host mode, if either the RX_STALL bit of the TXCSRL or ERROR bit of the TXCSRL becomes set following three failed attempts to transmit a packet, the DMA request line will be disabled until the RX_STALL/ERROR bit has been cleared.

The mode selected also affects the generation of Endpoint interrupts (if enabled). In DMA Request Mode 0, no interrupt is generated when packets are received but the appropriate Endpoint interrupt is generated to prompt the loading of all packets. In DMA Request Mode 1, the Endpoint interrupt is suppressed except following the receipt of a short packet (i.e. one of less than RXMAXP bytes).

The conditions under which Tx and Rx Endpoint interrupts are generated are summarized in the following tables.

Table 14-2 EP interrupt associated with RXPKTRDY being set

DMAREQENAB	DMAREQMODE	Interrupt generated?
0	X	Yes
1	0	No
1	1	Only if short packet

Table 14-3 EP interrupt associated with TXPKTRDY being cleared

DMAREQENAB	DMAREQENAB	Interrupt generated?
0	X	Yes
1	0	Yes
1	1	No

DMA Request Mode 0 can be used equally well for Bulk, Interrupt or Isochronous transfers. Indeed, if the endpoint is configured for Isochronous transfers, DMA Request Mode 0 should always be selected where DMA is used.

DMA Request Mode 1 is chiefly valuable where large blocks of data are transferred to a Bulk endpoint. The USB protocol requires such packets to be split into a series of packets of the maximum packet size for the endpoint (512 bytes for high speed, 64 bytes for full speed). Note that TXMAXP/RXMAXP must be set to an even number of bytes for proper interrupt generation. DMA Request Mode 1 can be

used, with a suitably programmed DMA controller, to avoid the overhead of having to interrupt the processor after each individual packet: instead the processor is only interrupted after the transfer has completed. In some cases, the block of data transferred will comprise a pre-defined number of these packets, that the controlling software 'counts' through the transfer process. In other cases, the last packet in the series may be less than the maximum packet size and the receiver may use this 'short' packet to signal the end of the transfer. (If the total size of the transfer is an exact multiple of the maximum packet size, the transmitting software should send a null packet for the receiver to detect.)

TXMAXP/RXMAXP must be set to an even number of bytes for proper interrupt generation in Mode 1.

Further information on using DMA for Bulk transfers is given in Employing DMA section.

DMA transfers may be 8-bit, 16-bit, or 32-bit as required. However, all the transfers associated with one packet (with the exception of the last) must be of the same width so that the data is consistently byte-, word- or double-word-aligned. The last transfer may contain fewer bytes than the previous transfers in order to complete an odd-byte or odd-word transfer.

DMA Requests should be disabled before the DMA Request Mode is changed. In particular, the DMAREQMODE bit in the TXCSRH should not be set to zero either before or in the same cycle as the corresponding DMAREQENAB bit is cleared to zero.

14.1.1.10. Included DMA Controller

This section covers the following topics:

- DMA Bus Cycles
- Bus Errors
- Transferring Packets

The USB 2.0 Controller include a multi-channel DMA controller, configured to support 5 channels.

This DMA controller supports two DMA modes, referred to as DMA Modes 0 and 1 and it can handle packet sizes up to 8k. In addition, the controller can be programmed to conduct transfers using INCR4, INCR8 and INCR16 4/8/16-beat incrementing bursts rather than bursts of unspecified length.

When operating in DMA Mode 0, the DMA controller can be only programmed to load/unload one packet, so processor intervention is required for each packet transferred over the USB. This mode can be used with any endpoint, whether it uses Control, Bulk, Isochronous, or Interrupt transactions (i.e. including Endpoint 0).

When operating in DMA Mode 1, the DMA controller can be programmed to load/unload a complete bulk transfer (which can be many packets). Once set up, the DMA controller will load/unload all packets of the transfer, interrupting the processor only when the transfer has completed. DMA Mode 1 can only be used with endpoints that use Bulk transactions.

Each channel can be independently programmed for the selected operating mode.

14.1.1.10.1. DMA Bus Cycles

The DMA controller uses incrementing bursts. It starts a new burst when it is first granted bus mastership (whether at the start of a USB packet or when regaining the bus after being thrown off part way through a packet), and when the address starts a new 1 KB block.

The requirement to start a new burst at 1 K boundaries is handled automatically by the DMA controller. The user does not need to take these boundaries into account in programming the DMA controller.

These bursts may be either 4-beat, 8-beat, 16-beat or of unspecified length, according to how the DMA channel is programmed, the size of packet being transferred and the location relative to the next 1 K boundary. Bits [10:9] of the `DMA_CNTL` select Burst Mode 0...3 which in turn define which burst types may be used (see Included DMA Controller section). For example, selection of Burst Mode 2 allows use of 8-beat (INCR8), 4-beat (INCR4) bursts and bursts of unspecified length but not 16-beat (INCR16) bursts.

There is no restriction on the Burst Mode selected for transfers in either DMA Mode 0 or DMA Mode 1.

Each transfer of a packet is generally carried out using word transfers (32 bits) but there may be additional byte or half-word transfers at the end of the packet transfer. Since the start address (written to the `DMA_ADDR`) must be word aligned, the packet transfer will start with a word transfer, but half-word and/or byte transfers may be added at the end to handle any residue. Split transactions and retries are supported.

14.1.1.10.2. Bus Errors

If a bus error occurs while the DMA controller is accessing memory, the DMA controller will immediately terminate the DMA transfer and interrupt the processor with the `DMA_ERR` bit of the `DMA_CNTL` register set.

The generation of this interrupt is not affected by the setting of the `DMA_CNTL.DMAIE`. The interrupt will still be generated when `DMA_CNTL.DMAIE` = 0.

14.1.1.10.3. Transferring Packets

This section covers the following topics:

- Individual Packet: Rx Endpoint
- Individual Packet: Tx Endpoint
- Multiple Packets: Rx Endpoint
- Multiple Packets: Tx Endpoint

Use of the built-in DMA controller to access the USB 2.0 Controller FIFOs requires both the DMA controller and the USB 2.0 Controller endpoint to be appropriately programmed. Many variations are possible. The following sections detail the standard set-ups used for the basic actions of transferring individual packets and multiple packets.

14.1.1.10.3.1. Individual Packet: Rx Endpoint

The transfer of individual packets will normally be carried out using DMA Mode 0.

For this, the USB 2.0 Controller Rx endpoint should be programmed as follows:

- The relevant interrupt enable bit in the `INTRRXE` set to 1.
- The `RXCSRH.DMAREQENAB` bit set to 0.

There is no need to set the USB 2.0 Controller to support DMA for this operation.

When a packet has been received by the USB 2.0 Controller, it will generate the appropriate Endpoint interrupt. The processor should then program selected channel of the DMA controller as follows:

- DMA_ADDR: Memory address to store packet.
- DMA_COUNT: Size of packet (determined by reading the RXCOUNT register).
- DMA_CNTL:
  - DMA_ENAB = 1
  - DMA_DIR = 0
  - DMAMODE = 0
  - DMAIE = 1
  - Required Burst Mode (DMA_BRSTM)

The DMA controller will then request bus mastership and transfer the packet to memory. When it has completed the transfer, it will generate a DMA interrupt. The processor should then clear the RXCSRL.RXPKTRDY bit.

14.1.1.10.3.2. Individual Packet: Tx Endpoint

To carry out this operation using DMA Mode 0, an USB 2.0 Controller Tx endpoint should be programmed as follows:

- The relevant interrupt enable bit in the INTRTXE register set to 1.
- The TXCSRH.DMAREQENAB bit set to 0.

There is no need to set the USB 2.0 Controller to support DMA for this operation.

When the FIFO in the USB 2.0 Controller becomes available, the USB 2.0 Controller will interrupt the processor with the appropriate Tx Endpoint interrupt. The processor should then program the DMA controller as follows:

- DMA_ADDR: Memory address of packet to send.
- DMA_COUNT: Size of packet to be sent.
- DMA_CNTL:
  - DMA_ENAB = 1
  - DMA_DIR = 1
  - DMAMODE = 0
  - DMAIE = 1
  - Required Burst Mode (DMA_BRSTM)

The DMA controller will then request bus mastership and transfer the packet to the USB 2.0 Controller FIFO. When it has completed the transfer, it will generate a DMA interrupt. The processor should then set the TXCSRL.TXPKTRDY bit.

14.1.1.10.3.3. Multiple Packets: Rx Endpoint

The transfer of multiple packets will normally be carried out using DMA Mode 1.

Where multiple packets are to be received using DMA Mode 1, the DMA Controller should be programmed as follows:

- DMA_ADDR: Memory address of the buffer in which to store transfer.
- DMA_COUNT: Maximum size of data buffer.
- DMA_CNTL:

- DMA_ENAB = 1
- DMA_DIR = 0
- DMAMODE = 1
- DMAIE = 1
- Required Burst Mode (DMA_BRSTM)

And the USB 2.0 Controller Rx endpoint should be programmed as follows:

- The relevant interrupt enable bit in the INTRRXE should be set to 1.
- The RXCSRH.AUTOCLEAR, RXCSRH.DMAREQENAB and RXCSRH.DMAREQMODE bits should be set to 1.

  In Host mode: the RXCSRH.AUTOREQ bit should also be set to 1 and the EPn_RQPKTCOUNT should be programmed with the number of packets in the transfer.

As each packet is received by the USB 2.0 Controller, the DMA controller will request bus mastership and transfer the packet to memory. With AUTOCLEAR set, the USB 2.0 Controller will automatically clear the RXCSRL.RXPKTRDY bit.

In Peripheral mode or where EPn_RQPKTCOUNT is zero, this process will continue automatically until the USB 2.0 Controller receives a 'short packet' (one of less than the maximum packet size for the endpoint) signifying the end of the transfer. This 'short packet' will not be transferred by the DMA controller: instead the USB 2.0 Controller will interrupt the processor by generating the appropriate Endpoint interrupt. The processor can then read the RXCOUNT to see the size of the 'short packet' and either unload it manually or reprogram the DMA controller in Mode 0 to unload the packet.

In Host mode with AUTOREQ set and EPn_RQPKTCOUNT non-zero, the core will decrement the value in the EPn_RQPKTCOUNT following each request. When the value decrements from 1 to 0, the AUTOREQ bit is cleared to prevent any further transactions being attempted.

The DMA controller DMA_ADDR will have been incremented as the packets were unloaded so the processor can determine the size of the transfer by comparing the current value of DMA_ADDR against the start address of the memory buffer.

If the size of the transfer exceeds the data buffer size, the DMA controller will stop unloading the FIFO and interrupt the processor via the dma_nint line.

14.1.1.10.3.4. Multiple Packets: Tx Endpoint

To carry out this operation using DMA Mode 1, the DMA controller should be programmed as follows:

- DMA_ADDR: Memory address of data block to send.
- DMA_COUNT: Size of data block.
- DMA_CNTL:
  - DMA_ENAB = 1
  - DMA_DIR = 1
  - DMAMODE = 1
  - DMAIE = 1
  - Required Burst Mode (DMA_BRSTM)

And the USB 2.0 Controller Tx endpoint should be programmed as follows:

- The relevant interrupt enable bit in the INTRTXE should be set to 1 (simply so that errors can be detected).
- The TXCSRH.AUTOSET, TXCSRH.DMAREQENAB and the TXCSRH.DMAREQMODE bits should be set to 1.

When the FIFO in the USB 2.0 Controller becomes available, the DMA controller will request bus mastership and transfer a packet to the FIFO. With AUTOSET set, the USB 2.0 Controller will automatically set the TXCSRL.TXPKTRDY bit. This process will continue until the entire data block has been transferred to the USB 2.0 Controller. The DMA controller will then interrupt the processor by taking dma_nint low. If the last packet to be loaded was less than the maximum packet size for the endpoint, the TXPKTRDY bit will not have been set for this packet: the processor should therefore respond to the DMA interrupt by setting the TXPKTRDY bit to allow the last 'short packet' to be sent. If the last packet to be loaded was of the maximum packet size, then the action to take depends on whether the transfer is under the control of an application such as the mass storage software on a Windows system that keeps count of the individual packets sent. If the transfer isn't under such control, the processor should still respond to the DMA interrupt by setting the TXPKTRDY bit. This has the effect of sending a null packet for the receiving software to interpret as indicating the end of the transfer.

14.1.1.11. VBus Events

This section covers the following topics:

- Actions as 'A' Device
- Actions as 'B' Device

The USB On-The-Go specification defines a series of thresholds to which the devices involved in point-to-point communications are required to respond:

- VBus Valid (required to be greater than 4.4 V and 4.75 V)
- Session Valid for 'A' device (required to be between 0.8 V and 2.1 V)
- Session End (required to be between 0.2 V and 0.8 V)

The actual thresholds used in a particular device are set through a series of comparators in the PHY, depending on the level of VBus.

Which of these thresholds are critical and the way in which the software controlling the USB 2.0 Controller needs to respond depends on whether the device is the 'A' device or the 'B' device and the circumstances under which the event happens. The required actions are summarized below.

14.1.1.11.1. Actions as 'A' Device

- **VBus > VBus Valid with session initiated by USB 2.0 Controller** (i.e. VBUS (DEVCTL[4:3]) = 0x3, SESSION bit of the DEVCTL set).

  When VBus becomes greater than VBus Valid, the USB 2.0 Controller selects Host Mode and waits for a device to be connected. It then generates a Connect interrupt (INTRUSB[4]). The software should reset and enumerate the connected 'B' device.

- **VBus > Session Valid with session initiated by 'B' device** (i.e. VBUS (bits [4:3]) of the DEVCTL) = 0x2, SESSION bit of the DEVCTL clear).

  When VBus becomes greater than Session Valid, the USB 2.0 Controller will generate a Session Request interrupt (INTRUSB.SESS_REQ bit). The software should set the SESSION bit. The USB 2.0 Controller will then either stay in Host mode or change to Peripheral mode depending on the state of the pull-up resistor on the 'B' device. The selected mode will be indicated by the state of the HOST_MODE bit of the DEVCTL.

- **VBus below VBus Valid while the Session bit remains set** (i.e. VBUS (bits [4:3]) of the DEVCTL) ≠ 0x3, SESSION bit of the DEVCTL set).

  This indicates a problem with the VBus power level. For example, the battery power may have dropped too low to sustain VBus Valid. Alternatively, the 'B' device may be drawing more current than the 'A' device can provide. In either case, the USB 2.0 Controller will automatically terminate the session and generate a VBus Error interrupt (INTRUSB.VBUS_ERROR bit).

14.1.1.11.2. Actions as 'B' Device

- **VBus > Session Valid** (i.e. VBUS (DEVCTL[4:3]) = 0x2, SESSION bit of the DEVCTL clear).

  This indicates activity from the 'A' device. The USB 2.0 Controller will set the SESSION bit and take the dppulldown output low in order to disconnect the pull-down resistor on the D+ line.

- **VBus < Session Valid while the Session bit remains set** (i.e. VBUS (DEVCTL[4:3]) = 0x1, SESSION bit of the DEVCTL set).

  This indicates that the 'A' device has lost power (or become disconnected). The USB 2.0 Controller will clear the SESSION bit of the DEVCTL and generate a Disconnect interrupt (INTRUSB.DISCON bit). The software should end the session.

- **VBus < Session End** (i.e. VBUS (DEVCTL[4:3]) = 0x0).

  This is the condition under which a 'B' device can initiate a session request. If the SESSION bit of the DEVCTL is set, then after 2 ms of SE0 on the bus, the USB 2.0 Controller will start SRP by first pulsing the data line, then pulsing VBus.

14.1.1.12. Dynamic FIFO Sizing

The USB 2.0 Controller have a single overall FIFO with size of 8 KB, areas of which may then be allocated to the different endpoints when the USB 2.0 Controller is initialized.

The allocation of FIFO space to the different endpoints requires the specification for each Tx and Rx endpoint of (the last two bullets below — together define the amount of space that needs to be allocated to the FIFO):

- The start address of the FIFO within the RAM block
- The maximum size of packet to be supported
- Whether double-buffering is required

These details may be specified through the following four registers:

Table 14-4 Endpoint #n FIFO parameters registers

Address	ID
0x62	Tx Endpoint #n FIFO Size (TXFIFOSZ)
0x63	Rx Endpoint #n FIFO Size (RXFIFOSZ)
0x64	Tx Endpoint #n FIFO Address (TXFIFOAD)
0x65	
0x66	Rx Endpoint #n FIFO Address (RXFIFOAD)
0x67	

The option of setting FIFO sizes dynamically applied only to Endpoints 1... 5. The Endpoint 0 FIFO has a fixed size (64 bytes) and a fixed location (start address 0).

14.1.1.13. Control Transactions (via Endpoint 0)

This section covers the following topics:

- In Peripheral Mode
- In Host Mode

14.1.1.13.1. In Peripheral Mode

This section covers the following topics:

- Zero Data Requests
- Write Requests
- Read Requests
- Endpoint 0 States
- Endpoint 0 Service Routine
- Error Handling
- Additional Actions

Endpoint 0 is the main control endpoint of the core. As such, the routines required to service Endpoint 0 are more complicated than those required to service other endpoints.

The software is required to handle all the Standard Device Requests that may be sent or received via Endpoint 0. These are described in ***Universal Serial Bus Specification, Revision 2.0, Chapter 9.*** The protocol for these device requests involves different numbers and types of transaction per transfer. To accommodate this, the software needs to take a state machine approach to command decoding and handling.

The Standard Device Requests received by a USB peripheral can be divided into three categories:

- Zero Data Requests (in which all the information is included in the command).
- Write Requests (in which the command will be followed by additional data).
- Read Requests (in which the device is required to send data back to the host).

This section looks at the sequence of actions that the software must perform to process these different types of device request.

The Setup packet associated with any Standard Device Request should include an 8-byte command. Any Setup packet containing a command field of anything other than 8 bytes will be automatically rejected by the USB 2.0 Controller core.

14.1.1.13.1.1. Zero Data Requests

Zero data requests have all their information included in the 8-byte command and require no additional data to be transferred. Examples of 'Zero Data' Standard Device Requests are: `SET_FEATURE`, `CLEAR_FEATURE`, `SET_ADDRESS`, `SET_CONFIGURATION`, `SET_INTERFACE`.

The sequence of events will begin, as with all requests, when the software receives an Endpoint 0 interrupt. The `RXPKTRDY` bit of the `CSR0L` will also have been set. The 8-byte command should then be read from the Endpoint 0 FIFO, decoded and the appropriate action taken. For example if the command is `SET_ADDRESS`, the 7-bit address value contained in the command should be written to the `FADDR`.

The CSR0L should then be written to set the SERVICED_RXPKTRDY bit (indicating that the command has been read from the FIFO) and to set the DATA_END bit (indicating that no further data is expected for this request).

When the host moves to the status stage of the request, a second Endpoint 0 interrupt will be generated to indicate that the request has completed. No further action is required from the software: the second interrupt is just a confirmation that the request completed successfully.

If the command is an unrecognized command, or for some other reason cannot be executed, then when it has been decoded, the CSR0L should be written to set the SERVICED_RXPKTRDY bit and to set the SEND_STALL bit. When the host moves to the status stage of the request, the USB 2.0 Controller will send a STALL to tell the host that the request was not executed. A second Endpoint 0 interrupt will be generated and SENT_STALL bit of the CSR0L will be set.

If the host sends more data after the DATA_END bit has been set, then the USB 2.0 Controller will send a STALL. An Endpoint 0 interrupt will be generated and SENT_STALL bit of the CSR0L will be set.

14.1.1.13.1.2. Write Requests

Write requests involve an additional packet (or packets) of data being sent from the host after the 8-byte command. An example of a 'Write' Standard Device Request is: SET_DESCRIPTOR.

The sequence of events will begin, as with all requests, when the software receives an Endpoint 0 interrupt. The RXPKTRDY bit of the CSR0L will also have been set. The 8-byte command should then be read from the Endpoint 0 FIFO and decoded.

As with a zero data request, the CSR0L should then be written to set the SERVICED_RXPKTRDY bit (indicating that the command has been read from the FIFO) but in this case the DATA_END bit should not be set (indicating that more data is expected).

When a second Endpoint 0 interrupt is received, the CSR0L should be read to check the endpoint status. The RXPKTRDY bit of the CSR0L should be set to indicate that a data packet has been received. The COUNT0 should then be read to determine the size of this data packet. The data packet can then be read from the Endpoint 0 FIFO.

If the length of the data associated with the request (indicated by the wLength field in the command) is greater than the maximum packet size for Endpoint 0, further data packets will be sent. In this case, CSR0L should be written to set the SERVICED_RXPKTRDY bit, but the DATA_END bit should not be set.

When all the expected data packets have been received, the CSR0L should be written to set the SERVICED_RXPKTRDY bit and to set the DATA_END bit (indicating that no more data is expected).

When the host moves to the status stage of the request, another Endpoint 0 interrupt will be generated to indicate that the request has completed. No further action is required from the software, the interrupt is just a confirmation that the request completed successfully.

If the command is an unrecognized command, or for some other reason cannot be executed, then when it has been decoded, the CSR0L should be written to set the SERVICED_RXPKTRDY bit and to set the SEND_STALL bit. When the host sends more data, the USB 2.0 Controller will send a STALL to tell the host that the request was not executed. An Endpoint 0 interrupt will be generated and the SENT_STALL bit of the CSR0L will be set.

If the host sends more data after the DATA_END has been set, then the USB 2.0 Controller will send a STALL. An Endpoint 0 interrupt will be generated and the SENT_STALL bit of the CSR0L will be set.

14.1.1.13.1.3. Read Requests

Read requests have a packet (or packets) of data sent from the function to the host after the 8-byte command. Examples of 'Read' Standard Device Requests are: `GET_CONFIGURATION`, `GET_INTERFACE`, `GET_DESCRIPTOR`, `GET_STATUS`, `SYNCH_FRAME`.

The sequence of events will begin, as with all requests, when the software receives an Endpoint 0 interrupt. The `RXPKTRDY` bit of the `CSR0L` will also have been set. The 8-byte command should then be read from the Endpoint 0 FIFO and decoded. The `CSR0L` should then be written to set the `SERVICED_RXPKTRDY` bit (indicating that the command has read from the FIFO).

The data to be sent to the host should then be written to the Endpoint 0 FIFO. If the data to be sent is greater than the maximum packet size for Endpoint 0, only the maximum packet size should be written to the FIFO. The `CSR0L` should then be written to set the `TXPKTRDY` bit (indicating that there is a packet in the FIFO to be sent). When the packet has been sent to the host, another Endpoint 0 interrupt will be generated and the next data packet can be written to the FIFO.

When the last data packet has been written to the FIFO, the `CSR0L` should be written to set the `TXPKTRDY` bit and to set the `DATA_END` bit (indicating that there is no more data after this packet).

When the host moves to the Status stage of the request, another Endpoint 0 interrupt will be generated to indicate that the request has completed. No further action is required from the software: the interrupt is just a confirmation that the request completed successfully.

If the command is an unrecognized command, or for some other reason cannot be executed, then when it has been decoded, the `CSR0L` should be written to set the `SERVICED_RXPKTRDY` bit and to set the `SEND_STALL` bit of the `CSR0L`. When the host requests data, the USB 2.0 Controller will send a STALL to tell the host that the request was not executed. An Endpoint 0 interrupt will be generated and the `SENT_STALL` bit of the `CSR0L` will be set.

If the host requests more data after `DATA_END` has been set, then the USB 2.0 Controller will send a STALL. An Endpoint 0 interrupt will be generated and the `SENT_STALL` bit of the `CSR0L` will be set.

14.1.1.13.1.4. Endpoint 0 States

When the USB 2.0 Controller is operating as a peripheral, the Endpoint 0 control needs three modes – IDLE, Tx and Rx – corresponding to the different phases of the control transfer and the states Endpoint 0 enters for the different phases of the transfer.

The default mode on power-up or reset should be IDLE.

`RXPKTRDY` bit of the `CSR0L` becoming set when Endpoint 0 is in IDLE state indicates a new device request. Once the device request is unloaded from the FIFO, the USB 2.0 Controller decodes the descriptor to find whether there is a Data phase and, if so, the direction of the Data phase of the control transfer (in order to set the FIFO direction).

Depending on the direction of the Data phase, Endpoint 0 goes into either Tx state or Rx state. If there is no Data phase, Endpoint 0 remains in IDLE state to accept the next device request.

The actions that the software needs to take at the different phases of the possible transfers (e.g. loading the FIFO, setting `TXPKTRDY`) are indicated in the diagram below.

Note that the USB 2.0 Controller changes the FIFO direction depending on the direction of the Data phase independently of the software.

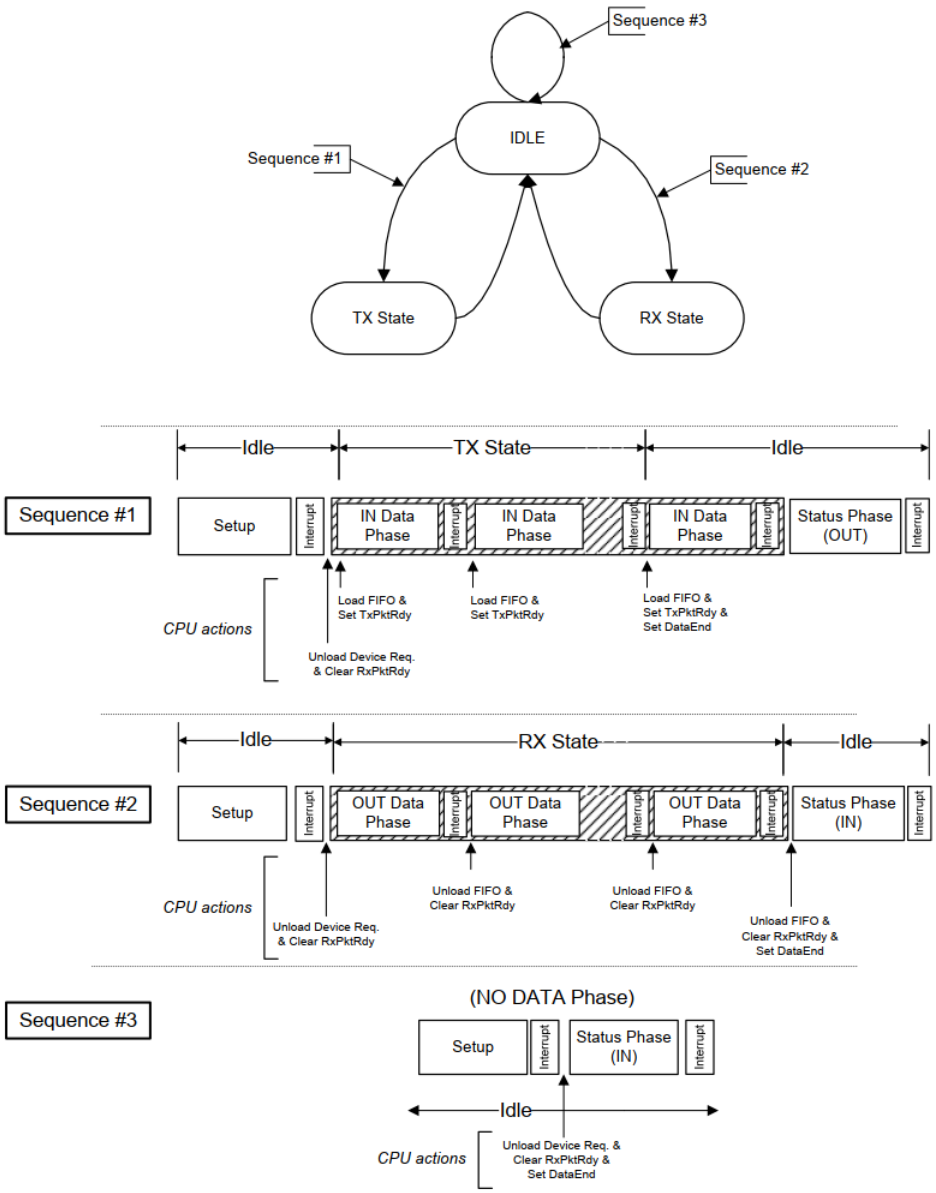

Figure 14-4 Endpoint 0 states for peripheral

14.1.1.13.1.5. Endpoint 0 Service Routine

An Endpoint 0 interrupt is generated:

- When the core sets the `RXPKTRDY` bit of the `CSR0L` after a valid token has been received and data has been written to the FIFO.
- When the core clears the `TXPKTRDY` bit of the `CSR0L` after the packet of data in the FIFO has been successfully transmitted to the host.
- When the core sets the `SENT_STALL` bit of the `CSR0L` after a control transaction is ended due to a protocol violation.
- When the core sets the `SETUP_END` bit of the `CSR0L` because a control transfer has ended before `DATA_END` bit of the `CSR0L` is set.

Whenever the Endpoint 0 service routine is entered, the firmware must first check to see if the current control transfer has been ended due to either a STALL condition or a premature end of control transfer. If the control transfer ends due to a STALL condition, the `SENT_STALL` bit would be set. If the control transfer ends due to a premature end of control transfer, the `SETUP_END` bit would be set. In either case, the firmware should abort processing the current control transfer and set the state to IDLE.

Once the firmware has determined that the interrupt was not generated by an illegal bus state, the next action taken depends on the Endpoint state.

If Endpoint 0 is in IDLE state, the only valid reason an interrupt can be generated is as a result of the core receiving data from the USB bus. The service routine must check for this by testing the `RXPKTRDY` bit of the `CSR0L`. If this bit is set, then the core has received a SETUP packet. This must be unloaded from the FIFO and decoded to determine the action the core must take.

Depending on the command contained within the SETUP packet, Endpoint 0 will enter one of three states:

- If the command is a single packet transaction (`SET_ADDRESS`, `SET_INTERFACE` etc.) without any data phase, the endpoint will remain in IDLE state.
- If the command has an OUT data phase (`SET_DESCRIPTOR` etc.), the endpoint will enter Rx state.
- If the command has an IN data phase (`GET_DESCRIPTOR` etc.), the endpoint will enter Tx state.

If the endpoint is in Tx state, the interrupt indicates that the core has received an IN token and data from the FIFO has been sent. The firmware must respond to this either by placing more data in the FIFO if the host is still expecting more data or by setting the `DATA_END` bit to indicate that the data phase is complete. Once the data phase of the transaction has been completed, Endpoint 0 should be returned to IDLE state to await the next control transaction.

If the endpoint is in Rx state, the interrupt indicates that a data packet has been received. The firmware must respond by unloading the received data from the FIFO. The firmware must then determine whether it has received all of the expected data. If it has, the firmware should set the `DATA_END` bit and return Endpoint 0 to IDLE state. If more data is expected, the firmware should set the `SERVICED_RXPKTRDY` bit of the `CSR0L` to indicate that it has read the data in the FIFO and leave the endpoint in Rx state.

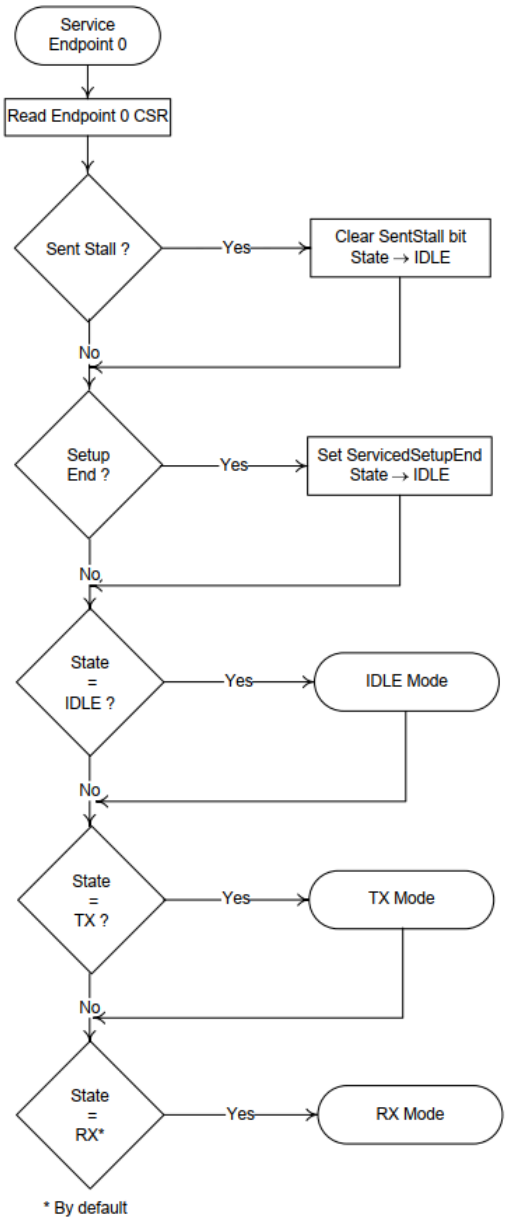

Figure 14-5 Flow for Endpoint 0 service routine

14.1.1.13.1.5.1. Idle

IDLE is the mode the Endpoint 0 control needs to select at power-on or reset and is the mode to which the Endpoint 0 control should return when the Rx and Tx modes are terminated.

It is also the mode in which the SETUP phase of control transfer is handled (as outlined in the figure below).

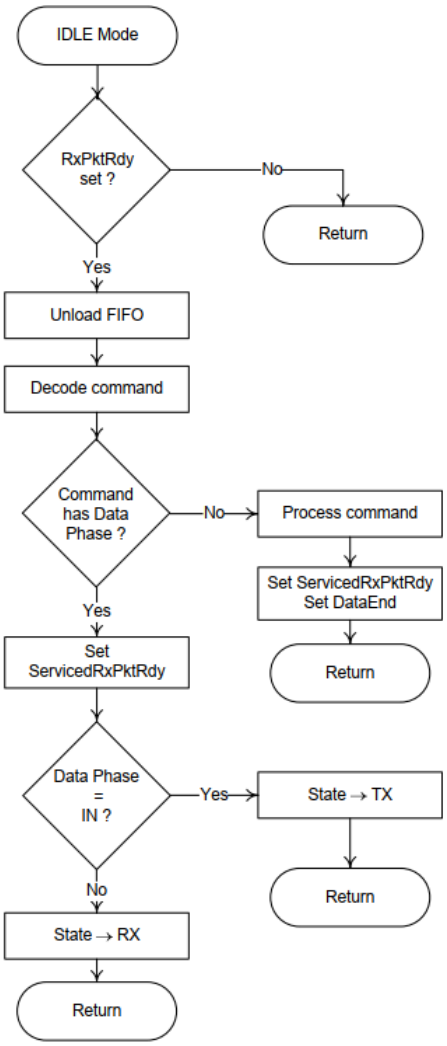

Figure 14-6 Flow for SETUP phase of control transfer

14.1.1.13.1.5.2. Tx

When the endpoint is in Tx state, all arriving IN tokens need to be treated as part of a Data phase until the required amount of data has been sent to the host. If either a SETUP or an OUT token is received whilst the endpoint is in the Tx state, this will cause a SetupEnd condition to occur as the core expects only IN tokens.

Three events can cause Tx mode to be terminated before the expected amount of data has been sent:

- The host sends an invalid token causing a SetupEnd condition (`SETUP_END` bit of the `CSR0L` set).
- The firmware sends a packet containing less than the maximum packet size for Endpoint 0 (`TXMAXP` register).
- The firmware sends an empty data packet.

Until the transaction is terminated, the firmware simply needs to load the FIFO when it receives an interrupt which indicates that a packet has been sent from the FIFO. (An interrupt is generated when `TXPKTRDY` is cleared.)

When the firmware forces the termination of a transfer (by sending a short or empty data packet), it should set the `DATA_END` bit of the `CSR0L` to indicate to the core that the Data phase is complete and that the core should next receive an acknowledge packet.

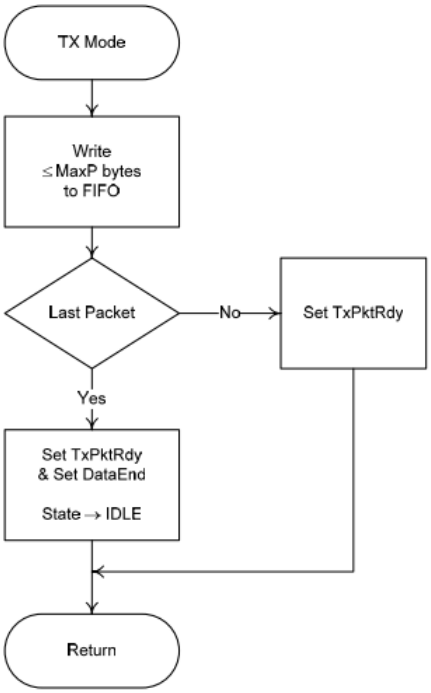

Figure 14-7 Flow for IN Data phase of control transfer

14.1.1.13.1.5.3. Rx

In Rx mode, all arriving data should be treated as part of a Data phase until the expected amount of data has been received. If either a SETUP or an IN token is received while the endpoint is in Rx state, this will cause a SetupEnd condition to occur as the core expects only OUT tokens.

Three events can cause Rx mode to be terminated before the expected amount of data has been received:

- The host sends an invalid token causing a SetupEnd condition (`SETUP_END` bit of the `CSR0L` set).
- The host sends a packet which contains less than the maximum packet size for Endpoint 0.
- The host sends an empty data packet.

Until the transaction is terminated, the firmware simply needs to unload the FIFO when it receives an interrupt which indicates that new data has arrived (`RXPKTRDY` bit of the `CSR0L`) set) and to clear `RXPKTRDY` by setting the `SERVICED_RXPKTRDY` bit of the `CSR0L`.

When the firmware detects the termination of a transfer (by receiving either the expected amount of data or an empty data packet), it should set the `DATA_END` bit of the `CSR0L` to indicate to the core that the Data phase is complete and that the core should receive an acknowledge packet next.

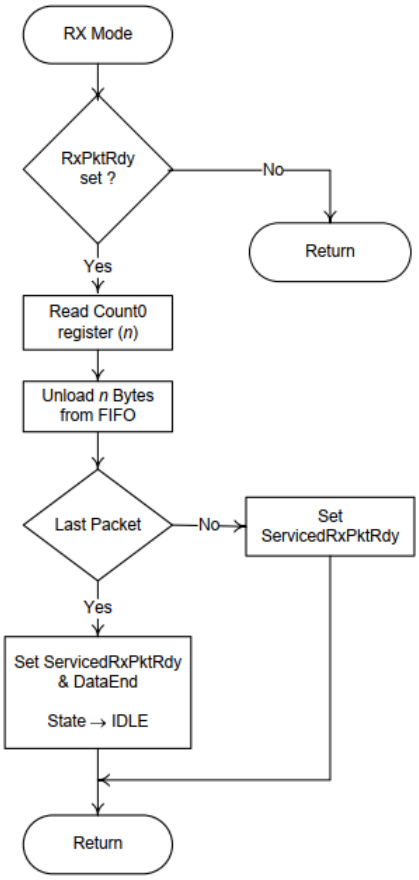

Figure 14-8 Flow for OUT Data phase of control transfer

14.1.1.13.1.6. Error Handling

A control transfer may be aborted due to a protocol error on the USB, the host prematurely ending the transfer, or if the function controller software wishes to abort the transfer (e.g. because it cannot process the command).

The USB 2.0 Controller will automatically detect protocol errors and send a STALL packet to the host under the following conditions:

1. The host sends more data during the OUT Data phase of a write request than was specified in the command. This condition is detected when the host sends an OUT token after the `DATA_END` bit of the `CSR0L` has been set.
2. The host request more data during the IN Data phase of a read request than was specified in the command. This condition is detected when the host sends an IN token after the `DATA_END` bit of the `CSR0L` has been set.
3. The host sends more than MaxP data bytes in an OUT data packet.
4. The host sends a non-zero length DATA1 packet during the STATUS phase of a read request.

When the USB 2.0 Controller has sent the STALL packet, it sets the `SENT_STALL` bit of the `CSR0L` and generates an interrupt. When the software receives an Endpoint 0 interrupt with the `SENT_STALL` bit set, it should abort the current transfer, clear the `SENT_STALL` bit, and return to the IDLE state.

If the host prematurely ends a transfer by entering the STATUS phase before all the data for the request has been transferred, or by sending a new SETUP packet before completing the current transfer, then the `SETUP_END` bit of the `CSR0L` will be set and an Endpoint 0 interrupt generated. When the software receives an Endpoint 0 interrupt with the `SETUP_END` bit set, it should abort the current transfer, set the `SERVICED_SETUP_END` bit of the `CSR0L`, and return to the IDLE state. If the `RXPKTRDY` bit of the

CSR0L is set this indicates that the host has sent another SETUP packet and the software should then process this command.

If the software wants to abort the current transfer, because it cannot process the command or has some other internal error, then it should set the SEND_STALL bit of the CSR0L. The USB 2.0 Controller will then send a STALL packet to the host, set the SENT_STALL bit of the CSR0L and generate an Endpoint 0 interrupt.

14.1.1.13.1.7. Additional Actions

When working as a peripheral, the USB 2.0 Controller core automatically responds to certain conditions on the USB bus or actions by the host. The details on it are given in below sections.

14.1.1.13.1.7.1. STALL Issued to Control Transfer

The USB 2.0 Controller core will automatically issue a STALL handshake to a Control transfer under the following conditions:

1. The host sends more data during an OUT Data phase of a Control transfer than was specified in the device request during the SETUP phase.

   This condition is detected by the USB 2.0 Controller when the host sends an OUT token (instead of an IN token) after the software has unloaded the last OUT packet and set DATA_END.

2. The host requests more data during an IN Data phase of a Control transfer than was specified in the device request during the SETUP phase.

   This condition is detected by the USB 2.0 Controller when the host sends an IN token (instead of an OUT token) after the software has cleared TXPKTRDY and set DATA_END in response to the ACK issued by the host to what should have been the last packet.
3. The host sends more than MaxP data with an OUT Data token.
4. The host sends more than a zero length data packet for the OUT Status phase.

14.1.1.13.1.7.2. Zero-Length OUT Data Packets in Control Transfers

A zero-length OUT data packet is used to indicate the end of a Control transfer. In normal operation, such packets should only be received after the entire length of the device request has been transferred (i.e. after the software has set DATA_END). If, however, the host sends a zero-length OUT data packet before the entire length of device request has been transferred, this signals the premature end of the transfer. In this case, the USB 2.0 Controller will automatically flush any IN token loaded by software ready for the Data phase from the FIFO and set SETUP_END bit of the CSR0L.

14.1.1.13.2. In Host Mode

This section covers the following topics:

- Setup Phase
- IN Data Phase
- OUT Data Phase
- IN Status Phase (Following Setup Phase or OUT Data Phase)
- OUT Status Phase (Following IN Data Phase)

Host Control Transactions are conducted through Endpoint 0 and the software is required to handle all the Standard Device Requests that may be sent or received via Endpoint 0 (as described in ***Universal Serial Bus Specification, Revision 2.0, Chapter 9***).

As for a USB peripheral, there are three categories of Standard Device Requests to be handled:

- Zero Data Requests (in which all the information is included in the command): comprise a SETUP command followed by an IN Status Phase.
- Write Requests (in which the command will be followed by additional data): comprise a SETUP command, followed by an OUT Data Phase which is in turn followed by an IN Status Phase.
- Read Requests (in which the device is required to send data back to the host): comprise a SETUP command, followed by an IN Data Phase which is in turn followed by an OUT Status Phase.

A timeout may be set to limit the length of time for which the USB 2.0 Controller will retry a transaction which is continually NAKed by the target. This limit can be between 2 and 2^{15} frames/microframes and is set through the `NAKLIMIT0` register.

The following sections describe the actions that the software needs to take in issuing these different types of request through looking at the steps to take in the different phases of a Control Transaction.

Before initiating any transactions as a Host, the `FADDR` needs to be set to address the peripheral device. When the device is first connected, `FADDR` should be set to zero. After a `SET_ADDRESS` command is issued, `FADDR` should be set the target's new address.

14.1.1.13.2.1. Setup Phase

For the SETUP Phase of a Control Transaction, the software driving the Host device needs to:

1. Load the 8 bytes of the required Device request command into the Endpoint 0 FIFO.
2. Then set `SETUPPKT` and `TXPKTRDY` bits of the `CSR0L`, respectively.

   

   These bits need to be set together.

   The USB 2.0 Controller then proceeds to send a SETUP token followed by the 8-byte command to Endpoint 0 of the addressed device, retrying as necessary (the details of this operation are shown in Dynamic FIFO Sizing section).
3. At the end of the attempt to send the data, the USB 2.0 Controller will generate an Endpoint 0 interrupt (i.e. set `INTRTX.EP0`). The software should then read `CSR0L` to establish whether the `RXSTALL` bit, the `ERROR` bit or the `NAK_TIMEOUT` bit has been set.

   If `RXSTALL` is set, it indicates that the target did not accept the command (e.g. because it is not supported by the target device) and so has issued a STALL response.

   If `ERROR` is set, it means that the USB 2.0 Controller has tried to send the SETUP Packet and the following data packet three times without getting any response.

   If `NAK_TIMEOUT` is set, it means that the USB 2.0 Controller has received a NAK response to each attempt to send the SETUP packet, for longer than the time set in the `NAKLIMIT0` register. The USB 2.0 Controller can then be directed either to continue trying this transaction (until it times out again) by clearing the `NAK_TIMEOUT` bit or to abort the transaction by flushing the FIFO before clearing the `NAK_TIMEOUT` bit.
4. If none of `RXSTALL`, `ERROR` or `NAK_TIMEOUT` is set, the SETUP Phase has been correctly ACKed and the software should proceed to the following IN Data Phase, OUT Data Phase or IN Status Phase specified for the particular Standard Device Request.

14.1.1.13.2.2. IN Data Phase

For the IN Data Phase of a Control Transaction, the software driving the Host device needs to:

1. Set REQPKT bit of the CSR0L.
2. Wait while the USB 2.0 Controller both sends the IN token and receives the required data back (the details of this operation are shown in Dynamic FIFO Sizing section).
3. When the USB 2.0 Controller generates the Endpoint 0 interrupt (i.e. sets EP0 bit of the INTRTX), read CSR0L to establish whether the RXSTALL bit, the ERROR bit or the NAK_TIMEOUT bit or RXPKTRDY has been set.

   If RXSTALL is set, it indicates that the target has issued a STALL response.

   If ERROR is set, it means that the USB 2.0 Controller has tried to send the required IN token three times without getting any response.

   If NAK_TIMEOUT is set, it means that the USB 2.0 Controller has received a NAK response to each attempt to send the IN token, for longer than the time set in the NAKLIMIT0. The USB 2.0 Controller can then be directed either to continue trying this transaction (until it times out again) by clearing the NAK_TIMEOUT bit or to abort the transaction by clearing REQPKT before clearing the NAK_TIMEOUT bit.
4. If RXPKTRDY has been set, the software should read the data from the Endpoint 0 FIFO, then clear RXPKTRDY.
5. If further data is expected, the software should repeat Steps 1 – 4.

When all the data has been successfully received, the software should proceed to the OUT Status Phase of the Control Transaction.

14.1.1.13.2.3. OUT Data Phase

For the OUT Data Phase of a Control Transaction, the software driving the Host device needs to:

1. Load the data to be sent into the Endpoint 0 FIFO.
2. Then set the TXPKTRDY bit of the CSR0L.

   The USB 2.0 Controller then proceeds to send an OUT token followed by the data from the FIFO to Endpoint 0 of the addressed device, retrying as necessary (the details of this operation are shown in Dynamic FIFO Sizing section).
3. At the end of the attempt to send the data, the USB 2.0 Controller will generate an Endpoint 0 interrupt (i.e. set EP0 bit of the INTRTX). The software should then read CSR0L to establish whether the RXSTALL bit, the ERROR bit or the NAK_TIMEOUT bit has been set.

   If RXSTALL is set, it indicates that the target has issued a STALL response.

   If ERROR is set, it means that the USB 2.0 Controller has tried to send the OUT token and the following data packet three times without getting any response.

   If NAK_TIMEOUT is set, it means that the USB 2.0 Controller has received a NAK response to each attempt to send the OUT token, for longer than the time set in the NAKLIMIT0. The USB 2.0 Controller can then be directed either to continue trying this transaction (until it times out again) by clearing the NAK_TIMEOUT bit or to abort the transaction by flushing the FIFO before clearing the NAK_TIMEOUT bit.

   If none of RXSTALL, ERROR or NAK_TIMEOUT is set, the OUT data has been correctly ACKed.
4. If further data needs to be sent, the software should repeat Steps 1 – 3.

When all the data has been successfully sent, the software should proceed to the IN Status Phase of the Control Transaction.

14.1.1.13.2.4. IN Status Phase (Following Setup Phase or OUT Data Phase)

For the IN Status Phase of a Control Transaction, the software driving the Host device needs to:

1. Set `REQPKT` and `STATUSPKT` bits of the `CSR0L`.

   

   These bits need to be set together i.e. in the same write operation.

2. Wait while the USB 2.0 Controller both sends an IN token and receives a response from the USB peripheral (the details of this operation are shown in Dynamic FIFO Sizing section).
3. When the USB 2.0 Controller generates the Endpoint 0 interrupt (i.e. sets `EP0` bit of the `INTRTX`), read `CSR0L` to establish whether the `RXSTALL` bit, the `ERROR` bit or the `NAK_TIMEOUT` bit or `RXPKTRDY` has been set.

   If `RXSTALL` is set, it indicates that the target could not complete the command and so has issued a STALL response.

   If `ERROR` is set, it means that the USB 2.0 Controller has tried to send the required IN token three times without getting any response.

   If `NAK_TIMEOUT` is set, it means that the USB 2.0 Controller has received a NAK response to each attempt to send the IN token, for longer than the time set in the `NAKLIMIT0`. The USB 2.0 Controller can then be directed either to continue trying this transaction (until it times out again) by clearing the `NAK_TIMEOUT` bit or to abort the transaction by clearing `REQPKT` and `STATUSPKT` before clearing the `NAK_TIMEOUT` bit.
4. The software should clear the `NAK_TIMEOUT` bit, together with (i.e. in the same write operation as) `RXPKTRDY` if this has been set.

14.1.1.13.2.5. OUT Status Phase (Following IN Data Phase)

For the OUT Status Phase of a Control Transaction, the software driving the Host device needs to:

1. Set `STATUSPKT` and `TXPKTRDY` bits of the `CSR0L`.

   

   These bits need to be set together.

2. Wait while the USB 2.0 Controller both sends the OUT token and a zero-length DATA1 packet (the details of this operation are shown in Dynamic FIFO Sizing section)
3. At the end of the attempt to send the data, the USB 2.0 Controller will generate an Endpoint 0 interrupt (i.e. set `EP0` bit of the `INTRTX`). The software should then read `CSR0L` to establish whether the `RXSTALL` bit, the `ERROR` bit or the `NAK_TIMEOUT` bit has been set.

   If `RXSTALL` is set, it indicates that the target could not complete the command and so has issued a STALL response.

   If `ERROR` is set, it means that the USB 2.0 Controller has tried to send the STATUS Packet and the following data packet three times without getting any response.

   If `NAK_TIMEOUT` is set, it means that the USB 2.0 Controller has received a NAK response to each attempt to send the IN token, for longer than the time set in the `NAKLIMIT0`. The USB 2.0 Controller can then be directed either to continue trying this transaction (until it times out again) by clearing the `NAK_TIMEOUT` bit or to abort the transaction by flushing the FIFO before clearing the `NAK_TIMEOUT` bit.
4. If none of `RXSTALL`, `ERROR` or `NAK_TIMEOUT` is set, the STATUS Phase has been correctly ACKed.

14.1.1.14. Bulk Transactions

This section covers the following topics:

- In Peripheral Mode
- In Host Mode
- Employing DMA

14.1.1.14.1. In Peripheral Mode

This section covers the following topics:

- Bulk IN Transactions
- Bulk OUT Transactions

14.1.1.14.1.1. Bulk IN Transactions

This section covers the following topics:

- Setup
- Operation

A Bulk IN transaction is used to transfer non-periodic data from the function controller to the host.

Four optional features are available for use with a Tx endpoint used in Peripheral mode for Bulk IN transactions:

- **Double packet buffering**

  Since the dynamic FIFO sizing is used by USB 2.0 Controller, the use of single or double packet buffering is part of the specification for the endpoint FIFO.

  When enabled, up to two packets can be stored in the FIFO awaiting transmission to the host.

- **DMA**

  If DMA is enabled for the endpoint, a DMA request will be generated whenever the endpoint is able to accept another packet in its FIFO. This feature can be used to allow a DMA controller to load packets into the FIFO without processor intervention. If DMA mode 1 is used the `TXMAXP[D10:0]` must be set to an even number for proper interrupt generation.

- **AutoSet**

  When the AutoSet feature is enabled, the `TXPKTRDY` bit of the `TXCSRL` will be automatically set when a packet of `TXMAXP` bytes has been loaded into the FIFO. This is particularly useful when DMA is used to load the FIFO as it avoids the need for any processor intervention when loading individual packets during a large Bulk transfer.

- **Automatic Packet Splitting**

  For some system designs, it may be convenient for the application software to write larger amounts of data to an endpoint in a single operation than can be transferred in a single USB operation. A particular case in point is where the same endpoint is used for high-speed transfers of 512 bytes under certain circumstances but for full-speed transfers under other circumstances. When operating at full-speed, the maximum amount of data transferred in a single operation is then just 64 bytes.

  To cater for such circumstances, the USB 2.0 Controller includes a feature, that allows larger data packets to be written to Bulk endpoints which are then split into packets of an appropriate

(specified) size for transfer across the USB bus. Whether this option is selected can be determined from the setting of the MPTXE bit of the CONFIGDATA. The necessary packet size information is set via the TXMAXP.

14.1.1.14.1.1.1. Setup

In configuring a Tx endpoint for Bulk transactions, the TXMAXP register must be written with the maximum packet size (in bytes) for the endpoint. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the endpoint. In addition, the relevant interrupt enable bit in the INTRTXE should be set to '1' (if an interrupt is required for this endpoint) and the high byte of the TXCSRH should be set as shown below:

Table 14-5 Values of TXCSRH register

Bits	Name	Value	Description
[7]	AUTOSET	0/1	Set to 1 if the AutoSet feature is required.
[6]	ISO	0	Set to 0 to enable Bulk protocol.
[5]	MODE	1	Set to 1 to ensure the FIFO is enabled (only necessary if the FIFO is shared with an Rx endpoint).
[4]	DMAREQENAB	0/1	Set to 1 if DMA Requests are required. If set to 1, will also need to select the chosen DMAREQMODE (TXCSRH.DMAREQMODE).
[3]	FRCDATATOG	0	Set to 0 to allow normal data toggle operation.

14.1.1.14.1.1.2. Operation

When data is to be transferred over a Bulk IN pipe, a data packet needs to be loaded into the FIFO and the TXCSRL written to set the TXPKTRDY bit. When the packet has been sent, the TXPKTRDY bit will be cleared by the USB 2.0 Controller and an interrupt generated so that the next packet can be loaded into the FIFO. If double packet buffering is enabled, then after the first packet has been loaded and the TXPKTRDY bit set, the TXPKTRDY bit will immediately be cleared by the USB 2.0 Controller and an interrupt generated so that a second packet can be loaded into the FIFO. The software should operate in the same way, loading a packet when it receives an interrupt, regardless of whether double packet buffering is enabled or not.

In the general case, the packet size must not exceed the size specified by the bottom 11 bits of the TXMAXP register. This part of the register defines the payload (packet size) for transfers over the USB and is required by the USB Specification to be either 8, 16, 32, 64 (Full-Speed or High-Speed) or 512 bytes (High-Speed only). If more than this amount of data is to be transferred, this needs to be sent as multiple USB packets which should all be TXMAXP[10:0] in size, except for the last packet which holds the residue.

In our case, we have an exception to this rule, since the automatic Bulk packet splitting feature is used (this determined from the setting of the MPTXE bit of the CONFIGDATA).

Since the this feature has been set, packets up to 32 times the size specified by TXMAXP[10:0] can be written to the FIFO (assuming that the FIFO is big enough to accept these larger packets) which are then split by the core into packets of the appropriate size for transfer over the USB. The size of the packets written to the FIFO is given by m × USB-payload where TXMAXP[15:11] = m – 1. All

the application software needs to do to take advantage of this feature is set the appropriate values in the TXMAXP register (and ensure that the value written to bits [10:0] matches the value given in the wMaxPacketSize field of the Standard Endpoint Descriptor for the associated endpoint). As far as the application software is concerned, the process of transferring these larger packets is no different from that used to transfer a standard-sized Bulk packet.

The host may determine that all the data for a transfer has been sent by knowing the total amount of data that is expected. Alternatively it may infer that all the data have been sent when it receives a packet which is smaller than the stated payload (TXMAXP[10:0]). In the latter case, if the total size of the data block is a multiple of this payload, it will be necessary for the function to send a null packet after all the data has been sent. This is done by setting TXPKTRDY when the next interrupt is received, without loading any data into the FIFO.

If large blocks of data are being transferred, then the overhead of calling an interrupt service routine to load each packet can be avoided by using a DMA Error Handling.

If the software wants to shut down the Bulk IN pipe, it should set the SEND_STALL bit of the TXCSRL. When the USB 2.0 Controller receives the next IN token, it will send a STALL to the host, set the SENT_STALL bit of the TXCSRL and generate an interrupt.

When the software receives an interrupt with the SENT_STALL bit of the TXCSRL set, it should clear the SENT_STALL bit. It should however leave the SEND_STALL bit of the TXCSRL set until it is ready to re-enable the Bulk IN pipe.

If the host failed to receive the STALL packet for some reason, it will send another IN token, so it is advisable to leave the SEND_STALL bit set until the software is ready to re-enable the Bulk IN pipe. When a pipe is re-enabled, the data toggle sequence should be restarted by setting the CLR_DATA_TOG bit in the TXCSRL.

14.1.1.14.1.2. Bulk OUT Transactions

This section covers the following topics:

- Setup
- Operation
- Error Handling

A Bulk OUT transaction is used to transfer non-periodic data from the host to the function controller.

Four optional features are available for use with an Rx endpoint used in Peripheral mode for Bulk OUT transactions:

- **Double packet buffering**

  Since the dynamic FIFO sizing is used by USB 2.0 Controller, the use of single or double packet buffering is part of the specification for the endpoint FIFO.

  When enabled, up to two packets can be stored in the FIFO.

- **DMA**

  If DMA is enabled for the endpoint, a DMA request will be generated whenever the endpoint has a packet in its FIFO. This feature can be used to allow a DMA without processor intervention. If DMA mode 1 is used the RXMAXP[10:0] must be set to an even number for proper interrupt generation.

- **AutoClear**

When the AutoClear feature is enabled, the RXPKTRDY bit of the RXCSRL will be automatically cleared when a packet of RXMAXP bytes has been unloaded from the FIFO (with exceptions, see register description). This is particularly useful when DMA is used to unload the FIFO as it avoids the need for any processor intervention when unloading individual packets during a large Bulk transfer.

- **Automatic Packet Combining**

  For some system designs, it may be convenient for the application software to read larger amounts of data from an endpoint in a single operation than can be transferred in a single USB operation. A particular case in point is where the same endpoint is used for high-speed transfers of 512 bytes under certain circumstances but for full-speed transfers under other circumstances. When operating at full-speed, the maximum amount of data transferred in a single operation is then just 64 bytes. To cater for such circumstances, the USB 2.0 Controller includes a feature, that causes the USB 2.0 Controller to combine the packets received across the USB bus into larger data packets prior to being read by the application software. Whether this feature is selected can be determined from the setting of the MPRXE bit of the CONFIGDATA. The necessary packet size information is set via the RXMAXP register, while the size of the amalgamated packet currently in line to be read is given in the RXCOUNT.

14.1.1.14.1.2.1. Setup

In configuring an Rx endpoint for Bulk OUT transactions, the RXMAXP register must be written with the maximum packet size (in bytes) for the endpoint. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the endpoint. In addition, the relevant interrupt enable bit in the INTRRXE should be set to '1' (if an interrupt is required for this endpoint) and the RXCSRH should be set as shown below:

Table 14-6 Values of RXCSRH register

Bits	Name	Value	Description
[7]	AUTOCLEAR	0/1	Set to 1 if the AutoSet feature is required.
[6]	ISO	0	Set to 0 to enable Bulk protocol.
[5]	DMAREQENAB	0/1	Set to 1 if a DMA request is required for this endpoint. If set to 1, will also need to select the chosen DMAREQMODE (RXCSRH.DMAREQMODE).
[4]	DISNYET	0	Set to 0 to allow normal PING flow control.

When the endpoint is first configured (following a SET_CONFIGURATION or SET_INTERFACE command on Endpoint 0), the RXCSRL should be written to set the CLRDATATOG bit. This will ensure that the data toggle (which is handled automatically by the USB 2.0 Controller) starts in the correct state. Also if there are any data packets in the FIFO (indicated by RXPKTRDY bit of the RXCSRL being set), they should be flushed by setting FLUSHFIFO bit of the RXCSRL.

It may be necessary to set this bit twice in succession if double buffering is enabled.

14.1.1.14.1.2.2. Operation

When a data packet is received by a Bulk Rx endpoint, the RXPKTRDY bit of the RXCSRL is set and an interrupt is generated. The software should read the RXCOUNT register for the endpoint to determine the size of the data packet. The data packet should be read from the FIFO, then RXPKTRDY should be cleared. If the FIFOFULL bit was set to 1 when RXPKTRDY is cleared, the USB 2.0 Controller will first clear the FIFOFULL bit. It will then set RXPKTRDY again to indicate that there is another packet waiting in the FIFO to be unloaded.

The packets received should not exceed the size specified in the RXMAXP register (as this should be the value set in the wMaxPacketSize field of the endpoint descriptor sent to the host). When a block of data larger than wMaxPacketSize needs to be sent to the function, it will be sent as multiple packets. All the packets will be wMaxPacketSize in size, except the last packet which will contain the residue. The software may use an application specific method of determining the total size of the block and hence when the last packet has been received. Alternatively it may infer that the entire block has been received when it receives a packet which is less than wMaxPacketSize in size (if the total size of the data block is a multiple of wMaxPacketSize, a null data packet will be sent after the data to signify that the transfer is complete).

In the general case, the application software will need to read each packet from the FIFO individually. The exception to this rule applies where the option for automatic combining of Bulk packets has been selected when the core was configured (this can be determined from the setting of the MPRXE bit of the CONFIGDATA. Where this option has been selected, the core can receive up to 32 packets at a time and combine them into a single packet within the FIFO (assuming that the FIFO is big enough to accept these larger packets). The size of the packets written to the FIFO is given by m × wMaxPacketSize where RXMAXP[15:11] = m – 1. All the application software needs to do to take advantage of this feature is set the appropriate values in the RXMAXP register (and ensure that the value written to bits [10:0] matches the value given in the wMaxPacketSize field of the endpoint descriptor). As far as the application software is concerned, the process of transferring these larger packets is no different from that used to transfer a standard-sized Bulk packet.

If large blocks of data are being transferred, the overhead of calling an interrupt service routine to unload each packet can be avoided by using DMA.

14.1.1.14.1.2.3. Error Handling

If the software wants to shut down the Bulk OUT pipe, it should set the SENDSTALL bit of the RXCSRL. When the USB 2.0 Controller receives the next packet it will send a STALL to the host, set the SENTSTALL bit of the RXCSRL and generate an interrupt.

When the software receives an interrupt with the SENTSTALL bit of the RXCSRL set, it should clear this bit. It should however leave the SENDSTALL bit of the RXCSRL set until it is ready to re-enable the Bulk OUT pipe.

If the host failed to receive the STALL packet for some reason, it will send another packet, so it is advisable to leave the SENDSTALL bit set until the software is ready to reenable the Bulk OUT pipe. When a Bulk OUT pipe is re-enabled, the data toggle sequence should be restarted by setting the CLRDATATOG bit in the RXCSRL.

14.1.1.14.2. In Host Mode

This section covers the following topics:

- Bulk IN Transactions
- Bulk OUT Transactions

14.1.1.14.2.1. Bulk IN Transactions

This section covers the following topics:

- Setup
- Operation
- Error Handling

A Bulk IN transaction may be used to transfer non-periodic data from the function controller to the host.

Five optional features are available for use with an Rx endpoint used in Host mode to receive this data:

- **Double packet buffering**

  Since the dynamic FIFO sizing is used by USB 2.0 Controller, the use of single or double packet buffering is part of the specification for the endpoint FIFO.

- **DMA**

  If DMA is enabled for the endpoint, a DMA request will be generated whenever the endpoint has a packet in its FIFO. This feature can be used to allow DMA to unload packets from the FIFO without processor intervention.

- **AutoReq(uest)**

  When the AutoReq(uest) feature is enabled, the `REQPKT` bit of the `RXCSRL` will be automatically set when the `RXPKTRDY` bit is cleared. This feature may be used in conjunction with the `EPn_RQPKTCOUNT` register to request the required number of maximum-size packets.

- **AutoClear**

  When the AutoClear feature is enabled, the `RXPKTRDY` bit of the `RXCSRL` will be automatically cleared when a packet of `RXMAXP` bytes has been unloaded from the FIFO (with exceptions, see register description). This is particularly useful together with AutoRequest when DMA is used to unload the FIFO as it avoids the need for any processor intervention when unloading individual packets during a large Bulk transfer.

- **Automatic Packet Combining**

  For some system designs, it may be convenient for the application software to read larger amounts of data from an endpoint in a single operation than can be transferred in a single USB operation. A particular case in point is where the same endpoint is used for high-speed transfers of 512 bytes under certain circumstances but for full-speed transfers under other circumstances. When operating at full-speed, the maximum amount of data transferred in a single operation is then just 64 bytes. To cater for such circumstances, the USB 2.0 Controller includes a feature, that causes the USB 2.0 Controller to combine the packets received across the USB bus into larger data packets prior to being read by the application software. Whether this option is selected can be determined from the setting of the `MPRXE` bit of the `CONFIGDATA`. The necessary packet size information is set via the `RXMAXP` register, while the size of the amalgamated packet currently in line to be read is given in the `EPn_RQPKTCOUNT`.

14.1.1.14.2.1.1. Setup

Before initiating any Bulk IN Transactions in Host mode:

- The RXTYPE for the USB 2.0 Controller endpoint that is to be used needs to be written with bits [7:6] set to select the operating speed, bits [5:4] = 10 (to select a Bulk transfer) and bits [3:0] set to the value of the endpoint number contained in the IN endpoint descriptor returned to the USB 2.0 Controller during device enumeration.
- The RXMAXP register for the USB 2.0 Controller endpoint must be written with the maximum packet size (in bytes) for the transmission. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the target endpoint.
- The RXINTERVAL needs to be written with the required value for the NAK Limit ($2 – 2^{15}$ frames/microframes), or set to zero if the NAK Timeout feature is not required.
- The relevant interrupt enable bit in the INTRRXE should be set to '1' (if an interrupt is required for this endpoint).
- The following bits of RXCSRH should be set as shown below:

Table 14-7 Values of RXCSRH register

Bits	Name	Value	Description
[7]	AUTOCLEAR	0/1	Set to 1 if the AutoClear feature is required.
[6]	AUTOREQ	0/1	Set to 1 if the AutoRequest feature is required.
[5]	DMAREQENAB	0/1	Set to 1 if a DMA request is required for this endpoint. If set to 1, will also need to select the chosen DMAREQMODE (RXCSRH.DMAREQMODE).
[4]	PID_ERROR	0	Set to 0 to allow normal PING flow control.

When the endpoint is first configured, the endpoint data toggle should be set to 0 either by using the DATA_TOGGLE_WRITE_ENABLE and DATA_TOGGLE bits of the RXCSRH to toggle the current setting or by writing the RXCSRL to set the CLRDATATOG bit. This will ensure that the data toggle (which is handled automatically by the USB 2.0 Controller) starts in the correct state. Also if there are any data packets in the FIFO (indicated by the RXPKTRDY bit of the RXCSRL being set), they should be flushed by setting the FLUSHFIFO bit of the RXCSRL.

It may be necessary to set this bit twice in succession if double buffering is enabled.

14.1.1.14.2.1.2. Operation

When Bulk data is required from the USB peripheral, the software should set the REQPKT bit in the corresponding RXCSRL. The USB 2.0 Controller will then send an IN token to the selected Peripheral endpoint and waits for data to be returned.

If data is correctly received, RXPKTRDY bit of the RXCSRL is set. If the USB peripheral responds with a STALL, RXSTALL bit of the RXCSRL is set. If a NAK is received, the USB 2.0 Controller tries again – and continues to try until either the transaction is successful or the NAK_LIMIT set in the RXINTERVAL is reached. If no response at all is received, two further attempts are made before the USB 2.0 Controller reports an error (RXCSRL.ERROR set).

The USB 2.0 Controller then generates the appropriate endpoint interrupt, whereupon the software should read the corresponding RXCSRL to determine whether the RXPKTRDY, RXSTALL, ERROR or DATAERROR_OR_NAK_TIMEOUT bit is set and act accordingly (if the DATAERROR_OR_NAK_TIMEOUT bit is

set, the USB 2.0 Controller can be directed either to continue trying this transaction (until it times out again) by clearing the DATAERROR_OR_NAK_TIMEOUT bit or to abort the transaction by clearing REQPKT before clearing the DATAERROR_OR_NAK_TIMEOUT bit).

The packets received should not exceed the size specified by RXMAXP[10:0] (as this should be the value set in the wMaxPacketSize field of the endpoint descriptor sent to the host).

When a block of data larger than wMaxPacketSize needs to be sent, it will be sent as multiple packets. All the packets will be wMaxPacketSize in size, except the last packet which will contain the residue. If the size of the block to be transferred is known, the number of packets of wMaxPacketSize to be transferred may be written to the EPn_RQPKTCOUNT register and the RXCSRH.AUTOREQ option set. As each packet is requested, the value in the EPn_RQPKTCOUNT register will be decremented. At the point that EPn_RQPKTCOUNT is decremented from 1 to 0, RXCSRH.AUTOREQ is cleared to stop any further requests being made. Alternatively it may be inferred that the entire block has been received when a packet which is less than wMaxPacketSize in size is received (if the total size of the data block is a multiple of wMaxPacketSize, the last packet may still be less than wMaxPacketSize in size as a null data packet is often sent after the data to signify that the transfer is complete) in this case, EPn_RQPKTCOUNT should be left set to zero. Then if AUTOREQ is set, it will be automatically cleared when the short packet is received.

The above describes the general case in which the application software read each packet from the FIFO individually. The behavior differs slightly where the option for automatic combining of Bulk packets was selected when the core was configured (this can be determined from the setting of the MPRXE bit of the CONFIGDATA). Where this option is selected, the core can receive up to 32 packets at a time and combine them into a single packet within the FIFO (assuming that the FIFO is big enough to accept these larger packets). The size of the packets written to the FIFO is given by m × wMaxPacketSize where RXMAXP[15:11] = m – 1. All the application software needs to do to take advantage of this feature is to set the appropriate values in the RXMAXP register (and ensure that the value written to bits [10:0] matches the value given in the wMaxPacketSize field of the endpoint descriptor). Values such as the number to set in the EPn_RQPKTCOUNT register are then calculated on the basis of packets of m × wMaxPacketSize, making the process of transferring these larger packets is no different from that used to transfer a standard-sized Bulk packet as far as the application software is concerned.

If large blocks of data are being transferred, the overhead of calling an interrupt service routine to unload each packet can be avoided by using DMA.

14.1.1.14.2.1.3. Error Handling

If the target wants to shut down the Bulk IN pipe, it will send a STALL response to the IN token. This will result in the RXSTALL bit of the RXCSRL being set.

14.1.1.14.2.2. Bulk OUT Transactions

This section covers the following topics:

- Setup
- Operation
- Error Handling

A Bulk OUT transaction may be used to transfer non-periodic data from the host to the function controller.

Four optional features are available for use with a Tx endpoint used in Host mode to transmit this data:

- **Double packet buffering**

  Since the dynamic FIFO sizing is used by USB 2.0 Controller, the use of single or double packet buffering is part of the specification for the endpoint FIFO. When enabled, up to two packets can be stored in the FIFO awaiting transmission to the peripheral.

- **DMA**

  If DMA is enabled for the endpoint, a DMA request will be generated whenever the endpoint is able to accept another packet in its FIFO. This feature can be used to allow DMA to load packets into the FIFO without processor intervention.

- **AutoSet**

  When the `TXCSRH.AUTOSET` feature is enabled, the `TXPKTRDY` bit of the `TXCSRL` will be automatically set when a packet of `TXMAXP` bytes has been loaded into the FIFO. This is particularly useful when DMA is used to load the FIFO as it avoids the need for any processor intervention when loading individual packets during a large Bulk transfer.

- **Automatic Packet Splitting**

  For some system designs, it may be convenient for the application software to write larger amounts of data to an endpoint in a single operation than can be transferred in a single USB operation. A particular case in point is where the same endpoint is used for high-speed transfers of 512 bytes under certain circumstances but for full-speed transfers under other circumstances. When operating at full-speed, the maximum amount of data transferred in a single operation is then just 64 bytes. To cater for such circumstances the USB 2.0 Controller includes a feature, that allows larger data packets to be written to Bulk endpoints which are then split into packets of an appropriate (specified) size for transfer across the USB bus. Whether this option is selected can be determined from the setting of the `MPTXE` bit of the `CONFIGDATA`.The necessary packet size information is set via the `TXMAXP` register.

14.1.1.14.2.2.1. Setup

Before initiating any Bulk OUT transactions:

- The `TXTYPE` for the USB 2.0 Controller endpoint that is to be used needs to be written with bits [7:6] set to select the operating speed, bits [5:4] = 0x2 (to select a Bulk transfer) and bits [3:0] set to the value of the endpoint number contained in the OUT endpoint descriptor returned to the USB 2.0 Controller during device enumeration.
- The `TXMAXP` register for the USB 2.0 Controller endpoint must be written with the maximum packet size (in bytes) for the transmission. This value should be the same as the `wMaxPacketSize` field of the Standard Endpoint Descriptor for the target endpoint.
- The `TXINTERVAL` needs to be written with the required value for the NAK Limit ($2 – 2^{15}$ frames/microframes), or set to zero if the NAK Timeout feature is not required.
- The relevant interrupt enable bit in the `INTRTXE` should be set to ‘1’ (if an interrupt is required for this endpoint).
- The following bits of the `TXCSRH` register should be set as shown below:

Table 14-8 Values of `TXCSRH` register

Bits	Name	Value	Description
[7]	`AUTOSET`	0/1	Set to 1 if the AutoSet feature is required.

Table 14-8 Values of TXCSRH register (continued)

Bits	Name	Value	Description
[5]	MODE	1	Set to 1 to ensure the FIFO is enabled (only necessary if the FIFO is shared with an Rx endpoint).
[4]	DMAREQENAB	0/1	Set to 1 if a DMA request is required for this endpoint. If set to 1, will also need to select the chosen DMAREQMODE (TXCSRH.DMAREQMODE).
[3]	FRCDATATOG	0	Set to 0 to allow normal data toggle operation.

When the endpoint is first configured, the endpoint data toggle should be set to 0 either by using the DATA_TOGGLE_WRITE_ENABLE and DATA_TOGGLE bits of the TXCSRL to toggle the current setting or by writing the TXCSRL to set the CLR_DATA_TOG bit. This will ensure that the data toggle (which is handled automatically by the USB 2.0 Controller) starts in the correct state. Also, if there are any data packets in the FIFO (indicated by the FIFO_NOT_EMPTY bit of the TXCSRL being set), they should be flushed by setting the FLUSH_FIFO bit of the TXCSRL.

It may be necessary to set this bit twice in succession if double buffering is enabled.

14.1.1.14.2.2.2. Operation

When Bulk data is required to be sent to the USB peripheral, the software should write the first packet of the data to the FIFO (or two packets if double-buffered) and set the TXPKTRDY bit in the corresponding TXCSRL. The USB 2.0 Controller will then send an OUT token to the selected Peripheral endpoint, followed by the first data packet from the FIFO.

If data is correctly received by the peripheral, an ACK should be received whereupon the USB 2.0 Controller will clear TXPKTRDY bit of the TXCSRL. If the USB peripheral responds with a STALL, RX_STALL bit of the TXCSRL is set. If a NAK is received, the USB 2.0 Controller tries again – and continues to try until either the transaction is successful or the NAK_LIMIT set in the TXINTERVAL is reached. If no response at all is received, two further attempts are made before the USB 2.0 Controller reports an error (TXCSRL.ERROR set).

The USB 2.0 Controller then generates the appropriate endpoint interrupt, whereupon the software should read the corresponding TXCSRL register to determine whether the RX_STALL, ERROR or NAK_TIMEOUT bit is set and act accordingly (if the NAK_TIMEOUT bit is set, the USB 2.0 Controller can be directed either to continue trying this transaction (until it times out again) by clearing the NAK_TIMEOUT bit or to abort the transaction by flushing the FIFO before clearing the NAK_TIMEOUT bit).

In the general case, packet sizes should not exceed the size specified by the bottom 11 bits of the TXMAXP register (which should have been set to match the value set in the wMaxPacketSize field of the appropriate endpoint descriptor). When a block of data larger than TXMAXP needs to be sent, it will need to be sent as multiple packets – each sent as described above. These packets should all be TXMAXP[10:0] in size, except the last packet which holds the residue.

The exception to this rule applies where the automatic Bulk packet splitting option has been selected when the core was configured (this can be determined from the setting of the MPTXE bit of the CONFIGDATA). Where this option has been selected, packets up to 32 times the size specified by TXMAXP[10:0] can be written to the FIFO (assuming that the FIFO is big enough to accept these larger packets) which are then split by the core into packets of the appropriate size for transfer over the USB.

The size of the packets written to the FIFO is given by `m × USB-payload` where `TXMAXP[15:11] = m – 1`. All the application software needs to do to take advantage of this feature is set the appropriate values in the `TXMAXP` register. As far as the application software is concerned, the process of transferring these larger packets is no different from that used to transfer a standard-sized Bulk packet.

If the total size of the data block is a multiple of `TXMAXP`, the host may need to send a null packet after all the data has been sent. This can be done by setting `TXPKTRDY` after the last interrupt is received, without loading any data into the FIFO.

If large blocks of data are being transferred, then the overhead of calling an interrupt service routine to load each packet can beavoided by using DMA.

14.1.1.14.2.2.3. Error Handling

If the target wants to shut down the Bulk OUT pipe, it will send a STALL response. This is indicated by the `RX_STALL` bit of the `TXCSRL` being set.

14.1.1.14.3. Employing DMA

This section covers the following topics:

- Using DMA with Bulk Tx Endpoints
- Using DMA with Bulk Rx Endpoints

The advantage of employing DMA is that it improves bus and processor utilization when loading or unloading the FIFOs.

DMA may be used in connection with any type of transfer but it is particularly useful when large blocks of data are to be transferred through a Bulk endpoint. The USB protocol requires that large data blocks are transferred by sending a series of packets of the maximum packet size for the endpoint (512 bytes for high speed, 64 bytes for full speed). The last packet in the series may be less than the maximum packet size. Indeed, the receiver may use the reception of this ‘short’ packet to signal the end of the transfer (a null packet may be sent at the end of the series if the size of the data block is an exact multiple of the maximum packet size).

The DMA facilities of the USB 2.0 Controller may be used in either Peripheral mode or Host mode to avoid the overhead of having to interrupt the processor after each individual packet, instead only interrupting the processor after the transfer has completed.

The following sections outline the basic actions that are involved in using DMA alongside some standard types of Bulk Tx and Bulk Rx transfer.

14.1.1.14.3.1. Using DMA with Bulk Tx Endpoints

For Tx endpoints, the DMA request line goes high when the endpoint FIFO is able to accept a data packet, and goes low when `TXMAXP` bytes have been loaded into the FIFO. Alternatively, the request line will go low when the `TXPKTRDY` bit in `TXCSRL` is set.

To use DMA to send a large block of data to the USB host over a Bulk Tx endpoint, we recommend setting up the DMA controller and the USB 2.0 Controller as follows.

The DMA controller should be programmed to perform a burst DMA read of the maximum size of packet for the endpoint (512 bytes for high speed, 64 bytes for full speed) when the DMA request line for the endpoint transitions from low to high. The controller should keep performing these burst reads on each DMA request until the entire data block has been transferred (the last burst may however be of less than the maximum packet size). It should then interrupt the software.

The USB 2.0 Controller should be programmed to enable AutoSet and DMA Request Mode 1 by setting the AUTOSET, DMAREQENAB and DMAREQMODE bits in the TXCSRH.

Programmed like this, the USB 2.0 Controller will take the DMA request line high whenever there is space in its FIFO to accept a packet. Further, the TXPKTRDY bit will be automatically set after the DMA controller has loaded the FIFO with a packet of the maximum packet size. The packet is then ready to be sent to the host. When the last packet has been loaded by the DMA controller, the controller should interrupt the processor. If the last packet loaded was less than the maximum packet size, the TXPKTRDY bit will not have been set and will therefore need to be set manually (i.e. by the software) to allow the last packet to be sent. The TXPKTRDY bit will also need to be set manually if the last packet was of the maximum packet size and a null packet is to be sent to indicate the end of the transfer.

If, when operating in Host mode, the core fails to successfully transmit a packet three times, the ERROR bit in the TXCSRL will become set and the DMA request line will be disabled until this ERROR bit is cleared again. It should also be noted that the DMAREQMODE bit in the TXCSRH must not be cleared either before or in the same cycle as the corresponding DMAREQENAB bit is cleared.

14.1.1.14.3.2. Using DMA with Bulk Rx Endpoints

The behavior of the DMA request line for an Rx Endpoint depends on the DMA Request Mode selected through the RXCSRH.DMAREQMODE:

- In DMA Request Mode 0, the Rx DMA request line goes high when a data packet is available in the endpoint FIFO and goes low either when the last byte of the data packet has been read – or when the RXPKTRDY bit in RXCSRL is cleared.
- In DMA Request Mode 1, the DMA request line only goes high when the packet received is of the maximum packet size (as set in the RXMAXP register). If the packet received is of some other size, the DMA request line stays low with the result that the packet remains in the FIFO with RXPKTRDY set. This causes an Rx Endpoint interrupt to be generated (if enabled).

The DMA Request Modes are primarily designed to be used where large packets of data are transferred to a Bulk endpoint. The USB protocol requires such packets to be split into a series of packets of maximum packet size (512 bytes for high speed, 64 bytes for full speed). The last packet in the series may be less than the maximum packet size (or a null packet if the total size of the transfer is an exact multiple of the maximum packet size) and the receiver may interpret this ‘short’ packet as signaling the end of the transfer. DMA Request Mode 1 can be used, with a suitably programmed DMA controller, to avoid the overhead of having to interrupt the processor after each individual packet – instead just interrupting the processor after the transfer has completed.

If the Request Mode is switched from Request Mode 1 to Request Mode 0, the request line will be asserted if there is a packet in the FIFO in order to allow this ‘pre-received’ packet to be downloaded.

14.1.1.15. Full-Speed/Low-Bandwidth Interrupt Transactions

This section covers the following topics:

- In Peripheral Mode
- In Host Mode

14.1.1.15.1. In Peripheral Mode

An Interrupt IN transaction uses the same protocol as a Bulk IN transaction (described in Bulk IN Transactions section) and can be used the same way. Similarly, an Interrupt OUT transaction uses

almost the same protocol as a Bulk OUT transaction (described in Bulk OUT Transactions section) and can be used the same way.

You should however note that Tx endpoints on a USB 2.0 Controller that is used as a peripheral support one feature for Interrupt IN transactions that they do not support in Bulk IN transactions, in that they support continuous toggle of the data toggle bit. This feature is enabled by setting the `FRCDATATOG` bit in the `TXCSRH`. When this bit is set to '1', the USB 2.0 Controller will consider the packet as having been successfully sent and toggle the data bit for the endpoint, regardless of whether an ACK was received from the host.

Another difference is that Interrupt endpoints do not support PING flow control. This means that the USB 2.0 Controller should never respond with a NYET handshake, only ACK/NAK/STALL. To ensure this, the `DISNYET_OR_PID_ERROR` in the `RXCSRH` should be set to '1' to disable the transmission of NYET handshakes in High-speed mode.

Though DMA can be used with an Interrupt OUT endpoint, it generally offers little benefit as Interrupt endpoints are usually expected to transfer all their data in a single packet.

14.1.1.15.2. In Host Mode

When the USB 2.0 Controller is operating as the host, interactions with an Interrupt endpoint on the USB peripheral are handled in very much the same way as the equivalent Bulk transactions (described in Bulk IN Transactions and Bulk OUT Transactions, respectively) – except that high-bandwidth Interrupt transactions are supported.

The principal difference as far as operational steps are concerned is that `RXTYPE[5:4]` and `TXTYPE[5:4]` need to be set to 0x3 (to represent an Interrupt transaction) rather than to 10.

The required polling interval also needs to be set in the `RXINTERVAL`/`TXINTERVAL`.

14.1.1.16. Full-Speed/Low Bandwidth Isochronous Transactions

This section covers the following topics:

- In Peripheral Mode
- In Host Mode

14.1.1.16.1. In Peripheral Mode

This section covers the following topics:

- IN Transactions
- OUT Transactions

14.1.1.16.1.1. IN Transactions

This section covers the following topics:

- Setup
- Operation
- Error Handling

An Isochronous IN transaction is used to transfer periodic data from the function controller to the host. This section describes the use in Peripheral mode of full-speed Isochronous Tx endpoints and low bandwidth (1 packet per microframe) high-speed Isochronous Tx endpoints. High bandwidth high-speed (> 8 Mbps) endpoints are described in a later section.

Three optional features are available for use with a Tx endpoint used in Peripheral mode for Isochronous IN transactions:

- **Double packet buffering**

  Since the dynamic FIFO sizing is used by USB 2.0 Controller, the use of single or double packet buffering is part of the specification for the endpoint FIFO. When enabled, up to two packets can be stored in the FIFO awaiting transmission to the host.

  

  Double packet buffering is generally advisable for Isochronous transactions in order to avoid Under run errors (see Operation below).

- **DMA**

  If DMA is enabled for the endpoint, DMA request will be generated whenever the endpoint is able to accept another packet in its FIFO. This feature can be used to allow a DMA controller to load packets into the FIFO without processor intervention. However, this feature is not particularly useful with Isochronous endpoints because the packets transferred are often not maximum packet size and the TxCSR registers (`TXCSRL` - `TXCSRH`) need to be accessed following every packet to check for Under run errors.

- **AutoSet**

  When the AutoSet feature is enabled for a low-bandwidth Isochronous endpoint, the `TXPKTRDY` bit of the `TXCSRL` will be automatically set when a packet of `TXMAXP` bytes has been loaded into the FIFO. However, this feature is not particularly useful with Isochronous endpoints because the packets transferred are often not maximum packet size and the TxCSR registers (`TXCSRL` - `TXCSRH`) need to be accessed following every packet to check for Under run errors.

14.1.1.16.1.1.1. Setup

In configuring a Tx endpoint for Isochronous IN transactions, the `TXMAXP` register must be written with the maximum packet size (in bytes) for the endpoint. This value should be the same as the `wMaxPacketSize` field of the Standard Endpoint Descriptor for the endpoint. In addition, the relevant interrupt enable bit in the `INTRTXE` should be set to 1 (if an interrupt is required for this endpoint) and the `TXCSRH` register should be set as shown below:

Table 14-9 Values of `TXCSRH` register

Bits	Name	Value	Description
[7]	`AUTOSET`	0/1	Set to 1 if the AutoSet feature is required.
[6]	`ISO`	1	Set to 0 to enable Isochronous protocol.
[5]	`MODE`	1	Set to 1 to ensure the FIFO is enabled (only necessary if the FIFO is shared with an Rx endpoint).
[4]	`DMAREQENAB`	0/1	Set to 1 if a DMA request is required for this endpoint. If set to 1, will also need to select the chosen `DMAREQMODE` (`TXCSRH.DMAREQMODE`).
[3]	`FRCDATATOG`	0	Ignored in Isochronous mode.

14.1.1.16.1.1.2. Operation

An Isochronous endpoint does not support data retries, so if data underrun is to be avoided, the data to be sent to the host must be loaded into the FIFO before the IN token is received. The host will send one IN token per frame (or microframe in Highspeed mode), however the timing within the frame (or microframe) can vary. If an IN token is received near the end of one frame and then at the start of the next frame, there will be little time to reload the FIFO. For this reason, double buffering of the endpoint is usually necessary.

The AutoSet feature can be used with a low-bandwidth Isochronous Tx endpoint, in the same way as with a Bulk Tx endpoint. However, unless the data arrives from the source at an absolutely consistent rate, synchronized to the host's frame clock, the size of the packets sent to the host will have to increase or decrease from frame to frame (or from microframe to microframe) to match the source data rate. This means that the actual packet sizes will not always be `TXMAXP` in size, rendering the AutoSet feature useless.

An interrupt is generated whenever a packet is sent to the host and the software may use this interrupt to load the next packet into the FIFO and set the `TXPKTRDY` bit in the `TXCSRL` in the same way as for a Bulk Tx endpoint. As the interrupt could occur almost any time within a frame(/microframe), depending on when the host has scheduled the transaction, this may result in irregular timing of FIFO load requests. If the data source for the endpoint is coming from some external hardware, it may be more convenient to wait until the end of each frame(/microframe) before loading the FIFO as this will minimize the requirement for additional buffering. This can be done by using either the SOF interrupt (`INTRUSB`.`SOF`) or the external `sof_pulse` signal from the USB 2.0 Controller to trigger the loading of the next data packet. The `sof_pulse` is generated once per frame(/microframe) when a SOF packet is received (the USB 2.0 Controller also maintains an external frames(/microframe) counter so it can still generate a `sof_pulse` when the SOF packet has been lost). The interrupts may still be used to set the `TXPKTRDY` bit in `TXCSRL` and to check for data overruns/under runs (see Error Handling below).

Starting up a double-buffered Isochronous IN pipe can be a source of problems. Double buffering requires that a data packet is not transmitted until the frame(/microframe) after it is loaded. There is no problem if the function loads the first data packet at least a frame(/microframe) before the host sets up the pipe (and therefore starts sending IN tokens). But if the host has already started sending IN tokens by the time the first packet is loaded, the packet may be transmitted in the same frame(/microframe) as it is loaded, depending on whether it is loaded before, or after, the IN token is received. This potential problem can be avoided by setting the `ISO_UPDATE` bit in the `POWER`. When this bit is set to 1, any data packet loaded into an Isochronous Tx endpoint FIFO will not be transmitted until after the next SOF packet has been received, thereby ensuring that the data packet is not sent too early.

14.1.1.16.1.1.3. Error Handling

If the endpoint has no data in its FIFO when an IN token is received, it will send a null data packet to the host and set the `UNDER_RUN` bit in the `TXCSRL`. This is an indication that the software is not supplying data fast enough for the host. It is up to the application to determine how this error condition is handled.

If the software is loading one packet per frame(/microframe) and it finds that the `TXPKTRDY` bit in the `TXCSRL` is set when it wants to load the next packet, this indicates that a data packet has not been sent (perhaps because an IN token from the host was corrupted). It is up to the application how it handles this condition: it may choose to flush the unsent packet by setting the `FLUSH_FIFO` bit in the `TXCSRL`, or it may choose to skip the current packet.

14.1.1.16.1.2. OUT Transactions

This section covers the following topics:

- Setup
- Operation
- Error Handling

An Isochronous OUT transaction is used to transfer periodic data from the host to the function controller. This section describes the use in Peripheral mode of full-speed Isochronous Rx endpoints and low bandwidth (1 packet per microframe) high-speed Isochronous Rx endpoints. High bandwidth high-speed (> 8 Mbps) endpoints are described in a later section.

Three optional features are available for use with an Rx endpoint used in Peripheral mode for Isochronous OUT transactions:

- **Double packet buffering**

  Since the dynamic FIFO sizing is used by USB 2.0 Controller, the use of single or double packet buffering is part of the specification for the endpoint FIFO. When enabled, up to two packets can be stored in the FIFO awaiting transmission to the host.

  

  Double packet buffering is generally advisable for Isochronous transactions in order to avoid Overrun errors (see Operation below).

- **DMA**

  If DMA is enabled for the endpoint, a DMA request will be generated whenever the endpoint has a packet in its FIFO. This feature can be used to allow a DMA controller to unload packets from the FIFO without processor intervention. However, this feature is not particularly useful with Isochronous endpoints because the packets transferred are often not maximum packet size and the RxCSR register needs to be accessed following every packet to check for Overrun or CRC errors.

- **AutoClear**

  When the AutoClear feature is enabled, the `RXPKTRDY` bit of the `RXCSRL` will be automatically cleared when a packet of `RXMAXP` bytes has been unloaded from the FIFO (with exceptions, see register description). However, this feature is not particularly useful with Isochronous endpoints because the packets transferred are often not maximum packet size and the RxCSR register needs to be accessed following every packet to check for Overrun or CRC errors.

14.1.1.16.1.2.1. Setup

In configuring an Rx endpoint for Isochronous OUT transactions, the `RXMAXP` register must be written with the maximum packet size (in bytes) for the endpoint. This value should be the same as the `wMaxPacketSize` field of the Standard Endpoint Descriptor for the endpoint. In addition, the relevant interrupt enable bit in the `INTRRXE` should be set to 1 (if an interrupt is required for this endpoint) and the `RXCSRH` should be set as shown below:

Table 14-10 Values of `RXCSRH` register

Bits	Name	Value	Description
[7]	`AUTOCLEAR`	0/1	Set to 1 if the AutoSet feature is required.
[6]	`ISO`	1	Set to 0 to enable Isochronous protocol.
[5]	`DMAREQENAB`	0/1	Set to 1 if a DMA request is required for this endpoint.

Table 14-10 Values of RXCSRH register (continued)

Bits	Name	Value	Description
			If set to 1, will also need to select the chosen DMAREQMODE (RXCSRH.DMAREQMODE).
[4]	DISNYET	0	Ignored in Isochronous mode.

14.1.1.16.1.2.2. Operation

An Isochronous endpoint does not support data retries so, if a data overrun is to be avoided, there must be space in the FIFO to accept a packet when it is received. The host will send one packet per frame (or microframe in High-speed mode), however the time within the frame can vary. If a packet is received near the end of one frame(/microframe) and another arrives at the start of the next frame, there will be little time to unload the FIFO. For this reason, double buffering of the endpoint is usually necessary.

The AutoClear feature can be used with an Isochronous Rx endpoint, in the same way as for a Bulk Rx endpoint. However, unless the data sink receives data at an absolutely consistent rate and is synchronized to the host's frame clock, the size of the packets sent from the host will have to increase or decrease from frame to frame (or from microframe to microframe) to match the required data rate. This means that the actual packet sizes will not always be RXMAXP in size, rendering the AutoClear feature useless.

An interrupt is generated whenever a packet is received from the host and the software may use this interrupt to unload the packet from the FIFO and clear the RXPKTRDY bit in the RXCSRL in the same way as for a Bulk Rx endpoint. As the interrupt could occur almost any time within a frame(/microframe), depending on when the host has scheduled the transaction, the timing of FIFO unload requests will probably be irregular. If the data sink for the endpoint is going to some external hardware, it may be better to minimize the requirement for additional buffering by waiting until the end of each frame(/microframe) before unloading the FIFO. This can be done by using the SOF interrupt (INTRUSB.SOF) to trigger the unloading of the data packet. The interrupts may still be used to clear the RXPKTRDY bit in RXCSRL and to check for data overruns/under runs (see Error Handling below).

14.1.1.16.1.2.3. Error Handling

If there is no space in the FIFO to store a packet when it is received from the host, the OVERRUN bit in the RXCSRL will be set. This is an indication that the software is not unloading data fast enough for the host. It is up to the application to determine how this error condition is handled.

If the USB 2.0 Controller finds that a received packet has a CRC error, it will still store the packet in the FIFO and set the RXPKTRDY bit and the DATAERROR bit of the RXCSRL). It is left up to the application how this error condition is handled.

14.1.1.16.2. In Host Mode

This section covers the following topics:

- IN Transactions
- OUT Transactions

14.1.1.16.2.1. IN Transactions

This section covers the following topics:

- Setup
- Operation
- Error Handling

An Isochronous IN transaction is used to transfer periodic data from the function controller to the host. This section describes the use in Host mode of full-speed Isochronous Rx endpoints and low bandwidth (1 packet per microframe) high-speed Isochronous Rx endpoints. High bandwidth high-speed (> 8 Mbps) endpoints are described in a later section.

Four optional features are available for use with an Rx endpoint used in Host mode to receive this data:

- **Double packet buffering**

  Since the dynamic FIFO sizing is used by USB 2.0 Controller, the use of single or double packet buffering is part of the specification for the endpoint FIFO.

  

  Double packet buffering is generally advisable for Isochronous transactions in order to avoid data overrun (see Operation below).

- **DMA**

  If DMA is enabled for the endpoint, a DMA request will be generated whenever the endpoint has a packet in its FIFO. This feature can be used to allow a DMA controller to unload packets from the FIFO without processor intervention. However, this feature is not particularly useful with Isochronous endpoints because the packets transferred are often not maximum packet size.

- **AutoClear**

  When the AutoClear feature is enabled, the `RXPKTRDY` bit of the `RXCSRL` will be automatically cleared when a packet of `RXMAXP` bytes has been unloaded from the FIFO (with exceptions, see register description). However, this feature is not particularly useful with Isochronous endpoints because the packets transferred are often not maximum packet size and the RxCSR register needs to be accessed following every packet to check for Overrun or CRC errors.

- **AutoReq(uest)**

  When the AutoReq(uest) feature is enabled, the `REQPKT` bit of the `RXCSRL` will be automatically set when the `RXPKTRDY` bit is cleared.

14.1.1.16.2.1.1. Setup

Before initiating an Isochronous IN Transaction:

- The `RXTYPE` for the USB 2.0 Controller endpoint that is to be used needs to be written with bits [7:6] set to select the operating speed, bits [5:4] = 0x1 (to select an Isochronous transfer) and bits [3:0] set to the value of the endpoint number contained in the IN endpoint descriptor returned to the USB 2.0 Controller during device enumeration.
- The `RXINTERVAL` for the USB 2.0 Controller endpoint needs to be written with the required transaction interval (usually one transaction every frame/microframe).
- The `RXMAXP` register for the USB 2.0 Controller endpoint must be written with the maximum packet size (in bytes) for the transmission. This value should be the same as the `wMaxPacketSize` field of the Standard Endpoint Descriptor for the target endpoint.
- The relevant interrupt enable bit in the `INTRRXE` should be set to '1' (if an interrupt is required for this endpoint)
- The following bits of the `RXCSRH` register should be set as shown below:

Table 14-11 Values of RXCSRH register

Bits	Name	Value	Description
[7]	AUTOCLEAR	0/1	Set to 1 if the AutoClear feature is required.
[6]	AUTOREQ	0/1	Set to 1 if the AutoRequest feature is required.
[5]	DMAREQENAB	0/1	Set to 1 if a DMA request is required for this endpoint. If set to 1, will also need to select the chosen DMAREQMODE (RXCSRH.DMAREQMODE).
[4]	PID_ERROR	0	Ignored in Isochronous mode.

14.1.1.16.2.1.2. Operation

The operation starts with the CPU setting REQPKT bit of the RXCSRL. This causes the USB 2.0 Controller to send an IN token to the target.

When a packet is received, an interrupt is generated which the software may use to unload the packet from the FIFO and clear the RXPKTRDY bit in the RXCSRL in the same way as for a Bulk Rx endpoint. As the interrupt could occur almost any time within a frame(/microframe), the timing of FIFO unload requests will probably be irregular. If the data sink for the endpoint is going to some external hardware, it may be better to minimize the requirement for additional buffering by waiting until the end of each frame before unloading the FIFO. This can be done by using the sof_pulse signal from the USB 2.0 Controller to trigger the unloading of the data packet. The sof_pulse is generated once per frame(/microframe). The interrupts may still be used to clear the RXPKTRDY bit in RXCSRL.

The AutoClear feature can be used with an Isochronous Rx endpoint, in the same way as for a Bulk Rx endpoint. However, unless the data sink receives data at an absolutely consistent rate and is synchronized to the USB 2.0 Controller's frame clock, the size of the packets will increase or decrease from frame to frame (or microframe to microframe) to match the required data rate. This means that the actual packet sizes will not always be RXMAXP in size, rendering the AutoClear feature useless.

14.1.1.16.2.1.3. Error Handling

If a CRC or bit-stuff error occurs during the reception of a packet, the packet will still be stored in the FIFO but the DATAERROR_OR_NAK_TIMEOUT bit of the RXCSRL is set to indicate that the data may be corrupt.

14.1.1.16.2.2. OUT Transactions

This section covers the following topics:

- Setup
- Operation

An Isochronous OUT transaction is used to transfer periodic data from the host to the function controller. This section describes the use of full-speed Isochronous Tx endpoints and low bandwidth (1 packet per microframe) high-speed Isochronous Tx endpoints. High bandwidth high-speed (> 8 Mbps) endpoints are described in a later section.

Three optional features are available for use with a Tx endpoint used in Host mode to transmit this data:

- **Double packet buffering**

  Since the dynamic FIFO sizing is used by USB 2.0 Controller, the use of single or double packet buffering is part of the specification for the endpoint FIFO. When enabled, up to two packets can be stored in the FIFO awaiting transmission to the peripheral.

- **DMA**

  If DMA is enabled for the endpoint, a DMA request will be generated whenever the endpoint is able to accept another packet in its FIFO. However, this feature is not particularly useful with Isochronous endpoints because the packets transferred are often not maximum packet size.

- **AutoSet**

  When the AutoSet feature is enabled with a low-bandwidth Isochronous endpoint, the `TXPKTRDY` bit of the `TXCSRL` will be automatically set when a packet of `TXMAXP` bytes has been loaded into the FIFO. However, this feature is not particularly useful with Isochronous endpoints because the packets transferred are often not maximum packet size.

14.1.1.16.2.2.1. Setup

Before initiating an Isochronous OUT Transaction:

- The `TXTYPE` for the USB 2.0 Controller endpoint that is to be used needs to be written with bits [7:6] set to select the operating speed, bits [5:4] = 0x1 (to select an Isochronous transfer) and bits [3:0] set to the value of the endpoint number contained in the OUT endpoint descriptor returned to the USB 2.0 Controller during device enumeration.
- The `TXINTERVAL` for the USB 2.0 Controller endpoint needs to be written with the required transaction interval (usually one transaction every frame/microframe).
- The `TXMAXP` for the USB 2.0 Controller endpoint must be written with the maximum packet size (in bytes) for the transmission. This value should be the same as the `wMaxPacketSize` field of the Standard Endpoint Descriptor for the target endpoint.
- The relevant interrupt enable bit in the `INTRTXE` should be set to '1' (if an interrupt is required for this endpoint)
- The following bits of the `TXCSRH` register should be set as shown below:

Table 14-12 Values of `TXCSRH` register

Bits	Name	Value	Description
[7]	`AUTOSET`	0/1	Set to 1 if the AutoSet feature is required.
[5]	`MODE`	1	Set to 1 to ensure FIFO is enabled (only necessary if the FIFO is shared with an Rx endpoint).
[4]	`DMAREQENAB`	0/1	Set to 1 if a DMA request is required for this endpoint.  If set to 1, will also need to select the chosen `DMAREQMODE` (`TXCSRH`.`DMAREQMODE`).
[3]	`FRCDATATOG`	0	Ignored in Isochronous mode.

14.1.1.16.2.2.2. Operation

The operation starts when the software writes to the FIFO then sets `TXPKTRDY` bit of the `TXCSRL`. This triggers the USB 2.0 Controller to send an OUT token followed by the first data packet from the FIFO.

An interrupt is generated whenever a packet is sent and the software may use this interrupt to load the next packet into the FIFO and set the `TXPKTRDY` bit in the `TXCSRL` in the same way as for a Bulk Tx endpoint. As the interrupt could occur almost any time within a frame, depending on when the host has scheduled the transaction, this may result in irregular timing of FIFO load requests. If the data source for the endpoint is coming from some external hardware, it may be more convenient to wait until the end of each frame before loading the FIFO as this will minimize the requirement for additional buffering. This can be done by using the `sof_pulse` signal from the USB 2.0 Controller to trigger the loading of the next data packet. The `sof_pulse` is generated once per frame(/microframe). The interrupts may still be used to set the `TXPKTRDY` bit in the `TXCSRL`.

The AutoSet feature can be used with a low-bandwidth Isochronous Tx endpoint, in the same way as with a Bulk Tx endpoint. However, unless the data arrives from the source at an absolutely consistent rate, synchronized to the USB 2.0 Controller's frame clock, the size of the packets will increase or decrease from frame to frame (or from microframe to microframe) to match the source data rate. This means that the actual packet sizes will not always be `TXMAXP` in size, rendering the AutoSet feature useless.

14.1.1.17. High Bandwidth Isochronous/Interrupt Transactions

High-Bandwidth Isochronous/Interrupt transactions use much the same protocol as other Isochronous/ Interrupt transactions.

There are, however, some special features to conducting High-Bandwidth transactions.

1. When setting the Maximum Packet size handled by the endpoint in the `TXMAXP`/`RXMAXP`, the maximum number of transactions per microframe also needs to be set via bits [11] and [12] of that register.

   This maximum number of transactions (2 or 3) also represents the maximum number of sections in which any single 'High-Bandwidth' packet can be transferred, which in turn sets the maximum size of packet to 2 or 3 times the maximum payload specified for the endpoint in the same register.

   

   The maximum payload that can be sent in any transaction is 1 KB.

2. When sending packets, `TXPKTRDY` needs to be set by the application software. Similarly, when unloading packets from the Rx endpoint FIFO, `RXPKTRDY` needs to be cleared by the application software.

   The AutoSet and AutoClear functions cannot be used to set and clear these bits in High-Bandwidth transactions.

3. The transmission of packets as a number of sections introduces a further type of error – the transmission of Incomplete packets.

   For Tx endpoints, the issue principally applies when the USB 2.0 Controller is in Peripheral mode and occurs when the USB 2.0 Controller fails to receive enough IN tokens from the host to send all the parts of the data packet. It can also apply to High-Bandwidth Interrupt transactions in Host mode where the core does not receive any response from the device to which the packet is being sent. In both cases, the USB 2.0 Controller will set the `INCOMP_TX` bit in the `TXCSRL`.

   For Rx endpoints, the issue occurs when the PIDs of the received parts of the data packet show that one or more parts of the data packet has not been received. When this happens, the USB 2.0 Controller sets the `INCOMPRX` bit in the `RXCSRH`. In the main, this bit will only be set when

the core is operating in Peripheral mode but it can also be set in Host mode if (and only if) the device with which the USB 2.0 Controller is communicating fails to respond in accordance with the USB protocol.

14.1.2. USB 2.0 Controller Registers

This section covers the following topics:

- Register Map
- Register Descriptions

The following table describes BE-U1000 microcontroller USB 2.0 subsystem register region.

Table 14-13 Memory address region for USB registers

Block Name	Size	Start Address	End Address
USB*	1 MB	0x1500_0000	0x150F_FFFF

* USB consists of two components — USB 2.0 Controller and USB 2.0 PHY, where the start address of USB 2.0 Controller register region coincides with USB 2.0 Subsystem register region (0x1500_0000), and USB 2.0 PHY registers can be accessed not directly, but using this region registers. For details, refer to USB 2.0 PHY Registers of USB 2.0 PHY section.

You can find list and description of USB 2.0 Controller registers in the Register Map and Register Descriptions sections below.

14.1.2.1. Register Map

This section tables contain information about the USB 2.0 Controller registers memory map.

The table below contains 8-bit and 16-bit registers, but the access to registers is implemented via the 32-bit wide bus. That means you will get more than a one register data in a case of access.

The following table provides a high-level summary of each register. To view the detailed description of a register, click the register name in the **Description** column.

Table 14-14 USB 2.0 Controller register map

Name	Offset	Description	Default Value
Common USB Registers			
FADDR	0x00	Function Address Register	0x0
POWER	0x01	Power Management Register	0x20
INTRTX	0x02	Interrupt Register for Endpoint 0 and Tx Endpoints	0x0
INTRRX	0x04	Interrupt Register for Rx Endpoints	0x0
INTRTXE	0x06	Interrupt Enable Register for INTRTX	0x3F
INTRRXE	0x08	Interrupt Enable Register for INTRRX	0x3E

Table 14-14 USB 2.0 Controller register map (continued)

Name	Offset	Description	Default Value
INTRUSB	0x0A	USB Common Interrupts Register	0x0
INTRUSBE	0x0B	USB Common Interrupts Enable Register	0x6
FRAME	0x0C	Frame Number Register	0x0
INDEX	0x0E	Index Register	0x0
TESTMODE	0x0F	USB Test Modes Register	0x0
Indexed Registers			
TXMAXP	0x10	Maximum Packet Size for Tx Endpoint #n **Exists:** INDEX.SELECTED_EP != 0x0	0x0
CSR0L	0x12	Endpoint 0 Control and Status Register 0 **Exists:** INDEX.SELECTED_EP = 0x0	0x0
TXCSRL	0x12	Tx Endpoint #n Control and Status Register 0 **Exists:** INDEX.SELECTED_EP != 0x0	0x0
CSR0H	0x13	Endpoint 0 Control and Status Register 1 **Exists:** INDEX.SELECTED_EP = 0x0	0x0
TXCSRH	0x13	Tx Endpoint #n Control and Status Register 1 **Exists:** INDEX.SELECTED_EP != 0x0	0x0
RXMAXP	0x14	Maximum Packet Size for Rx Endpoint #n **Exists:** INDEX.SELECTED_EP != 0x0	0x0
RXCSRL	0x16	Rx Endpoint #n Control and Status Register 0 **Exists:** INDEX.SELECTED_EP != 0x0	0x0
RXCSRH	0x17	Rx Endpoint #n Control and Status Register 1 **Exists:** INDEX.SELECTED_EP != 0x0	0x0
COUNT0	0x18*	Endpoint 0 Data Bytes Counter Register **Exists:** INDEX.SELECTED_EP = 0x0	0x0
RXCOUNT	0x18*	Rx Endpoint #n Data Bytes Counter Register **Exists:** INDEX.SELECTED_EP != 0x0	0x0
TYPE0	0x1A*	Endpoint 0 Speed Register **Exists:** DEVCTL.HOST_MODE = 0x1 && INDEX.SELECTED_EP = 0x0	0x0
TXTYPE	0x1A*	Tx Endpoint #n Settings Register **Exists:** DEVCTL.HOST_MODE = 0x1 && INDEX.SELECTED_EP != 0x0	0x0
NAKLIMIT0	0x1B*	Endpoint 0 NAK Limit Register **Exists:** DEVCTL.HOST_MODE = 0x1 && INDEX.SELECTED_EP = 0x0	0x0
TXINTERVAL	0x1B*	Tx Endpoint #n NAK Limit/Polling Interval Register	0x0

Table 14-14 USB 2.0 Controller register map (continued)

Name	Offset	Description	Default Value
		Exists: DEVCTL.HOST_MODE = 0x1 && INDEX.SELECTED_EP != 0x0	
RXTYPE	0x1C	Rx Endpoint #n Settings Register **Exists:** DEVCTL.HOST_MODE = 0x1 && INDEX.SELECTED_EP != 0x0	0x0
RXINTERVAL	0x1D	Rx Endpoint #n NAK Limit/Polling Interval Register **Exists:** DEVCTL.HOST_MODE = 0x1 && INDEX.SELECTED_EP != 0x0	0x0
CONFIGDATA	0x1F	Configuration Data Register **Exists:** INDEX.SELECTED_EP = 0x0	0xDE
FIFOs			
FIFO**n** (where **n** = 0...5)	0x20 + (**n***0x4)	Endpoint #n FIFO	0x0
Additional Control & Configuration Registers			
DEVCTL	0x60	Device Control Register	0x80
	0x61	Reserved	
TXFIFOSZ	0x62	Tx Endpoint #n FIFO Size **Exists:** INDEX.SELECTED_EP != 0x0	0x0
RXFIFOSZ	0x63	Rx Endpoint #n FIFO Size **Exists:** INDEX.SELECTED_EP != 0x0	0x0
TXFIFOADD	0x64	Tx Endpoint #n FIFO Address **Exists:** INDEX.SELECTED_EP != 0x0	0x0
RXFIFOADD	0x66	Rx Endpoint #n FIFO Address **Exists:** INDEX.SELECTED_EP != 0x0	0x0
VCONTROL	0x68**	PHY Signals Control Register **Exists:** This register is presented when the write at the address 0x68 is performed	0x0
VSTATUS	0x68**	PHY Signals Status Register **Exists:** This register is presented when the read at the address 0x68 is performed	0x0
EPINFO	0x78	Information About Numbers of Tx and Rx Endpoints	0x55
RAMINFO	0x79	Information About RAM	0xB*
LINKINFO	0x7A	Information About Delays	0x5C
VPLEN	0x7B	Duration of the VBus Pulsing Charge	0x3C

Table 14-14 USB 2.0 Controller register map (continued)

Name	Offset	Description	Default Value
HS_EOF1	0x7C	Time Buffer Available on High-Speed Transactions	0x80
FS_EOF1	0x7D	Time Buffer Available on Full-Speed Transactions	0x77
LS_EOF1	0x7E	Time Buffer Available on Low-Speed Transactions	0x72
	0x7F	Reserved	
Target Address Registers			
TXFUNCADDR (where **n** = 0...5)	0x80 + (**n***0x8)	Tx Endpoint #n Function Address **Exists:** DEVCTL.HOST_MODE = 0x1	0x0
TXHUBADDR (where **n** = 0...5)	0x82 + (**n***0x8)	Tx Endpoint #n Hub Address **Exists:** DEVCTL.HOST_MODE = 0x1	0x0
TXHUBPORT (where **n** = 0...5)	0x83 + (**n***0x8)	Tx Endpoint #n Hub Port **Exists:** DEVCTL.HOST_MODE = 0x1	0x0
RXFUNCADDR (where **n** = 0...5)	0x84 + (**n***0x8)	Rx Endpoint #n Function Address **Exists:** DEVCTL.HOST_MODE = 0x1	0x0
RXHUBADDR (where **n** = 0...5)	0x86 + (**n***0x8)	Rx Endpoint #n Hub Address **Exists:** DEVCTL.HOST_MODE = 0x1	0x0
RXHUBPORT (where **n** = 0...5)	0x87 + (**n***0x8)	Rx Endpoint #n Hub Port **Exists:** DEVCTL.HOST_MODE = 0x1	0x0
DMA Registers			
DMA_INTR	0x200	DMA Channel #n Interrupt Register	0x0
DMA_CNTL (where **n** = 1...5)	0x204 + (**n**-1)*0x10	DMA Channel #n Transfer Control Register	0x0
DMA_ADDR (where **n** = 1...5)	0x208 + (**n**-1)*0x10	DMA Channel #n Address Register	0x0
DMA_COUNT (where **n** = 1...5)	0x20C + (**n**-1)*0x10	DMA Channel #n Transfer Count Register	0x0
Extended Registers			
EPn_RQPKTCOUNT (where **n** = 1...5)	0x300 + (**n***0x4)	Number of Requested Packets for Rx Endpoint #n **Exists:** DEVCTL.HOST_MODE = 0x1	0x0
	0x340-0x343	Reserved	
C_T_UCH	0x344	Chirp Timeout Timer Register	0x101D0
C_T_HSRTN	0x346	High Speed Resume Signaling Delay	0x34BC

Table 14-14 USB 2.0 Controller register map (continued)

Name	Offset	Description	Default Value
C_T_HSBT	0x348	High Speed Timeout Increase Register	0x0
LPM Registers			
LPM_ATTR	0x360	LPM Attribute Register	0x0
LPM_CNTRL	0x362	LPM Control Register	0x0
LPM_INTREN	0x363	LPM Interrupt Enable Register	0x0
LPM_INTR	0x364	LPM Interrupt Register	0x0
LPM_FADDR	0x365	LPM Function Address Register **Exists:** DEVCTL.HOST_MODE = 0x1	0x0

*The same space, at the same offset can be used to read/write the different data, depending on we set control signals or read status signals this moment.

14.1.2.2. Register Descriptions

The following subsections describe the data fields of the USB 2.0 Controller registers.

Reserved bits in the USB registers cannot be used.

WS — means that the bit can only be written to set it

WC — means that the bit can only be written to clear it

R/WS — means that the bit can be read or set (written to set it) but it can't be cleared

R/WC — means that the bit can be read or cleared (written to clear it) but it can't be set

14.1.2.2.1. Function Address Register (FADDR)

The FADDR register is at offset 0x0.

FADDR is an 8-bit register that should be written with the 7-bit address of the peripheral part of the transaction.

When the USB 2.0 Controller is being used in Peripheral mode (DEVCTL.HOST_MODE=0), this register should be written with the address received through a SET_ADDRESS command, which will then be used for decoding the function address in subsequent token packets.

Peripheral Mode Only!

Core Configured with Multipoint support: This register is only applies to operations carried out when the USB 2.0 Controller is in Peripheral mode. In Host mode, this register is ignored.

The following table shows the register bit assignments.

Table 14-15 Fields for register: `FADDR`

Bits	Name	Software Access	USB Access	Description
[7]				Reserved
[6:0]	`FUNC_ADDR`	R/W	R	The function address. **Value After Reset:** 0x0

14.1.2.2.2. Power Management Register (`POWER`)

The `POWER` register is at offset 0x1.

`POWER` is an 8-bit register that is used for controlling Suspend and Resume signaling, and some basic operational aspects of the USB 2.0 Controller.

The following table shows the register bit assignments.

Table 14-16 Fields for register: `POWER`

Bits	Name	Software Access	USB Access	Description
[7]	`ISO_UPDATE`	R/W	R	When set by the software, the USB 2.0 Controller will wait for an SOF token from the time `TXPKTRDY` is set before sending the packet. If an IN token is received before an SOF token, then a zero length data packet will be sent. Only valid in Peripheral Mode. Also, this bit only affects endpoints performing Isochronous transfers. This bit is not valid in Host Mode. **Value After Reset:** 0x0
[6]	`SOFT_CONN`	R/W	R	If Soft Connect/Disconnect feature is enabled, then the USB D+/D- lines are enabled when this bit is set by the software and tri-stated when this bit is cleared by the software. Only valid in Peripheral Mode. This bit is not valid in Host Mode. **Value After Reset:** 0x0
[5]	`HS_ENAB`	R/W	R	When set by the software, the USB 2.0 Controller will negotiate for High-speed mode when the device is reset by the hub. If not set, the device will only operate in Full-speed mode. **Value After Reset:** 0x1
[4]	`HS_MODE`	R	R/W	When set, this bit indicates High-speed mode successfully negotiated during USB reset.

Table 14-16 Fields for register: `POWER` (continued)

Bits	Name	Software Access	USB Access	Description
				In Peripheral Mode, becomes valid when USB reset completes (as indicated by USB reset interrupt). In Host Mode, becomes valid when `RESET` bit is cleared. Remains valid for the duration of the session.  Allowance is made for Tiny-J signaling in determining the transfer speed to select. **Value After Reset:** 0x0
[3]	`RESET`	*Varies	R/W	This bit is set when Reset signaling is present on the bus.  This bit is Read/Write from the software in Host Mode but Read-Only in Peripheral Mode. The access attributes of this field are as follows: • Peripheral Mode ◦ Software Access: R • Host Mode ◦ Software Access: R/W **Value After Reset:** 0x0
[2]	`RESUME`	R/W	*Varies	Set by the software to generate Resume signaling when the device is in Suspend mode. In Peripheral mode, the software should clear this bit after 10 ms (a maximum of 15 ms), to end Resume signaling. In Host mode, the software should clear this bit after 20 ms. The access attributes of this field are as follows: • Peripheral Mode ◦ USB Access: R • Host Mode ◦ USB Access: R/WS **Value After Reset:** 0x0
[1]	`SUSPEND_MODE`	*Varies	*Varies	In Host mode, this bit is set by the software to enter Suspend mode. In Peripheral mode, this bit is set on entry into Suspend mode. It is cleared when the software reads the interrupt register, or sets the Resume bit above. The access attributes of this field are as follows:

Table 14-16 Fields for register: `POWER` (continued)

Bits	Name	Software Access	USB Access	Description
				• Peripheral Mode ◦ Software Access: R ◦ USB Access: R/W • Host Mode ◦ Software Access: WS ◦ USB Access: WC **Value After Reset:** 0x0
[0]	`ENABLE_SUSPENDM`	R/W	R	Set by the software to enable the `suspendm` output. **Value After Reset:** 0x0

14.1.2.2.3. Interrupt Register for Endpoint 0 and Tx Endpoints (`INTRTX`)

The `INTRTX` register is at offset 0x2.

`INTRTX` is a 16-bit read-only register that indicates which interrupts are currently active for Endpoint 0 and the Tx Endpoints 1...5.

Note that all active interrupts are cleared when this register is read.

The following table shows the register bit assignments.

Table 14-17 Fields for register: `INTRTX`

Bits	Name	Software Access	USB Access	Description
[15:6]				Reserved
[5]	`EP5_TX`	R	WS	Tx Endpoint 5 interrupt **Value After Reset:** 0x0
[4]	`EP4_TX`	R	WS	Tx Endpoint 4 interrupt **Value After Reset:** 0x0
[3]	`EP3_TX`	R	WS	Tx Endpoint 3 interrupt **Value After Reset:** 0x0
[2]	`EP2_TX`	R	WS	Tx Endpoint 2 interrupt **Value After Reset:** 0x0
[1]	`EP1_TX`	R	WS	Tx Endpoint 1 interrupt **Value After Reset:** 0x0
[0]	`EP0`	R	WS	Endpoint 0 interrupt **Value After Reset:** 0x0

14.1.2.2.4. Interrupt Register for Rx Endpoints (`INTRRX`)

The `INTRRX` register is at offset 0x4.

`INTRRX` is a 16-bit read-only register that indicates which of the interrupts for Rx Endpoints 1...5 are currently active.

Note that all active interrupts are cleared when this register is read.

The following table shows the register bit assignments.

Table 14-18 Fields for register: `INTRRX`

Bits	Name	Software Access	USB Access	Description
[15:6]				Reserved
[5]	`EP5_RX`	R	WS	Rx Endpoint 5 interrupt **Value After Reset:** 0x0
[4]	`EP4_RX`	R	WS	Rx Endpoint 4 interrupt **Value After Reset:** 0x0
[3]	`EP3_RX`	R	WS	Rx Endpoint 3 interrupt **Value After Reset:** 0x0
[2]	`EP2_RX`	R	WS	Rx Endpoint 2 interrupt **Value After Reset:** 0x0
[1]	`EP1_RX`	R	WS	Rx Endpoint 1 interrupt **Value After Reset:** 0x0
[0]				Reserved

14.1.2.2.5. Interrupt Enable Register for `INTRTX` (`INTRTXE`)

The `INTRTXE` register is at offset 0x6.

`INTRTXE` is a 16-bit register that provides interrupt enable bits.

Where a bit is set to 1, `mc_nint` will be asserted on the corresponding interrupt in the `INTRTX` register becoming set.

Where a bit is set to 0, the interrupt in `INTRTX` is still set but `mc_nint` is not asserted.

On reset, the bits corresponding to Endpoint 0 and the Tx endpoints are set to 1, while the remaining bits are set to 0.

The following table shows the register bit assignments.

Table 14-19 Fields for register: `INTRTXE`

Bits	Name	Software Access	USB Access	Description
[15:6]				Reserved
[5]	`EP5_TX_E`	RW	R	Tx Endpoint 5 interrupt enable **Value After Reset:** 0x1
[4]	`EP4_TX_E`	RW	R	Tx Endpoint 4 interrupt enable **Value After Reset:** 0x1
[3]	`EP3_TX_E`	RW	R	Tx Endpoint 3 interrupt enable

Table 14-19 Fields for register: `INTRTXE` (continued)

Bits	Name	Software Access	USB Access	Description
				Value After Reset: 0x1
[2]	`EP2_TX_E`	RW	R	Tx Endpoint 2 interrupt enable **Value After Reset:** 0x1
[1]	`EP1_TX_E`	RW	R	Tx Endpoint 1 interrupt enable **Value After Reset:** 0x1
[0]	`EP0_E`	RW	R	Endpoint 0 interrupt enable **Value After Reset:** 0x1

14.1.2.2.6. Interrupt Enable Register for `INTRRX` (`INTRRXE`)

The `INTRRXE` register is at offset 0x8.

`INTRRXE` is a 16-bit register that provides interrupt enable bits.

Where a bit is set to 1, `mc_nint` will be asserted on the corresponding interrupt in the `INTRRX` register becoming set.

Where a bit is set to 0, the interrupt in `INTRRX` is still set but `mc_nint` is not asserted.

On reset, the bits corresponding to the Rx endpoints are set to 1, while the remaining bits are set to 0.

The following table shows the register bit assignments.

Table 14-20 Fields for register: `INTRRXE`

Bits	Name	Software Access	USB Access	Description
[15:6]				Reserved
[5]	`EP5_RX_E`	RW	R	Rx Endpoint 5 interrupt enable **Value After Reset:** 0x1
[4]	`EP4_RX_E`	RW	R	Rx Endpoint 4 interrupt enable **Value After Reset:** 0x1
[3]	`EP3_RX_E`	RW	R	Rx Endpoint 3 interrupt enable **Value After Reset:** 0x1
[2]	`EP2_RX_E`	RW	R	Rx Endpoint 2 interrupt enable **Value After Reset:** 0x1
[1]	`EP1_RX_E`	RW	R	Rx Endpoint 1 interrupt enable **Value After Reset:** 0x1
[0]				Reserved

14.1.2.2.7. USB Common Interrupts Register (`INTRUSB`)

The `INTRUSB` register is at offset 0xA.

`INTRUSB` is an 8-bit read-only register that indicates which USB interrupts are currently active. All active interrupts will be cleared when this register is read.

The following table shows the register bit assignments.

Table 14-21 Fields for register: `INTRUSB`

Bits	Name	Software Access	USB Access	Description
[7]	`VBUS_ERROR`	R	WS	Set when VBus drops below the `vbusvalid` threshold during a session. Only valid when USB 2.0 Controller is A' device. **Value After Reset:** 0x0
[6]	`SESS_REQ`	R	WS	Set when Session Request signaling has been detected. Only valid when USB 2.0 Controller is A' device. **Value After Reset:** 0x0
[5]	`DISCON`	R	WS	Set in Host mode when a device disconnect is detected. Set in Peripheral mode when a session ends. Valid at all transaction speeds. **Value After Reset:** 0x0
[4]	`CONN`	R	WS	Set when a device connection is detected. Only valid in Host mode. Valid at all transaction speeds. **Value After Reset:** 0x0
[3]	`SOF`	R	WS	Set when a new frame starts. **Value After Reset:** 0x0
[2]	`RESET_OR_BABBLE`	R	WS	`RESET`: Set in Peripheral mode when Reset signaling is detected on the bus. `BABBLE`: Set in Host mode when babble is detected. `BABBLE` only active after first SOF has been sent. **Value After Reset:** 0x0
[1]	`RESUME`	R	WS	Set when Resume signaling is detected on the bus while the USB 2.0 Controller is in Suspend mode. **Value After Reset:** 0x0
[0]	`SUSPEND`	R	WS	Set when Suspend signaling is detected on the bus. Only valid in Peripheral mode. **Value After Reset:** 0x0

14.1.2.2.8. USB Common Interrupts Enable Register (`INTRUSBE`)

The `INTRUSBE` register is at offset 0xB.

`INTRUSBE` is an 8-bit register that provides interrupt enable bits for each of the interrupts in `INTRUSB`.

The following table shows the register bit assignments.

Table 14-22 Fields for register: `INTRUSBE`

Bits	Name	Software Access	USB Access	Description
[7]	`VBUS_ERROR_E`	R/W	R	`VBUS_ERROR` Enable **Value After Reset:** 0x0
[6]	`SESS_REQ_E`	R/W	R	`SESS_REQ` Enable **Value After Reset:** 0x0
[5]	`DISCON_E`	R/W	R	`DISCON` Enable **Value After Reset:** 0x0
[4]	`CONN_E`	R/W	R	`CONN` Enable **Value After Reset:** 0x0
[3]	`SOF_E`	R/W	R	`SOF` Enable **Value After Reset:** 0x0
[2]	`RESET_E_OR_BABBLE_E`	R/W	R	`RESET_E`: `RESET` Enable `BABBLE`: `BABBLE` Enable **Value After Reset:** 0x1
[1]	`RESUME_E`	R/W	R	`RESUME` Enable **Value After Reset:** 0x1
[0]	`SUSPEND_E`	R/W	R	`SUSPEND` Enable **Value After Reset:** 0x0

14.1.2.2.9. Frame Number Register (`FRAME`)

The `FRAME` register is at offset 0xC.

`FRAME` is a 16-bit read-only register that holds the last received frame number.

The following table shows the register bit assignments.

Table 14-23 Fields for register: `FRAME`

Bits	Name	Software Access	USB Access	Description
[15:11]				Reserved
[10:0]	`FRAME_NUMBER`	R	W	Last Received Frame Number This bit field holds the last received frame number, where `FRAME_NUMBER[10]` is the *Most Significant Bit* (**MSB**) and `FRAME_NUMBER[0]` is the *Last Significant Bit* (**LSB**) **Value After Reset:** 0x0

14.1.2.2.10. Index Register (`INDEX`)

The `INDEX` register is at offset 0xE.

Each Tx endpoint and each Rx endpoint have their own set of control/status registers located between 0x100...0x1FF. In addition one set of Tx control/status and one set of Rx control/status registers appear at 0x10...0x19.

`INDEX` is a 4-bit register that determines which endpoint control/status registers are accessed.

Before accessing control/status registers of an endpoint at 0x10...0x19, the endpoint number should be written to the `INDEX` register to ensure that the correct control/status registers appear in the memory map.

The following table shows the register bit assignments.

Table 14-24 Fields for register: `INDEX`

Bits	Name	Software Access	USB Access	Description
[4:0]	`SELECTED_EP`	R/W	R	Selected Endpoint Here `SELECTED_EP[4]` is the *Most Significant Bit* (**MSB**) and `SELECTED_EP[0]` is the *Last Significant Bit* (**LSB**) **Value After Reset:** 0x0

14.1.2.2.11. USB Test Modes Register (`TESTMODE`)

The `TESTMODE` register is at offset 0xF.

`TESTMODE` is an 8-bit register that is primarily used to put the USB 2.0 Controller into one of the four test modes for High-speed operation described in the USB 2.0 specification — in response to a `SET FEATURE: TESTMODE` command.It is not used in normal operation.

Only one of bits `[6:0]` should be set at any time.

The following table shows the register bit assignments.

Table 14-25 Fields for register: `TESTMODE`

Bits	Name	Software Access	USB Access	Description
[7]	`FORCE_HOST`	R/W	R	The software sets this bit to instruct the core to enter Host mode when the `SESSION` bit is set, regardless of whether it is connected to any peripheral. The state of the CID input, `hostdiscon` and `linestate[1:0]` signals are ignored. The core will then remain in Host mode until the `SESSION` bit is cleared, even if a device is disconnected, and if the `FORCE_HOST` bit remains set, will re-enter Host mode the next time the `SESSION` bit is set While in this mode, the status of the `HOSTDISCON` signal from the PHY may be read from bit `[7]` of the `DEVCTL`. The operating speed is determined from the `FORCE_HS` and `FORCE_FS` bits as follows: • `FORCE_HS=0x0` and `FORCE_FS=0x0`: Low Speed • `FORCE_HS=0x0` and `FORCE_FS=0x1`: Full Speed

Table 14-25 Fields for register: `TESTMODE` (continued)

Bits	Name	Software Access	USB Access	Description
				• `FORCE_HS=0x1` and `FORCE_FS=0x0`: High Speed • `FORCE_HS=0x1` and `FORCE_FS=0x1`: Undefined **Value After Reset:** 0x0
[6]	`FIFO_ACCESS`	WS	R	The software sets this bit to transfer the packet in the Endpoint 0 Tx FIFO to the Endpoint 0 Rx FIFO. It is cleared automatically. **Value After Reset:** 0x0
[5]	`FORCE_FS`	R/W	R	The software sets this bit either in conjunction with bit [7] above or to force the USB 2.0 Controller into Full-speed mode when it receives a USB reset. **Value After Reset:** 0x0
[4]	`FORCE_HS`	R/W	R	The software sets this bit either in conjunction with bit [7] above or to force the USB 2.0 Controller into High-speed mode when it receives a USB reset. **Value After Reset:** 0x0
[3]	`TEST_PACKET`	R/W	R	High-speed mode: the software sets this bit to enter the `TEST_PACKET` test mode. In this mode, the USB 2.0 Controller repetitively transmits on the bus a 53-byte test packet, the form of which is defined in the ***Universal Serial Bus Specification Revision 2.0, Section 7.1.20***. The test packet has a fixed format and must be loaded into the Endpoint 0 FIFO before the test mode is entered. **Value After Reset:** 0x0
[2]	`TEST_K`	R/W	R	High-speed mode: the software sets this bit to enter the `TEST_K` test mode. In this mode, the USB 2.0 Controller transmits a continuous `K` on the bus. **Value After Reset:** 0x0
[1]	`TEST_J`	R/W	R	High-speed mode: the software sets this bit to enter the `TEST_J` test mode. In this mode, the USB 2.0 Controller transmits a continuous `J` on the bus. **Value After Reset:** 0x0
[0]	`TEST_SE0_NAK`	R/W	R	High-speed mode: the software sets this bit to enter the `TEST_SE0_NAK` test mode. In this mode, the USB 2.0 Controller remains in High-speed mode but responds to any valid IN token with a NAK.

Table 14-25 Fields for register: `TESTMODE` (continued)

Bits	Name	Software Access	USB Access	Description
				Value After Reset: 0x0

14.1.2.2.12. Maximum Packet Size for Tx Endpoint #n (`TXMAXP`)

The `TXMAXP` register is at offset 0x10.

Exists: `INDEX.SELECTED_EP` != 0x0.

The `TXMAXP` register defines the maximum amount of data that can be transferred through the selected Tx endpoint in a single operation. There is a `TXMAXP` register for each Tx endpoint (except Endpoint 0)

Bits [10:0] define (in bytes) the maximum payload transmitted in a single transaction. The value set can be up to 1024 bytes but is subject to the constraints placed by the USB Specification on packet sizes for Bulk, Interrupt and Isochronous transfers in Full-speed and High-speed operations.

Where the option of High-bandwidth Isochronous/Interrupt endpoints or of packet splitting on Bulk endpoints has been taken when the core is configured, the register includes either 2 or 5 further bits that define a multiplier m which is equal to one more than the value recorded.

In the case of Bulk endpoints with the packet splitting option enabled, the multiplier "m" can be up to 32 and defines the maximum number of ‘USB’ packets (i.e. packets for transmission over the USB) of the specified payload into which a single data packet placed in the FIFO should be split, prior to transfer (if the packet splitting option is not enabled, bits [15:13] are not implemented and bits [12:11] (if included) are ignored).

The data packet is required to be an exact multiple of the payload specified by bits [10:0], which is itself required to be either 8, 16, 32, 64 or (in the case of High Speed transfers) 512 bytes.

For Isochronous/Interrupts endpoints operating in High-Speed mode and with the High-bandwidth option enabled, "m" may only be either 2 or 3 (corresponding to bit [11] set or bit [12] set, respectively) and it specifies the maximum number of such transactions that can take place in a single microframe.

If either bit [11] or bit [12] is non-zero, the USB 2.0 Controller will automatically split any data packet written to the FIFO into up to 2 or 3 ‘USB’ packets, each containing the specified payload (or less). The maximum payload for each transaction is 1024 bytes, so this allows up to 3072 bytes to be transmitted in each microframe (for Isochronous transfers in Full-speed mode or if High-bandwidth is not enabled, bits [11] and [12] are ignored).

The value written to bits [10:0] (multiplied by "m" in the case of high-bandwidth Isochronous transfers) must match the value given in the `wMaxPacketSize` field of the Standard Endpoint Descriptor for the associated endpoint (see ***Universal Serial Bus Specification, Revision 2.0, Chapter 9***). A mismatch could cause unexpected results.

The total amount of data represented by the value written to this register (specified payload x *m*) must not exceed the FIFO size for the Tx endpoint, and should not exceed half the FIFO size if double-buffering is required.

If this register is changed after packets have been sent from the endpoint, the Tx endpoint FIFO should be completely flushed (using the `FLUSH_FIFO` bit in `TXCSRL`) after writing the new value to this register.

`TXMAXP` must be set to an even number of bytes for proper interrupt generation in DMA Mode 1.

The following table shows the register bit assignments.

Table 14-26 Fields for register: `TXMAXP`

Bits	Name	Software Access	USB Access	Description
[15:11]	`MULTIPLER_VAL`	RW	R	This bit field defines the value of the multiplier m which is equal to one more than the value recorded. **Value After Reset:** 0x0
[10:0]	`MAX_PAYLOAD`	RW	R	This bit field defines (in bytes) the maximum payload transmitted in a single transaction. The value set can be up to 1024 bytes but is subject to the constraints placed by the USB Specification on packet sizes for Bulk, Interrupt and Isochronous transfers in Full-speed and High-speed operations. **Value After Reset:** 0x0

14.1.2.2.13. Endpoint 0 Control and Status Register 0 (`CSR0L`)

The `CSR0L` register is at offset 0x12.

Exists: `INDEX.SELECTED_EP` = 0x0.

`CSR0L` is an 8-bit register that provides control and status bits for Endpoint 0.

The interpretation of the register depends on whether the USB 2.0 Controller is acting as a peripheral or as a host. Users should also be aware that the value returned when the register is read reflects the status attained e.g. as a result of writing to the register.

The following table shows the register bit assignments for USB Controller in **Peripheral Mode**.

Table 14-27 Fields for register: `CSR0L` (`DEVCTL.HOST_MODE = 0x0`)

Bits	Name	Software Access	USB Access	Description
[7]	`SERVICED_SETUP_END`	WS	R	The software writes a 1 to this bit to clear the `SETUP_END` bit. It is cleared automatically. **Value After Reset:** 0x0
[6]	`SERVICED_RXPKTRDY`	WS	R	The software writes a 1 to this bit to clear the `RXPKTRDY` bit. It is cleared automatically. **Value After Reset:** 0x0
[5]	`SEND_STALL`	WS	R	The software writes a 1 to this bit to terminate the current transaction. The STALL handshake will be transmitted and then this bit will be cleared automatically. **Value After Reset:** 0x0
[4]	`SETUP_END`	R	WS	This bit will be set when a control transaction ends before the `DATA_END` bit has been set. An interrupt will be generated and the FIFO flushed at this time. The bit is cleared by the

Table 14-27 Fields for register: CSR0L (DEVCTL.HOST_MODE = 0x0) (continued)

Bits	Name	Software Access	USB Access	Description
				software writing a 1 to the SERVICED_SETUP_END bit. **Value After Reset:** 0x0
[3]	DATA_END	WS	R	The software sets this bit: 1. When setting TXPKTRDY for the last data packet. 2. When clearing RXPKTRDY after unloading the last data packet. 3. When setting TXPKTRDY for a zero length data packet. It is cleared automatically. **Value After Reset:** 0x0
[2]	SENT_STALL	R/WC	WS	This bit is set when a STALL handshake is transmitted. The software should clear this bit. **Value After Reset:** 0x0
[1]	TXPKTRDY	R/WS	R	The software sets this bit after loading a data packet into the FIFO. It is cleared automatically when a data packet has been transmitted. An interrupt is also generated at this point (if enabled). **Value After Reset:** 0x0
[0]	RXPKTRDY	R	WS	This bit is set when a data packet has been received. An interrupt is generated when this bit is set. The software clears this bit by setting the SERVICED_RXPKTRDY bit. **Value After Reset:** 0x0

The following table shows the register bit assignments for USB Controller in **Host Mode**.

Table 14-28 Fields for register: CSR0L (DEVCTL.HOST_MODE = 0x1)

Bits	Name	Software Access	USB Access	Description
[7]	NAK_TIMEOUT	R/WC	WS	This bit will be set when Endpoint 0 is halted following the receipt of NAK responses for longer than the time set by the NAKLIMIT0. The software should clear this bit to allow the endpoint to continue. **Value After Reset:** 0x0
[6]	STATUSPKT	RW	R	The software sets this bit at the same time as the TXPKTRDY or REQPKT bit is set, to perform a status stage transaction. Setting this bit ensures that the data toggle is set to 1 so that a DATA1 packet is used for the Status Stage transaction. **Value After Reset:** 0x0

Table 14-28 Fields for register: `CSR0L` (`DEVCTL.HOST_MODE = 0x1`) (continued)

Bits	Name	Software Access	USB Access	Description
[5]	REQPKT	RW	RW	The software sets this bit to request an IN transaction. It is cleared when RXPKTRDY is set. **Value After Reset:** 0x0
[4]	ERROR	R/WC	WS	This bit will be set when three attempts have been made to perform a transaction with no response from the peripheral. The software should clear this bit. An interrupt is generated when this bit is set. **Value After Reset:** 0x0
[3]	SETUPPKT	R/WC	R	The software sets this bit, at the same time as the TXPKTRDY bit is set, to send a SETUP token instead of an OUT token for the transaction. Setting this bit also clears the DATA_TOGGLE. **Value After Reset:** 0x0
[2]	RXSTALL	R/WC	WS	This bit is set when a STALL handshake is received. The software should clear this bit. **Value After Reset:** 0x0
[1]	TXPKTRDY	R/WS	WC	The software sets this bit after loading a data packet into the FIFO. It is cleared automatically when a data packet has been transmitted. An interrupt is also generated at this point (if enabled). **Value After Reset:** 0x0
[0]	RXPKTRDY	R/WC	R/W	This bit is set when a data packet has been received. An interrupt is generated (if enabled) when this bit is set. The software should clear this bit when the packet has been read from the FIFO. **Value After Reset:** 0x0

14.1.2.2.14. Tx Endpoint #n Control and Status Register 0 (`TXCSRL`)

The `TXCSRL` register is at offset 0x12.

Exists: `INDEX.SELECTED_EP` != 0x0.

`TXCSRL` is an 8-bit register that provides control and status bits for transfers through the currently-selected Tx endpoint. There is a `TXCSRL` register for each configured Tx endpoint (not including Endpoint 0).

The interpretation of the register depends on whether the USB 2.0 Controller is acting as a peripheral or as a host. Users should also be aware that the value returned when the register is read reflects the status attained e.g. as a result of writing to the register.

The following table shows the register bit assignments for USB Controller in **Peripheral Mode**.

Table 14-29 Fields for register: `TXCSRL` (`DEVCTL.HOST_MODE = 0x0`)

Bits	Name	Software Access	USB Access	Description
[7]	`INCOMP_TX`	R/WC	WS	When the endpoint is being used for high-bandwidth Isochronous, this bit is set to indicate where a large packet has been split into 2 or 3 packets for transmission but insufficient IN tokens have been received to send all the parts. In anything other than isochronous transfers, this bit will always return 0. **Value After Reset:** 0x0
[6]	`CLR_DATA_TOG`	WS	R/WC	The software writes a 1 to this bit to reset the endpoint data toggle to 0. **Value After Reset:** 0x0
[5]	`SENT_STALL`	R/WC	WS	This bit is set when a STALL handshake is transmitted. The FIFO is flushed and the `TXPKTRDY` bit is cleared (see below). The software should clear this bit. **Value After Reset:** 0x0
[4]	`SEND_STALL`	RW	R	The software writes a 1 to this bit to issue a STALL handshake to an IN token. The software clears this bit to terminate the stall condition. This bit has no effect where the endpoint is being used for Isochronous tranfers. **Value After Reset:** 0x0
[3]	`FLUSH_FIFO`	WS	R	The software writes a 1 to this bit to flush the latest packet from the endpoint Tx FIFO. The FIFO pointer is reset, the `TXPKTRDY` bit (below) is cleared and an interrupt is generated. May be set simultaneously with `TXPKTRDY` to abort the packet that is currently being loaded into the FIFO. `FLUSH_FIFO` should only be used when `TXPKTRDY` is set. At other times, it may cause data to be corrupted. Also note that, if the FIFO is double-buffered, `FLUSH_FIFO` may need to be set twice to completely clear the FIFO. **Value After Reset:** 0x0
[2]	`UNDER_RUN`	R/WC	WS	The USB sets this bit if an IN token is received when `TXPKTRDY` is not set. The software should clear this bit. **Value After Reset:** 0x0
[1]	`FIFO_NOT_EMPTY`	R/WC	WS	The USB sets this bit when there is at least 1 packet in the Tx FIFO. **Value After Reset:** 0x0

Table 14-29 Fields for register: `TXCSRL` (`DEVCTL.HOST_MODE = 0x0`) (continued)

Bits	Name	Software Access	USB Access	Description
[0]	`TXPKTRDY`	R/WS	WC	The software sets this bit after loading a data packet into the FIFO. It is cleared automatically when a data packet has been transmitted. An interrupt is also generated at this point (if enabled). `TXPKTRDY` is also automatically cleared prior to loading a second packet into a double-buffered FIFO. **Value After Reset:** 0x0

The following table shows the register bit assignments for USB Controller in **Host Mode**.

Table 14-30 Fields for register: `TXCSRL` (`DEVCTL.HOST_MODE = 0x1`)

Bits	Name	Software Access	USB Access	Description
[7]	`NAK_TIMEOUT_OR_INCOMP_TX`	R/WC	WS	Bulk endpoints only: This bit will be set when the Tx endpoint is halted following the receipt of NAK responses for longer than the time set as the NAK Limit by the `TXINTERVAL`. The CPU should clear this bit to allow the endpoint to continue. High-bandwidth Interrupt endpoints only: This bit will be set if no response is received from the device to which the packet is being sent. **Value After Reset:** 0x0
[6]	`CLR_DATA_TOG`	WS	R/WC	The software writes a 1 to this bit to reset the endpoint data toggle to 0. **Value After Reset:** 0x0
[5]	`RX_STALL`	R/WC	WS	This bit is set when a STALL handshake is received. When this bit is set, any DMA request that is in progress is stopped, the FIFO is completely flushed and the `TXPKTRDY` bit is cleared (see below). The software should clear this bit. **Value After Reset:** 0x0
[4]	`SETUP_PKT`	RW	R	The software sets this bit, at the same time as the `TXPKTRDY` bit is set, to send a SETUP token instead of an OUT token for the transaction. Setting this bit also clears the `DATA_TOGGLE`. **Value After Reset:** 0x0
[3]	`FLUSH_FIFO`	WS	R	The software writes a 1 to this bit to flush the latest packet from the endpoint Tx FIFO. The FIFO pointer is reset, the `TXPKTRDY` bit (below) is cleared and an interrupt is generated. May be set simultaneously

Table 14-30 Fields for register: `TXCSRL` (`DEVCTL.HOST_MODE = 0x1`) (continued)

Bits	Name	Software Access	USB Access	Description
				with `TXPKTRDY` to abort the packet that is currently being loaded into the FIFO. `FLUSH_FIFO` should only be used when `TXPKTRDY` is set. At other times, it may cause data to be corrupted. Also note that, if the FIFO is double-buffered, `FLUSH_FIFO` may need to be set twice to completely clear the FIFO. **Value After Reset:** 0x0
[2]	`ERROR`	R/WC	WS	The USB sets this bit when 3 attempts have been made to send a packet and no handshake packet has been received. When the bit is set, an interrupt is generated, `TXPKTRDY` is cleared and the FIFO is completely flushed. The software should clear this bit. Valid only when the endpoint is operating in Bulk or Interrupt mode. **Value After Reset:** 0x0
[1]	`FIFO_NOT_EMPTY`	R/WC	WS	The USB sets this bit when there is at least 1 packet in the Tx FIFO. **Value After Reset:** 0x0
[0]	`TXPKTRDY`	R/WS	WC	The software sets this bit after loading a data packet into the FIFO. It is cleared automatically when a data packet has been transmitted. An interrupt is also generated at this point (if enabled). `TXPKTRDY` is also automatically cleared prior to loading a second packet into a double-buffered FIFO. **Value After Reset:** 0x0

14.1.2.2.15. Endpoint 0 Control and Status Register 1 (`CSR0H`)

The `CSR0H` register is at offset 0x13.

Exists: `INDEX.SELECTED_EP` = 0x0.

`CSR0H` is an 8-bit register that provides control and status bits for Endpoint 0.

The interpretation of the register depends on whether the USB 2.0 Controller is acting as a peripheral or as a host. Users should also be aware that the value returned when the register is read reflects the status attained e.g. as a result of writing to the register.

The following table shows the register bit assignments for USB Controller in **Peripheral Mode**.

Table 14-31 Fields for register: CSR0H (DEVCTL.HOST_MODE=0x0)

Bits	Name	Software Access	USB Access	Description
[7:1]				Reserved
[0]	FLUSH_FIFO	WS	R	The software writes a 1 to this bit to flush the next packet to be transmitted/read from the Endpoint 0 FIFO. The FIFO pointer is reset and the CSR0L.TXPKTRDY/CSR0L.RXPKTRDY bit (below) is cleared. FLUSH_FIFO should only be used when TXPKTRDY/RXPKTRDY is set. At other times, it may cause data to be corrupted. **Value After Reset:** 0x0

The following table shows the register bit assignments for USB Controller in **Host Mode**.

Table 14-32 Fields for register: CSR0H (DEVCTL>.HOST_MODE=0x1)

Bits	Name	Software Access	USB Access	Description
[7:4]				Reserved
[3]	DIS_PING	R/W	R	The software writes a 1 to this bit to instruct the core not to issue PING tokens in data and status phases of a high-speed Control transfer (for use with devices that do not respond to PING). **Value After Reset:** 0x0
[2]	DATA_TOGGLE_WRITE_ENABLE	WS	R	The software writes a 1 to this bit to enable the current state of the Endpoint 0 data toggle to be written (see DATA_TOGGLE bit, below). This bit is automatically cleared once the new value is written. **Value After Reset:** 0x0
[1]	DATA_TOGGLE	R/W	R/W	When read, this bit indicates the current state of the Endpoint 0 data toggle. If DATA_TOGGLE_WRITE_ENABLE is high, this bit may be written with the required setting of the data toggle. If DATA_TOGGLE_WRITE_ENABLE is low, any value written to this bit is ignored. **Value After Reset:** 0x0
[0]	FLUSH_FIFO	WS	R	The software writes a 1 to this bit to flush the next packet to be transmitted/read from the Endpoint 0 FIFO. The FIFO pointer is reset and the CSR0L.TXPKTRDY/CSR0L.RXPKTRDY bit (below) is cleared.

Table 14-32 Fields for register: CSR0H (DEVCTL>.HOST_MODE=0x1) (continued)

Bits	Name	Software Access	USB Access	Description
				FLUSH_FIFO should only be used when TXPKTRDY/RXPKTRDY is set. At other times, it may cause data to be corrupted. **Value After Reset:** 0x0

14.1.2.2.16. Tx Endpoint #n Control and Status Register 1 (TXCSRH)

The TXCSRH register is at offset 0x13.

Exists: INDEX.SELECTED_EP != 0x0.

TXCSRH is an 8-bit register that provides additional control for transfers through the currently-selected Tx endpoint. There is a TXCSRH register for each configured Tx endpoint (not including Endpoint 0).

The interpretation of the register depends on whether the USB 2.0 Controller is acting as a peripheral or as a host. Users should also be aware that the value returned when the register is read reflects the status attained e.g. as a result of writing to the register.

The following table shows the register bit assignments for USB Controller in **Peripheral Mode**.

Table 14-33 Fields for register: TXCSRH (DEVCTL.HOST_MODE= 0x0)

Bits	Name	Software Access	USB Access	Description
[7]	AUTOSET	R/W	R	If the software sets this bit, TXPKTRDY will be automatically set when data of the maximum packet size (value in TXMAXP) is loaded into the Tx FIFO. If a packet of less than the maximum packet size is loaded, then TXPKTRDY will have to be set manually. Should not be set for either high-bandwidth Isochronous endpoints or high-bandwidth Interrupt endpoints. **Value After Reset:** 0x0
[6]	ISO	R/W	R	The software sets this bit to enable the Tx endpoint for Isochronous transfers, and clears it to enable the Tx endpoint for Bulk or Interrupt transfers. This bit only has any effect in Peripheral mode. In Host mode, it always returns zero. **Value After Reset:** 0x0
[5]	MODE	R/W	R	The software sets this bit to enable the endpoint direction as Tx, and clears the bit to enable it as Rx.

Table 14-33 Fields for register: TXCSRH (DEVCTL.HOST_MODE= 0x0) (continued)

Bits	Name	Software Access	USB Access	Description
				This bit only has any effect where the same endpoint FIFO is used for both Tx and Rx transactions. **Value After Reset:** 0x0
[4]	DMAREQENAB	R/W	R	The software sets this bit to enable the DMA request for the Tx endpoint. **Value After Reset:** 0x0
[3]	FRCDATATOG	R/W	R	The software sets this bit to force the endpoint data toggle to switch and the data packet to be cleared from the FIFO, regardless of whether an ACK was received. This can be used by Interrupt Tx endpoints that are used to communicate rate feedback for Isochronous endpoints. **Value After Reset:** 0x0
[2]	DMAREQMODE	R/W	R	Valid values: **0:** DMA Request Mode 0 **1:** DMA Request Mode 1 This bit must not be cleared either before or in the same cycle as the above DMAREQENAB bit is cleared. **Value After Reset:** 0x0
[1:0]				Reserved

The following table shows the register bit assignments for USB Controller in **Host Mode**.

Table 14-34 Fields for register: TXCSRH (DEVCTL.HOST_MODE= 0x1)

Bits	Name	Software Access	USB Access	Description
[7]	AUTOSET	R/W	R	If the software sets this bit, TXPKTRDY will be automatically set when data of the maximum packet size (value in TXMAXP register) is loaded into the Tx FIFO. If a packet of less than the maximum packet size is loaded, then TXPKTRDY will have to be set manually. Should not be set for either high-bandwidth Isochronous endpoints or high-bandwidth Interrupt endpoints. **Value After Reset:** 0x0
[6]				Reserved

Table 14-34 Fields for register: `TXCSRH` (`DEVCTL.HOST_MODE= 0x1`) (continued)

Bits	Name	Software Access	USB Access	Description
[5]	`MODE`	R/W	R	The software sets this bit to enable the endpoint direction as Tx, and clears it to enable the endpoint direction as Rx. This bit only has any effect where the seme endpoint FIFO is used for both Tx and Rx transactions. **Value After Reset:** 0x0
[4]	`DMAREQENAB`	R/W	R	The software sets this bit to enable the DMA request for the Tx endpoint. **Value After Reset:** 0x0
[3]	`FRCDATATOG`	R/W	R	The software sets this bit to force the endpoint data toggle to switch and the data packet to be cleared from the FIFO, regardless of whether an ACK was received. This can be used by Interrupt Tx endpoints that are used to communicate rate feedback for Isochronous endpoints. **Value After Reset:** 0x0
[2]	`DMAREQMODE`	R/W	R	Valid values: **0:** DMA Request Mode 0 **1:** DMA Request Mode 1 This bit must not be cleared either before or in the same cycle as the above `DMAREQENAB` bit is cleared. **Value After Reset:** 0x0
[1]	`DATA_TOGGLE_WRITE_ENABLE`	WS	R	Valid values: **0:** This bit is automatically cleared once the new value is written. **1:** The software writes a 1 to this bit to enable the current state of the Tx Endpoint data toggle to be written (see `DATA_TOGGLE` bit, below). **Value After Reset:** 0x0
[0]	`DATA_TOGGLE`	R/W	R/W	When read, this bit indicates the current state of the Tx Endpoint data toggle. If `DATA_TOGGLE_WRITE_ENABLE` is high, this bit may be written with the required setting of the data toggle. If `DATA_TOGGLE_WRITE_ENABLE` is low, any value written to this bit is ignored. **Value After Reset:** 0x0

14.1.2.2.17. Maximum Packet Size for Rx Endpoint #n (RXMAXP)

The RXMAXP register is at offset 0x14.

Exists: INDEX.SELECTED_EP != 0x0.

The RXMAXP register defines the maximum amount of data that can be transferred through the selected Rx endpoint in a single operation. There is a RXMAXP register for each Rx endpoint (except Endpoint 0).

Bits [10:0] define (in bytes) the maximum payload transmitted in a single transaction. The value set can be up to 1024 bytes but is subject to the constraints placed by the USB Specification on packet sizes for Bulk, Interrupt and Isochronous transfers in Full- speed and High-speed operations.

Where the option of High-bandwidth Isochronous/Interrupt endpoints or of packet splitting on Bulk endpoints has been taken when the core is configured, the register includes either 2 or 5 further bits that define a multiplier m which is equal to one more than the value recorded.

For Bulk endpoints with the packet splitting option enabled, the multiplier "m" can be up to 32 and defines the number of USB packets of the specified payload which are to be amalgamated into a single data packet within the FIFO (if the packet splitting option is not enabled, bits [15:13] are not implemented and bits [12:11] (if included) are ignored).

For Isochronous/Interrupt endpoints operating in High-Speed mode and with the High-bandwidth option enabled, "m" may only be either 2 or 3 (corresponding to bit [11] set or bit [12] set, respectively) and it specifies the maximum number of such transactions that can take place in a single microframe. If either bit [11] or bit [12] is non-zero, the USB 2.0 Controller will automatically combine the separate USB packets received in any microframe into a single packet within the Rx FIFO. The maximum payload for each transaction is 1024 bytes, so this allows up to 3072 bytes to be received in each microframe (for Isochronous transfers in Full-speed mode or if High-bandwidth is not enabled, bits [11] and [12] are ignored).

The value written to bits [10:0] (multiplied by m in the case of high-bandwidth Isochronous transfers) must match the value given in the wMaxPacketSize field of the Standard Endpoint Descriptor for the associated endpoint (see ***Universal Serial Bus Specification, Revision 2.0, Chapter 9***). A mismatch could cause unexpected results.

The total amount of data represented by the value written to this register (specified payload x *m*) must not exceed the FIFO size for the Rx endpoint, and should not exceed half the FIFO size if double-buffering is required.

RXMAXP must be set to an even number of bytes for proper interrupt generation in DMA Mode 1.

The following table shows the register bit assignments.

Table 14-35 Fields for register: RXMAXP

Bits	Name	Software Access	USB Access	Description
[15:11]	MULTIPLER_VAL	RW	R	This bit field defines the value of the multiplier m which is equal to one more than the value recorded. **Value After Reset:** 0x0
[10:0]	MAX_PAYLOAD	RW	R	This bit field defines (in bytes) the maximum payload transmitted in a single transaction.

Table 14-35 Fields for register: RXMAXP (continued)

Bits	Name	Software Access	USB Access	Description
				The value set can be up to 1024 bytes but is subject to the constraints placed by the ***Universal Serial Bus Specification, Revision 2.0*** on packet sizes for Bulk, Interrupt and Isochronous transfers in Full-speed and High-speed operations. **Value After Reset:** 0x0

14.1.2.2.18. Rx Endpoint #n Control and Status Register 0 (RXCSRL)

The RXCSRL register is at offset 0x16.

Exists: INDEX.SELECTED_EP != 0x0.

RXCSRL is an 8-bit register that provides control and status bits for transfers through the currently-selected Rx endpoint. There is a RXCSRL register for each configured Rx endpoint (not including Endpoint 0).

The interpretation of the register depends on whether the USB 2.0 Controller is acting as a peripheral or as a host. Users should also be aware that the value returned when the register is read reflects the status attained e.g. as a result of writing to the register.

The following table shows the register bit assignments for USB Controller in **Peripheral Mode**.

Table 14-36 Fields for register: RXCSRL (DEVCTL.HOST_MODE= 0x0)

Bits	Name	Software Access	USB Access	Description
[7]	CLRDATATOG	WS	R/WC	The software writes a 1 to this bit to reset the endpoint data toggle to 0. **Value After Reset:** 0x0
[6]	SENTSTALL	R/WC	WS	This bit is set when a STALL handshake is transmitted. The software should clear this bit. **Value After Reset:** 0x0
[5]	SENDSTALL	R/W	R	The software writes a 1 to this bit to issue a STALL handshake. The software clears this bit to terminate the stall condition. This bit has no effect where the endpoint is being usedfor Isochronous transfers. **Value After Reset:** 0x0
[4]	FLUSHFIFO	WS	R	The software writes a 1 to this bit to flush the next packet to be read from the endpoint Rx FIFO. The FIFO pointer is reset and the RXPKTRDY bit (below) is cleared. FLUSHFIFO should only be used when RXPKTRDY is set. At other times, it may cause data to be corrupted. Also note that, if the FIFO is double-buffered,

Table 14-36 Fields for register: RXCSRL (DEVCTL.HOST_MODE= 0x0) (continued)

Bits	Name	Software Access	USB Access	Description
				FLUSHFIFO may need to be set twice to completely clear the FIFO. **Value After Reset:** 0x0
[3]	DATAERROR	R	WS	This bit is set when RXPKTRDY is set if the data packet has a CRC or bit-stuff error. It is cleared when RXPKTRDY is cleared. This bit is only valid when the endpoint is operating in ISO mode. In Bulk mode, it always returns zero. **Value After Reset:** 0x0
[2]	OVERRUN	R/WC	WS	This bit is set if an OUT packet cannot be loaded into the Rx FIFO. The software should clear this bit. This bit is only valid when the endpoint is operating in ISO mode. In Bulk mode, it always returns zero. **Value After Reset:** 0x0
[1]	FIFOFULL	R	WS	This bit is set when no more packets can be loaded into the Rx FIFO. **Value After Reset:** 0x0
[0]	RXPKTRDY	R/WC	WS	This bit is set when a data packet has been received. The software should clear this bit when the packet has been unloaded from the Rx FIFO. An interrupt is generated when the bit is set. **Value After Reset:** 0x0

The following table shows the register bit assignments for USB Controller in **Host Mode**.

Table 14-37 Fields for register: RXCSRL (DEVCTL.HOST_MODE= 0x1)

Bits	Name	Software Access	USB Access	Description
[7]	CLRDATATOG	WS	R/WC	The software writes a 1 to this bit to reset the endpoint data toggle to 0. **Value After Reset:** 0x0
[6]	RXSTALL	R/WC	WS	When a STALL handshake is received, this bit is set and an interrupt is generated. The software should clear this bit. **Value After Reset:** 0x0
[5]	REQPKT	R/W	R/W	The software writes a 1 to this bit to request an IN transaction. It is cleared when RXPKTRDY is set. **Value After Reset:** 0x0

Table 14-37 Fields for register: RXCSRL (DEVCTL.HOST_MODE= 0x1) (continued)

Bits	Name	Software Access	USB Access	Description
[4]	FLUSHFIFO	WS	R	The software writes a 1 to this bit to flush the next packet to be read from the endpoint Rx FIFO. The FIFO pointer is reset and the RXPKTRDY bit (below) is cleared. FLUSHFIFO should only be used when RXPKTRDY is set. At other times, it may cause data to be corrupted. Also note that, if the FIFO is double-buffered, FLUSHFIFO may need to be set twice to completely clear the FIFO. **Value After Reset:** 0x0
[3]	DATAERROR_OR_NAK_TIMEOUT	R/WC	WS	When operating in ISO mode, this bit is set when RXPKTRDY is set if the data packet has a CRC or bit-stuff error and cleared when RXPKTRDY is cleared. In Bulk mode, this bit will be set when the Rx endpoint is halted following the receipt of NAK responses for longer than the time set as the NAK Limit by the RXINTERVAL. The software should clear this bit to allow the endpoint to continue. **Value After Reset:** 0x0
[2]	ERROR	R/WC	WS	The USB sets this bit when 3 attempts have been made to receive a packet and no data packet has been received. The software should clear this bit. An interrupt is generated when the bit is set. This bit is only valid when the Rx endpoint is operating in Bulk or Interrupt mode. In ISO mode, it always returns zero. **Value After Reset:** 0x0
[1]	FIFOFULL	R	WS	This bit is set when no more packets can be loaded into the Rx FIFO. **Value After Reset:** 0x0
[0]	RXPKTRDY	R/WC	WS	This bit is set when a data packet has been received. The software should clear this bit when the packet has been unloaded from the Rx FIFO. An interrupt is generated when the bit is set. **Value After Reset:** 0x0

14.1.2.2.19. Rx Endpoint #n Control and Status Register 1 (RXCSRH)

The RXCSRH register is at offset 0x17.

Exists: INDEX.SELECTED_EP != 0x0

RXCSRH is an 8-bit register that provides additional control and status bits for transfers through the currently-selected Rx endpoint. There is a RXCSRH register for each configured Rx endpoint (not including Endpoint 0).

The interpretation of the register depends on whether the USB 2.0 Controller is acting as a peripheral or as a host. Users should also be aware that the value returned when the register is read reflects the status attained e.g. as a result of writing to the register.

The following table shows the register bit assignments for USB Controller in **Peripheral Mode**.

Table 14-38 Fields for register: RXCSRH (DEVCTL.HOST_MODE= 0x0)

Bits	Name	Software Access	USB Access	Description
[7]	AUTOCLEAR	R/W	R	If the software sets this bit then the RXPKTRDY bit will be automatically cleared when a packet of RXMAXP bytes has been unloaded from the Rx FIFO. When packets of less than the maximum packet size are unloaded, RXPKTRDY will have to be cleared manually. When using a DMA to unload the Rx FIFO, data is read from the Rx FIFO in 4 byte chunks regardless of the RXMAXP. Therefore, the RXPKTRDY bit will be cleared as shown in **RXPKTRDY bit clearing** table below.  Should not be set for high-bandwidth Isochronous endpoints. **Value After Reset:** 0x0
[6]	ISO	R/W	R	The software sets this bit to enable the Rx endpoint for Isochronous transfers, and clears it to enable the Rx endpoint for Bulk/Interrupt transfers. **Value After Reset:** 0x0
[5]	DMAREQENAB	R/W	R	The software sets this bit to enable the DMA request for the Rx endpoint. **Value After Reset:** 0x0
[4]	DISNYET_OR_PID_ERROR	*Varies	*Varies	Bulk/Interrupt Transactions (DISNYET): The software sets this bit to disable the sending of NYET handshakes. When set, all successfully received Rx packets are ACK'd including at the point at which the FIFO becomes full.  This bit only has any effect in High-speed mode, in which mode it should be set for all Interrupt endpoints. The access attributes of this field are as follows:

Table 14-38 Fields for register: RXCSRH (DEVCTL.HOST_MODE= 0x0) (continued)

Bits	Name	Software Access	USB Access	Description
				• Software Access: R/W • USB Access: R ISO Transactions (PID_ERROR): The core sets this bit to indicate a PID error in the received packet. The access attributes of this field are as follows: • Software Access: R • USB Access: R/W **Value After Reset:** 0x0
[3]	DMAREQMODE	R/W	R	Valid values: **0:** DMA Request Mode 0 **1:** DMA Request Mode 1 **Value After Reset:** 0x0
[2:1]				Reserved
[0]	INCOMPRX	R/WC	WS	This bit is set in a high-bandwidth Isochronous/ Interrupt transfer if the packet in the Rx FIFO is incomplete because parts of the data were not received. It is cleared when RXPKTRDY is cleared. In anything other than Isochronous transfer, this bit will always return 0. **Value After Reset:** 0x0

The following table shows the register bit assignments for USB Controller in **Host Mode**.

Table 14-39 Fields for register: RXCSRH (DEVCTL.HOST_MODE= 0x1)

Bits	Name	Software Access	USB Access	Description
[7]	AUTOCLEAR	R/W	R	If the software sets this bit then the RXPKTRDY bit will be automatically cleared when a packet of RXMAXP bytes has been unloaded from the Rx FIFO. When packets of less than the maximum packet size are unloaded, RXPKTRDY will have to be cleared manually. When using a DMA to unload the Rx FIFO, data is read from the Rx FIFO in 4 byte chunks regardless of the RXMAXP. Therefore, the RXPKTRDY bit will be cleared as shown in **RXPKTRDY bit clearing** table below. Should not be set for high-bandwidth Isochronous endpoints.

Table 14-39 Fields for register: RXCSRH (DEVCTL.HOST_MODE= 0x1) (continued)

Bits	Name	Software Access	USB Access	Description
				Value After Reset: 0x0
[6]	AUTOREQ	R/W	R	If the software sets this bit, the REQPKT bit will be automatically set when the RXPKTRDY bit is cleared. This bit is automatically cleared when a short packet is received. **Value After Reset:** 0x0
[5]	DMAREQENAB	R/W	R	The software sets this bit to enable the DMA request for the Rx endpoint. **Value After Reset:** 0x0
[4]	PID_ERROR	R	R/W	ISO Transactions (PID Error): The core sets this bit to indicate a PID error in the received packet. The setting of this bit is ignored for Bulk/Interrupt Transactions. **Value After Reset:** 0x0
[3]	DMAREQMODE	R/W	R	Valid values: **0:** DMA Request Mode 0 **1:** DMA Request Mode 1 **Value After Reset:** 0x0
[2]	DATA_TOGGLE_WRITE_ENABLE	R	R	Valid values: **0:** This bit is automatically cleared once the new value is written. **1:** The software writes a 1 to this bit to enable the current state of the Endpoint 0 data toggle to be written (see DATA_TOGGLE bit, below). **Value After Reset:** 0x0
[1]	DATA_TOGGLE	R	R	When read, this bit indicates the current state of the Endpoint 0 data toggle. If DATA_TOGGLE_WRITE_ENABLE is high, this bit may be written with the required setting of the data toggle. If DATA_TOGGLE_WRITE_ENABLE is low, any value written to this bit is ignored. **Value After Reset:** 0x0
[0]	INCOMPRX	R/WC	WS	This bit will be set in a high-bandwidth Isochronous or Interrupt transfer if the packet received is incomplete. It will be cleared when RXPKTRDY is cleared.

Table 14-39 Fields for register: RXCSRH (DEVCTL.HOST_MODE= 0x1) (continued)

Bits	Name	Software Access	USB Access	Description
				If USB protocols are followed correctly, this bit should never be set. The bit becoming set indicates a failure of the associated Peripheral device to behave correctly (in anything other than Isochronous transfer, this bit will always return 0). **Value After Reset:** 0x0

Table 14-40 RXPKTRDY bit clearing

Remainder (RXMAXP/4)	Actual Bytes Read	Packet Sizes that will clear RXPKTRDY
0 (i.e. RXMAXP = 64 bytes)	RXMAXP	RXMAXP RXMAXP-1 RXMAXP-2 RXMAXP-3
3 (i.e. RXMAXP = 63 bytes)	RXMAXP+1	RXMAXP RXMAXP-1 RXMAXP-2
2 (i.e. RXMAXP = 62 bytes)	RXMAXP+2	RXMAXP RXMAXP-1
1 (i.e. RXMAXP = 61 bytes)	RXMAXP+3	RXMAXP

14.1.2.2.20. Endpoint 0 Data Bytes Counter Register (COUNT0)

The COUNT0 register is at offset 0x18.

Exists: INDEX.SELECTED_EP = 0x0.

COUNT0 is a 7-bit read-only register that indicates the number of received data bytes in the Endpoint 0 FIFO.

The following table shows the register bit assignments.

Table 14-41 Fields for register: COUNT0

Bits	Name	Software Access	USB Access	Description
[6:0]	COUNT	R	W	Number of received data bytes in the Endpoint 0 FIFO. The value returned changes as the contents of the FIFO change and is only valid while CSR0L.RXPKTRDY is set. **Value After Reset:** 0x0

14.1.2.2.21. Rx Endpoint #n Data Bytes Counter Register (RXCOUNT)

The RXCOUNT register is at offset 0x18.

Exists: INDEX.SELECTED_EP != 0x0.

RXCOUNT is a 14-bit read-only register that holds the number of data bytes in the packet currently in line to be read from the Rx FIFO.

The following table shows the register bit assignments.

Table 14-42 Fields for register: RXCOUNT

Bits	Name	Software Access	USB Access	Description
[13:0]	COUNT	R	W	Number of data bytes in the packet currently in line to be read from the Rx FIFO. If the packet was transmitted as multiple bulk packets, the number given will be for the combined packet. The value returned changes as the FIFO is unloaded and is only valid while RXCSRL.RXPKTRDY is set. **Value After Reset:** 0x0

14.1.2.2.22. Tx Endpoint #n Settings Register (TXTYPE)

The TXTYPE register is at offset 0x1A.

Exists: DEVCTL.HOST_MODE = 0x1 && INDEX.SELECTED_EP != 0x0.

TXTYPE is an 8-bit register that should be written with the endpoint number to be targeted by the endpoint, the transaction protocol to use for the currently-selected Tx endpoint, and its operating speed. There is a TXTYPE register for each configured Tx endpoint (except Endpoint 0, which has its own Type register at 0x1A).

The following table shows the register bit assignments.

Table 14-43 Fields for register: TXTYPE

Bits	Name	Software Access	USB Access	Description
[7:6]	SPEED	R/W	R	Operating speed of the target device when the core is configured with the multipoint option. Valid values: **0x0:** Unused If selected, the target will be assumed to be using the same connection speed as the core. **0x1:** High **0x2:** Full **0x3:** Low **Value After Reset:** 0x0
[5:4]	PROTOCOL	R/W	R	The software should set this to select the required protocol for the Tx endpoint. Valid values:

Table 14-43 Fields for register: TXTYPE (continued)

Bits	Name	Software Access	USB Access	Description
				0x0: Control **0x1:** Isochronous **0x2:** Bulk **0x3:** Interrupt **Value After Reset:** 0x0
[3:0]	TARGET_ENDPOINT_NUMBER	R/W	R	The software should set this value to the endpoint number contained in the Tx endpoint descriptor returned to the USB 2.0 Controller during device enumeration. **Value After Reset:** 0x0

14.1.2.2.23. Endpoint 0 Speed Register (TYPE0)

The TYPE0 register is at offset 0x1A.

Exists: DEVCTL.HOST_MODE = 0x1 && INDEX.SELECTED_EP = 0x0

The following table shows the register bit assignments.

Table 14-44 Fields for register: TYPE0

Bits	Name	Software Access	USB Access	Description
[7:6]	SPEED	R/W	R	Operating speed of the target device. **Values:** **0x0:** Unused If selected, the target will be assumed to be using the same connection speed as the core. **0x1:** High **0x2:** Full **0x3:** Low **Value After Reset:** 0x0
[5:0]				Reserved

14.1.2.2.24. Endpoint 0 NAK Limit Register (NAKLIMIT0)

The NAKLIMIT0 register is at offset 0x1B.

Exists: DEVCTL.HOST_MODE = 0x1 && INDEX.SELECTED_EP = 0x0

NAKLIMIT0 is a 5-bit register that sets the number of frames/microframes (High-Speed transfers) after which Endpoint 0 should timeout on receiving a stream of NAK responses. (Equivalent settings for other endpoints can be made through their TXINTERVAL and RXINTERVAL.)

The number of frames/microframes selected is $2^{(m-1)}$ (where *m* is the value set in the register, valid values 2 — 16). If the host receives NAK responses from the target for more frames than the number represented by the Limit set in this register, the endpoint will be halted.

A value of 0 or 1 disables the NAK timeout function.

The following table shows the register bit assignments.

Table 14-45 Fields for register: `NAKLIMIT0`

Bits	Name	Software Access	USB Access	Description
[4:0]	`LIMIT_VAL`	R/W	R	**Value After Reset:** 0x0

14.1.2.2.25. Tx Endpoint #n NAK Limit/Polling Interval Register (`TXINTERVAL`)

The `TXINTERVAL` register is at offset 0x1B.

Exists: `DEVCTL`.`HOST_MODE` = 0x1 && `INDEX`.`SELECTED_EP` != 0x0

`TXINTERVAL` is an 8-bit register that, for Interrupt and Isochronous transfers, defines the polling interval for the currently-selected Tx endpoint. For Bulk endpoints, this register sets the number of frames/microframes after which the endpoint should timeout on receiving a stream of NAK responses.

There is a `TXINTERVAL` register for each configured Tx endpoint (except Endpoint 0). In each case the value that is set defines a number of frames/microframes (High Speed transfers) and it is described in the table below.

The following table shows the register bit assignments.

Table 14-46 Fields for register: `TXINTERVAL`

Bits	Name	Software Access	USB Access	Description

[7:0]	`POLLING_INTERVAL_OR_NAK_LIMIT`	R/W	R	`POLLING_INTERVAL:`
				• For Low Speed or Full Speed Interrupt transfers polling interval is m frames. Here *m* corresponds to `POLLING_INTERVAL` field value and can be set from 1 to 255.
				• For High Speed Interrupt transfers polling interval is $2^{(m-1)}$ microframes. Here *m* corresponds to `POLLING_INTERVAL` field value and can be set from 1 to 16.
				• For Full Speed or High Speed Isochronous transfers polling interval is $2^{(m-1)}$ frames/microframes. Here *m* corresponds to `POLLING_INTERVAL` field value and can be set from 1 to 16.
				`NAK_LIMIT:`

Table 14-46 Fields for register: `TXINTERVAL` (continued)

Bits	Name	Software Access	USB Access	Description
				• For Full Speed or High Speed Bulk transfers NAK Limit is $2^{(m-1)}$ frames/microframes. Here *m* corresponds to `NAK_LIMIT` field value and can be set from 2 to 16. A value of 0 or 1 disables the NAK timeout function. **Value After Reset:** 0x0

14.1.2.2.26. Rx Endpoint #n Settings Register (`RXTYPE`)

The `RXTYPE` register is at offset 0x1C.

Exists: `DEVCTL`.`HOST_MODE` = 0x1 && `INDEX`.`SELECTED_EP` != 0x0.

`RXTYPE` is an 8-bit register that should be written with the endpoint number to be targeted by the endpoint, the transaction protocol to use for the currently-selected Rx endpoint, and its operating speed.

There is a `RXTYPE` register for each configured Rx endpoint (except Endpoint 0, which has its own Type register at 0x1A).

The following table shows the register bit assignments.

Table 14-47 Fields for register: `RXTYPE`

Bits	Name	Software Access	USB Access	Description
[7:6]	`SPEED`	R/W	R	Operating speed of the target device when the core is configured with the multipoint option. Valid values: **0x0:** Unused If selected, the target will be assumed to be using the same connection speed as the core. **0x1:** High **0x2:** Full **0x3:** Low **Value After Reset:** 0x0
[5:4]	`PROTOCOL`	R/W	R	The software should set this to select the required protocol for the Rx endpoint. Valid values: **0x0:** Control **0x1:** Isochronous **0x2:** Bulk **0x3:** Interrupt

Table 14-47 Fields for register: RXTYPE (continued)

Bits	Name	Software Access	USB Access	Description
				Value After Reset: 0x0
[3:0]	TARGET_ENDPOINT_NUMBER	R/W	R	The software should set this value to the endpoint number contained in the Rx endpoint descriptor returned to the USB 2.0 Controller during device enumeration. **Value After Reset:** 0x0

14.1.2.2.27. Rx Endpoint #n NAK Limit/Polling Interval Register (RXINTERVAL)

The RXINTERVAL register is at offset 0x1D.

Exists: DEVCTL.HOST_MODE = 0x1 && INDEX.SELECTED_EP != 0x0.

RXINTERVAL is an 8-bit register that, for Interrupt and Isochronous transfers, defines the polling interval for the currently-selected Rx endpoint. For Bulk endpoints, this register sets the number of frames/microframes after which the endpoint should timeout on receiving a stream of NAK responses.

There is a RXINTERVAL register for each configured Rx endpoint (except Endpoint 0). In each case the value that is set defines a number of frames/microframes (High Speed transfers) and it is described in the table below.

The following table shows the register bit assignments.

Table 14-48 Fields for register: RXINTERVAL

Bits	Name	Software Access	USB Access	Description
[7:0]	POLLING_INTERVAL_OR_NAK_LIMIT	R/W	R	POLLING_INTERVAL: • For Low Speed or Full Speed Interrupt transfers polling interval is m frames. Here m corresponds to POLLING_INTERVAL field value and can be set from 1 to 255 • For High Speed Interrupt transfers polling interval is $2^{(m-1)}$ microframes. Here m corresponds to POLLING_INTERVAL field value and can be set from 1 to 16. • For Full Speed or High Speed Isochronous transfers polling interval is $2^{(m-1)}$ frames/microframes. Here *m* corresponds to POLLING_INTERVAL field value and can be set from 1 to 16. NAK_LIMIT: • For Full Speed or High Speed Bulk transfers NAK Limit is $2^{(m-1)}$ frames/microframes. Here m corresponds to NAK_LIMIT field value and can be set from 2 to 16.

Table 14-48 Fields for register: `RXINTERVAL` (continued)

Bits	Name	Software Access	USB Access	Description
				A value of 0 or 1 disables the NAK timeout function. **Value After Reset:** 0x0

14.1.2.2.28. Configuration Data Register (`CONFIGDATA`)

The `CONFIGDATA` register is at offset 0x1F.

`CONFIGDATA` is an 8-bit Read-Only register that returns information about the selected core configuration.

The following table shows the register bit assignments.

Table 14-49 Fields for register: `CONFIGDATA`

Bits	Name	Software Access	USB Access	Description
[7]	`MPRXE`	R	R	When set to '1', automatic amalgamation of bulk packets is selected (see Bulk Transactions section). **Value After Reset:** 0x1
[6]	`MPTXE`	R	R	When set to '1', automatic splitting of bulk packets is selected (see Bulk Transactions section). **Value After Reset:** 0x1
[5]	`BIGENDIAN`	R	R	Always '0'. Indicates Little Endian ordering. **Value After Reset:** 0x0
[4]	`HBRXE`	R	R	When set to '1' indicates High-bandwidth Rx ISO Endpoint Support selected. **Value After Reset:** 0x1
[3]	`HBTXE`	R	R	When set to '1' indicates High-bandwidth Tx ISO Endpoint Support selected. **Value After Reset:** 0x1
[2]	`DYNFIFO_SIZING`	R	R	When set to '1' indicates Dynamic FIFO Sizing option selected. **Value After Reset:** 0x1
[1]	`SOFTCONE`	R	R	Always '1'. Indicates Soft Connect/Disconnect. **Value After Reset:** 0x1
[0]				Reserved

14.1.2.2.29. Endpoint #n FIFO (`FIFOn`)

The `FIFOn` register is at offset 0x20 + (**n***0x4), where **n** = 0...5.

This address range provides 6 addresses for software access to the FIFOs for each endpoint. Writing to these addresses loads data into the Tx FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Rx FIFO for the corresponding endpoint.

The address range is 0x20...0x37 and the FIFOs are located on 32-bit double-word boundaries (Endpoint 0 at 0x20, Endpoint 1 at 0x24 ... Endpoint 5 at 0x34).

- Transfers to and from FIFOs may be 8-bit, 16-bit or 32-bit as required, and any combination of access is allowed provided the data accessed is contiguous. However, all the transfers associated with one packet must be of the same width so that the data is consistently byte-, word- or double-word-aligned. The last transfer may however contain fewer bytes than the previous transfers in order to complete an odd-byte or odd-word transfer.
- Depending on the size of the FIFO and the expected maximum packet size, the FIFOs support either single-packet or doublepacket buffering. However, burst writing of multiple packets is not supported as flags need to be set after each packet is written.
- Following a STALL response or a Tx Strike Out error on Endpoint 1...5, the associated FIFO is completely flushed.

The following table shows the register bit assignments.

Table 14-50 Fields for register: `FIFOn`

Bits	Name	Software Access	USB Access	Description
[31:0]	DATA	R/W	R/W	FIFO**n** data. **Value After Reset:** 0x0

14.1.2.2.30. Device Control Register (`DEVCTL`)

The `DEVCTL` register is at offset 0x60.

`DEVCTL` is an 8-bit register that is used to select whether the USB 2.0 Controller is operating in Peripheral mode or in Host mode, and for controlling and monitoring the USB VBus line. If the PHY is suspended no PHY clock (`clk60`) is received and the VBus is not sampled.

The following table shows the register bit assignments.

Table 14-51 Fields for register: `DEVCTL`

Bits	Name	Software Access	USB Access	Description
[7]	B_DEVICE	R	R/W	This bit indicates whether the USB 2.0 Controller is operating as the 'A' device or the 'B' device. Valid values: **0x0:** 'A' device **0x1:** 'B' device Only valid while a session is in progress. To determine the role when no session is in progress, set the session bit and read this bit. **Value After Reset:** 0x1
[6]	FSDEV	R	R/W	This bit is set when a full-speed or high-speed device has been detected being connected to the port (high-speed devices are distinguished from

Table 14-51 Fields for register: `DEVCTL` (continued)

Bits	Name	Software Access	USB Access	Description
				full-speed by checking for high-speed chirps when the device is reset). Only valid in Host mode. **Value After Reset:** 0x0
[5]	`LSDEV`	R	R/W	Thisbit is set when a low-speed device has been detected being connected to the port. Only valid in Host mode. **Value After Reset:** 0x0
[4:3]	`VBUS`	R	R/W	These bits encode the current VBus level as follows: **0x0:** Below `sessend` **0x1:** Above `sessend`, below `avalid` **0x2:** Above `avalid`, below `vbusvalid` **0x3:** Above `vbusvalid` **Value After Reset:** 0x0
[2]	`HOST_MODE`	R	R/W	This bit is set when the USB 2.0 Controller is acting as a Host. **Value After Reset:** 0x0
[1]	`HOST_REQ`	R/W	R/WC	When set, the USB 2.0 Controller will initiate the Host Negotiation when Suspend mode is entered. It is cleared when Host Negotiation is completed. See Host Negotiation section (‘B’ device only). **Value After Reset:** 0x0
[0]	`SESSION`	R/W	R/W	When operating as an ‘A’ device, this bit is set or cleared by the software to start or end a session. When operating as a ‘B’ device, this bit is set/cleared by the USB 2.0 Controller when a session starts/ends. It is also set by the software to initiate the Session Request Protocol. When the USB 2.0 Controller is in Suspend mode, the bit may be cleared by the software to perform a software disconnect. Clearing this bit when the core is not suspended will result in undefined behavior. **Value After Reset:** 0x0

14.1.2.2.31. Tx Endpoint #n FIFO Size (`TXFIFOSZ`)

The `TXFIFOSZ` register is at offset 0x62.

`TXFIFOSZ` is a 5-bit register which controls the size of the selected Tx endpoint FIFO.

The following table shows the register bit assignments.

Table 14-52 Fields for register: `TXFIFOSZ`

Bits	Name	Software Access	USB Access	Description
[4]	DPB	R/W	R	Defines whether double-packet buffering supported. Valid values: **0x0:** Only single-packet buffering **0x1:** Double-packet buffering **Value After Reset:** 0x0
[3:0]	SZ	R/W	R	Maximum packet size to be allowed for (before any splitting within the FIFO of Bulk/High-Bandwidth packets prior to transmission). Valid values: **0x0:** 8 bytes **0x1:** 16 bytes **0x2:** 32 bytes **0x3:** 64 bytes **0x4:** 128 bytes **0x5:** 256 bytes **0x6:** 512 bytes **0x7:** 1024 bytes **0x8:** 2048 bytes **0x9:** 4096 bytes • If `DPB` = 0, the FIFO will also be this size • If `DPB` = 1, the FIFO will be twice this size **Value After Reset:** 0x0

14.1.2.2.32. Rx Endpoint #n FIFO Size (`RXFIFOSZ`)

The `RXFIFOSZ` register is at offset 0x63.

`RXFIFOSZ` is a 5-bit register which controls the size of the selected Rx endpoint FIFO.

The following table shows the register bit assignments.

Table 14-53 Fields for register: `RXFIFOSZ`

Bits	Name	Software Access	USB Access	Description
[4]	DPB	R/W	R	Defines whether double-packet buffering supported. Valid values: **0x0:** Only single-packet buffering **0x1:** Double-packet buffering **Value After Reset:** 0x0

Table 14-53 Fields for register: RXFIFOSZ (continued)

Bits	Name	Software Access	USB Access	Description
[3:0]	SZ	R/W	R	Maximum packet size to be allowed for (after any combination within the FIFO of Bulk/High-Bandwidth packets following their reception). Valid values: **0x0:** 8 bytes **0x1:** 16 bytes **0x2:** 32 bytes **0x3:** 64 bytes **0x4:** 128 bytes **0x5:** 256 bytes **0x6:** 512 bytes **0x7:** 1024 bytes **0x8:** 2048 bytes **0x9:** 4096 bytes • If DPB = 0, the FIFO will also be this size • If DPB = 1, the FIFO will be twice this size **Value After Reset:** 0x0

14.1.2.2.33. Tx Endpoint #n FIFO Address (TXFIFOAD)

The TXFIFOAD register is at offset 0x64.

TXFIFOAD is a 14-bit register which controls the start address of the selected Tx endpoint FIFO.

The following table shows the register bit assignments.

Table 14-54 Fields for register: TXFIFOAD

Bits	Name	Software Access	USB Access	Description
[13]				Reserved
[12:0]	AD	R/W	R	Start address of the endpoint FIFO in units of 8 bytes. Valid values: **0x0000:** 0x0000 **0x0001:** 0x0008 **0x0002:** 0x0010 **...** **0x1FFF:** 0xFFF8 **Value After Reset:** 0x0

14.1.2.2.34. Rx Endpoint #n FIFO Address (RXFIFOAD)

The RXFIFOAD register is at offset 0x66.

RXFIFOAD is a 14-bit register which controls the start address of the selected Rx endpoint FIFO.

The following table shows the register bit assignments.

Table 14-55 Fields for register: RXFIFOAD

Bits	Name	Software Access	USB Access	Description
[13]				Reserved
[12:0]	AD	R/W	R	Start address of the endpoint FIFO in units of 8 bytes. Valid values: **0x0000:** 0x0000 **0x0001:** 0x0008 **0x0002:** 0x0010 ... **0x1FFF:** 0xFFF8 **Value After Reset:** 0x0

14.1.2.2.35. PHY signals Control Register (VCONTROL)

The VCONTROL register is at offset 0x68.

If a register of more than 8 bits is specified, any write must be made to the entire register, using either a 16-bit or a 32-bit write as appropriate. Writes to individual bytes are not supported.

The following table shows the register bit assignments.

Table 14-56 Fields for register: VCONTROL

Bits	Name	Software Access	USB Access	Description
[23]	OTG_SUSPENDM	W	N/A	Controls otg_suspendm signal. This signal is used for VBUS negotiation in OTG application and HOST disconnect in HOST application, in DEVICE mode this signal should be tie to 0x0. In OTG/HOST application this signal should be connected to 0x1. Valid values: **0x0:** VBUS detection output vbus_valid, sessvalid, sessend, bvalid will give inaccurate output, but good enough for VBUS negotiation in low power status. **0x1:** VBUS detection output vbus_valid, sessvalid, sessend, bvalid will give accurate VBUS detection result. **Value After Reset:** 0x0
[22]	FSLS_SERIALMODE	W	N/A	Controls fsls_serialmode signal. This signal is used to indicate how the digital core signals the FS and LS packets to the transceiver for reference clock input selection.

Table 14-56 Fields for register: `VCONTROL` (continued)

Bits	Name	Software Access	USB Access	Description
				Valid values: **0x0:** FS packets are sent using the parallel interface **0x1:** FS and LS packets are sent using the serial interface **Value After Reset:** 0x0
[21]	`PLL_EN`	W	N/A	Controls `pll_en` signal. **Value After Reset:** 0x0
[20]	`TX_SE0`	W	N/A	Controls `tx_se0` signal. **Value After Reset:** 0x0
[19]	`TX_DAT`	W	N/A	Controls `tx_dat` signal. **Value After Reset:** 0x0
[18]	`TX_ENABLE_N`	W	N/A	Controls `tx_enable_n` signal. **Value After Reset:** 0x0
[17]	`REFCLK_MODE`	W	N/A	Controls `refclk_mode` signal. This signal is used for reference clock input selection. Valid values: **0x0:** Input clock is 25 MHz **0x1:** Input clock is 12 MHz ⚠ It is recommended to use the 25 MHz reference clock frequency. **Value After Reset:** 0x0
[16]	`CFG_RSTN`	W	N/A	Controls `cfg_rstn` signal. **Active Level:** Low **Value After Reset:** 0x0
[15]	`CFG_RDEN`	W	N/A	Controls `cfg_rden` signal. **Active Level:** High **Value After Reset:** 0x0
[14]	`CFG_WREN`	W	N/A	Controls `cfg_wren` signal. **Active Level:** High **Value After Reset:** 0x0
[13:8]	`CFG_REGADDR`	W	N/A	Controls `cfg_regaddr[5:0]` signal. **Value After Reset:** 0x0
[7:0]	`CFG_DATAWR`	W	N/A	Controls `cfg_datawr[7:0]` signal. **Value After Reset:** 0x0

14.1.2.2.36. PHY Signals Status Register (`VSTATUS`)

The `VSTATUS` register is at offset 0x68.

- The `vstatus` input bus is sampled once every 6 `clk60` cycles.
- The latency between the `vstatus` input bus from the PHY changing and the new value being read from the `VSTATUS` register (measured to the positive edge of `hclk` at the end of the read cycle) will be between:

  2 Hc + Xc and 3 Hc + 6 Xc,

  where *Hc* is a cycle of `hclk` and *Xc* is a cycle of `clk60`.

The following table shows the register bit assignments.

Table 14-57 Fields for register: `VSTATUS`

Bits	Name	Software Access	USB Access	Description
[7:0]	`CFG_DATARD`	R	N/A	Controls `cfg_datard[7:0]` signal. **Value After Reset:** 0x0

14.1.2.2.37. Information About Numbers of Tx and Rx Endpoints (`EPINFO`)

The `EPINFO` register is at offset 0x78.

This 8-bit read-only register allows read-back of the number of Tx and Rx endpoints included in the design.

The following table shows the register bit assignments.

Table 14-58 Fields for register: `EPINFO`

Bits	Name	Software Access	USB Access	Description
[7:4]	`RXENDPOINTS`	R	R	The number of Rx endpoints implemented in the design. **Value After Reset:** 0x5
[3:0]	`TXENDPOINTS`	R	R	The number of Tx endpoints implemented in the design. **Value After Reset:** 0x5

14.1.2.2.38. Information About RAM (`RAMINFO`)

The `RAMINFO` register is at offset 0x79.

This 8-bit read-only register provides information about the width of the RAM.

The following table shows the register bit assignments.

Table 14-59 Fields for register: `RAMINFO`

Bits	Name	Software Access	USB Access	Description
[7:4]				Reserved
[3:0]	`RAMBITS`	R	R	The width of the RAM address bus. **Value After Reset:** 0xB

14.1.2.2.39. Information About Delays (LINKINFO)

The LINKINFO register is at offset 0x7A.

This 8-bit register allows some delays to be specified.

The following table shows the register bit assignments.

Table 14-60 Fields for register: LINKINFO

Bits	Name	Software Access	USB Access	Description
[7:4]	WTCON	R/W	R	Sets the wait to be applied to allow for the user's connect/disconnect filter in units of 533.3 ns (the default setting corresponds to 2.667 µs). **Value After Reset:** 0x5
[3:0]	WTID	R/W	R	Sets the delay to be applied from idpullup being asserted to iddig being considered valid in units of 4.369 ms (the default setting corresponds to 52.43 ms). **Value After Reset:** 0xC

14.1.2.2.40. Duration of the VBus Pulsing Charge (VPLEN)

The VPLEN register is at offset 0x7B.

This 8-bit register sets the duration of the VBus pulsing charge.

The following table shows the register bit assignments.

Table 14-61 Fields for register: VPLEN

Bits	Name	Software Access	USB Access	Description
[7:0]	VPLEN	R/W	R	Sets the duration of the VBus pulsing charge in units of 546.1 µs (the default setting corresponds to 32.77 ms). **Value After Reset:** 0x3C

14.1.2.2.41. Time Buffer Available on High-Speed Transactions (HS_EOF1)

The HS_EOF1 register is at offset 0x7C.

This 8-bit register sets the minimum time gap that is to be allowed between the start of the last transaction and the EOF for High-speed transactions.

The following table shows the register bit assignments.

Table 14-62 Fields for register: HS_EOF1

Bits	Name	Software Access	USB Access	Description
[7:0]	HS_EOF1	R/W	R	Sets for High-speed transactions the time before EOF to stop beginning new transactions, in units of 133.3 ns (the default setting corresponds to 17.07 µs).

Table 14-62 Fields for register: HS_EOF1 (continued)

Bits	Name	Software Access	USB Access	Description
				Value After Reset: 0x80

14.1.2.2.42. Time Buffer Available on Full-Speed Transactions (FS_EOF1)

The FS_EOF1 register is at offset 0x7D.

This 8-bit register sets the minimum time gap that is to be allowed between the start of the last transaction and the EOF for Full-speed transactions.

The following table shows the register bit assignments.

Table 14-63 Fields for register: FS_EOF1

Bits	Name	Software Access	USB Access	Description
[7:0]	FS_EOF1	R/W	R	Sets for Full-speed transactions the time before EOF to stop beginning new transactions, in units of 533.3 ns (the default setting corresponds to 63.46 µs). **Value After Reset:** 0x77

14.1.2.2.43. Time Buffer Available on Low-Speed Transactions (LS_EOF1)

The LS_EOF1 register is at offset 0x7E.

This 8-bit register sets the minimum time gap that is to be allowed between the start of the last transaction and the EOF for Low-speed transactions.

The following table shows the register bit assignments.

Table 14-64 Fields for register: LS_EOF1

Bits	Name	Software Access	USB Access	Description
[7:0]	LS_EOF1	R/W	R	Sets for Low-speed transactions the time before EOF to stop beginning new transactions, in units of 1.067 µs (the default setting corresponds to 121.6 µs). **Value After Reset:** 0x72

14.1.2.2.44. Tx Endpoint #n Function Address (TXFUNCADDR)

The TXFUNCADDR register is at offset 0x80 + (n*0x8), where n = 0...5.

Exists: DEVCTL.HOST_MODE = 0x1

TXFUNCADDR is an 7-bit read/write register that record the address of the target function that is to be accessed through the associated endpoint (EPn). TXFUNCADDR needs to be defined for each Tx endpoint that is used.

The following table shows the register bit assignments.

Table 14-65 Fields for register: TXFUNCADDR

Bits	Name	Software Access	USB Access	Description
[6:0]	ADDR	R/W	R	Address of target function. **Value After Reset:** 0x0

14.1.2.2.45. Tx Endpoint #n Hub Address (TXHUBADDR)

The TXHUBADDR register is at offset 0x82 + (n*0x8), where **n** = 0...5.

Exists: DEVCTL.HOST_MODE = 0x1

TXHUBADDR is an 8-bit read/write register which, like TXHUBPORT, only need to be written where a full- or low-speed device is connected to Tx Endpoint EPn via a high-speed USB 2.0 hub which carries out the necessary transaction translation to convert between high-speed transmission and full-/low-speed transmission. In such circumstances:

- The lower 7 bits should record the address of this USB 2.0 hub
- The top bit should record whether the hub has multiple transaction translators (set to '0' if single transaction translator; set to '1' if multiple transaction translators).

If Endpoint 0 is connected to a hub, then TXHUBADDR must be defined for EP0.

The following table shows the register bit assignments.

Table 14-66 Fields for register: TXHUBADDR

Bits	Name	Software Access	USB Access	Description
[7]	MULTIPLE_TRANSLATORS	R/W	R	Multiple transaction translators. Valid values: **0:** Single transaction translator **1:** Multiple transaction translators **Value After Reset:** 0x0
[6:0]	HUB_ADDRESS	R/W	R	Address of USB 2.0 hub. **Value After Reset:** 0x0

14.1.2.2.46. Tx Endpoint #n Hub Port (TXHUBPORT)

The TXHUBPORT register is at offset 0x83 + (n*0x8), where n = 0...5.

Exists: DEVCTL.HOST_MODE = 0x1

TXHUBPORT only need to be written where a full- or low-speed device is connected to Tx Endpoint EPn via a high-speed USB 2.0 hub which carries out the necessary transaction translation. In such circumstances, these 7-bit read/write registers need to be used to record the port of that USB 2.0 hub through which the target associated with the endpoint is accessed.

If Endpoint 0 is connected to a hub, then TXHUBPORT must be defined.

This information, together with the Hub Address details recorded above, allows the USB 2.0 Controller to support split transactions.

The following table shows the register bit assignments.

Table 14-67 Fields for register: TXHUBPORT

Bits	Name	Software Access	USB Access	Description
[6:0]	HUB_PORT	R/W	R	Record the port of that USB 2.0 hub through which the target associated with the endpoint is accessed. **Value After Reset:** 0x0

14.1.2.2.47. Rx Endpoint #n Function Address (RXFUNCADDR)

The RXFUNCADDR register is at offset 0x84 + (n*0x8), where n = 0...5.

Exists: DEVCTL.HOST_MODE = 0x1

RXFUNCADDR is an 7-bit read/write register that record the address of the target function that is to be accessed through the associated endpoint (EPn). RXFUNCADDR needs to be defined for each Rx endpoint (except Endpoint 0) that is used.

The RXFUNCADDR register does not exist on EP0.

The following table shows the register bit assignments.

Table 14-68 Fields for register: RXFUNCADDR

Bits	Name	Software Access	USB Access	Description
[6:0]	ADDR	R/W	R	Address of target function. **Value After Reset:** 0x0

14.1.2.2.48. Rx Endpoint #n Hub Address (RXHUBADDR)

The RXHUBADDR register is at offset 0x86 + (n*0x8), where n = 0...5.

Exists: DEVCTL.HOST_MODE = 0x1.

RXHUBADDR is an 8-bit read/write register which, like RXHUBPORT, only need to be written where a full- or low-speed device is connected to Rx Endpoint EPn via a high-speed USB 2.0 hub which carries out the necessary transaction translation to convert between high-speed transmission and full-/low-speed transmission. In such circumstances:

- The lower 7 bits should record the address of this USB 2.0 hub.
- The top bit should record whether the hub has multiple transaction translators (set to '0' if single transaction translator; set to '1' if multiple transaction translators).

The RXHUBADDR register does not exist on EP0.

The following table shows the register bit assignments.

Table 14-69 Fields for register: RXHUBADDR

Bits	Name	Software Access	USB Access	Description
[7]	MULTIPLE_TRANSLATORS	R/W	R	Multiple transaction translators. Valid values: **0x0:** Single transaction translator **0x1:** Multiple transaction translators **Value After Reset:** 0x0
[6:0]	HUB_ADDRESS	R/W	R	Address of USB 2.0 hub. **Value After Reset:** 0x0

14.1.2.2.49. Rx Endpoint #n Hub Port (RXHUBPORT)

The RXHUBPORT register is at offset 0x87 + (n*0x8), where n = 0...5.

Exists: DEVCTL.HOST_MODE = 0x1.

RXHUBPORT only need to be written where a full- or low-speed device is connected to Rx Endpoint EPn via a high-speed USB 2.0 hub which carries out the necessary transaction translation. In such circumstances, these 7-bit read/write registers need to be used to record the port of that USB 2.0 hub through which the target associated with the endpoint is accessed.

The RXHUBPORT register does not exist on EP0.

This information, together with the Hub Address details recorded above, allows the USB 2.0 Controller to support split transactions.

The following table shows the register bit assignments.

Table 14-70 Fields for register: RXHUBPORT

Bits	Name	Software Access	USB Access	Description
[6:0]	HUB_PORT	R/W	R	Record the port of that USB 2.0 hub through which the target associated with the endpoint is accessed. **Value After Reset:** 0x0

14.1.2.2.50. DMA Channel #n Interrupt Register (DMA_INTR)

The DMA_INTR register is at offset 0x200.

The DMA_INTR register provides an interrupt for each DMA channel. This interrupt register is cleared when read. When any bit of this register is set, the output dma_nint is asserted low. Events that cause interrupts to be set is described in Included DMA Controller section. Bits in this register will only be set if the DMA Interrupt Enable bit for the corresponding channel is enabled (DMA_CNTL.DMAIE).

The following table shows the register bit assignments.

Table 14-71 Fields for register: `DMA_INTR`

Bits	Name	Software Access	USB Access	Description
[7:5]				Reserved
[4]	`DMA_CHANNEL_5_INTERRUPT`	R/W	W	DMA Channel 5 Interrupt **Value After Reset:** 0x0
[3]	`DMA_CHANNEL_4_INTERRUPT`	R/W	W	DMA Channel 4 Interrupt **Value After Reset:** 0x0
[2]	`DMA_CHANNEL_3_INTERRUPT`	R/W	W	DMA Channel 3 Interrupt **Value After Reset:** 0x0
[1]	`DMA_CHANNEL_2_INTERRUPT`	R/W	W	DMA Channel 2 Interrupt **Value After Reset:** 0x0
[0]	`DMA_CHANNEL_1_INTERRUPT`	R/W	W	DMA Channel 1 Interrupt **Value After Reset:** 0x0

14.1.2.2.51. DMA Channel #n Transfer Control Register (`DMA_CNTL`)

The `DMA_CNTL` register is at offset 0x204 + (n-1)*0x10, where n=1...5.

The `DMA_CNTL` register provides the DMA transfer control for each channel. The enabling, transfer direction, transfer mode, the DMA burst modes are all controlled by this register.

The following table shows the register bit assignments.

Table 14-72 Fields for register: `DMA_CNTL`

Bits	Name	Software Access	USB Access	Description
[10:9]	`DMA_BRSTM`	R/W	R	Burst Mode. Valid values: **0x0:** Burst Mode 0: Bursts of unspecified length **0x1:** Burst Mode 1: INCR4 or unspecified length **0x2:** Burst Mode 2: INCR8, INCR4 or unspecified length **0x3:** Burst Mode 3: INCR16, INCR8, INCR4 or unspecified length **Value After Reset:** 0x0
[8]	`DMA_ERR`	R/W	R/W	Bus Error Bit. Indicates that a bus error has been observed on the input `hrespm[1:0]`. This bit is cleared by software. **Value After Reset:** 0x0
[7:4]	`DMAEP`	R/W	R	The endpoint number this channel is assigned to. Valid values: **0x0:** Endpoint 0 **0x1:** Endpoint 1

Table 14-72 Fields for register: `DMA_CNTL` (continued)

Bits	Name	Software Access	USB Access	Description
				0x2: Endpoint 2 **0x3:** Endpoint 3 **0x4:** Endpoint 4 **0x5:** Endpoint 5 **Value After Reset:** 0x0
[3]	`DMAIE`	R/W	R	DMA Interrupt Enable. **Value After Reset:** 0x0
[2]	`DMAMODE`	R/W	R	This bit selects the DMA Transfer Mode. Valid values: **0x0:** DMA Mode0 Transfer **0x1:** DMA Mode1 Transfer **Value After Reset:** 0x0
[1]	`DMA_DIR`	R/W	R	This bit selects the DMA Transfer Direction. Valid values: **0x0:** DMA Write (Rx Endpoint) **0x1:** DMa Read (Tx Endpoint) **Value After Reset:** 0x0
[0]	`DMA_ENAB`	R/W	R	This bit enables the DMA transfer and will cause the transfer to begin. **Value After Reset:** 0x0

14.1.2.2.52. DMA Channel #n Address Register (`DMA_ADDR`)

The `DMA_ADDR` register is at offset 0x208 + (n-1)*0x10, where n=1...5.

The `DMA_ADDR` register identifies the current memory address of the corresponding DMA channel. The Initial memory address written to this register must have a value such that its modulo 4 value is equal to 0. That is, `DMA_ADDR[1:0]` must be equal to 2'b00. The lower two bits of this register are read only and cannot be set by software. As the DMA transfer progresses, the memory address will increment as bytes are tranfered.

The following table shows the register bit assignments.

Table 14-73 Fields for register: `DMA_ADDR`

Bits	Name	Software Access	USB Access	Description
[31:2]	`DMA_ADDR`	R/W	R/W	The DMA memory address. Note that the initial memory address written to this register must have a value such that it's modulo 4 value is equal to 0. That is, `DMA_ADDR[1:0]` must be equal to 2'b00. The lower two bits of this register are read only and cannot be set by software. **Value After Reset:** 0x0
[1:0]		R	R/W	

14.1.2.2.53. DMA Channel #n Transfer Count Register (DMA_COUNT)

The DMA_COUNT register is at offset 0x20C + (n-1)*0x10, where n=1...5.

The register identifies the current DMA count of the transfer. Software will set the initial count of the transfer which identifies the entire transfer length. As the count progresses this count is decremented as bytes are transfered.

The following table shows the register bit assignments.

Table 14-74 Fields for register: DMA_COUNT

Bits	Name	Software Access	USB Access	Description
[31:0]	DMA_COUNT	R/W	R/W	The DMA memory address for the corresponding DMA channel. If DMA is enabled with a count of 0, the bus will not be requested and a DMA interrupt will be generated. **Value After Reset:** 0x0

14.1.2.2.54. Number of Requested Packets for Rx Endpoint #n (EPn_RQPKTCOUNT)

The EPn_RQPKTCOUNT register is at offset 0x300 + (n*0x4), where n = 1...5.

Exists: DEVCTL.HOST_MODE = 0x1.

For each Rx Endpoint 1...5, the USB 2.0 Controller provides a 16-bit EPn_RQPKTCOUNT register.

This read/write register is used in Host mode to specify the number of packets that are to be transferred in a block transfer of one or more Bulk packets of length MaxP to Rx Endpoint "n".

The core uses the value recorded in this register to determine the number of requests to issue where the RXCSRH.AUTOREQ has been set.

Multiple packets combined into a single bulk packet within the FIFO count as one packet.

The following table shows the register bit assignments.

Table 14-75 Fields for register: EPn_RQPKTCOUNT

Bits	Name	Software Access	USB Access	Description
[15:0]	RQPKTCOUNT	R/W	R/W	Sets the number of packets of size MaxP that are to be transferred in a block transfer. Only used in Host mode when RXCSRH.AUTOREQ is set. Has no effect in Peripheral mode or when AUTOREQ is not set. **Value After Reset:** 0x0

14.1.2.2.55. Chirp Timeout Timer Register(C_T_UCH)

The C_T_UCH register is at offset 0x344.

This register sets the chirp timeout. This number when multiplied by 4 represents the number of 60 MHz cycles before the timeout occurs. This number represents the number of 67 ns time intervals before the timeout occurs.

The following table shows the register bit assignments.

Table 14-76 Fields for register: `C_T_UCH`

Bits	Name	Software Access	USB Access	Description
[15:0]	`C_T_UCH`	R/W	N/A	Configurable Chirp Timeout timer. **Value After Reset:** 0x101D0

14.1.2.2.56. High Speed Resume Signaling Delay Register (`C_T_HSRTN`)

The `C_T_HSRTN` register is at offset 0x346.

This register sets the delay from the end of High Speed resume signaling to enable the normal operating mode. This number when multiplied by 4 represents the number of 60 MHz cycles before the timeout occurs. This number represents the number of 67 ns time intervals before the timeout occurs.

The following table shows the register bit assignments.

Table 14-77 Fields for register: `C_T_HSRTN`

Bits	Name	Software Access	USB Access	Description
[15:0]	`C_T_HSRTN`	R/W	N/A	The delay from the end of High Speed resume signaling to enabling normal operating mode. **Value After Reset:** 0x34BC

14.1.2.2.57. High Speed Timeout Increase Register (`C_T_HSBT`)

The `C_T_HSBT` register is at offset 0x348.

Per ***Universal Serial Bus Specification, Revision 2.0, Section 7.1.19.2***, a high-speed host or device expecting a response to a transmission must not timeout the transaction if the interpacket delay is less than 736 bit times, and it must timeout the transaction if no signaling is seen within 816 bit times.

This register represents the value to be added to the mimumum high speed (HS) timeout period of 736 bit times. The timeout period can be increased in increments of 64 high speed (HS) bit times (133 ns). There are 16 possible values.

By default, the adder is 0 thus setting the high speed (HS) timeout to its minimum value. Use of this register will allow the high speed (HS) timeout to be set to values that are greater the maximum specifed in ***Universal Serial Bus Specification, Revision 2.0*** making the USB 2.0 Controller non-complient.

The following table shows the register bit assignments.

Table 14-78 Fields for register: `C_T_HSBT`

Bits	Name	Software Access	USB Access	Description
[3:0]	`C_T_HSBT`	R/W	R	The value added to the minimum high speed (HS) timeout period (736 bit times) in increments of

Table 14-78 Fields for register: C_T_HSBT (continued)

Bits	Name	Software Access	USB Access	Description
				64 high speed (HS) bit times. This allows the turn around timeout period to be set to 16 possible values, as shown in **C_T_HSBT values decoding** table below. **Value After Reset:** 0x0

Table 14-79 C_T_HSBT values decoding

C_T_HSBT Value	HS Turnaround Timeout (HS Bit Times)	HS Turnaround Timeout (µs)
0x0	736	1.534
0x1	800	1.667
0x2	864	1.801
0x3	928	1.934
0x4	992	2.067
0x5	1056	2.201
0x6	1120	2.334
0x7	1184	2.601
0x8	1248	2.601
0x9	1312	2.734
0xA	1376	2.868
0xB	1440	3.001
0xC	1504	3.134
0xD	1568	3.268
0xE	1632	3.401
0xF	1696	3.534

14.1.2.2.58. LPM Attribute Register (LPM_ATTR)

The LPM_ATTR register is at offset 0x360.

This register is used to define the attributes of an LPM transaction and sleep cycle. In both the Host mode and the Peripheral mode, the meaning of this register is the same however the source of the data is different for Host and Peripheral as follows:

In Peripheral mode:

In Peripheral mode, the values in this register will contain the equivalent attributes that were received in the last LPM transaction that was accepted. This register is updated with the LPM packet contents if the response to the LPM transaction was an ACK. This register can be update via software. In all other cases, this register will hold its current value.

In Host mode:

In Host mode software will set-up the values in this register to define the next LPM transaction that will be transmitted. These values will be inserted in the payload of the next LPM Transaction.

The following table shows the register bit assignments.

Table 14-80 Fields for register: `LPM_ATTR`

Bits	Name	Software Access	USB Access	Description
[15:12]	`ENDPNT`	R	R	End Point number that in the Token Packet of the LPM transaction. **Value After Reset:** 0x0
[11:9]				Reserved
[8]	`RMTWAK`	R	R/W	Remote wakeup enable bit. Valid values: **0x0:** Not enabled **0x1:** Remote wakeup is enabled This bit is applied on a temporary basis only and is only applied to the current suspend state. After the current suspend cycle, the remote wakeup capability that was negotiated upon enumeration will apply. **Value After Reset:** 0x0
[7:4]	`HIRD`	R	R/W	Host Initiated Resume Duration. This value is the minimum time the host will drive resume on the Bus. The value in this register corresponds to an actual resume time of: *Resume Time = 50 µs + HIRD*75 µs*. This results a range 50 µs to 1,200 µs. **Value After Reset:** 0x0
[3:0]	`LINKSTATE`	R	R/W	This value is provided by the host to the peripheral to indicate what state the peripheral must transition to after the receipt and acceptance of a LPM transaction. Valid values: **0x0:** Reserved **0x1:** Sleep State (L1) **0x2, 0x3:** Reserved **Value After Reset:** 0x0

14.1.2.2.59. LPM Control Register (`LPM_CNTRL`)

The `LPM_CNTRL` register is at offset 0x362.

The following table shows the register bit assignments.

The following table shows the register bit assignments for USB Controller in **Peripheral Mode**.

Table 14-81 Fields for register: `LPM_CNTRL` (`DEVCTL.HOST_MODE = 0x0`)

Bits	Name	Software Access	USB Access	Description
[7:5]				Reserved
[4]	`LPMNAK`	R/W	R	This bit is used to place all end points in a state such that the response to all transactions other then an LPM transaction will be a NAK. This bit will only take effect after the USB 2.0 Controller has been LPM suspended. In this case, the USB 2.0 Controller will continue to NAK until this bit has been cleared by software. **Value After Reset:** 0x0
[3:2]	`LPMEN`	R/W	R	This bits are used to enable LPM in the USB 2.0 Controller. There are three levels in which LPM can be enabled which will determine the response of the USB 2.0 Controller to LPM transactions. The three levels are listed below. Valid values: **0x0, 0x2:** LPM and Extended transactions are not supported. In this case, the USB 2.0 Controller will not respond to LPM transactions and the transaction will timeout. **0x1:** LPM is not supported but extended transactions are supported. In this case, the USB 2.0 Controller will respond to an LPM transaction with a STALL. **0x1:** The USB 2.0 Controller supports LPM extended transactions. In this case, the USB 2.0 Controller will respond with a NYET or an ACK as determined by the value of `LPMXMT` field below and other conditions. **Value After Reset:** 0x0
[1]	`LPMRES`	R/W	WC	This bit is used by software to initiate resume (remote wakeup). This bit differs from the classic `POWER.RESUME` bit, where the `resume` signal timing is controlled by hardware. When software writes this bit, resume signaling will be asserted for 50 µs. This bit is self clearing. **Value After Reset:** 0x0
[0]	`LPMXMT`	R/W	WC	This bit is set by software to instruct the USB 2.0 Controller to transition to the L1 state upon the receipt of the next LPM transaction.

Table 14-81 Fields for register: LPM_CNTRL (DEVCTL.HOST_MODE = 0x0) (continued)

Bits	Name	Software Access	USB Access	Description
				This bit is only effective if LPMEN field above is set to 2'b11. This bit can be set in the same cycle as LPMEN. If this bit is set to 1'b1 and LPMEN = 2'b11, the USB 2.0 Controller can respond in the following ways: • If no data is pending (All Tx FIFOs are empty), the USB 2.0 Controller will respond with an ACK. In this case this bit will self clear and a software interrupt will be generated. • If data is pending (data resides in at least one Tx FIFO), the USB 2.0 Controller will respond with a NYET. In this case, this bit will NOT self clear however a software interrupt will be generated. **Value After Reset:** 0x0

The following table shows the register bit assignments for USB Controller in **Host Mode**.

Table 14-82 Fields for register: LPM_CNTRL (DEVCTL.HOST_MODE = 0x1)

Bits	Name	Software Access	USB Access	Description
[7:2]				Reserved
[1]	LPMRES	R/W	R/W	This bit is used by software to initiate a RESUME from the L1 State. When software writes this bit, resume signaling will be asserted for a time specified by the LPM_ATTR.HIRD field. This bit is self clearing. **Value After Reset:** 0x0
[0]	LPMXMT	R/W	R/W	Software should set this bit to transmit an LPM transaction. This bit is self clearing. This bit will be immediately cleared upon receipt of any Token or three timeouts have occurred. **Value After Reset:** 0x0

14.1.2.2.60. LPM Interrupt Enable Register (LPM_INTREN)

The LPM_INTREN register is at offset 0x363.

The LPM_INTREN is a 6 bit register that provides enable bits for the interrupts in the LPM_INTR.

If a bit in this register is set to 1, mc_nint will be asserted (low) when the corresponding interrupt in the LPM_INTR is set.

If a bit in this register is set to 0, the corresponding register in LPM_INTR is still set but mc_nint will not be asserted (low).

On reset, all bits in this register are reset to 0.

The following table shows the register bit assignments.

Table 14-83 Fields for register: `LPM_INTREN`

Bits	Name	Software Access	USB Access	Description
[7:6]				Reserved
[5]	`LPMERREN`	R	R	Valid values: **0x0:** Disable the `LPMERR` interrupt **0x1:** Enable the `LPMERR` interrupt **Value After Reset:** 0x0
[4]	`LPMRESEN`	R	R	Valid values: **0x0:** Disable the `LPMRES` interrupt **0x1:** Enable the `LPMRES` interrupt **Value After Reset:** 0x0
[3]	`LPMNCEN`	R/W	R	Valid values: **0x0:** Disable the `LPMNC` interrupt **0x1:** Enable the `LPMNC` interrupt **Value After Reset:** 0x0
[2]	`LPMACKEN`	R/W	R	Valid values: **0x0:** Disable the `LPMACK` interrupt **0x1:** Enable the `LPMACK` interrupt **Value After Reset:** 0x0
[1]	`LPMNYEN`	R/W	R	Valid values: **0x0:** Disable the `LPMNY` interrupt **0x1:** Enable the `LPMNY` interrupt **Value After Reset:** 0x0
[0]	`LPMSTEN`	R/W	R	Valid values: **0x0:** Disable the `LPMST` interrupt **0x1:** Enable the `LPMST` interrupt **Value After Reset:** 0x0

14.1.2.2.61. LPM Interrupt Register (`LPM_INTR`)

The `LPM_INTR` register is at offset 0x364.

The following table shows the register bit assignments for USB Controller in **Peripheral Mode**.

Table 14-84 Fields for register: `LPM_INTR` (`DEVCTL.HOST_MODE=0x0`)

Bits	Name	Software Access	USB Access	Description
[7:6]				Reserved

Table 14-84 Fields for register: `LPM_INTR` (`DEVCTL.HOST_MODE=0x0`) (continued)

Bits	Name	Software Access	USB Access	Description
[5]	LPMERR	R	WS	This bit is set if an LPM transaction is received that has a `LINKSTATE` field that is not supported. In this case, the response to the transaction will be a STALL. However, the `LPM_INTR` register will be updated so that software can observe the non compliant LPM packet payload. **Value After Reset:** 0x0
[4]	LPMRES	R	WS	This bit is set if the USB 2.0 Controller has been resumed for any reason. This bit is mutually exclusive from the `POWER.RESUME` bit. **Value After Reset:** 0x0
[3]	LPMNC	R	WS	This bit is set when an LPM transaction is received and the USB 2.0 Controller responds with a NYET due to data pending in the Rx FIFOs. **Value After Reset:** 0x0
[2]	LPMACK	R	WS	This bit is set when an LPM transaction is received and the USB 2.0 Controller responds with an ACK. **Value After Reset:** 0x0
[1]	LPMNY	R	WS	This bit is set when an LPM transaction is received and the USB 2.0 Controller responds with a NYET. **Value After Reset:** 0x0
[0]	LPMST	R	WS	This bit is set when an LPM transaction is received and the USB 2.0 Controller responds with a STALL. **Value After Reset:** 0x0

The following table shows the register bit assignments for USB Controller in **Host Mode**.

Table 14-85 Fields for register: `LPM_INTR` (`DEVCTL.HOST_MODE=0x1`)

Bits	Name	Software Access	USB Access	Description
[7:6]				Reserved
[5]	LPMERR	R	WS	This bit is set if a response to the LPM transaction is received with a Bit Stuff error or a PID error. In this case, no suspend occurs and the state of the device is now unknown. **Value After Reset:** 0x0
[4]	LPMRES	R	WS	This bit is set when the USB 2.0 Controller has been resumed for any reason.

Table 14-85 Fields for register: `LPM_INTR` (`DEVCTL.HOST_MODE=0x1`) (continued)

Bits	Name	Software Access	USB Access	Description
				This bit is mutually exclusive from the `POWER.RESUME` bit. **Value After Reset:** 0x0
[3]	`LPMNC`	R	WS	This bit is set when an LPM transaction has been transmitted and has failed to complete. The transaction will have failed because a timeout occurred or there were bit errors in the response for three attempts. **Value After Reset:** 0x0
[2]	`LPMACK`	R	WS	This bit is set when an LPM transaction is transmitted and the device responds with an ACK. **Value After Reset:** 0x0
[1]	`LPMNY`	R	WS	This bit is set when an LPM transaction is transmitted and the device responds with a NYET. **Value After Reset:** 0x0
[0]	`LPMST`	R	WS	This bit is set when an LPM transaction is transmitted and the device responds with a STALL. **Value After Reset:** 0x0

14.1.2.2.62. LPM Function Address Register (`LPM_FADDR`)

The `LPM_FADDR` register is at offset 0x365.

Exists: `DEVCTL.HOST_MODE` = 0x1

The `LPM_FADDR` is the function address that will be placed in the LPM payload.

The following table shows the register bit assignments.

Table 14-86 Fields for register: `LPM_FADDR`

Bits	Name	Software Access	USB Access	Description
[7]				Reserved
[6:0]	`LPM_FADDR`	R/W	R	The LPM function address. **Value After Reset:** 0x0

14.2. USB 2.0 PHY

A simplified block diagram of the USB 2.0 PHY is shown in the following figure.

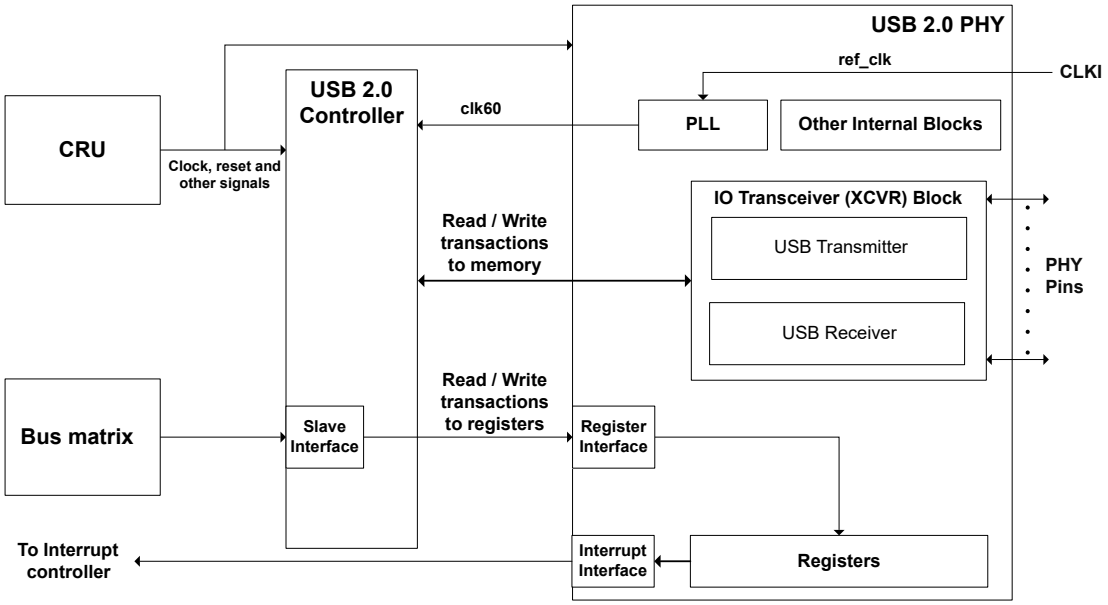

Figure 14-9 USB 2.0 PHY block diagram

USB 2.0 PHY features:

- Supports HS (480 Mbps)/FS (12 Mbps)/LS (1.5 Mbps) modes.
- Serializing for transmitting data stream and De-serializing for receiving data stream.
- USB Data Recovery and Clock Recovery on receiving.
- Integrated Bit Stuffing and NRZI encoding for Transmit.
- Integrated Bit Un-Stuffing and NRZI decoding for Receive.
- SYNC and EOP generation on transmit packets and detection on receive packets.
- Supports USB suspend state and remote wakeup.
- Supports resume send in serial mode when PHY PLL is off.
- Supports detection of USB reset, suspend and resume signaling.
- Support high speed host disconnection detection.

14.2.1. USB 2.0 PHY Registers

This section covers the following topics:

- Register Map
- Register Descriptions

The BE-U1000 microcontroller USB 2.0 PHY registers can be accessed not directly, but using USB 2.0 Controller `VCONTROL` and `VSTATUS` registers.

The register configuration interface will be used to write or read register, when it is worked in write mode, user should make `VCONTROL`.`CFG_REGADDR` and `VCONTROL`.`CFG_DATAWR` value stable and make `VCONTROL`.`CFG_WREN` to be asserted after. When it is worked in read mode, user should make `VCONTROL`.`CFG_REGADDR` value stable and make `VCONTROL`.`CFG_RDEN` to be assert after, `VSTATUS`.`CFG_DATARD` will be displayed with some delay time. When write or read next data, please make `VCONTROL`.`CFG_WREN` or `VCONTROL`.`CFG_RDEN` to be de-asserted first, and do the same operation step as the before.

You can find list and description of USB 2.0 PHY registers in the Register Map and Register Descriptions sections below.

14.2.1.1. Register Map

This section tables contain information about the USB 2.0 PHY registers memory map.

The following table provides a high-level summary of each register. To view the detailed description of a register, click the register name in the **Description** column.

Table 14-87 USB 2.0 PHY register map

Name	Offset	Description	Default Value
HS_RES_ADJ_TX_CLK_GATE_CTL	0x0	HS Resistor Adjust and Tx Clock Gate Control Register	0x80
SPWR_SVNG_SQLCH_CTL	0x1	Super Power Saving and Squelch Control Register	0xF8
OUT_EYE_SHAPE_INT_BIAS_CTL	0x2	Output Eye Shape and Internal Bias Current Control Register	0x47
TX_CLK_PHASE_PLL_ADJ_CTL	0x3	Tx Clock Phase and PLL Adjust Control Register	0x5
PARAM_CTL_0	0x5	Parameters Control Register 0	0xC1
PARAM_CTL_1	0x6	Parameters Control Register 1	0x40
PARAM_CTL_2	0x7	Parameters Control Register 2	0x0
RSVD_OUT1_0	0x8	Reserved Out1 Register 0	0xF0
RSVD_OUT1_1	0x9	Reserved Out1 Register 1	0xF0
RSVD_OUT2_0	0xA	Reserved Out2 Register 0	0xF0
RSVD_OUT2_1	0xB	Reserved Out2 Register 1	0xF0
RSVD_IN1_0	0xC	Reserved Input1 Register 0	0x0
RSVD_IN1_1	0xD	Reserved Input1 Register 1	0x0
RSVD_IN2_0	0xE	Reserved Input2 Register 0	0x0
RSVD_IN2_1	0xF	Reserved Input2 Register 1	0x0

14.2.1.2. Register Descriptions

14.2.1.2.1. HS Resistor Adjust and Tx Clock Gate Control Register (HS_RES_ADJ_TX_CLK_GATE_CTL)

The HS_RES_ADJ_TX_CLK_GATE_CTL register is at offset 0x0.

The following table shows the register bit assignments.

Table 14-88 Fields for register: `HS_RES_ADJ_TX_CLK_GATE_CTL`

Bits	Name	R/W	Description
[7:5]	`HS_RES_ADJ`	R/W	Fine tune the 45 ohm termination resistor (HS). **Valid values:** **0x0:** 40 ohm **0x4:** 45 ohm (default) **0x6:** 50 ohm **Value After Reset:** 0x4
[4:3]			Reserved
[2:0]	`CLK_MODE`	R/W	Clock Gating Control Signal. `CLK_MODE[2]:` Clock Gating Control Signal for Tx. **Valid values:** **0x0:** Low power consumption mode (default) **0x1:** Normal power consumption mode `CLK_MODE[1:0]:` Clock Gating Control Signal for Rx. **Valid values:** **00:** Lower power consumption mode (default) **01:** Lowest power consumption mode **1X:** Normal power consumption mode **Value After Reset:** 0x0

14.2.1.2.2. Super Power Saving and Squelch Control Register (`SPWR_SVNG_SQLCH_CTL`)

The `SPWR_SVNG_SQLCH_CTL` register is at offset 0x1.

The following table shows the register bit assignments.

Table 14-89 Fields for register: `SPWR_SVNG_SQLCH_CTL`

Bits	Name	R/W	Description
[7:4]	`ADJ_PW_HS[3:0]`	R/W	Super power saving with reduced output swing mode control bits (for HS mode only). **Valid values:** **0x1:** 100 mV **0x3:** 200 mV **0x7:** 300 mV **0xF:** 400 mV (default) **Value After Reset:** 0xF
[3:0]	`ADJ_VREF_SQ[3:0]`	R/W	Squelch detection threshold voltage control bits. **Valid values:** **0x0:** 92 mV **0x8:** 124 mV (default) **0xF:** 152 mV **Value After Reset:** 0x8

14.2.1.2.3. Output Eye Shape and Internal Bias Current Control Register (OUT_EYE_SHAPE_INT_BIAS_CTL)

The OUT_EYE_SHAPE_INT_BIAS_CTL register is at offset 0x2.

The following table shows the register bit assignments.

Table 14-90 Fields for register: OUT_EYE_SHAPE_INT_BIAS_CTL

Bits	Name	R/W	Description
[7]			Reserved
[6:4]	ADJ_VSW_HS[2:0]	R/W	Output eye shape adjustment control bits. **Valid values:** **0x0:** 320 mV **0x4:** 400 mV (default) **0x7:** 460 mV **Value After Reset:** 0x4
[3:0]	ADJ_IREF_RES[3:0]	R/W	Internal bias current adjustment control bits. **Valid values:** **0x0:** 125 µA **0x7:** 100 µA (default) **0xF:** 78 µA **Value After Reset:** 0x7

14.2.1.2.4. Tx Clock Phase and PLL Adjust Control Register (TX_CLK_PHASE_PLL_ADJ_CTL)

The TX_CLK_PHASE_PLL_ADJ_CTL register is at offset 0x3.

The following table shows the register bit assignments.

Table 14-91 Fields for register: TX_CLK_PHASE_PLL_ADJ_CTL

Bits	Name	R/W	Description
[7]			Reserved
[6:4]	ADJ_TXCLK_PHASE[2:0]	R/W	Tx Clock phase adjust signal. **Value After Reset:** 0x0
[3:0]	ADJ_PLL[3:0]	R/W	PLL adjustment signal. **Value After Reset:** 0x5

14.2.1.2.5. Parameters Control Register 0 (PARAM_CTL_0)

The PARAM_CTL_0 register is at offset 0x5.

The following table shows the register bit assignments.

Table 14-92 Fields for register: PARAM_CTL_0

Bits	Name	R/W	Description
[7:5]			Reserved
[4]	PHY_MODE_INTER	R/W	PHY mode internal.

Table 14-92 Fields for register: `PARAM_CTL_0` (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0
[3]	PHY_MODE_SEL	R/W	PHY mode select. **Value After Reset:** 0x0
[2]	CLK_SEL	R/W	clk60 source select. **Value After Reset:** 0x0
[1]	COUNTER_SEL	R/W	SIE input sample enable. **Value After Reset:** 0x0
[0]	RXGAP_FIX_EN	R/W	RXGAP fix enable. **Value After Reset:** 0x1

14.2.1.2.6. Parameters Control Register 1 (`PARAM_CTL_1`)

The `PARAM_CTL_1` register is at offset 0x6.

The following table shows the register bit assignments.

Table 14-93 Fields for register: `PARAM_CTL_1`

Bits	Name	R/W	Description
[7]			Reserved
[6:4]	HS_TX2RX_DLY_CNT_SEL	R/W	PHY High-Speed bus turn-around time select. **Valid values:** **0x0:** 25*T **0x1:** 1*T **0x2:** 3*T **0x3:** 5*T **0x4:** 7*T (Default) **0x5:** 11*T **0x6:** 15*T **0x7:** 19*T T - is the period of clk_480. **Value After Reset:** 0x4
[3]	LS_KPALV_EN	R/W	LS mode keep alive enable. **Value After Reset:** 0x0
[2]	PRE_HPHY_LSIE	R/W	dis_preamble enable. **Value After Reset:** 0x0
[1]	DET_FSEOP_EN	R/W	FS EOP detect enable. **Value After Reset:** 0x0
[0]	LS_PAR_EN	R/W	LS mode with parallel enable. **Value After Reset:** 0x0

14.2.1.2.7. Parameters Control Register 2 (PARAM_CTL_2)

The PARAM_CTL_2 register is at offset 0x7.

The following table shows the register bit assignments.

Table 14-94 Fields for register: PARAM_CTL_2

Bits	Name	R/W	Description
[7:5]			Reserved
[4]	VREF_PD18_SEL	R/W	Inner vref_pd18 control signal. **Valid values:** **0x0:** vref_pd18 is only controlled by pll_en **0x1:** vref_pd18 is controlled by pll_en | suspendm **Value After Reset:** 0x0
[3]	PLL_PD_SEL	R/W	Inner pd control signal. **Valid values:** **0x0:** pll_pd is only controlled by pll_en **0x1:** pll_pd is controlled by pll_en | suspendm **Value After Reset:** 0x0
[2]	CLK480_SEL	R/W	clk_480 output time select. **Valid values:** **0x0:** clk_480 is valid after PLL is locked and add about 300 µs delay **0x1:** clk_480 is valid after PLL is locked **Value After Reset:** 0x0
[1:0]	CNT_NUM	R/W	3 ms counter select. **Valid values:** **0x0:** 392 µs (Default) **0x1:** 682 µs **0x2:** 1.36 ms **0x3:** 2.72 ms **Value After Reset:** 0x0

14.2.1.2.8. Reserved Out1 Register 0 (RSVD_OUT1_0)

The RSVD_OUT1_0 register is at offset 0x8.

The following table shows the register bit assignments.

Table 14-95 Fields for register: RSVD_OUT1_0

Bits	Name	R/W	Description
[7:0]	RESERVED_OUT1[7:0]	R/W	**Value After Reset:** 0xF0

14.2.1.2.9. Reserved Out1 Register 1 (RSVD_OUT1_1)

The RSVD_OUT1_1 register is at offset 0x9.

The following table shows the register bit assignments.

Table 14-96 Fields for register: RSVD_OUT1_1

Bits	Name	R/W	Description
[7:0]	RESERVED_OUT1[15:8]	R/W	**Value After Reset:** 0xF0

14.2.1.2.10. Reserved Out2 Register 0 (RSVD_OUT2_0)

The RSVD_OUT2_0 register is at offset 0xA.

The following table shows the register bit assignments.

Table 14-97 Fields for register: RSVD_OUT2_0

Bits	Name	R/W	Description
[7:0]	RESERVED_OUT2[7:0]	R/W	**Value After Reset:** 0xF0

14.2.1.2.11. Reserved Out2 Register 1 (RSVD_OUT2_1)

The RSVD_OUT2_1 register is at offset 0xB.

The following table shows the register bit assignments.

Table 14-98 Fields for register: RSVD_OUT2_1

Bits	Name	R/W	Description
[7:0]	RESERVED_OUT2[15:8]	R/W	**Value After Reset:** 0xF0

14.2.1.2.12. Reserved Input1 Register 0 (RSVD_IN1_0)

The RSVD_IN1_0 register is at offset 0xC.

The following table shows the register bit assignments.

Table 14-99 Fields for register: RSVD_IN1_0

Bits	Name	R/W	Description
[7:0]	RESERVED_IN1[7:0]	R	**Value After Reset:** 0x0

14.2.1.2.13. Reserved Input1 Register 1 (RSVD_IN1_1)

The RSVD_IN1_1 register is at offset 0xD.

The following table shows the register bit assignments.

Table 14-100 Fields for register: RSVD_IN1_1

Bits	Name	R/W	Description
[7:0]	RESERVED_IN1[15:8]	R	**Value After Reset:** 0x0

14.2.1.2.14. Reserved Input2 Register 0 (RSVD_IN2_0)

The RSVD_IN2_0 register is at offset 0xE.

The following table shows the register bit assignments.

Table 14-101 Fields for register: RSVD_IN2_0

Bits	Name	R/W	Description
[7:0]	RESERVED_IN2[7:0]	R	**Value After Reset:** 0x0

14.2.1.2.15. Reserved Input2 Register 1 (RSVD_IN2_1)

The RSVD_IN2_1 register is at offset 0xF.

The following table shows the register bit assignments.

Table 14-102 Fields for register: RSVD_IN2_1

Bits	Name	R/W	Description
[7:0]	RESERVED_IN2[15:8]	R	**Value After Reset:** 0x0

15. Core 2 Software JTAG

JTAG interface for the BR-350 Core 0 and BR-350 Core 1 can be software emulated via the DEBUG register that is located in the TCMB memory acessible via the TCM Arbiter of the BM-310 Core 2. DEBUG register is available only for the BM-310 Core 2 and is multiplexed with a hardware debugging JTAG interface by setting the SOFTDBGEN bit of the SYSCR0 CRU register. SYSCR0.SOFTDBGEN bit can be set by software in any time or will be set automatically when PC[8]=1 and the value of the strap pins {PC[4], PB[10], PA[10]}=100.

When PC[8]=0 and the value of the strap pins {PC[4], PB[10], PA[10]}=100, the SYSCR0.SOFTDBGEN is not set automatically and a hardware debugger is connected to the BR-350 Core 0, BR-350 Core 1 and BM-310 Core 2. In this case, Software JTAG can be enabled by setting the SOFTDBGEN bit by software.

If the SYSCR0.SOFTDBGEN bit is set, debugging is not provided for the BM-310 Core 2.

15.1. Software JTAG Register

The register region of the Software JTAG is mapped in the BE-U1000 microcontroller memory map as the following table shows.

Table 15-1 Memory address region for software JTAG register

Region	Block Name	Size	Start Address	End Address
Core 2 Resources	Software JTAG	4 B	0x4000_A010	0x4000_A013

For details, refer to Memory Map chapter.

15.1.1. Register Map

The following table provides a top-level summary of each register. To view the detailed description of a register, click the register name in the **Description** column.

Table 15-2 Software JTAG register map

Register	Offset	Description
DEBUG	0x0	Software Debug

15.1.2. Register Description

In the table below, the **R/W** column specifies how the application and the core can access the register fields.

15.1.2.1. Software Debug (DEBUG)

The DEBUG register is at offset 0x0.

The following table shows the register bit assignments.

Table 15-3 Fields for register: `DEBUG`

Bits	Name	R/W	Description
[31:5]			Reserved
[4]	`NTRST`	R/W	Data that is propagated to the PA[4] I/O pin **Value After Reset:** 0x0
[3]	`TDI`	R/W	Data that is propagated to the PA[3] I/O pin **Value After Reset:** 0x0
[2]	`TDO`	R	Data that is present at the PA[2] I/O pin **Value After Reset:** 0x0
[1]	`TMS`	R/W	Data that is propagated to the PA[1] I/O pin **Value After Reset:** 0x0
[0]	`TCK`	R/W	Data that is propagated to the PA[0] I/O pin **Value After Reset:** 0x0

16. Tracer

The BE-U1000 contains a Tracing System that is part of the Debug System. The Tracing System consists of a number of components, the most significant of which is the Tracer Block, as shown in the block diagram below.

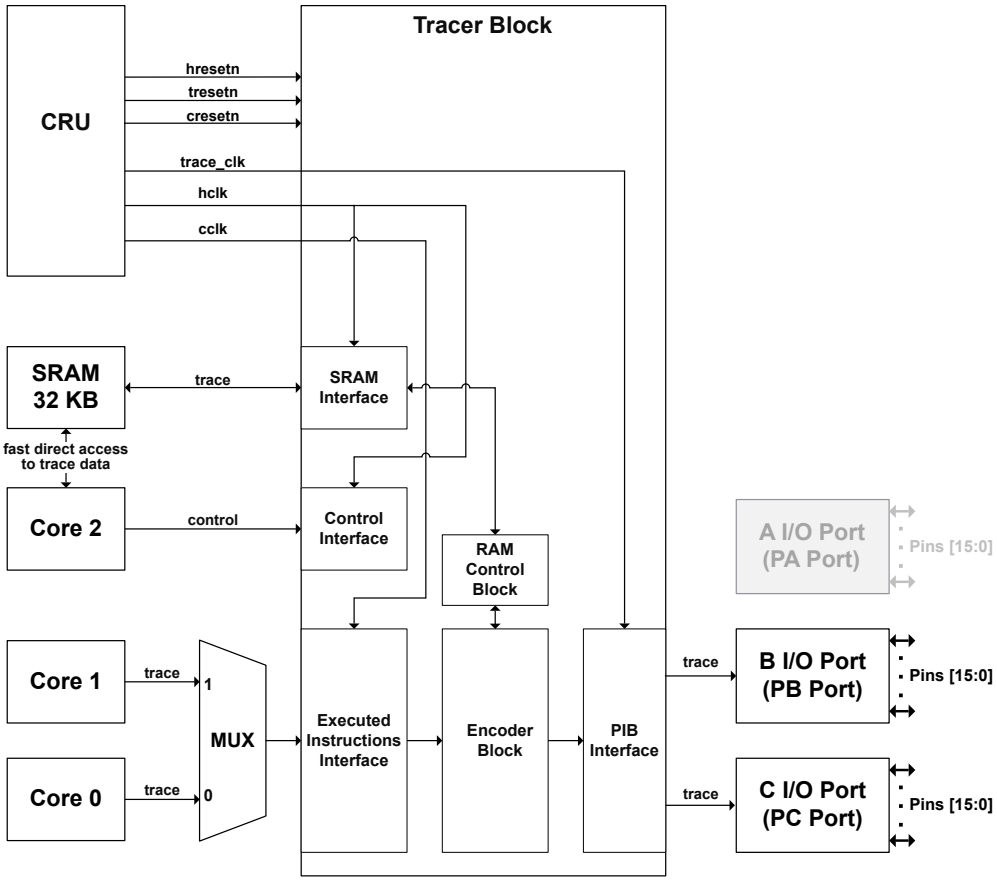

Figure 16-1 Tracer block diagram

Of the three cores (Core 0, Core 1, and Core 2) in the BE-U1000, only Core 2 is responsible for supporting the Tracing System interfaces, while Core 0 and Core 1 operate as trace sources in the trace collection process.

The Tracer Block components meet the following specifications and standards:

- ***IEEE-Industry Standards and Technology Organization. The Nexus 5001 Forum. Standard for a Global Embedded Processor. Debug Interface. Version 3.0, June 2012.***
- ***RISC-V International. Efficient Trace for RISC-V. Version 1.1.3, November 2021.***
- ***RISC-V International. RISC-V N-Trace (Nexus-based Trace) Specification. Version 1.0, July 2024.***
- ***RISC-V International. RISC-V Control interface. Version 1.0, July 2024.***

16.1. Tracer Registers

This section covers the following topics:

- Register Maps
- Register Descriptions

The register region of the Tracer is mapped in the BE-U1000 memory map as the following tables show.

Table 16-1 Memory address regions for Tracer registers

Device	Size	Start Address	End Address	Access
Encoder Block	4 KB	0x1510_4000	0x1510_4FFF	Core 0...2
RAM Control Block	4 KB	0x1510_5000	0x1510_5FFF	
Pin Interface Block (PIB)	4 KB	0x1510_6000	0x1510_6FFF	
SRAM	4 KB	0x1510_7000	0x1510_7FFF	

SRAM address space is used for fast direct access to read trace data.

Table 16-2 Memory address region for Trace Select register

Device	Size	Start Address	End Address	Access
Trace Select	4 B	0x4000_A014	0x4000_A017	Only Core 2

Trace source selection (i.e., choosing whether Core 0 or Core 1 provides the trace data) is performed via a multiplexer controlled by the logical OR of the `TRACE_SEL.ALTTRACESEL` bit and the `SYSCR0.TRACESEL` bit of the CRU register.

You can find list and description of the registers in the Register Maps and Register Descriptions sections below.

16.1.1. Register Maps

This section covers the following topics:

- Trace Select Register Map
- Encoder Block Register Map
- RAM Control Block Register Map
- Pin Interface Block (PIB Interface) Register Map

16.1.1.1. Trace Select Register Map

The following table provides a high-level summary of the register. To view the detailed description of the register, click the register name in the **Description** column.

Table 16-3 Trace Select register map

Name	Offset	Description	Default Value
TRACE_SEL	0x0	Trace Source Select Register	0x0

16.1.1.2. Encoder Block Register Map

The following table provides a high-level summary of each register. To view the detailed description of the register, click the register name in the **Description** column.

Table 16-4 Encoder Block register map

Name	Offset	Description	Default Value
TRTECONTROL	0x0	Control Register	0x1008008
TRTEIMPL	0x4	Encoder Block Implementation Parameters Register	0x10101
TRTEINSTFEATURES	0x8	Trace Instruction Features Register	0x0
TRTSCONTROL	0x40	Timestamp Control Register	0x20000000
TRTSCOUNTERLOW	0x48	Timestamp Counter Lower Bits	0x0
TRTSCOUNTERHIGH	0x4C	Timestamp Counter Upper Bits	0x0

16.1.1.3. RAM Control Block Register Map

The following table provides a high-level summary of each register. To view the detailed description of the register, click the register name in the **Description** column.

Table 16-5 RAM Control Block register map

Name	Offset	Description	Default Value
TRRAMCONTROL	0x0	RAM Memory Control Register	0x8
TRRAMIMPL	0x4	RAM Memory Implementation Parameters Register	0x1901
TRRAMSTARTLOW	0x10	RAM Memory Start Address Low Bits Register	0x0
TRRAMSTARTHIGH	0x14	RAM Memory Start Address High Bits Register	0x0
TRRAMLIMITLOW	0x18	RAM Memory End Address Low Bits Register	0x0
TRRAMLIMITHIGH	0x1C	RAM Memory End Address High Bits Register	0x0
TRRAMWPLOW	0x20	RAM Memory Write Pointer Address Low Bits Register	0x0
TRRAMWPHIGH	0x24	RAM Memory Write Pointer Address High Bits	0x0
TRRAMRPLOW	0x28	RAM Memory Read Pointer Address Low Bits Register	0x0
TRRAMRPHIGH	0x2C	RAM Memory Read Pointer Address High Bits Register	0x0
TRRAMDATA	0x40	RAM Memory Read Data Register	0x0

16.1.1.4. *Pin Interface Block* (PIB Interface) Register Map

The following table provides a high-level summary of the register. To view the detailed description of the register, click the register name in the **Description** column.

Table 16-6 PIB Block register map

Name	Offset	Description	Default Value
TRPIBCONTROL	0x0	PIB Block Control Register	0x1A8

16.1.2. Register Descriptions

16.1.2.1. Trace Select Register

16.1.2.1.1. Trace Source Select Register (TRACE_SEL)

The TRACE_SEL register is at offset 0x0.

The following table shows the register bit assignments.

Table 16-7 Fields for register: TRACE_SEL

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	ALTTRACESEL	R/W	Alternative trace source select. Trace source selection (i.e., choosing whether Core 0 or Core 1 provides the trace data) is performed via a multiplexer controlled by the logical OR of the ALTTRACESEL bit and the SYSCR0.TRACESEL bit of the CRU register. Valid values: **0x0:** Core 0 **0x1:** Core 1 **Value After Reset:** 0x0

16.1.2.2. Encoder Block Registers

16.1.2.2.1. Control Register (TRTECONTROL)

The TRTECONTROL register is at offset 0x0.

The following table shows the register bit assignments.

Table 16-8 Fields for register: TRTECONTROL

Bits	Name	R/W	Description
[31:27]			Reserved
[26:24]	TR_TE_FORMAT	R	Trace recording/protocol format. Valid values: **0x1:** Nexus format: 6 data bits (MDO) + 2 control bits (MSEO) **Value After Reset:** 0x1

Table 16-8 Fields for register: `TRTECONTROL` (continued)

Bits	Name	R/W	Description
[23:20]	`TR_TE_INST_SYNC_MAX`	R/W	The maximum interval (in units determined by `TR_TE_SYNC_MODE` bits below) between instruction trace synchronization messages/packets. Generate synchronization when count reaches $2^{(\texttt{TR\_TE\_INST\_SYNC\_MAX}+4)}$. If an instruction trace synchronization message/packet is generated for another reason, the internal counter should be reset. **Value After Reset:** 0x0
[17:16]	`TR_TE_SYNC_MODE`	R/W	Select the periodic instruction trace synchronization message/packet generation mechanism. At least one non-zero mechanism must be implemented. Valid values: **0x0:** Periodic synchronization disabled **0x1:** Tracing messages counting **0x2:** Core clock ticks counting **0x3:** Executed instructions counting (16 bit) Once the max value of periodic counter is reached, an instruction trace synchronization message/packet should be generated. **Value After Reset:** 0x0
[15]	`TR_TE_INHIBIT_SRC`	R	Adding a core number to the trace messages. Valid values: **0x0:** Messages/packets generated by the trace encoder include a message source field if the source width held in `TRTEINSTFEATURES.TR_TE_SRC_BITS` is not 0. **0x1:** Disable inclusion of source field in trace messages/packets. **Value After Reset:** 0x1
[14]			Reserved
[13]	`TR_TE_INST_STALL_EN`	R/W	Permission of `stall` signal generation in a case of internal queues overflow. Valid values: **0x0:** If Encoder Block cannot send a message, the message is dropped. The protocol dependent overflow instruction trace synchronization message/packet is generated when the trace is restarted, so the decoder will know that trace is lost and must reset any internal decoder state. **0x1:** If Encoder Block cannot send a message, the hart is stalled until it can. With this option execution of instructions by the hart may be intrusively affected, but in many cases it is acceptable.

Table 16-8 Fields for register: `TRTECONTROL` (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0
[12]	`TR_TE_INST_STALL_OR_OVERFLOW`	RW1C	Set to 1 by hardware when trace buffer overflow (also known as trace lost) occurs, or when the Encoder Block requests a hart stall. Clears to 0 at Encoder Block reset or when the trace is enabled (`TR_TE_ENABLE` bit below set to 1). Write 1 to clear. **Value After Reset:** 0x0
[11]	`TR_TE_TRIG_ENABLE`	R/W	Permission of `TR_TE_TRACING` control bit managing by the outer triggers. Valid values: **0x1:** Allows `TR_TE_TRACING` bit below to be set or cleared by Trace-on and Trace-off signals generated by the corresponding trigger module. **Value After Reset:** 0x0
[10]			Reserved
[9]	`TR_TE_CONTEXT`	R	Enable sending trace messages/fields with scontext/mcontext values and/or privilege levels. **Value After Reset:** 0x0
[8:7]			Reserved
[6:4]	`TR_TE_INST_MODE`	R/W	Instruction trace generation mode. Valid values: **0x0:** Full Instruction trace is disabled, but other trace (data trace) may be emitted **0x2:** Branch History Messaging **0x3:** Branch Trace messaging **Value After Reset:** 0x0
[3]	`TR_TE_EMPTY`	R	Internal queues fullness state. If this bit =1, this means that all generated messages are sent to selected devices and no new messages in present. **Value After Reset:** 0x1
[2]	`TR_TE_TRACING`	R/W	Trace generation permission. Valid values: **0x1:** Instruction trace is being generated. Written from a trace tool (after a write to `TR_TE_ENABLE` bit below) or controlled by triggers. **Value After Reset:** 0x0
[1]	`TR_TE_ENABLE`	R/W	Encoder Block enable. Valid values:

Table 16-8 Fields for register: `TRTECONTROL` (continued)

Bits	Name	R/W	Description
			0x0: If this bit is set to 0, all internal queues will sent their messages to the selected devices and a new messages generation will be stopped. **0x1:** Encoder Block is enabled. This bit can be set to 1 only by direct writing to it. This write of 1 should be done after all other settings are done. **Value After Reset:** 0x0
[0]	`TR_TE_ACTIVE`	R/W	Trace system enable. If this bit is set to 0, all other registers are not accessible: when 0, the Encoder Block may have clocks gated off or be powered down, and other register locations may be inaccessible. Hardware may take an arbitrarily long time to process power-up and power-down and will indicate completion when the read value of this bit matches what was written. **Value After Reset:** 0x0

16.1.2.2.2. Encoder Block Implementation Parameters Register (`TRTEIMPL`)

The `TRTEIMPL` register is at offset 0x4.

The following table shows the register bit assignments.

Table 16-9 Fields for register: `TRTEIMPL`

Bits	Name	R/W	Description
[31:24]			Reserved
[23:20]	`TR_TE_PROTOCOL_MINOR`	R	Trace Protocol Minor Version. As specified by specification governing `TRTECONTROL.TR_TE_FORMAT`. **Value After Reset:** 0x0
[19:16]	`TR_TE_PROTOCOL_MAJOR`	R	Trace Protocol Major Version. As specified by specification governing `TRTECONTROL.TR_TE_FORMAT`. **Value After Reset:** 0x1
[15:12]			Reserved
[11:8]	`TR_TE_COMP_TYPE`	R	Component type. **Value After Reset:** 0x1
[7:4]	`TR_TE_VER_MINOR`	R	Encoder Block Minor Version. Value 0 means the component is compliant with ***RISC-V Trace Control Interface Specification, Version 1.0_rc42, July 22, 2024***. **Value After Reset:** 0x0

Table 16-9 Fields for register: TRTEIMPL (continued)

Bits	Name	R/W	Description
[3:0]	TR_TE_VER_MAJOR	R	Encoder Block Major Version. Value 1 means the component is compliant with ***RISC-V Trace Control Interface Specification, Version 1.0_rc42, July 22, 2024***. **Value After Reset:** 0x1

16.1.2.2.3. Trace Instruction Features Register (TRTEINSTFEATURES)

The TRTEINSTFEATURES register is at offset 0x8.

The following table shows the register bit assignments.

Table 16-10 Fields for register: TRTEINSTFEATURES

Bits	Name	R/W	Description
[31:28]	TR_TE_SRC_BITS	R	The number of bits in the trace source field, unless disabled by TRTECONTROL.TR_TE_INHIBIT_SRC. **Value After Reset:** 0x0
[27:16]	TR_TE_SRC_ID	R	Trace source ID assigned to this trace encoder. If TR_TE_SRC_BITS is not 0 and trace source is not disabled by TRTECONTROL.TR_TE_INHIBIT_SRC, then trace messages from this Trace Encoder will all include a trace source field of TR_TE_SRC_BITS bits and all messages from this Trace Encoder will use this value as trace source field. **Value After Reset:** 0x0
[15:0]			Reserved

16.1.2.2.4. Timestamp Control Register (TRTSCONTROL)

The TRTSCONTROL register is at offset 0x40.

The following table shows the register bit assignments.

Table 16-11 Fields for register: TRTSCONTROL

Bits	Name	R/W	Description
[31:30]			Reserved
[29:24]	TR_TS_WIDTH	R	Width of timestamp in bits. **Value After Reset:** 0x20
[23:16]			Reserved
[15]	TR_TS_ENABLE	R/W	Enable for timestamp field in trace messages/packets (for Trace Encoder only). **Value After Reset:** 0x0
[14:10]			Reserved
[9:8]	TR_TS_PRESCALE	R/W	Internal System or Core timestamp only.

Table 16-11 Fields for register: `TRTSCONTROL` (continued)

Bits	Name	R/W	Description
			Prescale timestamp input clock by 2^(2*`TR_TS_PRESCALE`). It will be divided by 1, 4, 16, 64 respectively. **Value After Reset:** 0x0
[7]			Reserved
[6:4]	`TR_TS_TYPE`	R/W	Mode used by Timestamp unit (clock source types): **0x0:** None (Encoder Block disabled) **0x1:** External **0x2:** Internal System Bus clock based **0x3:** Internal Core clock based **0x4:** Shared with other blocks counter **Value After Reset:** 0x0
[3]	`TR_TS_RUN_IN_DEBUG`	R/W	Counter run mode (even in debug mode): **0x0:** Stopped **0x1:** Counter runs when hart is halted (in debug mode) **Value After Reset:** 0x0
[2]	`TR_TS_RESET`	R/W	Timestamp counter reset bit — write 1 to reset. **Value After Reset:** 0x0
[1]	`TR_TS_COUNT`	R/W	Counter enable: **0x0:** Stopped **0x1:** Counter enable **Value After Reset:** 0x0
[0]	`TR_TS_ACTIVE`	R/W	Primary activate/reset bit for timestamp unit. If this bit is 0 — all other registers are not accessible. **Value After Reset:** 0x0

16.1.2.2.5. Timestamp Counter Lower Bits (`TRTSCOUNTERLOW`)

The `TRTSCOUNTERLOW` register is at offset 0x48.

The following table shows the register bit assignments.

Table 16-12 Fields for register: `TRTSCOUNTERLOW`

Bits	Name	R/W	Description
[31:0]	`TR_TS_COUNTER_LOW`	R	Lower 32 bits of timestamp counter. **Value After Reset:** 0x0

16.1.2.2.6. Timestamp Counter Upper Bits (`TRTSCOUNTERHIGH`)

The `TRTSCOUNTERHIGH` register is at offset 0x4C.

The following table shows the register bit assignments.

Table 16-13 Fields for register: `TRTSCOUNTERHIGH`

Bits	Name	R/W	Description
[31:0]	`TR_TS_COUNTER_HIGH`	R	Upper bits of timestamp counter, zero-extended. **Value After Reset:** 0x0

16.1.2.3. RAM Control Block Registers

16.1.2.3.1. RAM Memory Control Register (`TRRAMCONTROL`)

The `TRRAMCONTROL` register is at offset 0x0.

The following table shows the register bit assignments.

Table 16-14 Fields for register: `TRRAMCONTROL`

Bits	Name	R/W	Description
[31:15]			Reserved
[14:12]	`TR_RAM_ASYNQ_FREQ`	R	Synchronization of aligning packets size control. Valid values: **0x0:** Alignment synchronization (Async) packets disabled (may be the only choice for some protocols) **0x1 to 0x7:** Different levels of alignment synchronization (bigger number, bigger distance) Details should be defined in the specification of each trace protocol. **Value After Reset:** 0x0
[11:9]			Reserved
[8]	`TR_RAM_STOP_ON_WRAP`	R/W	RAM Control Block disable in a case of it cyclic buffer is full (the `TR_RAM_ENABLE` field below of this register will be 0). **Value After Reset:** 0x0
[7:5]			Reserved
[4]	`TR_RAM_MODE`	R/W	RAM Control Block operating mode. Valid values: **0x0:** SMEM mode **0x1:** SRAM mode SRAM — is a cyclic internal memory buffer. This memory can be accessed through the RAM registers or directly, through the SRAM interface (the base address is 0x14000000). This memory can be accessed even if the processor core is in the debug mode or halted. SMEM — is an outer cyclic system memory buffer. To store data in this memory, need to

Table 16-14 Fields for register: `TRRAMCONTROL` (continued)

Bits	Name	R/W	Description
			share address range and set it in the RAM control registers. **Value After Reset:** 0x0
[3]	`TR_RAM_EMPTY`	R	Reads 1 when Trace RAM Sink internal buffers are empty, which means that all trace data is flushed. **Value After Reset:** 0x1
[2]			Reserved
[1]	`TR_RAM_ENABLE`	R/W	RAM Control Block enable. Valid values: **0x0:** If this bit is set to 0, all internal queues will be forced to transfer their data to selected transfer devices and generation of new messages will be stopped. **0x1:** Trace RAM Sink enabled. **Value After Reset:** 0x0
[0]	`TR_RAM_ACTIVE`	R/W	RAM Control Block activating. If this bit is 0, all other bits are unaccessible: when 0, the Trace RAM Sink may have clocks gated off or be powered down, and other register locations may be inaccessible. Hardware may take an arbitrarily long time to process power-up and power-down and will indicate completion when the read value of this bit matches what was written. **Value After Reset:** 0x0

16.1.2.3.2. RAM Memory Implementation Parameters Register (`TRRAMIMPL`)

The `TRRAMIMPL` register is at offset 0x4.

The following table shows the register bit assignments.

Table 16-15 Fields for register: `TRRAMIMPL`

Bits	Name	R/W	Description
[31:14]			Reserved
[13]	`TR_RAM_HAS_SMEM`	R	SMEM mode supported or not. **Value After Reset:** 0x0
[12]	`TR_RAM_HAS_SRAM`	R	SRAM mode supported or not. **Value After Reset:** 0x1
[11:8]	`TR_RAM_COMP_TYPE`	R	Trace RAM Sink Component Type (RAM Sink). **Value After Reset:** 0x9
[7:4]	`TR_RAM_VER_MINOR`	R	Trace RAM Sink Component Minor Version. Value 0 means the component is compliant with ***RISC-V Trace Control Interface Specification, Version 1.0_rc42, July 22, 2024***.

Table 16-15 Fields for register: `TRRAMIMPL` (continued)

Bits	Name	R/W	Description
			Value After Reset: 0x0
[3:0]	TR_RAM_VER_MAJOR	R	Trace RAM Sink Component Major Version. Value 1 means the component is compliant with ***RISC-V Trace Control Interface Specification, Version 1.0_rc42, July 22, 2024***. **Value After Reset:** 0x1

16.1.2.3.3. RAM Memory Start Address Low Bits Register (`TRRAMSTARTLOW`)

The `TRRAMSTARTLOW` register is at offset 0x10.

The following table shows the register bit assignments.

Table 16-16 Fields for register: `TRRAMSTARTLOW`

Bits	Name	R/W	Description
[31:0]	TR_RAM_START_LOW	R/W	RAM memory (cyclic buffer) start address low bits. **Value After Reset:** 0x0

16.1.2.3.4. RAM Memory Start Address High Bits Register (`TRRAMSTARTHIGH`)

The `TRRAMSTARTHIGH` register is at offset 0x14.

The following table shows the register bit assignments.

Table 16-17 Fields for register: `TRRAMSTARTHIGH`

Bits	Name	R/W	Description
[31:0]	TR_RAM_START_HIGH	R/W	RAM memory (cyclic buffer) start address high bits. **Value After Reset:** 0x0

16.1.2.3.5. RAM Memory End Address Low Bits Register (`TRRAMLIMITLOW`)

The `TRRAMLIMITLOW` register is at offset 0x18.

The following table shows the register bit assignments.

Table 16-18 Fields for register: `TRRAMLIMITLOW`

Bits	Name	R/W	Description
[31:0]	TR_RAM_LIMIT_LOW	R/W	RAM memory (cyclic buffer) end address low bits. **Value After Reset:** 0x0

16.1.2.3.6. RAM Memory End Address High Bits Register (`TRRAMLIMITHIGH`)

The `TRRAMLIMITHIGH` register is at offset 0x1C.

The following table shows the register bit assignments.

Table 16-19 Fields for register: `TRRAMLIMITHIGH`

Bits	Name	R/W	Description
[31:0]	TR_RAM_LIMIT_HIGH	R/W	RAM memory (cyclic buffer) end address high bits. **Value After Reset:** 0x0

16.1.2.3.7. RAM Memory Write Pointer Address Low Bits Register (`TRRAMWPLOW`)

The `TRRAMWPLOW` register is at offset 0x20.

The following table shows the register bit assignments.

Table 16-20 Fields for register: `TRRAMWPLOW`

Bits	Name	R/W	Description
[31:1]	TR_RAM_WP_LOW	R/W	RAM memory (cyclic buffer) write pointer address low bits. **Value After Reset:** 0x0
[0]	TR_RAM_WRAP	RC1	RAM memory (cyclic buffer) is full flag. **Value After Reset:** 0x0

16.1.2.3.8. RAM Memory Write Pointer Address High Bits Register (`TRRAMWPHIGH`)

The `TRRAMWPHIGH` register is at offset 0x24.

The following table shows the register bit assignments.

Table 16-21 Fields for register: `TRRAMWPHIGH`

Bits	Name	R/W	Description
[31:0]	TR_RAM_WP_HIGH	R/W	RAM memory (cyclic buffer) write pointer address high bits. **Value After Reset:** 0x0

16.1.2.3.9. RAM Memory Read Pointer Address Low Bits Register (`TRRAMRPLOW`)

The `TRRAMRPLOW` register is at offset 0x28.

The following table shows the register bit assignments.

Table 16-22 Fields for register: `TRRAMRPLOW`

Bits	Name	R/W	Description
[31:0]	TR_RAM_RP_LOW	R/W	RAM memory (cyclic buffer) read pointer address low bits. **Value After Reset:** 0x0

16.1.2.3.10. RAM Memory Read Pointer Address High Bits Register (`TRRAMRPHIGH`)

The `TRRAMRPHIGH` register is at offset 0x2C.

The following table shows the register bit assignments.

Table 16-23 Fields for register: `TRRAMRPHIGH`

Bits	Name	R/W	Description
[31:0]	`TR_RAM_RP_HIGH`	R/W	RAM memory (cyclic buffer) read pointer address high bits. **Value After Reset:** 0x0

16.1.2.3.11. RAM Memory Read Data Register (`TRRAMDATA`)

The `TRRAMDATA` register is at offset 0x40.

The following table shows the register bit assignments.

Table 16-24 Fields for register: `TRRAMDATA`

Bits	Name	R/W	Description
[31:0]	`TR_RAM_DATA`	R	Read from RAM memory (cyclic buffer) data. **Value After Reset:** 0x0

16.1.2.4. PIB Block Registers

16.1.2.4.1. PIB Block Control Register (`TRPIBCONTROL`)

The `TRPIBCONTROL` register is at offset 0x0.

The following table shows the register bit assignments.

Table 16-25 Fields for register: `TRPIBCONTROL`

Bits	Name	R/W	Description
[31:16]	`TR_PIB_DIVIDER`	R	PIB Interface clock divider value, that is constant and always = 0. This means that the divider is disactivated. **Value After Reset:** 0x0
[15:10]			Reserved
[9]	`TR_PIB_CALIBRATE`	R/W	Set this to 1 to generate a repeating calibration pattern to help tune a probe's signal delays, bit rate, etc. In this mode input to the sink is not consumed. **Value After Reset:** 0x0
[8]	`TR_PIB_CLK_CENTER`	R	Support of message transfer mode, where the debugger can vary clock periods (set it in the middle of the data stream). This value is constant and equal to 1. This can be set only if `TR_PIB_MODE` bit below selects one of the parallel protocols. **Value After Reset:** 0x1
[7:4]	`TR_PIB_MODE`	R	Select mode for output pins. Valid values: **0xA**: (means 10 in decimal) Parallel mode*

Table 16-25 Fields for register: `TRPIBCONTROL` (continued)

Bits	Name	R/W	Description
			*For description on this mode, refer to ***RISC-V Control interface Specification, Version 1.0_rc42, July 22, 2024***. **Value After Reset:** 0xA
[3]	`TR_PIB_EMPTY`	R	Reads 1 when PIB internal buffers are empty. **Value After Reset:** 0x1
[2]			Reserved
[1]	`TR_PIB_ENABLE`	R/W	Enable of data transmitting via the PIB interface. Valid values: **0x0:** PIB does not accept input but holds output(s) at idle state defined by PIB Mode. **0x1:** Enable PIB to generate output. **Value After Reset:** 0x0
[0]	`TR_PIB_ACTIVE`	R/W	Activating of PIB Interface. When 0, the PIB Sink may have clocks gated off or be powered down, and other register locations may be inaccessible. Hardware may take an arbitrarily long time to process power-up and power-down and will indicate completion when the read value of this bit matches what was written. **Value After Reset:** 0x0

17. Bus Matrix

This chapter describes the Bus matrix of the BE-U1000 microcontroller.

The BE-U1000 microcontroller contains one Bus matrix.

The Bus matrix provides the bus fabric to connect Bus matrix masters and slaves to form part of a *System-on-Chip* (**SoC**) bus solution.

17.1. Bus Matrix Registers

The register region of the Bus matrix is mapped in the BE-U1000 memory map as the following table shows.

Table 17-1 Memory address region for Bus matrix registers

Region	Block Name	Size	Start Address	End Address
High-speed Periphery	Bus matrix	4 KB	0x1510_8000	0x1510_8FFF

For details, refer to Memory Map chapter.

This section covers the following topics:

- Register Map
- Register Descriptions

In this section, the register offsets are given relative to the start address (0x15108000).

17.1.1. Register Map

The following table provides a top-level summary of each register. To view the detailed description of a register, click the register name in the **Description** column.

Table 17-2 Bus matrix register map

Name	Offset	Description	Default Value
PL1	0x00	Arbitration Priority Master 1 Register	0x1
PL2	0x04	Arbitration Priority Master 2 Register	0x2
EBTCOUNT	0x3C	Early Burst Termination Count Register	0x0
EBT_EN	0x40	Early Burst Termination Enable	0x0
EBT	0x44	Early Burst Termination Register	0x0
DFLT_MASTER	0x48	Default Master ID Number Register	0x0

17.1.2. Register Descriptions

In the tables below, the **R/W** column specifies how the application and the core can access the register fields.

17.1.2.1. Arbitration Priority Master 1 Register (PL1)

The PL1 register is at offset 0x0.

The following table shows the register bit assignments.

Table 17-3 Fields for register: PL1

Bits	Name	R/W	Description
[31:4]			Reserved
[3:0]	PRIORITY_LEVEL1	R/W	Arbitration priority for master 1 register used for programming the Core 0 and Core 1 priorities. The priority level ranges from 0 to 15 (in decimal), with a priority 0 signifying that the master has been disabled. The highest value indicates the highest priority. **Value After Reset:** 0x1

17.1.2.2. Arbitration Priority Master 2 Register (PL2)

The PL2 register is at offset 0x4.

The following table shows the register bit assignments.

Table 17-4 Fields for register: PL2

Bits	Name	R/W	Description
[31:4]			Reserved
[3:0]	PRIORITY_LEVEL2	R/W	Arbitration priority for master 2 register used for programming the Core 2 priority. The priority level ranges from 0 to 15 (in decimal), with a priority 0 signifying that the master has been disabled. The highest value indicates the highest priority. **Value After Reset:** 0x2

17.1.2.3. Early Burst Termination Count Register (EBTCOUNT)

The EBTCOUNT register is at offset 0x3C.

The following table shows the register bit assignments.

Table 17-5 Fields for register: EBTCOUNT

Bits	Name	R/W	Description
[31:10]			Reserved
[9:0]	EBTCOUNT	R/W	Early burst termination count register. Maximum number of cycles a transfer can take before being subject to an early burst termination. **Value After Reset:** 0x0

17.1.2.4. Early Burst Termination Enable (EBT_EN)

The EBT_EN register is at offset 0x40.

The following table shows the register bit assignments.

Table 17-6 Fields for register: `EBT_EN`

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	EBT_EN	R/W	Early burst termination enable is to enable or disable *Early Burst Termination* (**EBT**) through the software programming. **Values:** **0x0**: Disables the Early Burst Termination **0x1**: Enables the Early Burst Termination **Value After Reset:** 0x0

17.1.2.5. Early Burst Termination Register (`EBT`)

The `EBT` register is at offset 0x44.

The following table shows the register bit assignments.

Table 17-7 Fields for register: `EBT`

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	EBT	R	Early burst termination register. Set when an Early Burst Termination takes place. The register is cleared when read by the processor. **Values:** **0x0**: No Early Burst Termination Occurred **0x1**: Early Burst Termination Occured **Value After Reset:** 0x0

17.1.2.6. Default Master ID Number Register (`DFLT_MASTER`)

The `DFLT_MASTER` register is at offset 0x48.

The following table shows the register bit assignments.

Table 17-8 Fields for register: `DFLT_MASTER`

Bits	Name	R/W	Description
[31:4]			Reserved
[3:0]	DFLT_MASTER	R	Default master ID number register. The default master is the master that is granted by the bus when no master has requested ownership. **Value After Reset:** 0x0