

# BE-U1000

## Reference Manual

---

Document ID: BE-U-RM#1740

BAIKAL ELECTRONICS, JSC. Registration code: 7707767484

Revision 1.0.0

December 26, 2025

The content of the document is protected under the copyright law of the Russian Federation and International Copyright Law. BAIKAL ELECTRONICS JSC is the holder of exclusive rights to this document.

This document should not be modified in any way unless prior written consent is granted by BAIKAL ELECTRONICS JSC.

BAIKAL ELECTRONICS JSC reserves the right to modify or complement this and related documents without prior notice.

BAIKAL ELECTRONICS JSC is not liable for any losses that may be incurred due to using this and related documents.

The providing of this document does not imply a transfer of copyright or any related rights.

All trademarks belong to respective owners.

## Revision History

Revision	Date	Description
1.0.0	December 26, 2025	Initial Release

## Glossary

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

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

## Table of contents

List of Tables.....	27
List of Figures.....	41
<b>Glossary.....</b>	<b>4</b>
<b>1. Introduction.....</b>	<b>43</b>
<b>2. Architecture.....</b>	<b>44</b>
<b>3. Memory Map.....</b>	<b>45</b>
<b>4. Interrupt List.....</b>	<b>50</b>
4.1. Core 0/1 Interrupts.....	50
4.2. Core 2 Interrupts.....	54
<b>5. Control Registers Unit.....</b>	<b>55</b>
5.1. CRU Functional Description .....	55
5.1.1. Clock Generation.....	55
5.1.1.1. Clock Management .....	55
5.1.1.1.1. Clock Sources.....	57
5.1.1.1.2. PLL.....	59
5.1.1.1.2.1. PLL Output Frequency.....	59
5.1.1.1.2.2. PLL Stability Tracking.....	60
5.1.1.1.3. Dividers.....	61
5.1.1.1.4. Output Clocks.....	61
5.1.1.2. Clock Domains .....	62
5.1.1.2.1. <code>cclk</code> Clock Domain.....	62
5.1.1.2.2. <code>pclk_0</code> Clock Domain.....	63
5.1.1.2.3. <code>pclk_1</code> Clock Domain.....	64
5.1.1.2.4. <code>pclk_2</code> Clock Domain.....	65
5.1.1.2.5. <code>hclk</code> Clock Domain.....	66
5.1.1.2.6. <code>can_clk</code> and <code>can_clk_x2</code> Clock Domains.....	66
5.1.1.2.7. <code>tclk</code> Clock Domain.....	67
5.1.2. Reset Sources .....	67
5.1.3. I/O Pins Configuration.....	68
5.1.3.1. Alternate Functions.....	69
5.1.3.2. Input Enable.....	70
5.1.3.3. Drive Strength.....	70
5.1.3.4. Pull-Up and Pull-Down Resistors.....	70
5.2. Control Registers.....	71
5.2.1. Registers Map.....	71
5.2.2. Register Descriptions.....	72
5.2.2.1. Clock Select Register ( <code>CLKSEL</code> ).....	72
5.2.2.2. PLL Control Register ( <code>PLLSET</code> ).....	74
5.2.2.3. Clock Control Register 0 ( <code>CLKCR0</code> ).....	75
5.2.2.4. <code>pclk0</code> Clock Gating and Resets Control Register ( <code>PCLK0EN</code> ).....	77
5.2.2.5. <code>pclk1</code> Clock Gating and Resets Control Register ( <code>PCLK1EN</code> ).....	81

5.2.2.6. <code>pclk2</code> Clock Gating and Resets Control Register ( <code>PCLK2EN</code> ).....	84
5.2.2.7. System Control Register 0 ( <code>SYSCR0</code> ).....	86
5.2.2.8. System Control Register 1 ( <code>SYSCR1</code> ).....	88
5.2.2.9. System Control Register 2 ( <code>SYSCR2</code> ).....	89
5.2.2.10. Main Bus Matrix Priority Control Register 0 ( <code>PRIOR0</code> ).....	89
5.2.2.11. Main Bus Matrix Priority Control Register 1 ( <code>PRIOR1</code> ).....	90
5.2.2.12. I/O Pull-Up Control Register 0 ( <code>IOPUCR0</code> ).....	92
5.2.2.13. I/O Pull-Up Control Register 1 ( <code>IOPUCR1</code> ).....	93
5.2.2.14. I/O Pull-Down Control Register 0 ( <code>IOPDCR0</code> ).....	93
5.2.2.15. I/O Pull-Down Control Register 1 ( <code>IOPDCR1</code> ).....	94
5.2.2.16. I/O Alternate Function Control Register 0 ( <code>IOAFCR0</code> ).....	94
5.2.2.17. I/O Alternate Function Control Register 1 ( <code>IOAFCR1</code> ).....	96
5.2.2.18. I/O Alternate Function Control Register 2 ( <code>IOAFCR2</code> ).....	98
5.2.2.19. I/O Alternate Function Control Register 3 ( <code>IOAFCR3</code> ).....	100
5.2.2.20. I/O Alternate Function Control Register 4 ( <code>IOAFCR4</code> ).....	102
5.2.2.21. I/O Alternate Function Control Register 5 ( <code>IOAFCR5</code> ).....	104
5.2.2.22. I/O Drive Strength Control Register 0 ( <code>IODSCR0</code> ).....	106
5.2.2.23. I/O Drive Strength Control Register 1 ( <code>IODSCR1</code> ).....	107
5.2.2.24. I/O Drive Strength Control Register 2 ( <code>IODSCR2</code> ).....	108
5.2.2.25. LDO Control Register ( <code>LDOCR</code> ).....	109
5.2.2.26. USB PHY control and status register ( <code>USBCR</code> ).....	110
5.2.2.27. System Status Register ( <code>SYSSR0</code> ).....	112
5.2.2.28. WDT Clear Key Register ( <code>WDTCLRKEY</code> ).....	113
5.2.2.29. Interrupt Source Control Register ( <code>INTCR0</code> ).....	113
5.2.2.30. Clock Control Register 1 ( <code>CLKCR1</code> ).....	114
5.2.2.31. Flash NVR Control Register ( <code>FLASHNVRCR</code> ).....	115
5.2.2.32. Core 1 Control Register ( <code>CORE1CR</code> ).....	115
<b>6. RISC-V Cores.....</b>	<b>116</b>
6.1. BR-350 Core .....	116
6.1.1. BR-350 Core Functional Description .....	117
6.1.1.1. Non-Maskable Interrupts ( <code>int_nmi</code> ).....	117
6.1.1.2. Physical Memory Protection.....	117
6.1.1.3. Hardware Performance Monitor.....	117
6.1.1.4. Debug.....	119
6.1.1.4.1. Core Debug Mode.....	120
6.1.1.4.2. Trigger Module.....	122
6.1.1.5. CLINT.....	122
6.1.1.6. Bus Error Unit .....	122
6.1.1.7. CLIC .....	122
6.1.2. BR-350 Core Registers.....	122
6.1.2.1. Control and Status Registers.....	123
6.1.2.1.1. CLIC CSRs.....	123
6.1.2.1.1.1. CSRs Map.....	123
6.1.2.1.2. Trigger Module CSRs.....	123

6.1.2.1.2.1. CSRs Map.....	123
6.1.2.1.3. Core Debug CSRs.....	124
6.1.2.1.3.1. CSRs Map.....	124
6.1.2.1.4. Core CSRs.....	124
6.1.2.1.4.1. CSRs Map.....	124
6.1.2.1.4.2. CSR Descriptions.....	125
6.1.2.1.4.2.1. Machine NMI Trap Handler Address ( <code>NMITVEC</code> ).....	125
6.1.2.1.4.2.2. Core Configuration ( <code>XCCFG</code> ).....	125
6.1.2.1.4.2.3. Instruction Cache Configuration ( <code>XICFG</code> ).....	125
6.1.2.2. DTM Registers.....	126
6.1.2.2.1. Registers Map.....	126
6.1.2.3. DM Registers (access via DMI) .....	126
6.1.2.3.1. Registers Map.....	126
6.1.2.4. CLINT Registers.....	127
6.1.2.4.1. Registers Map.....	127
6.1.2.4.2. Register Descriptions.....	128
6.1.2.4.2.1. Machine Software Interrupt Pending ( <code>MSIPO</code> ).....	128
6.1.2.4.2.2. Machine Timer Compare ( <code>MTIMECMP0</code> ).....	128
6.1.2.4.2.3. Machine Timer Value ( <code>MTIME</code> ).....	128
6.1.2.5. Bus Error Unit Registers.....	129
6.1.2.5.1. Registers Map.....	129
6.1.2.5.2. Register Descriptions.....	129
6.1.2.5.2.1. CPU Status Register ( <code>STATUS_CPU</code> ).....	129
6.1.2.5.2.2. CPU Interrupt Control Register ( <code>CONTROL_CPU</code> ).....	130
6.1.2.5.2.3. CPU Error Address Low Register ( <code>ADDR_LO_CPU</code> ).....	130
6.1.2.5.2.4. CPU Error Address High Register ( <code>ADDR_HI_CPU</code> ).....	130
6.1.2.6. CLIC Registers.....	131
6.1.2.6.1. Registers Map.....	131
6.2. BM-310 Core .....	132
6.2.1. BM-310 Core Functional Description .....	132
6.2.1.1. Debug.....	132
6.2.1.1.1. Core Debug Mode.....	133
6.2.1.1.2. Trigger Module.....	134
6.2.1.2. CLINT.....	135
6.2.2. BM-310 Core Registers.....	135
6.2.2.1. Control and Status Registers.....	135
6.2.2.1.1. Trigger Module CSRs.....	135
6.2.2.1.1.1. CSRs Map.....	135
6.2.2.1.2. Core Debug CSRs.....	136
6.2.2.1.2.1. CSRs Map.....	136
6.2.2.1.3. Core CSRs.....	136
6.2.2.1.3.1. CSRs Map.....	136
6.2.2.1.3.2. CSR Descriptions.....	136
6.2.2.1.3.2.1. Core Configuration ( <code>XCCFG</code> ).....	136

6.2.2.2. DTM Registers.....	137
6.2.2.2.1. Registers Map.....	137
6.2.2.3. DM Registers (access via DMI) .....	137
6.2.2.3.1. Registers Map.....	137
6.2.2.4. CLINT Registers.....	138
6.2.2.4.1. Registers Map.....	138
6.2.2.4.2. Register Descriptions.....	139
6.2.2.4.2.1. Machine Software Interrupt Pending ( <code>MSIPO</code> ).....	139
6.2.2.4.2.2. Machine Timer Compare ( <code>MTIMECMP0</code> ).....	139
6.2.2.4.2.3. Machine Timer Value ( <code>MTIME</code> ).....	139
<b>7. Core 2 PIO.....</b>	<b>141</b>
7.1. PIO Registers.....	141
7.1.1. Registers Map.....	141
7.1.2. Register Descriptions.....	142
7.1.2.1. Interrupt Register ( <code>INT</code> ).....	142
7.1.2.2. Input Data Register ( <code>PIO_IN</code> ).....	142
7.1.2.3. Enable Register ( <code>PIO_EN</code> ).....	142
7.1.2.4. Output Data Register ( <code>PIO_OU</code> ).....	143
<b>8. DMA Controller.....</b>	<b>144</b>
8.1. DMA Functional Description.....	145
8.1.1. Handshaking Interface.....	145
8.1.2. Basic Interface Definitions.....	151
8.1.3. Memory Peripherals.....	152
8.1.4. Software Handshaking.....	152
8.1.5. Handshaking Interface Operation.....	153
8.1.5.1. Single Transaction Region.....	153
8.1.5.2. Early-Terminated Burst Transaction.....	154
8.1.5.3. Software Handshaking Operation.....	154
8.1.5.4. Single Transactions.....	155
8.1.6. Setting Up Transfers.....	155
8.1.7. Peripheral Burst Transaction Requests.....	156
8.1.7.1. Transmit Watermark Level and Transmit FIFO Underflow.....	156
8.1.7.2. Choosing the Transmit Watermark Level.....	156
8.1.7.3. Selecting <code>CTLx.DEST_MSIZE</code> and Transmit FIFO Overflow.....	157
8.1.7.4. Receive Watermark Level and Receive FIFO Overflow.....	157
8.1.7.5. Choosing the Receive Watermark level.....	157
8.1.7.6. Selecting <code>CTLx.SRC_MSIZE</code> and Receive FIFO Underflow.....	158
8.1.8. Generating Requests for the Master Bus Interface.....	158
8.2. DMA Registers.....	159
8.2.1. Registers Map.....	160
8.2.2. Register Descriptions.....	163
8.2.2.1. Channel Registers.....	163
8.2.2.1.1. Source Address Register ( <code>SARx</code> ) for Channel x.....	163
8.2.2.1.2. Destination Address Register ( <code>DARx</code> ) for Channel x .....	164

8.2.2.1.3. Hardware Realignment of SAR/DAR Registers.....	164
8.2.2.1.4. Linked List Pointer Register ( <code>LLPx</code> ) for Channel x .....	165
8.2.2.1.5. Control Register ( <code>CTLx</code> ) for Channel x.....	165
8.2.2.1.6. Configuration Register ( <code>CFGx</code> ) for Channel x.....	169
8.2.2.2. Interrupt Registers.....	173
8.2.2.2.1. Raw Status for IntTfr Interrupt ( <code>RAW_TFR</code> ).....	173
8.2.2.2.2. Raw Status for IntBlock Interrupt ( <code>RAW_BLOCK</code> ).....	173
8.2.2.2.3. Raw Status for IntSrcTran Interrupt ( <code>RAW_SRC_TRAN</code> ).....	174
8.2.2.2.4. Raw Status for IntDstTran Interrupt ( <code>RAW_DST_TRAN</code> ).....	174
8.2.2.2.5. Raw Status for IntErr Interrupt ( <code>RAW_ERR</code> ).....	175
8.2.2.2.6. Status for IntTfr Interrupt ( <code>STAT_TFR</code> ).....	175
8.2.2.2.7. Status for IntBlock Interrupt ( <code>STAT_BLOCK</code> ).....	176
8.2.2.2.8. Status for IntSrcTran Interrupt ( <code>STAT_SRC_TRAN</code> ).....	176
8.2.2.2.9. Status for IntDstTran Interrupt ( <code>STAT_DST_TRAN</code> ).....	176
8.2.2.2.10. Status for IntErr Interrupt ( <code>STAT_ERR</code> ).....	177
8.2.2.2.11. Mask for IntTfr Interrupt ( <code>MASK_TFR</code> ).....	177
8.2.2.2.12. Mask for IntBlock Interrupt ( <code>MASK_BLOCK</code> ).....	178
8.2.2.2.13. Mask for IntSrcTran Interrupt ( <code>MASK_SRC_TRAN</code> ).....	178
8.2.2.2.14. Mask for IntDstTran Interrupt ( <code>MASK_DST_TRAN</code> ).....	179
8.2.2.2.15. Mask for IntErr Interrupt ( <code>MASK_ERR</code> ).....	179
8.2.2.2.16. Clear for IntTfr Interrupt ( <code>CLR_TFR</code> ).....	180
8.2.2.2.17. Clear for IntBlock Interrupt ( <code>CLR_BLOCK</code> ).....	180
8.2.2.2.18. Clear for IntSrcTran Interrupt ( <code>CLR_SRC_TRAN</code> ).....	180
8.2.2.2.19. Clear for IntDstTran Interrupt ( <code>CLR_DST_TRAN</code> ).....	181
8.2.2.2.20. Clear for IntErr Interrupt ( <code>CLR_ERR</code> ).....	181
8.2.2.2.21. Status for each interrupt type ( <code>STAT</code> ).....	182
8.2.2.3. Software Handshaking Registers.....	183
8.2.2.3.1. Source Software Transaction Request Register ( <code>REQ_SRC</code> ).....	183
8.2.2.3.2. Destination Software Transaction Request Register ( <code>REQ_DST</code> ).....	183
8.2.2.3.3. Source Single Transaction Request Register ( <code>SGL_REQ_SRC</code> ).....	184
8.2.2.3.4. Destination Single Transaction Request register ( <code>SGL_REQ_DST</code> ).....	184
8.2.2.3.5. Source Last Transaction Request register ( <code>LST_SRC</code> ).....	185
8.2.2.3.6. Destination Last Transaction Request Register ( <code>LST_DST</code> ).....	186
8.2.2.4. Miscellaneous DMA Controller Registers.....	186
8.2.2.4.1. DMA Configuration Register ( <code>CFG</code> ).....	186
8.2.2.4.2. DMA Channel Enable Register ( <code>CH_EN</code> ).....	187
8.2.2.4.3. DMA Test Register ( <code>TEST</code> ).....	187
8.3. Programming the DMA .....	188
8.3.1. Register Access.....	188
8.3.2. DMA Controller Transfer Types.....	188
8.3.2.1. Multi-Block Transfers.....	189
8.3.2.1.1. Block Chaining Using Linked Lists.....	189
8.3.2.1.2. Auto-Reloading of Channel Registers.....	190
8.3.2.1.3. Contiguous Address Between Blocks.....	191

8.3.2.1.4. Suspension of Transfers Between Blocks.....	191
8.3.2.2. Ending Multi-Block Transfers.....	191
8.3.3. Programming the Linked List Multi-Block Transfer.....	192
8.3.4. Programming a Channel.....	193
8.3.4.1. Programming Examples.....	194
8.3.4.1.1. Single-block Transfer.....	194
8.3.4.1.2. Multi-Block Transfer with Linked List for Source and Linked List for Destination.....	195
8.3.5. Disabling a Channel Prior to Transfer Completion.....	197
8.3.5.1. Abnormal Transfer Termination.....	197
8.3.6. Defined-Length Burst Support on DMA Controller.....	198
<b>9. eFLASH Controller.....</b>	<b>199</b>
9.1. eFLASH Controller Registers.....	200
9.1.1. Registers Map.....	200
9.1.2. Register Descriptions.....	201
9.1.2.1. eFLASH Controller Control Register ( <code>EFLASH_CR</code> ).....	201
9.1.2.2. eFLASH Controller Address Register ( <code>EFLASH_ADDR</code> ).....	202
9.1.2.3. eFLASH Controller Write Data Register ( <code>EFLASH_WDATA</code> ).....	202
9.1.2.4. eFLASH Controller Read Data Register ( <code>EFLASH_RDATA</code> ).....	202
9.1.2.5. eFLASH Controller Status Register ( <code>EFLASH_SR</code> ).....	203
9.1.2.6. eFLASH Controller Time Intervals Register 0 ( <code>EFLASH_TIME0</code> ).....	203
9.1.2.7. eFLASH Controller Time Intervals Register 1 ( <code>EFLASH_TIME1</code> ).....	203
9.1.2.8. eFLASH Controller Time Intervals Register 2 ( <code>EFLASH_TIME2</code> ).....	204
9.1.2.9. eFLASH Controller Time Intervals Register 3 ( <code>EFLASH_TIME3</code> ).....	205
9.1.2.10. eFLASH Controller Time Intervals Register 4 ( <code>EFLASH_TIME4</code> ).....	205
9.1.2.11. eFLASH Controller Time Intervals Register 5 ( <code>EFLASH_TIME5</code> ).....	206
9.1.2.12. eFLASH Controller Time Intervals Register 6 ( <code>EFLASH_TIME6</code> ).....	207
9.2. Programming the eFLASH Controller.....	207
9.2.1. Time Intervals Configuration.....	207
9.2.2. Memory Configuration Operation (CFG).....	209
9.2.3. Memory Write Operation (WRITE).....	209
9.2.4. Memory Read Operation (READ).....	209
9.2.5. Chip Erase Operation (CHIP_ERASE).....	209
9.2.6. Block Erase Operation (BLOCK_ERASE).....	210
9.2.7. Sector Erase Operation (SECTOR_ERASE).....	210
9.2.8. Entering the DPD Mode Operation (ENTER_DPD).....	210
9.2.9. Exiting the DPD Mode Operation (EXIT_DPD).....	210
9.2.10. Direct access to the eFLASH Memory.....	210
<b>10. Mailbox.....</b>	<b>212</b>
10.1. Mailbox Registers.....	212
10.1.1. Registers Map.....	213
10.1.2. Register Descriptions.....	214
10.1.2.1. Messages Exchange Register 0 ( <code>MBnDATA0</code> ).....	214
10.1.2.2. Message Ready Flag Register 0 ( <code>MBnFULL0</code> ).....	214
10.1.2.3. Message Not Ready Flag Register 0 ( <code>MBnEMPTY0</code> ).....	215

10.1.2.4. Messages Exchange Register 1 ( <code>MBnDATA1</code> ).....	215
10.1.2.5. Message Ready Flag Register 1 ( <code>MBnFULL1</code> ).....	215
10.1.2.6. Message Not Ready Flag Register 1 ( <code>MBnEMPTY1</code> ).....	216
<b>11. ADC.....</b>	<b>217</b>
11.1. ADC Functional Description.....	217
11.1.1. ADC Pins.....	218
11.1.2. Operation Modes.....	218
11.1.2.1. Single Mode.....	219
11.1.2.2. Scan Mode.....	219
11.1.2.3. Discontinous Mode.....	220
11.1.2.4. Temperature Sensor Mode.....	220
11.1.2.4.1. Measurement Without Trimming or Calibration.....	220
11.1.2.4.2. Measurement With Trimming and Calibration.....	221
11.1.3. Conversion on PWMA Event.....	221
11.1.4. Direct Clocking.....	222
11.2. ADC Registers.....	222
11.2.1. Registers Map.....	223
11.2.2. Register Descriptions.....	223
11.2.2.1. ADC Control Register 1 ( <code>ADC_CR1</code> ).....	223
11.2.2.2. ADC Set Register ( <code>ASER</code> ).....	226
11.2.2.3. ADC Status Register ( <code>ASTR</code> ).....	227
11.2.2.4. ADC Sample Time Register ( <code>ASMPR</code> ).....	227
11.2.2.5. ADC Sequence Register ( <code>ASQR</code> ).....	229
11.2.2.6. ADC Data Register ( <code>ADR</code> ).....	230
<b>12. Timers.....</b>	<b>231</b>
12.1. PWMA.....	231
12.1.1. PWMA Functional Description.....	232
12.1.1.1. Inter-Block Trigger Connections.....	232
12.1.1.2. Control Unit.....	233
12.1.1.2.1. External Trigger Control.....	233
12.1.1.2.2. Trigger Selection.....	234
12.1.1.2.3. Slave Mode Controller.....	235
12.1.1.2.4. Master Mode Controller.....	236
12.1.1.3. Time-base Unit.....	237
12.1.1.3.1. Counter.....	238
12.1.1.3.2. Prescaler.....	238
12.1.1.3.3. Repetition Counter.....	238
12.1.1.4. Counter modes.....	239
12.1.1.4.1. Upcounting mode.....	239
12.1.1.4.2. Downcounting mode.....	239
12.1.1.4.3. Center-aligned mode (up/down counting).....	240
12.1.1.5. Capture/Compare Channels.....	241
12.1.1.5.1. Input Stage.....	242
12.1.1.5.2. Output Stage.....	243

12.1.1.5.3. Capture/Compare Stage.....	245
12.1.1.6. XOR Function.....	246
12.1.1.7. Dead-time Insertion.....	247
12.1.1.8. Break Function.....	247
12.1.1.9. Shadow Registers.....	248
12.1.1.10. Quadrature Encoder.....	248
12.1.1.10.1. Operation Modes.....	249
12.1.1.10.1.1. Quadrature Mode.....	249
12.1.1.10.1.2. <sup>qa</sup> Rising/Falling Edge Mode.....	249
12.1.1.10.2. Initialization and Reset.....	249
12.1.2. PWMA Registers.....	250
12.1.2.1. Registers Map.....	250
12.1.2.2. Register Descriptions.....	251
12.1.2.2.1. PWMA Control Register 1 ( <code>PWMA_CR1</code> ).....	251
12.1.2.2.2. PWMA Control Register 2 ( <code>PWMA_CR2</code> ).....	253
12.1.2.2.3. PWMA Slave Mode Control Register ( <code>PWMA_SMCR</code> ).....	256
12.1.2.2.4. PWMA DMA/Interrupt Enable Register ( <code>PWMA_DIER</code> ).....	259
12.1.2.2.5. PWMA Status Register ( <code>PWMA_SR</code> ).....	261
12.1.2.2.6. PWMA Event Generation Register ( <code>PWMA_EGR</code> ).....	265
12.1.2.2.7. PWMA Capture/Compare Mode Register 1 ( <code>PWMA_CCMR1</code> ).....	268
12.1.2.2.8. PWMA Capture/Compare Mode Register 2 ( <code>PWMA_CCMR2</code> ).....	273
12.1.2.2.9. PWMA Capture/Compare Enable Register ( <code>PWMA_CCER</code> ).....	277
12.1.2.2.10. PWMA Counter ( <code>PWMA_CNT</code> ).....	280
12.1.2.2.11. PWMA Prescaler ( <code>PWMA_PSC</code> ).....	280
12.1.2.2.12. PWMA Auto-reload Register ( <code>PWMA_ARR</code> ).....	280
12.1.2.2.13. PWMA Repetition Counter Register ( <code>PWMA_RCR</code> ).....	280
12.1.2.2.14. PWMA Capture/Compare Register 0 ( <code>PWMA_CCR0</code> ).....	281
12.1.2.2.15. PWMA Capture/Compare Register 1 ( <code>PWMA_CCR1</code> ).....	282
12.1.2.2.16. PWMA Capture/Compare Register 2 ( <code>PWMA_CCR2</code> ).....	282
12.1.2.2.17. PWMA Capture/Compare Register 3 ( <code>PWMA_CCR3</code> ).....	283
12.1.2.2.18. PWMA Break and Dead-time Register ( <code>PWMA_BDTR</code> ).....	283
12.1.2.2.19. PWMA DMA Control Register ( <code>PWMA_DCR</code> ).....	285
12.1.2.2.20. PWMA DMA Address for Full Transfer ( <code>PWMA_DMAR</code> ).....	286
12.1.2.2.21. PWMA Quadrature Encoder Setting Register ( <code>PWMA_QESET</code> ).....	286
12.1.2.2.22. PWMA Quadrature Encoder Filters Setting ( <code>PWMA_FILTSET</code> ).....	289
12.1.2.2.23. PWMA Quadrature Encoder Max Count Value ( <code>PWMA_QEMAX</code> ).....	290
12.1.2.2.24. PWMA Quadrature Encoder Init Value ( <code>PWMA_INITSET</code> ).....	290
12.1.2.2.25. PWMA Quadrature Encoder Counter Value ( <code>PWMA_QECNT</code> ).....	290
12.2. PWMG.....	291
12.2.1. PWMG Functional Description.....	291
12.2.1.1. Time-base Unit.....	291
12.2.1.1.1. Counter.....	292
12.2.1.1.2. Prescaler.....	292
12.2.1.2. Counter Upcounting Mode.....	293

12.2.1.3. Capture/Compare Channel.....	293
12.2.1.3.1. Input Stage.....	294
12.2.1.3.2. Output Stage.....	294
12.2.1.3.3. Capture/Compare Stage.....	295
12.2.1.4. Shadow Registers.....	296
12.2.2. PWMG Registers.....	296
12.2.2.1. Registers Map.....	296
12.2.2.2. Register Descriptions.....	297
12.2.2.2.1. PWMG Control Register 1 ( <code>PWMG_CR1</code> ).....	297
12.2.2.2.2. PWMG Interrupt Enable Register ( <code>PWMG_IER</code> ).....	298
12.2.2.2.3. PWMG Status Register ( <code>PWMG_SR</code> ).....	299
12.2.2.2.4. PWMG Event Generation Register ( <code>PWMG_EGR</code> ).....	300
12.2.2.2.5. PWMG Capture/Compare Mode Register 1 ( <code>PWMG_CCMR1</code> ).....	301
12.2.2.2.6. PWMG Capture/Compare Enable Register ( <code>PWMG_CCER</code> ).....	303
12.2.2.2.7. PWMG Counter ( <code>PWMG_CNT</code> ).....	304
12.2.2.2.8. PWMG Prescaler ( <code>PWMG_PSC</code> ).....	304
12.2.2.2.9. PWMG Auto-reload Register ( <code>PWMG_ARR</code> ).....	305
12.2.2.2.10. PWMG Capture/Compare Register 0 ( <code>PWMG_CCR0</code> ).....	305
12.3. TIM.....	305
12.3.1. TIM Functional Description.....	306
12.3.1.1. Timer Operation.....	306
12.3.1.1.1. To Use The Timer.....	306
12.3.1.1.2. To Enable a Timer.....	307
12.3.1.1.3. To Disable a Timer.....	307
12.3.1.1.4. To Load a Timer Countdown Value.....	307
12.3.1.1.5. To Work with Interrupts.....	308
12.3.1.1.6. Generating Toggled Outputs.....	308
12.3.1.1.6.1. Pulse Width Modulation of Toggle Outputs.....	308
12.3.1.1.6.2. Pulse Width Modulation with 0% and 100% Duty Cycle.....	308
12.3.1.1.6.3. Timer Pause Mode.....	310
12.3.2. TIM Registers.....	310
12.3.2.1. Registers Map.....	311
12.3.2.2. Register Descriptions.....	312
12.3.2.2.1. Timer N Load Count Register ( <code>TIMER&lt;N&gt;LOAD_COUNT</code> ).....	312
12.3.2.2.2. Timer N Load Count 2 Register ( <code>TIMER&lt;N&gt;LOAD_COUNT2</code> ).....	313
12.3.2.2.3. Timer N Current Value Register ( <code>TIMER&lt;N&gt;CURRENT_VALUE</code> ).....	313
12.3.2.2.4. Timer N Control Register ( <code>TIMER&lt;N&gt;CONTROL_REG</code> ).....	313
12.3.2.2.5. Timer N End-of-Interrupt Register ( <code>TIMER&lt;N&gt;EOI</code> ).....	314
12.3.2.2.6. Timer N Interrupt Status Register ( <code>TIMER&lt;N&gt;INT_STATUS</code> ).....	315
12.3.2.2.7. Timers Interrupt Status Register ( <code>TIMERS_INT_STATUS</code> ).....	315
12.3.2.2.8. Timers End-of-Interrupt Register ( <code>TIMERS_EOI</code> ).....	316
12.3.2.2.9. Timers Raw Interrupt Status Register ( <code>TIMERS_RAW_INT_STATUS</code> ).....	316
12.4. WDT.....	316
12.4.1. WDT Functional Description.....	317

12.4.1.1. Interrupts.....	318
12.4.1.2. Counter.....	318
12.4.1.3. System Resets.....	318
12.4.1.4. Reset Pulse Length.....	318
12.4.2. WDT Registers.....	318
12.4.2.1. Registers Map.....	319
12.4.2.2. Register Descriptions.....	319
12.4.2.2.1. Control Register ( <code>WDT_CR</code> ).....	319
12.4.2.2.2. Timeout Range Register ( <code>WDT_TORR</code> ).....	320
12.4.2.2.3. Current Counter Value Register ( <code>WDT_CCVR</code> ).....	321
12.4.2.2.4. Counter Restart Register ( <code>WDT_CRR</code> ).....	321
12.4.2.2.5. Interrupt Status Register ( <code>WDT_STAT</code> ).....	322
12.4.2.2.6. Interrupt Clear Register ( <code>WDT_EOI</code> ).....	322
12.4.2.2.7. Component Parameters Register 5 ( <code>WDT_COMP_PARAM_5</code> ).....	322
12.4.2.2.8. Component Parameters Register 4 ( <code>WDT_COMP_PARAM_4</code> ).....	323
12.4.2.2.9. Component Parameters Register 3 ( <code>WDT_COMP_PARAM_3</code> ).....	323
12.4.2.2.10. Component Parameters Register 2 ( <code>WDT_COMP_PARAM_2</code> ).....	323
12.4.2.2.11. Component Parameters Register 1 ( <code>WDT_COMP_PARAM_1</code> ).....	323
<b>13. Low Speed Peripherals.....</b>	<b>326</b>
13.1. GPIO.....	326
13.1.1. GPIO Functional Description.....	326
13.1.1.1. Data and Control Flow.....	326
13.1.1.1.1. Software Control.....	327
13.1.1.1.2. Reading External Signals.....	327
13.1.1.2. Interrupts.....	327
13.1.1.2.1. Debounce Operation.....	327
13.1.1.2.2. Level-Sensitive Interrupts.....	328
13.1.2. GPIO Registers.....	328
13.1.2.1. Programming Considerations.....	328
13.1.2.2. Registers Map.....	328
13.1.2.3. Register Descriptions.....	329
13.1.2.3.1. Data Output Register ( <code>GPIO_DR</code> ).....	329
13.1.2.3.2. Data Direction Control Register ( <code>GPIO_DDR</code> ).....	329
13.1.2.3.3. Interrupt Enable Register ( <code>GPIO_INTEN</code> ).....	330
13.1.2.3.4. Interrupt Mask Register ( <code>GPIO_INTMSK</code> ).....	330
13.1.2.3.5. Interrupt Level Register ( <code>GPIO_INTLVL</code> ).....	330
13.1.2.3.6. Interrupt Polarity Register ( <code>GPIO_INTPOLARITY</code> ).....	331
13.1.2.3.7. Interrupt Status Register ( <code>GPIO_INTSTAT</code> ).....	331
13.1.2.3.8. Raw Interrupt Status Register ( <code>GPIO_INTSTATRAW</code> ).....	331
13.1.2.3.9. Debounce Enable Register ( <code>GPIO_DEBOUNCE</code> ).....	332
13.1.2.3.10. Clear Interrupt Register ( <code>GPIO_EOI</code> ).....	332
13.1.2.3.11. External Port Register ( <code>GPIO_EXT</code> ).....	333
13.1.2.3.12. Synchronization Level Register ( <code>GPIO_SYNC</code> ).....	333
13.1.2.3.13. Interrupt Both Edge Type Register ( <code>GPIO_BOTHEDGE</code> ).....	333

13.2. CAN FD.....	334
13.2.1. CAN FD Functional Description.....	335
13.2.1.1. Operating Modes and States.....	335
13.2.1.2. Interrupts.....	338
13.2.2. CAN FD Registers.....	338
13.2.2.1. Registers Map.....	339
13.2.2.2. Register Descriptions.....	340
13.2.2.2.1. CAN FD Core Register Space.....	340
13.2.2.2.1.1. Software Reset Register ( <u>SRR</u> ).....	340
13.2.2.2.1.2. Mode Select Register ( <u>MSR</u> ).....	341
13.2.2.2.1.3. Arbitration Phase (Nominal) Baud Rate Prescaler Register ( <u>BRPR</u> ).....	344
13.2.2.2.1.4. Arbitration Phase (Nominal) Bit Timing Register ( <u>BTR</u> ).....	344
13.2.2.2.1.5. Error Count Register ( <u>ECR</u> ).....	345
13.2.2.2.1.6. Error Status Register ( <u>ESR</u> ).....	345
13.2.2.2.1.7. Status Register ( <u>SR</u> ).....	347
13.2.2.2.1.8. Interrupt Status Register ( <u>ISR</u> ).....	349
13.2.2.2.1.9. Interrupt Enable Register ( <u>IER</u> ).....	353
13.2.2.2.1.10. Interrupt Clear Register ( <u>ICR</u> ).....	356
13.2.2.2.1.11. Timestamp Register ( <u>TSR</u> ).....	359
13.2.2.2.1.12. Data Phase Baud Rate Prescaler Register ( <u>DP_BRPR</u> ).....	359
13.2.2.2.1.13. Data Phase Bit Timing Register ( <u>DP_BTR</u> ).....	360
13.2.2.2.1.14. TX Buffer Ready Request Register ( <u>TRR</u> ).....	361
13.2.2.2.1.15. Interrupt Enable TX Buffer Ready Request Served/Cleared ( <u>IETRS</u> ).....	362
13.2.2.2.1.16. TX Buffer Cancel Request ( <u>TCR</u> ).....	362
13.2.2.2.1.17. Interrupt Enable TX Buffer Cancellation Served/Cleared ( <u>IETCS</u> ).....	363
13.2.2.2.1.18. TX Event FIFO Status Register ( <u>TXE_FSR</u> ).....	363
13.2.2.2.1.19. TX Event FIFO Watermark Register ( <u>TXE_WMR</u> ).....	364
13.2.2.2.1.20. Acceptance Filter (Control) Register ( <u>AFR</u> ).....	365
13.2.2.2.1.21. RX FIFO Status Register ( <u>FSR</u> ).....	365
13.2.2.2.1.22. RX FIFO Watermark Register ( <u>WMR</u> ).....	366
13.2.2.2. CAN FD TX Message Space .....	367
13.2.2.2.1. TBi Message ID Register ( <u>TBi_ID</u> ).....	367
13.2.2.2.2. TBi Data Length Code Register ( <u>TBiDLC</u> ).....	368
13.2.2.2.3. TBi Data Byte n Register ( <u>TBiDWN</u> ).....	369
13.2.2.2.4. Acceptance Filter i Mask Register ( <u>AFMRI</u> ).....	369
13.2.2.2.5. Acceptance Filter i ID registers Register ( <u>AFIRi</u> ).....	371
13.2.2.2.3. CAN FD TX Event FIFO Status Register Space.....	371
13.2.2.2.3.1. TXE FIFO TBi Message ID Register ( <u>TXE_FIFO_TBi_ID</u> ).....	371
13.2.2.2.3.2. TXE FIFO TBi Data Length Code Register ( <u>TXE_FIFO_TBi_DLC</u> ).....	372
13.2.2.2.4. CAN FD RX Message Space .....	373
13.2.2.2.4.1. RBi Message ID Register ( <u>RBi_ID</u> ).....	373
13.2.2.2.4.2. RBi Data Length Code Register ( <u>RBiDLC</u> ).....	375
13.2.2.2.4.3. RBi Data Byte n Register ( <u>RBiDWN</u> ).....	375
13.3. UART.....	376

13.3.1. UART Functional Description.....	377
13.3.1.1. Programmable THRE Interrupt.....	377
13.3.1.2. UART Serial Protocol.....	377
13.3.1.3. 9-bit Data Transfer.....	378
13.3.1.3.1. Transmit Mode.....	378
13.3.1.3.1.1. Transmit Mode 0.....	378
13.3.1.3.1.2. Transmit Mode 1.....	379
13.3.1.3.2. Receive Mode.....	379
13.3.1.3.2.1. Hardware Address Match Receive Mode.....	379
13.3.1.3.2.2. Software Address Match Receive Mode.....	380
13.3.1.4. RS485 Support (For UART_6 and UART_7 Only).....	381
13.3.1.4.1. <del>de</del> Assertion and De-assertion Timing.....	381
13.3.1.4.2. RS485 Modes.....	381
13.3.1.5. Fractional Baud Rate Support.....	383
13.3.1.6. IrDA 1.0 SIR Support.....	384
13.3.2. UART Registers.....	385
13.3.2.1. Registers Map.....	385
13.3.2.2. Register Descriptions.....	387
13.3.2.2.1. Receive Buffer Register ( <u>RBR</u> ).....	387
13.3.2.2.2. Transmit Holding Register ( <u>THR</u> ).....	387
13.3.2.2.3. Divisor Latch Low Register ( <u>DLL</u> ).....	388
13.3.2.2.4. Divisor Latch High Register ( <u>DLH</u> ).....	388
13.3.2.2.5. Interrupt Enable Register ( <u>IER</u> ).....	389
13.3.2.2.6. Interrupt Identity Register ( <u>IIR</u> ).....	390
13.3.2.2.7. FIFO Control Register ( <u>FCR</u> ).....	391
13.3.2.2.8. Line Control Register ( <u>LCR</u> ).....	392
13.3.2.2.9. Mode Control Register ( <u>MCR</u> ).....	394
13.3.2.2.10. Line Status Register ( <u>LSR</u> ).....	394
13.3.2.2.11. Scratchpad Register ( <u>SCR</u> ).....	397
13.3.2.2.12. Shadow Receive Buffer Register ( <u>SRBRI</u> ).....	397
13.3.2.2.13. Shadow Transmit Holding Register ( <u>STHRI</u> ).....	397
13.3.2.2.14. FIFO Access Register ( <u>FAR</u> ).....	398
13.3.2.2.15. Transmit FIFO Read Register ( <u>TFR</u> ).....	398
13.3.2.2.16. Receive FIFO Write Register ( <u>RFW</u> ).....	399
13.3.2.2.17. UART Status Register ( <u>USR</u> ).....	399
13.3.2.2.18. Transmit FIFO Level Register ( <u>TFL</u> ).....	400
13.3.2.2.19. Receive FIFO Level Register ( <u>RFL</u> ).....	401
13.3.2.2.20. Software Reset Register ( <u>SRR</u> ).....	401
13.3.2.2.21. Shadow Break Control Register ( <u>SBCR</u> ).....	402
13.3.2.2.22. FIFO Enable Register ( <u>SFE</u> ).....	402
13.3.2.2.23. Shadow RCVR Trigger Register ( <u>SRT</u> ).....	402
13.3.2.2.24. Shadow Tx Empty Trigger Register ( <u>STET</u> ).....	403
13.3.2.2.25. Halt Tx Register ( <u>HTx</u> ).....	403
13.3.2.2.26. DMA Software Acknowledge Register ( <u>DMASA</u> ).....	404

13.3.2.2.27. Transceiver Control Register ( <b>TCR</b> ) .....	404
13.3.2.2.28. Driver Output Enable Register ( <b>DE_EN</b> ).....	405
13.3.2.2.29. Receiver Output Enable Register ( <b>RE_EN</b> ) .....	406
13.3.2.2.30. Driver Output Enable Timing Register ( <b>DET</b> ) .....	406
13.3.2.2.31. TurnAround Timing Register ( <b>TAT</b> ) .....	407
13.3.2.2.32. Divisor Latch Fraction Register ( <b>DLF</b> ).....	407
13.3.2.2.33. Receive Address Register ( <b>RAR</b> ).....	408
13.3.2.2.34. Transmit Address Register ( <b>TAR</b> ).....	408
13.3.2.2.35. Line Extended Control Register ( <b>LCR_EXT</b> ).....	409
13.3.2.2.36. Register Timeout Counter Reset Value Register ( <b>REG_TIMEOUT_RST</b> ).....	410
13.4. SPI & QSPI.....	410
13.4.1. *SPI Functional Description.....	411
13.4.1.1. *SPI Clock.....	412
13.4.1.2. Endian Conversion Support.....	412
13.4.1.3. Transmit and Receive FIFO Buffers.....	413
13.4.1.4. Transfer Modes.....	414
13.4.1.5. Dual Data-Rate Support (QSPI only).....	415
13.4.1.6. Interrupts.....	415
13.4.2. *SPI Registers.....	416
13.4.2.1. Registers Map.....	417
13.4.2.2. Register Descriptions.....	418
13.4.2.2.1. Control Register 0 ( <b>CTRLR0</b> ).....	418
13.4.2.2.2. Control Register 1 ( <b>CTRLR1</b> ).....	421
13.4.2.2.3. SPI Enable Register ( <b>SPIENR</b> ).....	422
13.4.2.2.4. Microwire Control Register ( <b>MWCR</b> ).....	422
13.4.2.2.5. Slave Enable Register ( <b>SER</b> ).....	423
13.4.2.2.6. Baud Rate Select Register ( <b>BAUDR</b> ).....	424
13.4.2.2.7. Transmit FIFO Threshold Level Register ( <b>TXFTLR</b> ).....	424
13.4.2.2.8. Receive FIFO Threshold Level Register ( <b>RXFTLR</b> ).....	425
13.4.2.2.9. Transmit FIFO Level Register ( <b>TXFLR</b> ).....	425
13.4.2.2.10. Receive FIFO Level Register ( <b>RXFLR</b> ).....	426
13.4.2.2.11. Status Register ( <b>SR</b> ).....	426
13.4.2.2.12. Interrupt Mask Register ( <b>IMR</b> ).....	428
13.4.2.2.13. Interrupt Status Register ( <b>ISR</b> ).....	429
13.4.2.2.14. Raw Interrupt Status Register ( <b>RISR</b> ).....	430
13.4.2.2.15. Transmit FIFO Overflow Interrupt Clear Register ( <b>TXOICR</b> ).....	431
13.4.2.2.16. Receive FIFO Overflow Interrupt Clear Register ( <b>RXOICR</b> ).....	431
13.4.2.2.17. Receive FIFO Underflow Interrupt Clear Register ( <b>RXUICR</b> ).....	432
13.4.2.2.18. Multi-Master Interrupt Clear Register ( <b>MSTICR</b> ).....	432
13.4.2.2.19. Interrupt Clear Register ( <b>ICR</b> ).....	432
13.4.2.2.20. DMA Control Register ( <b>DMACR</b> ).....	433
13.4.2.2.21. DMA Transmit Data Level Register ( <b>DMATDLR</b> ).....	433
13.4.2.2.22. DMA Receive Data Level Register ( <b>DMARDLR</b> ).....	434
13.4.2.2.23. Data Registers ( <b>DRI</b> ).....	434

13.4.2.2.24. RX Sample Delay Register ( <code>RX_SAMPLE_DLY</code> ).....	434
13.4.2.2.25. SPI Control Register ( <code>SPI_CTRLR0</code> ).....	435
13.4.2.2.26. Transmit Drive Edge Register ( <code>TXD_DRIVE_EDGE</code> ).....	437
13.4.2.2.27. XIP Command Register ( <code>XIP_CMD</code> ).....	437
13.5. I2C.....	438
13.5.1. I2C Functional Description.....	439
13.5.1.1. Addressing Slave Protocol.....	439
13.5.1.2. Operation Modes.....	440
13.5.1.2.1. Slave Mode Operation.....	440
13.5.1.2.2. Master Mode Operation.....	443
13.5.1.2.3. Disabling I2C Controller.....	444
13.5.1.2.4. Aborting I2C Transfers.....	445
13.5.1.3. Fast Mode Plus Operation.....	445
13.5.1.4. SDA Hold Time.....	445
13.5.1.4.1. SDA Hold Timings in Receiver.....	446
13.5.1.4.2. SDA Hold Timings in Transmitter.....	446
13.5.1.5. Interrupts.....	446
13.5.2. I2C Registers.....	448
13.5.2.1. Registers Map.....	448
13.5.2.2. Register Descriptions.....	450
13.5.2.2.1. I2C Control Register ( <code>IC_CON</code> ).....	450
13.5.2.2.2. I2C Target Address Register ( <code>IC_TAR</code> ).....	453
13.5.2.2.3. I2C Slave Address Register ( <code>IC_SAR</code> ).....	455
13.5.2.2.4. I2C High Speed Master Mode Code Address Register ( <code>IC_HS_MADDR</code> ).....	455
13.5.2.2.5. I2C Rx/Tx Data Buffer and Command Register ( <code>IC_DATA_CMD</code> ).....	456
13.5.2.2.6. Standard Speed I2C Clock SCL High Count Register ( <code>IC_SS_SCL_HCNT</code> ).....	457
13.5.2.2.7. Standard Speed I2C Clock SCL Low Count Register ( <code>IC_SS_SCL_LCNT</code> ).....	457
13.5.2.2.8. Fast Mode or Fast Mode Plus I2C Clock SCL High Count Register ( <code>IC_FS_SCL_HCNT</code> ).....	458
13.5.2.2.9. Fast Mode or Fast Mode Plus I2C Clock SCL Low Count Register ( <code>IC_FS_SCL_LCNT</code> ).....	458
13.5.2.2.10. High Speed I2C Clock SCL High Count Register ( <code>IC_HS_SCL_HCNT</code> ).....	459
13.5.2.2.11. High Speed I2C Clock SCL Low Count Register ( <code>IC_HS_SCL_LCNT</code> ).....	459
13.5.2.2.12. I2C Interrupt Status Register ( <code>IC_INTR_STAT</code> ).....	460
13.5.2.2.13. I2C Interrupt Mask Register ( <code>IC_INTR_MASK</code> ).....	462
13.5.2.2.14. I2C Raw Interrupt Status Register ( <code>IC_RAW_INTR_STAT</code> ).....	464
13.5.2.2.15. I2C Receive FIFO Threshold Register ( <code>IC_RX_TL</code> ).....	468
13.5.2.2.16. I2C Transmit FIFO Threshold Register ( <code>IC_TX_TL</code> ).....	469
13.5.2.2.17. Clear Combined and Individual Interrupt Register ( <code>IC_CLR_INTR</code> ).....	469
13.5.2.2.18. Clear <code>RX_UNDER</code> Interrupt Register ( <code>IC_CLR_RX_UNDER</code> ).....	470
13.5.2.2.19. Clear <code>RX_OVER</code> Interrupt Register ( <code>IC_CLR_RX_OVER</code> ).....	470
13.5.2.2.20. Clear <code>TX_OVER</code> Interrupt Register ( <code>IC_CLR_TX_OVER</code> ).....	470
13.5.2.2.21. Clear <code>RD_REQ</code> Interrupt Register ( <code>IC_CLR_RD_REQ</code> ).....	470
13.5.2.2.22. Clear <code>TX_ABRT</code> Interrupt Register ( <code>IC_CLR_TX_ABRT</code> ).....	471

13.5.2.2.23. Clear RX_DONE Interrupt Register ( <code>IC_CLR_RX_DONE</code> ).....	471
13.5.2.2.24. Clear ACTIVITY Interrupt Register ( <code>IC_CLR_ACTIVITY</code> ).....	471
13.5.2.2.25. Clear STOP_DET Interrupt Register ( <code>IC_CLR_STOP_DET</code> ).....	472
13.5.2.2.26. Clear START_DET Interrupt Register ( <code>IC_CLR_START_DET</code> ).....	472
13.5.2.2.27. Clear GEN_CALL Interrupt Register ( <code>IC_CLR_GEN_CALL</code> ).....	472
13.5.2.2.28. I2C ENABLE Register ( <code>IC_ENABLE</code> ).....	473
13.5.2.2.29. I2C STATUS Register ( <code>IC_STATUS</code> ).....	474
13.5.2.2.30. I2C Transmit FIFO Level Register ( <code>IC_TXFLR</code> ).....	477
13.5.2.2.31. I2C Receive FIFO Level Register ( <code>IC_RXFLR</code> ).....	477
13.5.2.2.32. I2C SDA Hold Time Length Register ( <code>IC_SDA_HOLD</code> ).....	478
13.5.2.2.33. I2C Transmit Abort Source Register ( <code>IC_TX_ABRT_SOURCE</code> ).....	478
13.5.2.2.34. DMA Control Register ( <code>IC_DMA_CR</code> ) .....	484
13.5.2.2.35. DMA Transmit Data Level Register ( <code>IC_DMA_TDLR</code> ).....	484
13.5.2.2.36. DMA Receive Data Level Register ( <code>IC_DMA_RDLR</code> ) .....	485
13.5.2.2.37. I2C SDA Setup Register ( <code>IC_SDA_SETUP</code> ).....	485
13.5.2.2.38. I2C ACK General Call Register ( <code>IC_ACK_GENERAL_CALL</code> ).....	486
13.5.2.2.39. I2C Enable Status Register ( <code>IC_ENABLE_STATUS</code> ).....	486
13.5.2.2.40. I2C SS, FS or FM+ Spike Suppression Limit ( <code>IC_FS_SPKLEN</code> ).....	488
13.5.2.2.41. I2C HS Spike Suppression Limit Register ( <code>IC_HS_SPKLEN</code> ) .....	489
13.5.2.2.42. I2C SCL Stuck at Low Timeout register ( <code>IC_SCL_STUCK_AT_LOW_TIMEOUT</code> ).....	489
13.5.2.2.43. I2C SDA Stuck at Low Timeout register ( <code>IC_SDA_STUCK_AT_LOW_TIMEOUT</code> ).....	490
13.5.2.2.44. Clear SCL Stuck at Low Detect interrupt Register ( <code>IC_CLR_SCL_STUCK_DET</code> ).....	490
13.5.2.2.45. Register Timeout Counter Reset Value ( <code>REG_TIMEOUT_RST</code> ).....	490
13.6. I2S.....	491
13.6.1. I2S Functional Description.....	491
13.6.1.1. I2S Controller Enable.....	492
13.6.1.2. I2S Controller as Transmitter.....	492
13.6.1.2.1. Transmitter Block Enable.....	493
13.6.1.2.2. Transmit Channel Enable.....	493
13.6.1.2.3. Transmit Channel Audio Data Resolution.....	493
13.6.1.2.4. Transmit Channel FIFOs.....	494
13.6.1.2.5. Transmit Channel Interrupts.....	494
13.6.1.2.6. Writing to the Transmit Channel.....	495
13.6.1.3. I2S Controller as Receiver.....	495
13.6.1.3.1. Receiver Block Enable.....	496
13.6.1.3.2. Receive Channel Enable.....	497
13.6.1.3.3. Receive Channel Audio Data Resolution.....	497
13.6.1.3.4. Receive Channel FIFOs.....	497
13.6.1.3.5. Receive Channel Interrupts.....	498
13.6.1.3.6. Reading from the Receive Channel.....	498
13.6.2. I2S Registers.....	498
13.6.2.1. Registers Map.....	499
13.6.2.2. Register Descriptions.....	500
13.6.2.2.1. I2S Controller Enable Register ( <code>IER</code> ).....	500

13.6.2.2.2. I2S Receiver Block Enable Register ( <u>I<sub>RER</sub></u> ).....	501
13.6.2.2.3. I2S Transmitter Block Enable Register ( <u>I<sub>TER</sub></u> ).....	501
13.6.2.2.4. Receiver Block FIFO Reset Register ( <u>RXFFR</u> ).....	501
13.6.2.2.5. Transmitter Block FIFO Reset Register ( <u>TXFFR</u> ).....	502
13.6.2.2.6. Left Receive Buffer Register ( <u>LRBR0</u> ).....	502
13.6.2.2.7. Left Transmit Holding Register ( <u>LTHR0</u> ).....	502
13.6.2.2.8. Right Receive Buffer Register ( <u>RRBR0</u> ).....	503
13.6.2.2.9. Right Transmit Holding Register ( <u>RTHR0</u> ).....	503
13.6.2.2.10. Receive Enable Register ( <u>RERO</u> ).....	504
13.6.2.2.11. Transmit Enable Register ( <u>TER0</u> ).....	504
13.6.2.2.12. Receive Configuration Register ( <u>RCR0</u> ).....	504
13.6.2.2.13. Transmit Configuration Register ( <u>TCR0</u> ).....	505
13.6.2.2.14. Interrupt Status Register ( <u>ISR0</u> ).....	505
13.6.2.2.15. Interrupt Mask Register ( <u>IMR0</u> ).....	506
13.6.2.2.16. Receive Overrun Register ( <u>ROR0</u> ).....	507
13.6.2.2.17. Transmit Overrun Register ( <u>TOR0</u> ).....	507
13.6.2.2.18. Receive FIFO Configuration Register ( <u>RFCR0</u> ).....	508
13.6.2.2.19. Transmit FIFO Configuration Register ( <u>TFCR0</u> ).....	508
13.6.2.2.20. Receive FIFO Flush Register ( <u>RFF0</u> ).....	508
13.6.2.2.21. Transmit FIFO Flush Register ( <u>TFF0</u> ).....	509
13.6.2.2.22. Receiver Block DMA Register ( <u>RXDMA</u> ).....	509
13.6.2.2.23. Reset Receiver Block DMA Register ( <u>RRXDMA</u> ).....	510
13.6.2.2.24. Transmitter Block DMA Register ( <u>TXDMA</u> ).....	511
13.6.2.2.25. Reset Transmitter Block DMA Register ( <u>RTXDMA</u> ).....	511
13.6.2.2.26. DMA Control Register ( <u>DMACR</u> ).....	511
<b>14. USB 2.0.....</b>	<b>513</b>
14.1. USB 2.0 Controller.....	513
14.1.1. USB 2.0 Controller Functional Description.....	514
14.1.1.1. Resets.....	514
14.1.1.1.1. In Peripheral Mode.....	515
14.1.1.1.2. In Host Mode.....	515
14.1.1.2. Modes of Operation.....	515
14.1.1.3. Suspend/Resume.....	516
14.1.1.3.1. Control by Inactivity on USB Bus (L0 to L2 State).....	517
14.1.1.3.1.1. In Peripheral Mode.....	517
14.1.1.3.1.2. In Host Mode.....	517
14.1.1.3.2. Control by LPM Transaction (L0 to L1 State).....	518
14.1.1.3.2.1. In Peripheral Mode.....	518
14.1.1.3.2.2. In Host Mode.....	520
14.1.1.4. Multiple Devices Support.....	520
14.1.1.4.1. Allocating Devices to Endpoints.....	521
14.1.1.4.2. Operation.....	522
14.1.1.4.3. Bandwidth Issues.....	523
14.1.1.5. Connect/Disconnect.....	523

14.1.1.5.1. In Host Mode.....	523
14.1.1.5.2. In Peripheral Mode.....	523
14.1.1.6. Programming Scheme.....	523
14.1.1.6.1. Soft Connect/Disconnect.....	524
14.1.1.6.2. USB Interrupt Handle.....	524
14.1.1.7. OTG Session Request.....	525
14.1.1.7.1. Starting Session.....	526
14.1.1.7.2. Detecting Activity.....	526
14.1.1.8. Host Negotiation.....	527
14.1.1.9. Fundamental DMA Support.....	527
14.1.1.10. Included DMA Controller.....	529
14.1.1.10.1. DMA Bus Cycles.....	530
14.1.1.10.2. Bus Errors.....	530
14.1.1.10.3. Transferring Packets.....	530
14.1.1.10.3.1. Individual Packet: RX Endpoint.....	530
14.1.1.10.3.2. Individual Packet: TX Endpoint.....	531
14.1.1.10.3.3. Multiple Packets: RX Endpoint.....	532
14.1.1.10.3.4. Multiple Packets: TX Endpoint.....	532
14.1.1.11. VBus Events.....	533
14.1.1.11.1. Actions as 'A' Device.....	533
14.1.1.11.2. Actions as 'B' Device.....	534
14.1.1.12. Dynamic FIFO Sizing.....	534
14.1.1.13. Control Transactions (via Endpoint 0).....	535
14.1.1.13.1. In Peripheral Mode.....	535
14.1.1.13.1.1. Zero Data Requests.....	536
14.1.1.13.1.2. Write Requests.....	536
14.1.1.13.1.3. Read Requests.....	537
14.1.1.13.1.4. Endpoint 0 States.....	537
14.1.1.13.1.5. Endpoint 0 Service Routine.....	538
14.1.1.13.1.5.1. Idle.....	540
14.1.1.13.1.5.2. Tx.....	541
14.1.1.13.1.5.3. Rx.....	542
14.1.1.13.1.6. Error Handling.....	543
14.1.1.13.1.7. Additional Actions .....	544
14.1.1.13.1.7.1. STALL Issued to Control Transfer.....	544
14.1.1.13.1.7.2. Zero-Length OUT Data Packets in Control Transfers.....	544
14.1.1.13.2. In Host Mode.....	544
14.1.1.13.2.1. Setup Phase.....	545
14.1.1.13.2.2. IN Data Phase.....	545
14.1.1.13.2.3. OUT Data Phase.....	546
14.1.1.13.2.4. IN Status Phase (Following Setup Phase or OUT Data Phase).....	547
14.1.1.13.2.5. OUT Status Phase (Following IN Data Phase).....	547
14.1.1.14. Bulk Transactions.....	548
14.1.1.14.1. In Peripheral Mode.....	548

14.1.1.14.1.1. Bulk IN Transactions.....	548
14.1.1.14.1.1.1. Setup.....	549
14.1.1.14.1.1.2. Operation.....	549
14.1.1.14.1.2. Bulk OUT Transactions.....	550
14.1.1.14.1.2.1. Setup.....	551
14.1.1.14.1.2.2. Operation.....	552
14.1.1.14.1.2.3. Error Handling.....	552
14.1.1.14.2. In Host Mode.....	552
14.1.1.14.2.1. Bulk IN Transactions.....	553
14.1.1.14.2.1.1. Setup.....	553
14.1.1.14.2.1.2. Operation.....	554
14.1.1.14.2.1.3. Error Handling.....	555
14.1.1.14.2.2. Bulk OUT Transactions.....	555
14.1.1.14.2.2.1. Setup.....	556
14.1.1.14.2.2.2. Operation.....	557
14.1.1.14.2.2.3. Error Handling.....	558
14.1.1.14.3. Employing DMA.....	558
14.1.1.14.3.1. Using DMA with Bulk Tx Endpoints.....	558
14.1.1.14.3.2. Using DMA with Bulk Rx Endpoints.....	559
14.1.1.15. Full-Speed/Low-Bandwidth Interrupt Transactions.....	559
14.1.1.15.1. In Peripheral Mode.....	560
14.1.1.15.2. In Host Mode.....	560
14.1.1.16. Full-Speed/Low Bandwidth Isochronous Transactions.....	560
14.1.1.16.1. In Peripheral Mode.....	560
14.1.1.16.1.1. IN Transactions.....	560
14.1.1.16.1.1.1. Setup.....	561
14.1.1.16.1.1.2. Operation.....	562
14.1.1.16.1.1.3. Error Handling.....	562
14.1.1.16.1.2. OUT Transactions.....	563
14.1.1.16.1.2.1. Setup.....	563
14.1.1.16.1.2.2. Operation.....	564
14.1.1.16.1.2.3. Error Handling.....	564
14.1.1.16.2. In Host Mode.....	565
14.1.1.16.2.1. IN Transactions.....	565
14.1.1.16.2.1.1. Setup.....	565
14.1.1.16.2.1.2. Operation.....	566
14.1.1.16.2.1.3. Error Handling.....	566
14.1.1.16.2.2. OUT Transactions.....	567
14.1.1.16.2.2.1. Setup.....	567
14.1.1.16.2.2.2. Operation.....	568
14.1.1.17. High Bandwidth Isochronous/Interrupt Transactions.....	568
14.1.2. USB 2.0 Controller Registers.....	569
14.1.2.1. Data Field Format.....	569
14.1.2.2. Registers Map.....	570

14.1.2.3. Register Descriptions.....	574
14.1.2.3.1. Function Address Register ( <b>FADDR</b> ).....	574
14.1.2.3.2. Power Management Register ( <b>POWER</b> ).....	574
14.1.2.3.3. Interrupt Register for Endpoint 0 and TX Endpoints ( <b>INTRTX</b> ).....	576
14.1.2.3.4. Interrupt Register for RX Endpoints ( <b>INTRRX</b> ).....	577
14.1.2.3.5. Interrupt Enable Register for <b>INTRTX</b> ( <b>INTRTXE</b> ).....	578
14.1.2.3.6. Interrupt Enable Register for <b>INTRRX</b> ( <b>INTRRXE</b> ).....	578
14.1.2.3.7. USB Common Interrupts Register ( <b>INTRUSB</b> ).....	579
14.1.2.3.8. USB Common Interrupts Enable Register ( <b>INTRUSBE</b> ).....	580
14.1.2.3.9. Frame Number Register ( <b>FRAME</b> ).....	581
14.1.2.3.10. Index Register ( <b>INDEX</b> ).....	581
14.1.2.3.11. USB Test Modes Register ( <b>TESTMODE</b> ).....	582
14.1.2.3.12. Maximum Packet Size for TX Endpoint #n ( <b>TXMAXP</b> ).....	583
14.1.2.3.13. Endpoint 0 Control and Status Register 0 ( <b>CSR0L</b> ).....	585
14.1.2.3.14. TX Endpoint #n Control and Status Register 0 ( <b>TXCSRL</b> ).....	587
14.1.2.3.15. Endpoint 0 Control and Status Register 1 ( <b>CSR0H</b> ).....	590
14.1.2.3.16. TX Endpoint #n Control and Status Register 1 ( <b>TXCSRH</b> ).....	592
14.1.2.3.17. Maximum Packet Size for RX Endpoint #n ( <b>RXMAXP</b> ).....	595
14.1.2.3.18. RX Endpoint #n Control and Status Register 0 ( <b>RXCSRL</b> ).....	596
14.1.2.3.19. RX Endpoint #n Control and Status Register 1 ( <b>RXCSRH</b> ).....	598
14.1.2.3.20. Endpoint 0 Data Bytes Counter Register ( <b>COUNT0</b> ).....	602
14.1.2.3.21. RX Endpoint #n Data Bytes Counter Register ( <b>RXCOUNT</b> ).....	603
14.1.2.3.22. TX Endpoint #n Settings Register ( <b>TXTYPE</b> ).....	603
14.1.2.3.23. Endpoint 0 Speed Register ( <b>TYPE0</b> ).....	604
14.1.2.3.24. Endpoint 0 NAK Limit Register ( <b>NAKLIMIT0</b> ).....	605
14.1.2.3.25. TX Endpoint #n NAK Limit/Polling Interval Register ( <b>TXINTERVAL</b> ).....	605
14.1.2.3.26. RX Endpoint #n Settings Register ( <b>RXTYPE</b> ).....	606
14.1.2.3.27. RX Endpoint #n NAK Limit/Polling Interval Register ( <b>RXINTERVAL</b> ).....	607
14.1.2.3.28. Configuration Data Register ( <b>CONFIGDATA</b> ).....	608
14.1.2.3.29. Endpoint #n FIFO ( <b>FIFOn</b> ).....	609
14.1.2.3.30. Device Control Register ( <b>DEVCTL</b> ).....	610
14.1.2.3.31. TX Endpoint #n FIFO Size ( <b>TXFIFOSZ</b> ).....	611
14.1.2.3.32. RX Endpoint #n FIFO Size ( <b>RXFIFOSZ</b> ).....	612
14.1.2.3.33. TX Endpoint #n FIFO Address ( <b>TXFIFOAD</b> ).....	613
14.1.2.3.34. RX Endpoint #n FIFO Address ( <b>RXFIFOAD</b> ).....	613
14.1.2.3.35. PHY signals Control Register ( <b>VCONTROL</b> ).....	613
14.1.2.3.36. PHY Signals Status Register ( <b>VSTATUS</b> ).....	615
14.1.2.3.37. Information About Numbers of TX and Rx Endpoints ( <b>EPINFO</b> ).....	615
14.1.2.3.38. Information About RAM ( <b>RAMINFO</b> ).....	616
14.1.2.3.39. Information About Delays ( <b>LINKINFO</b> ).....	616
14.1.2.3.40. Duration of the VBus Pulsing Charge ( <b>VPLEN</b> ).....	616
14.1.2.3.41. Time Buffer Available on High-Speed Transactions ( <b>HS_EOF1</b> ).....	617
14.1.2.3.42. Time Buffer Available on Full-Speed Transactions ( <b>FS_EOF1</b> ).....	617
14.1.2.3.43. Time Buffer Available on Low-Speed Transactions ( <b>LS_EOF1</b> ).....	617

14.1.2.3.44. TX Endpoint #n Function Address ( <code>TXFUNCADDR</code> ).....	618
14.1.2.3.45. TX Endpoint #n Hub Address ( <code>TXHUBADDR</code> ).....	618
14.1.2.3.46. TX Endpoint #n Hub Port ( <code>TXHUBPORT</code> ).....	619
14.1.2.3.47. RX Endpoint #n Function Address ( <code>RXFUNCADDR</code> ).....	619
14.1.2.3.48. RX Endpoint #n Hub Address ( <code>RXHUBADDR</code> ).....	620
14.1.2.3.49. RX Endpoint #n Hub Port ( <code>RXHUBPORT</code> ).....	620
14.1.2.3.50. DMA Channel #n Interrupt Register ( <code>DMA_INTR</code> ).....	621
14.1.2.3.51. DMA Channel #n Transfer Control Register ( <code>DMA_CNTL</code> ).....	621
14.1.2.3.52. DMA Channel #n Address Register ( <code>DMA_ADDR</code> ).....	623
14.1.2.3.53. DMA Channel #n Transfer Count Register ( <code>DMA_COUNT</code> ).....	623
14.1.2.3.54. Number of Requested Packets for RX Endpoint #n ( <code>EPn_RQPKTCOUNT</code> ).....	623
14.1.2.3.55. Chirp Timeout Timer Register( <code>C_T_UCH</code> ).....	624
14.1.2.3.56. High Speed Resume Signaling Delay Register ( <code>C_T_HSRTN</code> ).....	624
14.1.2.3.57. High Speed Timeout Increase Register ( <code>C_T_HSBT</code> ).....	625
14.1.2.3.58. LPM Attribute Register ( <code>LPM_ATTR</code> ).....	625
14.1.2.3.59. LPM Control Register ( <code>LPM_CTR</code> ).....	627
14.1.2.3.60. LPM Interrupt Enable Register ( <code>LPM_INTREN</code> ).....	628
14.1.2.3.61. LPM Interrupt Register ( <code>LPM_INTR</code> ).....	629
14.1.2.3.62. LPM Function Address Register ( <code>LPM_FADDR</code> ).....	631
<b>14.2. USB 2.0 PHY.....</b>	<b>631</b>
<b>14.2.1. USB 2.0 PHY Registers.....</b>	<b>632</b>
<b>14.2.1.1. Registers Map.....</b>	<b>633</b>
<b>14.2.1.2. Register Descriptions.....</b>	<b>633</b>
<b>14.2.1.2.1. HS Resistor Adjust and Tx Clock Gate Control Register (<code>HS_RES_ADJ_TX_CLK_GATE_CTL</code>).....</b>	<b>633</b>
<b>14.2.1.2.2. Super Power Saving and Squelch Control Register (<code>SPWR_SVNG_SQLCH_CTL</code>).....</b>	<b>634</b>
<b>14.2.1.2.3. Output Eye Shape and Internal Bias Current Control Register (<code>OUT_EYE_SHAPE_INT_BIAS_CTL</code>).....</b>	<b>635</b>
<b>14.2.1.2.4. TX Clock Phase and PLL Adjust Control Register (<code>TX_CLK_PHASE_PLL_ADJ_CTL</code>).....</b>	<b>635</b>
<b>14.2.1.2.5. Parameters Control Register 0 (<code>PARAM_CTL_0</code>).....</b>	<b>635</b>
<b>14.2.1.2.6. Parameters Control Register 1 (<code>PARAM_CTL_1</code>).....</b>	<b>636</b>
<b>14.2.1.2.7. Parameters Control Register 2 (<code>PARAM_CTL_2</code>).....</b>	<b>637</b>
<b>14.2.1.2.8. Reserved Out1 Register 0 (<code>RSVD_OUT1_0</code>).....</b>	<b>637</b>
<b>14.2.1.2.9. Reserved Out1 Register 1 (<code>RSVD_OUT1_1</code>).....</b>	<b>637</b>
<b>14.2.1.2.10. Reserved Out2 Register 0 (<code>RSVD_OUT2_0</code>).....</b>	<b>638</b>
<b>14.2.1.2.11. Reserved Out2 Register 1 (<code>RSVD_OUT2_1</code>).....</b>	<b>638</b>
<b>14.2.1.2.12. Reserved Input1 Register 0 (<code>RSVD_IN1_0</code>).....</b>	<b>638</b>
<b>14.2.1.2.13. Reserved Input1 Register 1 (<code>RSVD_IN1_1</code>).....</b>	<b>638</b>
<b>14.2.1.2.14. Reserved Input2 Register 0 (<code>RSVD_IN2_0</code>).....</b>	<b>638</b>
<b>14.2.1.2.15. Reserved Input2 Register 1 (<code>RSVD_IN2_1</code>).....</b>	<b>639</b>
<b>15. Core 2 Software JTAG.....</b>	<b>640</b>
<b>15.1. Software JTAG Register.....</b>	<b>640</b>
<b>15.1.1. Register Map.....</b>	<b>640</b>
<b>15.1.2. Register Description.....</b>	<b>640</b>

15.1.2.1. Software Debug ( <code>DEBUG</code> ).....	640
<b>16. Tracer.....</b>	<b>642</b>
16.1. Tracer Registers.....	642
16.1.1. Registers Maps.....	643
16.1.1.1. Trace Select Register Map.....	643
16.1.1.2. Encoder Block Registers Map.....	643
16.1.1.3. RAM Control Block Registers Map.....	644
16.1.1.4. <i>Pin Interface Block (PIB) Interface</i> Registers Map.....	644
16.1.2. Register Descriptions.....	645
16.1.2.1. Trace Select Register.....	645
16.1.2.1.1. Trace Source Select Register ( <code>TRACE_SEL</code> ).....	645
16.1.2.2. Encoder Block Registers.....	645
16.1.2.2.1. Control Register ( <code>TRTECONTROL</code> ).....	645
16.1.2.2.2. Encoder Block Implementation Parameters Register ( <code>TRTEIMPL</code> ).....	648
16.1.2.2.3. Trace Instruction Features Register ( <code>TRTEINSTFEATURES</code> ).....	649
16.1.2.2.4. Timestamp Control Register ( <code>TRTSCONTROL</code> ).....	649
16.1.2.2.5. Timestamp Counter Lower Bits ( <code>TRTSCOUNTERLOW</code> ).....	650
16.1.2.2.6. Timestamp Counter Upper Bits ( <code>TRTSCOUNTERHIGH</code> ).....	650
16.1.2.3. RAM Control Block Registers.....	651
16.1.2.3.1. RAM Memory Control Register ( <code>TRRAMCONTROL</code> ).....	651
16.1.2.3.2. RAM Memory Implementation Parameters Register ( <code>TRRAMIMPL</code> ).....	652
16.1.2.3.3. RAM Memory Start Address Low Bits Register ( <code>TRRAMSTARTLOW</code> ) .....	653
16.1.2.3.4. RAM Memory Start Address High Bits Register ( <code>TRRAMSTARTHIGH</code> ).....	653
16.1.2.3.5. RAM Memory End Address Low Bits Register ( <code>TRRAMLIMITLOW</code> ) .....	653
16.1.2.3.6. RAM Memory End Address High Bits Register ( <code>TRRAMLIMITHIGH</code> ).....	653
16.1.2.3.7. RAM Memory Write Pointer Address Low Bits Register ( <code>TRRAMWPLOW</code> ).....	654
16.1.2.3.8. RAM Memory Write Pointer Address High Bits Register ( <code>TRRAMWPHIGH</code> ).....	654
16.1.2.3.9. RAM Memory Read Pointer Address Low Bits Register ( <code>TRRAMRPLOW</code> ).....	654
16.1.2.3.10. RAM Memory Read Pointer Address High Bits Register ( <code>TRRAMRPHIGH</code> ).....	654
16.1.2.3.11. RAM Memory Read Data Register ( <code>TRRAMDATA</code> ).....	655
16.1.2.4. PIB Block Registers.....	655
16.1.2.4.1. PIB Block Control Register ( <code>TRPIBCONTROL</code> ).....	655
<b>17. Bus Matrix.....</b>	<b>657</b>
17.1. Bus Matrix Registers.....	657
17.1.1. Registers Map.....	657
17.1.2. Register Descriptions.....	657
17.1.2.1. Arbitration Priority Master 1 Register ( <code>PL1</code> ) .....	658
17.1.2.2. Arbitration Priority Master 2 Register ( <code>PL2</code> ) .....	658
17.1.2.3. Early Burst Termination Count Register ( <code>EBTCOUNT</code> ) .....	658
17.1.2.4. Early Burst Termination Enable ( <code>EBT_EN</code> ) .....	658
17.1.2.5. Early Burst Termination Register ( <code>EBT</code> ) .....	659
17.1.2.6. Default Master ID Number Register ( <code>DFLT_MASTER</code> ) .....	659

## List of Tables

Table 3-1 BE-U1000 Core 0/1 Address Space.....	45
Table 3-2 BE-U1000 Core 2 Address Space.....	48
Table 4-1 int_local interrupts distribution for Core 0 and Core 1.....	50
Table 4-2 int_nmi interrupt for Core 1.....	54
Table 4-3 int_local interrupts distribution for Core 2.....	54
Table 5-1 Clock Sources.....	56
Table 5-2 Output Clocks.....	56
Table 5-3 refsel signal values.....	58
Table 5-4 Influence of the alarm and refsel signals to the sys_clk and alt_clk outputs .....	58
Table 5-5 Recommended PLL Settings for the i_pll_clk Frequency of 25 MHz.....	60
Table 5-6 Reset Sources block input reset signals.....	68
Table 5-7 Reset Sources block output reset signals.....	68
Table 5-8 Memory Address Region for Control Registers.....	71
Table 5-9 CRU Registers Map.....	71
Table 5-10 Fields For Register: CLKSEL.....	73
Table 5-11 Fields For Register: PLLSET.....	74
Table 5-12 Fields For Register: CLKCR0.....	75
Table 5-13 Fields For Register: PCLK0EN.....	78
Table 5-14 Fields For Register: PCLK1EN.....	81
Table 5-15 Fields For Register: PCLK2EN.....	84
Table 5-16 Fields For Register: SYSCR0.....	86
Table 5-17 Fields For Register: SYSCR1.....	89
Table 5-18 Fields For Register: SYSCR2.....	89
Table 5-19 Fields For Register: PRIOR0.....	89
Table 5-20 Fields For Register: PRIOR1.....	90
Table 5-21 Fields For Register: IOPUCR0.....	92
Table 5-22 Fields For Register: IOPUCR1.....	93
Table 5-23 Fields For Register: IOPDCR0.....	93
Table 5-24 Fields For Register: IOPDCR1.....	94
Table 5-25 Fields For Register: IOAFCR0.....	94
Table 5-26 Fields For Register: IOAFCR1.....	96
Table 5-27 Fields For Register: IOAFCR2.....	98
Table 5-28 Fields For Register: IOAFCR3.....	100
Table 5-29 Fields For Register: IOAFCR4.....	102
Table 5-30 Fields For Register: IOAFCR5.....	104
Table 5-31 Fields For Register: IODSCR0.....	106
Table 5-32 Fields For Register: IODSCR1.....	107
Table 5-33 Fields For Register: IODSCR2.....	108
Table 5-34 Fields For Register: LDOCR.....	109
Table 5-35 Fields For Register: USBCR.....	110
Table 5-36 Fields For Register: SYSSR0.....	112
Table 5-37 Fields For Register: WDTCLRKEY.....	113

Table 5-38 Fields For Register: <code>INTCR0</code> .....	113
Table 5-39 Fields For Register: <code>CLKCR1</code> .....	114
Table 5-40 Fields For Register: <code>FLASHNVRCR</code> .....	115
Table 5-41 Fields For Register: <code>CORE1CR</code> .....	115
Table 6-1 HPM Events.....	117
Table 6-2 Memory Map of the Core Debug block.....	121
Table 6-3 CPU Flags ( <code>DM_CPU_FLAGS</code> ) Description.....	121
Table 6-4 CLIC CSRs.....	123
Table 6-5 Trigger Module CSRs.....	124
Table 6-6 Core Debug CSRs.....	124
Table 6-7 Core CSRs.....	124
Table 6-8 Fields For Register: <code>NMITVEC</code> .....	125
Table 6-9 Fields For Register: <code>XCCFG</code> .....	125
Table 6-10 Fields For Register: <code>XICFG</code> .....	125
Table 6-11 JTAG DTM Registers.....	126
Table 6-12 DM Registers (access via DMI).....	126
Table 6-13 Memory Address Region for CLINT Registers.....	127
Table 6-14 CLINT Registers Map.....	127
Table 6-15 Fields For Register: <code>MSIP0</code> .....	128
Table 6-16 Fields For Register: <code>MTIMECMP0</code> .....	128
Table 6-17 Fields For Register: <code>MTIME</code> .....	129
Table 6-18 Memory Address Region for Bus Error Unit Registers.....	129
Table 6-19 Bus Error Unit Registers Map.....	129
Table 6-20 Fields For Register: <code>STATUS_CPU</code> .....	130
Table 6-21 Fields For Register: <code>CONTROL_CPU</code> .....	130
Table 6-22 Fields For Register: <code>ADDR_LO_CPU</code> .....	130
Table 6-23 Fields For Register: <code>ADDR_HI_CPU</code> .....	130
Table 6-24 Memory Address Region for CLIC Registers.....	131
Table 6-25 CLIC Registers Map.....	131
Table 6-26 Memory Map of the Core Debug block.....	133
Table 6-27 CPU Flags ( <code>DM_CPU_FLAGS</code> ) Description.....	134
Table 6-28 Trigger Module CSRs.....	135
Table 6-29 Core Debug CSRs.....	136
Table 6-30 Core CSRs.....	136
Table 6-31 Fields For Register: <code>XCCFG</code> .....	136
Table 6-32 JTAG DTM Registers.....	137
Table 6-33 DM Registers (access via DMI).....	137
Table 6-34 Memory Address Region for CLINT Registers.....	138
Table 6-35 CLINT Registers Map.....	138
Table 6-36 Fields For Register: <code>MSIP0</code> .....	139
Table 6-37 Fields For Register: <code>MTIMECMP0</code> .....	139
Table 6-38 Fields For Register: <code>MTIME</code> .....	140
Table 7-1 Memory Address Region for PIO Block Registers.....	141
Table 7-2 PIO Registers Map.....	141

Table 7-3 Fields For Register: INT.....	142
Table 7-4 Fields For Register: PIO_IN.....	142
Table 7-5 Fields For Register: PIO_EN.....	142
Table 7-6 Fields For Register: PIO_OU.....	143
Table 8-1 Table of connectivity of DMA Handshake channels.....	146
Table 8-2 Programming the Alternate Functions.....	147
Table 8-3 Parameters Used in Transfer Examples.....	155
Table 8-4 Memory Address Regions for DMA Controllers.....	160
Table 8-5 DMA Registers Map.....	160
Table 8-6 Fields For Register: SARx.....	164
Table 8-7 Fields For Register: DARx.....	164
Table 8-8 Fields For Register: LLPx.....	165
Table 8-9 Fields For Register: CTLx.....	166
Table 8-10 CTLx.SRC_MSIZE and CTLx.DEST_MSIZE Decoding.....	168
Table 8-11 CTLx.SRC_TR_WIDTH and CTLx.DST_TR_WIDTH Decoding.....	169
Table 8-12 CTLx.TT_FC Field Decoding.....	169
Table 8-13 Fields For Register: CFGx.....	169
Table 8-14 PROTCTL field to HPROT Mapping.....	173
Table 8-15 Fields For Register: RAW_TFR.....	173
Table 8-16 Fields For Register: RAW_BLOCK.....	174
Table 8-17 Fields For Register: RAW_SRC_TRAN.....	174
Table 8-18 Fields For Register: RAW_DST_TRAN.....	174
Table 8-19 Fields For Register: RAW_ERR.....	175
Table 8-20 Fields For Register: STAT_TFR.....	175
Table 8-21 Fields For Register: STAT_BLOCK.....	176
Table 8-22 Fields For Register: STAT_SRC_TRAN.....	176
Table 8-23 Fields For Register: STAT_DST_TRAN.....	177
Table 8-24 Fields For Register: STAT_ERR.....	177
Table 8-25 Fields For Register: MASK_TFR.....	177
Table 8-26 Fields For Register: MASK_BLOCK.....	178
Table 8-27 Fields For Register: MASK_SRC_TRAN.....	178
Table 8-28 Fields For Register: MASK_DST_TRAN.....	179
Table 8-29 Fields For Register: MASK_ERR.....	179
Table 8-30 Fields For Register: CLR_TFR.....	180
Table 8-31 Fields For Register: CLR_BLOCK.....	180
Table 8-32 Fields For Register: CLR_SRC_TRAN.....	181
Table 8-33 Fields For Register: CLR_DST_TRAN.....	181
Table 8-34 Fields For Register: CLR_ERR.....	181
Table 8-35 Fields For Register: STAT.....	182
Table 8-36 Fields For Register: REQ_SRC.....	183
Table 8-37 Fields For Register: REQ_DST.....	184
Table 8-38 Fields For Register: SGL_REQ_SRC.....	184
Table 8-39 Fields For Register: SGL_REQ_DST.....	185
Table 8-40 Fields For Register: LST_SRC.....	185

Table 8-41 Fields For Register: LST_DST.....	186
Table 8-42 Fields For Register: CFG.....	187
Table 8-43 Fields For Register: CH_EN.....	187
Table 8-44 Fields For Register: TEST.....	188
Table 8-45 Transfer Types.....	189
Table 8-46 Single-block Transfer parameters.....	194
Table 8-47 Multi-Block Transfer with Linked Lists for Source and Destination.....	195
Table 9-1 CRU registers that affect the operation of the eFLASH Controller.....	199
Table 9-2 Memory Address Region for the eFLASH Controller Registers.....	200
Table 9-3 Memory Map of the eFLASH Controller Registers.....	200
Table 9-4 Fields For Register: EFLASH_CR.....	201
Table 9-5 Fields For Register: EFLASH_ADDR.....	202
Table 9-6 Fields For Register: EFLASH_WDATA.....	202
Table 9-7 Fields For Register: EFLASH_RDATA.....	202
Table 9-8 Fields For Register: EFLASH_SR.....	203
Table 9-9 Fields For Register: EFLASH_TIME0.....	203
Table 9-10 Fields For Register: EFLASH_TIME1.....	204
Table 9-11 Fields For Register: EFLASH_TIME2.....	204
Table 9-12 Fields For Register: EFLASH_TIME3.....	205
Table 9-13 Fields For Register: EFLASH_TIME4.....	206
Table 9-14 Fields For Register: EFLASH_TIME5.....	206
Table 9-15 Fields For Register: EFLASH_TIME6.....	207
Table 9-16 Time intervals values.....	208
Table 10-1 Memory Address Regions for Mailbox Registers.....	213
Table 10-2 Mailbox Registers Map.....	213
Table 10-3 Fields For Register: MBnDATA0.....	214
Table 10-4 Fields For Register: MBnFULL0.....	214
Table 10-5 Fields For Register: MBnEMPTY0.....	215
Table 10-6 Fields For Register: MBnDATA1.....	215
Table 10-7 Fields For Register: MBnFULL1.....	216
Table 10-8 Fields For Register: MBnEMPTY1.....	216
Table 11-1 ADC Pins.....	218
Table 11-2 ADC channels connection to the VIN pins.....	218
Table 11-3 Temperature parameters.....	220
Table 11-4 PWMA Event Selection.....	222
Table 11-5 Memory Address Regions for ADC Controller Registers.....	222
Table 11-6 ADC Registers Map.....	223
Table 11-7 Fields For Register: ADC_CRI.....	223
Table 11-8 Fields For Register: ASER.....	226
Table 11-9 Fields For Register: ASTR.....	227
Table 11-10 Fields For Register: ASMPR.....	228
Table 11-11 Fields For Register: ASQR.....	229
Table 11-12 Fields For Register: ADR.....	230
Table 12-1 ITR to TRGO connection.....	232

Table 12-2 Timer slave operation modes.....	236
Table 12-3 Output control bits for complementary OC <sub>x</sub> and OC <sub>xN</sub> channels.....	244
Table 12-4 Memory Address Regions for PWMA Controller Registers.....	250
Table 12-5 PWMA Registers Map.....	250
Table 12-6 Fields For Register: PWMA_CR1.....	251
Table 12-7 Fields For Register: PWMA_CR2.....	253
Table 12-8 Fields For Register: PWMA_SMCR.....	256
Table 12-9 Fields For Register: PWMA_DIER.....	259
Table 12-10 Fields For Register: PWMA_SR.....	261
Table 12-11 Fields For Register: PWMA_EGR.....	265
Table 12-12 Fields For Register: PWMA_CCMR1.....	268
Table 12-13 PWMA_CCMR1[15:10] bits functions for different CC1S values.....	269
Table 12-14 PWMA_CCMR1[7:2] bits functions for different CC0S values.....	270
Table 12-15 Fields For Register: PWMA_CCMR2.....	273
Table 12-16 PWMA_CCMR2[15:10] bits functions for different CC3S values.....	274
Table 12-17 PWMA_CCMR2[7:2] bits functions for different CC2S values.....	275
Table 12-18 Fields For Register: PWMA_CCER.....	278
Table 12-19 Fields For Register: PWMA_CNT.....	280
Table 12-20 Fields For Register: PWMA_PSC.....	280
Table 12-21 Fields For Register: PWMA_ARR.....	280
Table 12-22 Fields For Register: PWMA_RCR.....	281
Table 12-23 Fields For Register: PWMA_CCR0.....	281
Table 12-24 Fields For Register: PWMA_CCR1.....	282
Table 12-25 Fields For Register: PWMA_CCR2.....	282
Table 12-26 Fields For Register: PWMA_CCR3.....	283
Table 12-27 Fields For Register: PWMA_BDTR.....	283
Table 12-28 Fields For Register: PWMA_DCR.....	285
Table 12-29 Fields For Register: PWMA_DMAR.....	286
Table 12-30 Fields For Register: PWMA_QESET.....	286
Table 12-31 Fields For Register: PWMA_FILTSET.....	289
Table 12-32 Fields For Register: PWMA_QEMAX.....	290
Table 12-33 Fields For Register: PWMA_INITSET.....	290
Table 12-34 Fields For Register: PWMA_QECNT.....	291
Table 12-35 Memory Address Regions for PWMG Controller Registers.....	296
Table 12-36 PWMG Registers Map.....	296
Table 12-37 Fields For Register: PWMG_CR1.....	297
Table 12-38 Fields For Register: PWMG_IER.....	298
Table 12-39 Fields For Register: PWMG_SR.....	299
Table 12-40 Fields For Register: PWMG_EGR.....	300
Table 12-41 Fields For Register: PWMG_CCMR1.....	301
Table 12-42 PWMG_CCMR1[7:2] bits functions for different CC0S values.....	302
Table 12-43 Fields For Register: PWMG_CCER.....	303
Table 12-44 Fields For Register: PWMG_CNT.....	304
Table 12-45 Fields For Register: PWMG_PSC.....	304

Table 12-46 Fields For Register: PWMG_ARR.....	305
Table 12-47 Fields For Register: PWMG_CCR0.....	305
Table 12-48 Duty Cycle, TIMER<N>LOAD_COUNT, TIMER<N>LOAD_COUNT2 Relationship.....	309
Table 12-49 Memory Address Regions for TIM Registers.....	310
Table 12-50 TIM Address Range.....	311
Table 12-51 TIM Registers Map.....	311
Table 12-52 Fields For Register: TIMER<N>LOAD_COUNT.....	313
Table 12-53 Fields For Register: TIMER<N>LOAD_COUNT2.....	313
Table 12-54 Fields For Register: TIMER<N>CURRENT_VALUE.....	313
Table 12-55 Fields For Register: TIMER<N>CONTROL_REG.....	314
Table 12-56 Fields For Register: TIMER<N>EOI.....	314
Table 12-57 Fields For Register: TIMER<N>INT_STATUS.....	315
Table 12-58 Fields For Register: TIMERS_INT_STATUS.....	315
Table 12-59 Fields For Register: TIMERS_EOI.....	316
Table 12-60 Fields For Register: TIMERS_RAW_INT_STATUS.....	316
Table 12-61 Memory Address Regions for WDT Registers.....	318
Table 12-62 WDT Registers Map.....	319
Table 12-63 Fields For Register: WDT_CR.....	320
Table 12-64 Fields For Register: WDT_TORR.....	320
Table 12-65 Fields For Register: WDT_CCVR.....	321
Table 12-66 Fields For Register: WDT_CRR.....	321
Table 12-67 Fields For Register: WDT_STAT.....	322
Table 12-68 Fields For Register: WDT_EOI.....	322
Table 12-69 Fields For Register: WDT_COMP_PARAM_5.....	322
Table 12-70 Fields For Register: WDT_COMP_PARAM_4.....	323
Table 12-71 Fields For Register: WDT_COMP_PARAM_3.....	323
Table 12-72 Fields For Register: WDT_COMP_PARAM_2.....	323
Table 12-73 Fields For Register: WDT_COMP_PARAM_1.....	324
Table 13-1 Memory Address Regions for GPIO Controller Registers.....	328
Table 13-2 GPIO Registers Map.....	329
Table 13-3 Fields For Register: GPIO_DR.....	329
Table 13-4 Fields For Register: GPIO_DDR.....	330
Table 13-5 Fields For Register: GPIO_INTEN.....	330
Table 13-6 Fields For Register: GPIO_INTMSK.....	330
Table 13-7 Fields For Register: GPIO_INTLVL.....	331
Table 13-8 Fields For Register: GPIO_INTPOLARITY.....	331
Table 13-9 Fields For Register: GPIO_INTSTAT.....	331
Table 13-10 Fields For Register: GPIO_INTSTATRAW.....	332
Table 13-11 Fields For Register: GPIO_DEBOUNCE.....	332
Table 13-12 Fields For Register: GPIO_EOI.....	332
Table 13-13 Fields For Register: GPIO_EXT.....	333
Table 13-14 Fields For Register: GPIO_SYNC.....	333
Table 13-15 Fields For Register: GPIO_BOTHEDGE.....	333
Table 13-16 CAN FD Operating Mode Transitions.....	335

Table 13-17 Memory Address Regions for CAN FD Registers.....	338
Table 13-18 CAN FD Registers Map.....	339
Table 13-19 Fields For Register: SRR.....	341
Table 13-20 Fields For Register: MSR.....	341
Table 13-21 Fields For Register: BRPR.....	344
Table 13-22 Fields For Register: BTR.....	344
Table 13-23 Fields For Register: ECR.....	345
Table 13-24 Fields For Register: ESR.....	345
Table 13-25 Fields For Register: SR.....	347
Table 13-26 Fields For Register: ISR.....	349
Table 13-27 Fields For Register: IER.....	353
Table 13-28 Fields For Register: ICR.....	356
Table 13-29 Fields For Register: TSR.....	359
Table 13-30 Fields For Register: DP_BRPR.....	359
Table 13-31 Fields For Register: DP_BTR.....	360
Table 13-32 Fields For Register: TRR.....	361
Table 13-33 Fields For Register: IETRS.....	362
Table 13-34 Fields For Register: TCR.....	362
Table 13-35 Fields For Register: IETCS.....	363
Table 13-36 Fields For Register: TXE_FSR.....	363
Table 13-37 Fields For Register: TXE_WMR.....	364
Table 13-38 Fields For Register: AFR.....	365
Table 13-39 Fields For Register: FSR.....	365
Table 13-40 Fields For Register: WMR.....	366
Table 13-41 Fields For Register: TBIID.....	367
Table 13-42 Fields For Register: TBIDLC.....	368
Table 13-43 Fields For Register: TBIDWN.....	369
Table 13-44 Fields For Register: AFMRI.....	370
Table 13-45 Fields For Register: AFIRi.....	371
Table 13-46 Fields For Register: TXE_FIFO_TBI_ID.....	371
Table 13-47 Fields For Register: TXE_FIFO_TBI_DLC.....	373
Table 13-48 Fields For Register: RBIID.....	374
Table 13-49 Fields For Register: RBIDLC.....	375
Table 13-50 Fields For Register: RBIDWN.....	376
Table 13-51 Divisor Latch Fractional Values .....	384
Table 13-52 Memory Address Regions for UART Controller Registers.....	385
Table 13-53 UART Registers Map.....	385
Table 13-54 Fields For Register: RBR.....	387
Table 13-55 Fields For Register: THR.....	387
Table 13-56 Fields For Register: DLL.....	388
Table 13-57 Fields For Register: DLH.....	388
Table 13-58 Fields For Register: IER.....	389
Table 13-59 Fields For Register: IIR.....	390
Table 13-60 Interrupt Control Functions.....	391

Table 13-61 Fields For Register: FCR.....	391
Table 13-62 Fields For Register: LCR.....	392
Table 13-63 Fields For Register: MCR.....	394
Table 13-64 Fields For Register: LSR.....	394
Table 13-65 Fields For Register: SCR.....	397
Table 13-66 Fields For Register: SRBRI.....	397
Table 13-67 Fields For Register: STHRI.....	398
Table 13-68 Fields For Register: FAR.....	398
Table 13-69 Fields For Register: TFR.....	398
Table 13-70 Fields For Register: RFW.....	399
Table 13-71 Fields For Register: USR.....	399
Table 13-72 Fields For Register: TFL.....	401
Table 13-73 Fields For Register: RFL.....	401
Table 13-74 Fields For Register: SRR.....	401
Table 13-75 Fields For Register: SBCR.....	402
Table 13-76 Fields For Register: SFE.....	402
Table 13-77 Fields For Register: SRT.....	403
Table 13-78 Fields For Register: STET.....	403
Table 13-79 Fields For Register: HTx.....	403
Table 13-80 Fields For Register: DMASA.....	404
Table 13-81 Fields For Register: TCR.....	404
Table 13-82 Fields For Register: DE_EN.....	406
Table 13-83 Fields For Register: RE_EN.....	406
Table 13-84 Fields For Register: DET.....	406
Table 13-85 Fields For Register: TAT.....	407
Table 13-86 Fields For Register: DLF.....	407
Table 13-87 Fields For Register: RAR.....	408
Table 13-88 Fields For Register: TAR.....	408
Table 13-89 Fields For Register: LCR_EXT.....	409
Table 13-90 Fields For Register: REG_TIMEOUT_RST.....	410
Table 13-91 *SPI feature table.....	411
Table 13-92 Transmit FIFO Threshold (TFT) Decode Values .....	413
Table 13-93 Receive FIFO Threshold (RFT) Decode Values .....	414
Table 13-94 Memory Address Regions for *SPI Controller Registers.....	417
Table 13-95 *SPI Registers Map.....	417
Table 13-96 Fields For Register: CTRLR0.....	419
Table 13-97 Fields For Register: CTRLR1.....	422
Table 13-98 Fields For Register: SPIENR.....	422
Table 13-99 Fields For Register: MWCR.....	423
Table 13-100 Fields For Register: SER.....	424
Table 13-101 Fields For Register: BAUDR.....	424
Table 13-102 Fields For Register: TXFTLR.....	425
Table 13-103 Fields For Register: RXFTLR.....	425
Table 13-104 Fields For Register: TXFLR.....	425

Table 13-105 Fields For Register: RXFLR.....	426
Table 13-106 Fields For Register: SR.....	426
Table 13-107 Fields For Register: IMR.....	428
Table 13-108 Fields For Register: ISR.....	429
Table 13-109 Fields For Register: RISR.....	430
Table 13-110 Fields For Register: TXOICR.....	431
Table 13-111 Fields For Register: RXOICR.....	432
Table 13-112 Fields For Register: RXUICR.....	432
Table 13-113 Fields For Register: MSTICR.....	432
Table 13-114 Fields For Register: ICR.....	432
Table 13-115 Fields For Register: DMACR.....	433
Table 13-116 Fields For Register: DMATDLR.....	433
Table 13-117 Fields For Register: DMARDLR.....	434
Table 13-118 Fields For Register: DRi.....	434
Table 13-119 Fields For Register: RX_SAMPLE_DLY.....	435
Table 13-120 Fields For Register: SPI_CTRLR0.....	435
Table 13-121 Fields For Register: TXD_DRIVE_EDGE.....	437
Table 13-122 Fields For Register: XIP_CMD.....	437
Table 13-123 I2C Definition of Bits in First Byte.....	440
Table 13-124 Maximum Values for IC_SDA_RX_HOLD.....	446
Table 13-125 I2C events monitored by individual interrupts.....	447
Table 13-126 Memory Address Regions for I2C Registers.....	448
Table 13-127 I2C Registers Map.....	448
Table 13-128 Fields for Register: IC_CON.....	450
Table 13-129 Fields for Register: IC_TAR.....	454
Table 13-130 Fields for Register: IC_SAR.....	455
Table 13-131 Fields for Register: IC_HS_MADDR.....	455
Table 13-132 Fields for Register: IC_DATA_CMD.....	456
Table 13-133 Fields for Register: IC_SS_SCL_HCNT.....	457
Table 13-134 Fields for Register: IC_SS_SCL_LCNT.....	458
Table 13-135 Fields for Register: IC_FS_SCL_HCNT.....	458
Table 13-136 Fields for Register: IC_FS_SCL_LCNT.....	458
Table 13-137 Fields for Register: IC_HS_SCL_HCNT.....	459
Table 13-138 Fields for Register: IC_HS_SCL_LCNT.....	459
Table 13-139 Fields for Register: IC_INTR_STAT.....	460
Table 13-140 Fields for Register: IC_INTR_MASK.....	462
Table 13-141 Fields for Register: IC_RAW_INTR_STAT.....	464
Table 13-142 Fields for Register: IC_RX_TL.....	469
Table 13-143 Fields for Register: IC_TX_TL.....	469
Table 13-144 Fields for Register: IC_CLR_INTR.....	469
Table 13-145 Fields for Register: IC_CLR_RX_UNDER.....	470
Table 13-146 Fields for Register: IC_CLR_RX_OVER.....	470
Table 13-147 Fields for Register: IC_CLR_TX_OVER.....	470
Table 13-148 Fields for Register: IC_CLR_RD_REQ.....	471

Table 13-149 Fields for Register: IC_CLR_TX_ABRT.....	471
Table 13-150 Fields for Register: IC_CLR_RX_DONE.....	471
Table 13-151 Fields for Register: IC_CLR_ACTIVITY.....	472
Table 13-152 Fields for Register: IC_CLR_STOP_DET.....	472
Table 13-153 Fields for Register: IC_CLR_START_DET.....	472
Table 13-154 Fields for Register: IC_CLR_GEN_CALL.....	473
Table 13-155 Fields for Register: IC_ENABLE.....	473
Table 13-156 Fields for Register: IC_STATUS.....	475
Table 13-157 Fields for Register: IC_TXFLR.....	477
Table 13-158 Fields for Register: IC_RXFLR.....	478
Table 13-159 Fields for Register: IC_SDA_HOLD.....	478
Table 13-160 Fields for Register: IC_TX_ABRT_SOURCE.....	479
Table 13-161 Fields for Register: IC_DMA_CR.....	484
Table 13-162 Fields for Register: IC_DMA_TDRL.....	485
Table 13-163 Fields for Register: IC_DMA_RDLR.....	485
Table 13-164 Fields for Register: IC_SDA_SETUP.....	485
Table 13-165 Fields for Register: IC_ACK_GENERAL_CALL.....	486
Table 13-166 Fields for Register: IC_ENABLE_STATUS.....	486
Table 13-167 Fields for Register: IC_FS_SPKLEN.....	488
Table 13-168 Fields for Register: IC_HS_SPKLEN.....	489
Table 13-169 Fields for Register: IC_SCL_STUCK_AT_LOW_TIMEOUT.....	489
Table 13-170 Fields for Register: IC_SDA_STUCK_AT_LOW_TIMEOUT.....	490
Table 13-171 Fields for Register: IC_CLR_SCL_STUCK_DET.....	490
Table 13-172 Fields for Register: REG_TIMEOUT_RST.....	491
Table 13-173 Memory Address Regions for I2S Controllers.....	498
Table 13-174 I2S Registers Map.....	499
Table 13-175 Fields For Register: IER.....	500
Table 13-176 Fields For Register: IRER.....	501
Table 13-177 Fields For Register: ITER.....	501
Table 13-178 Fields For Register: RXFFR.....	501
Table 13-179 Fields For Register: TXFFR.....	502
Table 13-180 Fields For Register: LRBRO.....	502
Table 13-181 Fields For Register: LTHR0.....	503
Table 13-182 Fields For Register: RRBR0.....	503
Table 13-183 Fields For Register: RTHR0.....	503
Table 13-184 Fields For Register: RER0.....	504
Table 13-185 Fields For Register: TER0.....	504
Table 13-186 Fields For Register: RCR0.....	504
Table 13-187 Fields For Register: TCR0.....	505
Table 13-188 Fields For Register: ISR0.....	506
Table 13-189 Fields For Register: IMR0.....	506
Table 13-190 Fields For Register: ROR0.....	507
Table 13-191 Fields For Register: TOR0.....	507
Table 13-192 Fields For Register: RFCR0.....	508

Table 13-193 Fields For Register: <code>TFCR0</code> .....	508
Table 13-194 Fields For Register: <code>RFF0</code> .....	509
Table 13-195 Fields For Register: <code>TF0</code> .....	509
Table 13-196 Fields For Register: <code>RXDMA</code> .....	510
Table 13-197 Fields For Register: <code>RRXDMA</code> .....	510
Table 13-198 Fields For Register: <code>TXDMA</code> .....	511
Table 13-199 Fields For Register: <code>RTXDMA</code> .....	511
Table 13-200 Fields For Register: <code>DMACR</code> .....	511
Table 14-1 Response to LPM Transaction from USB 2.0 Controller .....	518
Table 14-2 EP Interrupt Associated with <code>RXPKTRDY</code> Being Set.....	528
Table 14-3 EP Interrupt Associated with <code>TXPKTRDY</code> Being Cleared .....	528
Table 14-4 Endpoint #n FIFO Parameters Registers.....	534
Table 14-5 Values of <code>TXCSRH</code> Register.....	549
Table 14-6 Values of <code>RXCSRH</code> Register.....	551
Table 14-7 Values of <code>RXCSRH</code> Register.....	554
Table 14-8 Values of <code>TXCSRH</code> Register.....	556
Table 14-9 Values of <code>TXCSRH</code> Register.....	561
Table 14-10 Values of <code>RXCSRH</code> Register.....	564
Table 14-11 Values of <code>RXCSRH</code> Register.....	566
Table 14-12 Values of <code>TXCSRH</code> Register.....	567
Table 14-13 Memory Address Region for USB Registers.....	569
Table 14-14 USB 2.0 Controller Registers Map.....	570
Table 14-15 Fields For Register: <code>FADDR</code> .....	574
Table 14-16 Fields For Register: <code>POWER</code> .....	575
Table 14-17 Fields For Register: <code>INTRTX</code> .....	577
Table 14-18 Fields For Register: <code>INTRRX</code> .....	577
Table 14-19 Fields For Register: <code>INTRTXE</code> .....	578
Table 14-20 Fields For Register: <code>INTRRXE</code> .....	579
Table 14-21 Fields For Register: <code>INTRUSB</code> .....	579
Table 14-22 Fields For Register: <code>INTRUSBE</code> .....	580
Table 14-23 Fields For Register: <code>FRAME</code> .....	581
Table 14-24 Fields For Register: <code>INDEX</code> .....	581
Table 14-25 Fields For Register: <code>TESTMODE</code> .....	582
Table 14-26 Fields For Register: <code>TXMAXP</code> .....	584
Table 14-27 Fields For Register: <code>CSR0L</code> ( <code>DEVCTL.HOST_MODE = 0x0</code> ).....	585
Table 14-28 Fields For Register: <code>CSR0L</code> ( <code>DEVCTL.HOST_MODE = 0x1</code> ).....	586
Table 14-29 Fields For Register: <code>TXCSRL</code> ( <code>DEVCTL.HOST_MODE = 0x0</code> ).....	588
Table 14-30 Fields For Register: <code>TXCSRL</code> ( <code>DEVCTL.HOST_MODE = 0x1</code> ).....	589
Table 14-31 Fields For Register: <code>CSR0H</code> ( <code>DEVCTL.HOST_MODE=0x0</code> ).....	591
Table 14-32 Fields For Register: <code>CSR0H</code> ( <code>DEVCTL.HOST_MODE=0x1</code> ).....	591
Table 14-33 Fields For Register: <code>TXCSRH</code> ( <code>DEVCTL.HOST_MODE= 0x0</code> ).....	592
Table 14-34 Fields For Register: <code>TXCSRH</code> ( <code>DEVCTL.HOST_MODE= 0x1</code> ).....	593
Table 14-35 Fields For Register: <code>RXMAXP</code> .....	595
Table 14-36 Fields For Register: <code>RXCSR0L</code> ( <code>DEVCTL.HOST_MODE= 0x0</code> ).....	596

Table 14-37 Fields For Register: RXCSRL (DEVCTL.HOST_MODE= 0x1).....	597
Table 14-38 Fields For Register: RXCSRH (DEVCTL.HOST_MODE= 0x0).....	599
Table 14-39 .....	599
Table 14-40 Fields For Register: RXCSRH (DEVCTL.HOST_MODE= 0x1).....	601
Table 14-41 .....	601
Table 14-42 Fields For Register: COUNT0.....	603
Table 14-43 Fields For Register: RXCOUNT.....	603
Table 14-44 Fields For Register: TXTYPE.....	604
Table 14-45 Fields For Register: TYPE0.....	604
Table 14-46 Fields For Register: NAKLIMIT0.....	605
Table 14-47 Fields For Register: TXINTERVAL.....	605
Table 14-48 Fields For Register: RXTYPE.....	607
Table 14-49 Fields For Register: RXINTERVAL.....	607
Table 14-50 Fields For Register: CONFIGDATA.....	608
Table 14-51 Fields For Register: FIFO.....	609
Table 14-52 Fields For Register: DEVCTL.....	610
Table 14-53 Fields For Register: TXFIFOSZ.....	611
Table 14-54 Fields For Register: RXFIFOSZ.....	612
Table 14-55 Fields For Register: TXFIFOAD.....	613
Table 14-56 Fields For Register: RXFIFOAD.....	613
Table 14-57 Fields For Register: VCONTROL.....	614
Table 14-58 Fields For Register: VSTATUS.....	615
Table 14-59 Fields For Register: EPINFO.....	616
Table 14-60 Fields For Register: RAMINFO.....	616
Table 14-61 Fields For Register: LINKINFO.....	616
Table 14-62 Fields For Register: VPLEN.....	617
Table 14-63 Fields For Register: HS_EOF1.....	617
Table 14-64 Fields For Register: FS_EOF1.....	617
Table 14-65 Fields For Register: LS_EOF1.....	618
Table 14-66 Fields For Register: TXFUNCADDR.....	618
Table 14-67 Fields For Register: TXHUBADDR.....	618
Table 14-68 Fields For Register: TXHUBPORT.....	619
Table 14-69 Fields For Register: RXFUNCADDR.....	620
Table 14-70 Fields For Register: RXHUBADDR.....	620
Table 14-71 Fields For Register: RXHUBPORT.....	621
Table 14-72 Fields For Register: DMA_INTR.....	621
Table 14-73 Fields For Register: DMA_CNTL.....	621
Table 14-74 Fields For Register: DMA_ADDR.....	623
Table 14-75 Fields For Register: DMA_COUNT.....	623
Table 14-76 Fields For Register: EPn_RQPKTCOUNT.....	624
Table 14-77 Fields For Register: C_T_UCH.....	624
Table 14-78 Fields For Register: C_T_HSRTN.....	624
Table 14-79 Fields For Register: C_T_HSBT.....	625
Table 14-80 Fields For Register: LPM_ATTR.....	626

Table 14-81 Fields For Register: LPM_CNTRL (DEVCTL.HOST_MODE = 0x0).....	627
Table 14-82 Fields For Register: LPM_CNTRL (DEVCTL.HOST_MODE = 0x1).....	628
Table 14-83 Fields For Register: LPM_INTREN.....	629
Table 14-84 Fields For Register: LPM_INTR (DEVCTL.HOST_MODE=0x0).....	630
Table 14-85 Fields For Register: LPM_INTR (DEVCTL.HOST_MODE=0x1).....	630
Table 14-86 Fields For Register: LPM_FADDR.....	631
Table 14-87 USB 2.0 PHY Registers Mp.....	633
Table 14-88 Fields For Register: HS_RES_ADJ_TX_CLK_GATE_CTL.....	634
Table 14-89 Fields For Register: SPWR_SVNG_SQLCH_CTL.....	634
Table 14-90 Fields For Register: OUT_EYE_SHAPE_INT_BIAS_CTL.....	635
Table 14-91 Fields For Register: TX_CLK_PHASE_PLL_ADJ_CTL.....	635
Table 14-92 Fields For Register: PARAM_CTL_0.....	635
Table 14-93 Fields For Register: PARAM_CTL_1.....	636
Table 14-94 Fields For Register: PARAM_CTL_2.....	637
Table 14-95 Fields For Register: RSVD_OUT1_0.....	637
Table 14-96 Fields For Register: RSVD_OUT1_1.....	638
Table 14-97 Fields For Register: RSVD_OUT2_0.....	638
Table 14-98 Fields For Register: RSVD_OUT2_1.....	638
Table 14-99 Fields For Register: RSVD_IN1_0.....	638
Table 14-100 Fields For Register: RSVD_IN1_1.....	638
Table 14-101 Fields For Register: RSVD_IN2_0.....	639
Table 14-102 Fields For Register: RSVD_IN2_1.....	639
Table 15-1 Memory Address Region for Software JTAG Register.....	640
Table 15-2 Software JTAG Register Map.....	640
Table 15-3 Fields For Register: DEBUG.....	641
Table 16-1 Memory Address Regions for Tracer Registers.....	643
Table 16-2 Memory Address Region for Trace Select Register .....	643
Table 16-3 Trace Select Register Map.....	643
Table 16-4 Encoder Block Registers Map.....	644
Table 16-5 RAM Control Block Registers Map.....	644
Table 16-6 PIB Block Registers Map.....	645
Table 16-7 Fields For Register: TRACE_SEL.....	645
Table 16-8 Fields For Register: TRTECONTROL.....	645
Table 16-9 Fields For Register: TRTEIMPL.....	648
Table 16-10 Fields For Register: TRTEINSTFEATURES.....	649
Table 16-11 Fields For Register: TRTSCONTROL.....	649
Table 16-12 Fields For Register: TRTSCOUNTERLOW.....	650
Table 16-13 Fields For Register: TRTSCOUNTERHIGH.....	651
Table 16-14 Fields For Register: TRRAMCONTROL.....	651
Table 16-15 Fields For Register: TRRAMIMPL.....	652
Table 16-16 Fields For Register: TRRAMSTARTLOW.....	653
Table 16-17 Fields For Register: TRRAMSTARTHIGH.....	653
Table 16-18 Fields For Register: TRRAMILIMITLOW.....	653
Table 16-19 Fields For Register: TRRAMILIMITHIGH.....	654

---

Table 16-20 Fields For Register: TRRAMWPLOW.....	654
Table 16-21 Fields For Register: TRRAMWPHIGH.....	654
Table 16-22 Fields For Register: TRRAMRPLLOW.....	654
Table 16-23 Fields For Register: TRRAMRPHIGH.....	655
Table 16-24 Fields For Register: TRRAMDATA.....	655
Table 16-25 Fields For Register: TRPIBCONTROL.....	655
Table 17-1 Memory Address Region for Bus matrix Registers.....	657
Table 17-2 Bus Matrix Registers Map.....	657
Table 17-3 Fields For Register: PL1.....	658
Table 17-4 Fields For Register: PL2.....	658
Table 17-5 Fields For Register: EBTCOUNT.....	658
Table 17-6 Fields For Register: EBT_EN.....	659
Table 17-7 Fields For Register: EBT.....	659
Table 17-8 Fields For Register: DFLT_MASTER.....	659

## List of Figures

Figure 2-1 BE-U1000 Architecture.....	44
Figure 3-1 Core 0/1 Address Space.....	48
Figure 3-2 Core 2 Address Space.....	49
Figure 5-1 CRU Block Diagram.....	55
Figure 5-2 Clock and Reset Generation Block Diagram.....	56
Figure 5-3 Clock Sources Block Diagram.....	57
Figure 5-4 <code>sys_clk</code> Selection Block.....	58
Figure 5-5 PLL Block Diagram.....	59
Figure 5-6 PLL Stability Tracking.....	61
Figure 5-7 Dividers Block Diagram.....	61
Figure 5-8 Output Clocks Block Diagram.....	62
Figure 5-9 <code>cclk</code> Clock Domain.....	63
Figure 5-10 <code>pclk_0</code> Clock Domain.....	64
Figure 5-11 <code>pclk_1</code> Clock Domain.....	65
Figure 5-12 <code>pclk_2</code> Clock Domain.....	66
Figure 5-13 <code>hclk</code> Clock Domain.....	66
Figure 5-14 <code>can_clk</code> and <code>can_clk_x2</code> Clock Domains.....	67
Figure 5-15 <code>tclk</code> Clock Domain.....	67
Figure 5-16 Reset Sources Block Diagram.....	67
Figure 5-17 PA[0] I/O pin multiplexor.....	69
Figure 5-18 I/O pins functions and control for Port A.....	69
Figure 5-19 I/O pins functions and control for Port B.....	70
Figure 5-20 I/O pins functions and control for Port C.....	70
Figure 6-1 BR-350 Core Block Diagram.....	116
Figure 6-2 Debug Subsystem.....	120
Figure 6-3 BM-310 Core Block Diagram.....	132
Figure 6-4 Debug Subsystem.....	133
Figure 7-1 PIO Block in the BE-U1000.....	141
Figure 8-1 DMA Controller Functional Block Diagram.....	144
Figure 8-2 Handshaking Interface for DMA_0.....	150
Figure 8-3 Handshaking Interface for DMA_1.....	151
Figure 8-4 Software Controlled DMA Transfers.....	152
Figure 8-5 SPI Receive FIFO.....	158
Figure 8-6 Flowchart for DMA Programming Example.....	193
Figure 9-1 eFLASH Controller functional block diagram.....	199
Figure 10-1 Mailbox in Block Diagram.....	212
Figure 11-1 ADC Functional Block Diagram.....	217
Figure 12-1 PWMA Functional Block Diagram.....	231
Figure 12-2 Control Unit.....	233
Figure 12-3 External Trigger Control block.....	234
Figure 12-4 Trigger Selection block.....	235
Figure 12-5 Slave Mode Controller.....	236

Figure 12-6 Master Mode Controller.....	237
Figure 12-7 TRGO to ITR connection.....	237
Figure 12-8 Time-base Unit.....	237
Figure 12-9 Capture/Compare Channels.....	241
Figure 12-10 Channel 0 Input Stage.....	242
Figure 12-11 Channel 0 Output Stage.....	243
Figure 12-12 Channel 3 Output Stage.....	245
Figure 12-13 Capture/Compare Stage 0.....	246
Figure 12-14 Quadrature Encoder.....	248
Figure 12-15 PWMG Functional Block Diagram.....	291
Figure 12-16 Time-base Unit.....	292
Figure 12-17 Capture/Compare Channel.....	293
Figure 12-18 Channel 0 Input Stage.....	294
Figure 12-19 Channel 0 Output Stage.....	294
Figure 12-20 Capture/Compare Stage 0.....	295
Figure 12-21 TIM Functional block diagram.....	306
Figure 12-22 Timer Usage Flow.....	307
Figure 12-23 WDT_0 Block Diagram.....	317
Figure 12-24 WDT_1 Block Diagram.....	317
Figure 13-1 GPIO Functional block diagram .....	326
Figure 13-2 CAN FD Functional block diagram.....	334
Figure 13-3 UART Functional block diagram.....	376
Figure 13-4 Serial data format.....	377
Figure 13-5 Auto address transmit flow chart.....	379
Figure 13-6 Hardware Address Match Receive Mode.....	380
Figure 13-7 *SPI controller functional block diagram.....	411
Figure 13-8 Endian Conversion for Data Register and XIP Reads.....	412
Figure 13-9 I2C Functional Block Diagram.....	439
Figure 13-10 7-bit Address Format.....	439
Figure 13-11 10-bit Address Format.....	440
Figure 13-12 IC_SDA_HOLD Register.....	445
Figure 13-13 I2S Block Diagram.....	491
Figure 13-14 Basic Usage Flow - I2S Controller as Transmitter.....	492
Figure 13-15 Basic Usage Flow - I2S Controller as Receiver.....	496
Figure 14-1 USB 2.0 Controller Block Diagram.....	513
Figure 14-2 Link Power Management Diagram.....	517
Figure 14-3 USB Interrupt Service Routine Diagram.....	525
Figure 14-4 Endpoint 0 States for Peripheral.....	538
Figure 14-5 Flow for Endpoint 0 Service Routine.....	540
Figure 14-6 Flow for SETUP Phase of Control Transfer.....	541
Figure 14-7 Flow for IN Data Phase of Control Transfer.....	542
Figure 14-8 Flow for OUT Data Phase of Control Transfer.....	543
Figure 14-9 USB 2.0 PHY Block Diagram.....	632
Figure 16-1 Tracer Block Diagram.....	642

## 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.

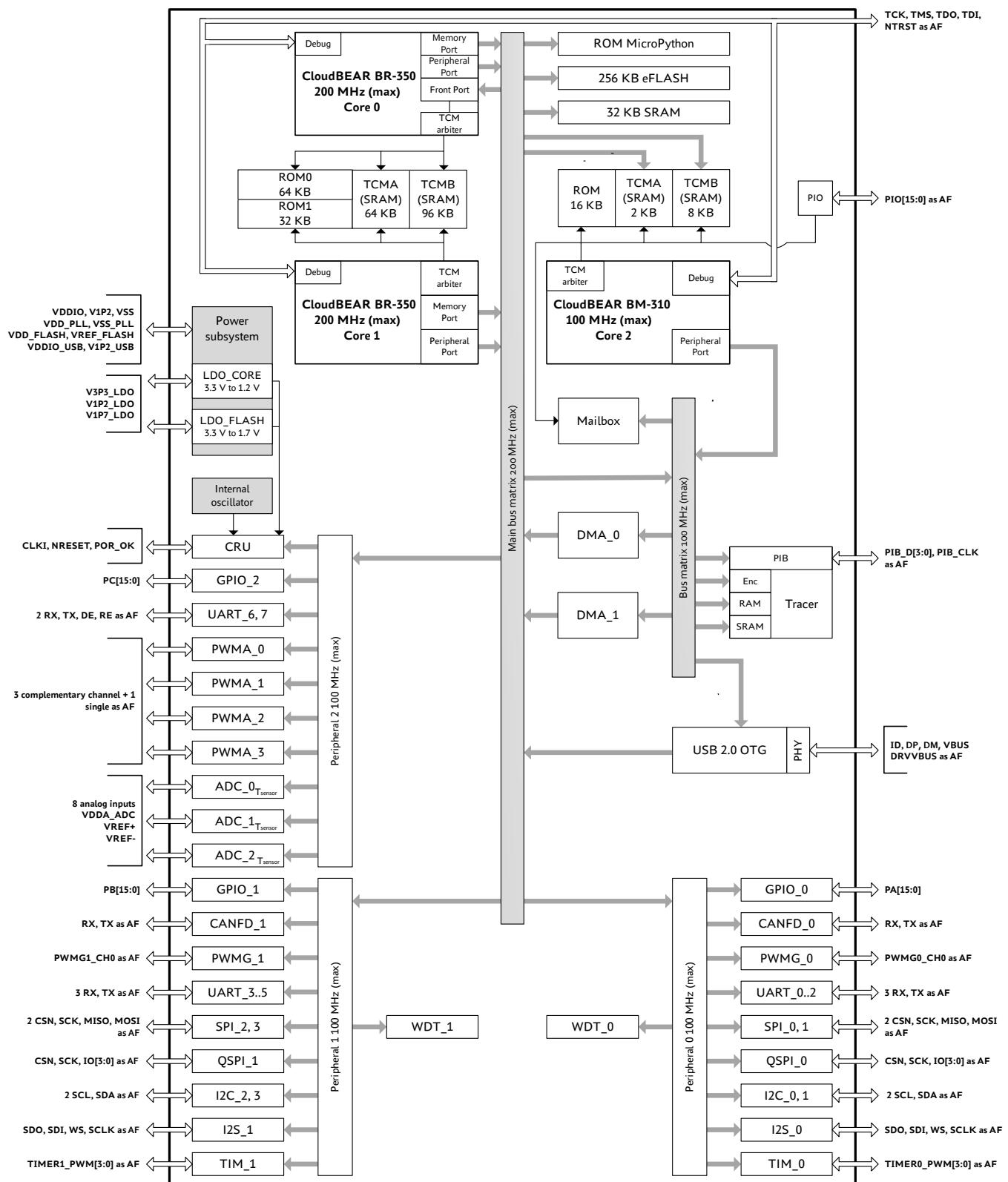


Figure 2-1 BE-U1000 Architecture

### 3. Memory Map

BE-U1000 supports 4GB (0x0000\_0000 - 0xFFFF\_FFFF) of adressable memory.

The tables below give the blocks adresses from the Core 0/1 and Core 2 side.



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	4KB	0x0000_0000	0x0000_0FFF	
	CLINT	64KB	0x0200_0000	0x0200_FFFF	See <a href="#">CLINT Registers Map</a> table
	Bus Error Unit	4KB	0x0202_0000	0x0202_0FFF	See <a href="#">Bus Error Unit Registers Map</a> table
	CLIC	20KB	0x0D00_0000	0x0D00_4FFF	See <a href="#">CLIC Registers Map</a>
Periphery 0	QSPI_0	4KB	0x1100_0000	0x1100_0FFF	See <a href="#">*SPI Registers Map</a> table
	UART_0	4KB	0x1100_1000	0x1100_1FFF	<a href="#">UART Registers Map</a> table
	UART_1	4KB	0x1100_2000	0x1100_2FFF	
	UART_2	4KB	0x1100_3000	0x1100_3FFF	
	I2C_0	4KB	0x1100_4000	0x1100_4FFF	<a href="#">I2C Registers Map</a> table
	I2C_1	4KB	0x1100_5000	0x1100_5FFF	
	I2S_0	4KB	0x1100_6000	0x1100_6FFF	See <a href="#">I2S Registers Map</a> table
	TIM_0	4KB	0x1100_7000	0x1100_7FFF	See <a href="#">TIM Registers Map</a> table
	WDT_0	4KB	0x1100_8000	0x1100_8FFF	See <a href="#">WDT Registers Map</a> table
	GPIO_0	4KB	0x1100_9000	0x1100_9FFF	See <a href="#">GPIO Registers Map</a> table
	PWMG_0	4KB	0x1100_A000	0x1100_AFFF	See <a href="#">PWMG Registers Map</a> table
	SPI_0 (Slave)	4KB	0x1100_B000	0x1100_BFFF	<a href="#">*SPI Registers Map</a> table
	SPI_1 (Master)	4KB	0x1100_C000	0x1100_CFFF	
	CANFD_0	32KB	0x1101_0000	0x1101_7FFF	See <a href="#">CAN FD Registers Map</a> table
Periphery 1	QSPI_1	4KB	0x1300_0000	0x1300_0FFF	See <a href="#">*SPI Registers Map</a> table

**Table 3-1 BE-U1000 Core 0/1 Address Space (continued)**

Region	Block Name	Size	Start Address	End Address	Register Map
Periphery 1	UART_3	4KB	0x1300_1000	0x1300_1FFF	See <a href="#">UART Registers Map</a> table
	UART_4	4KB	0x1300_2000	0x1300_2FFF	
	UART_5	4KB	0x1300_3000	0x1300_3FFF	
	I2C_2	4KB	0x1300_4000	0x1300_4FFF	See <a href="#">I2C Registers Map</a> table
	I2C_3	4KB	0x1300_5000	0x1300_5FFF	
	I2S_1	4KB	0x1300_6000	0x1300_6FFF	See <a href="#">I2S Registers Map</a> table
	TIM_1	4KB	0x1300_7000	0x1300_7FFF	See <a href="#">TIM Registers Map</a> table
	WDT_1	4KB	0x1300_8000	0x1300_8FFF	See <a href="#">WDT Registers Map</a> table
	GPIO_1	4KB	0x1300_9000	0x1300_9FFF	See <a href="#">GPIO Registers Map</a> table
	PWMG_1	4KB	0x1300_A000	0x1300_AFFF	See <a href="#">PWMG Registers Map</a> table
	SPI_2 (Slave)	4KB	0x1300_B000	0x1300_BFFF	See <a href="#">*SPI Registers Map</a> table
	SPI_3 (Master)	4KB	0x1300_C000	0x1300_CFFF	
Periphery 2	CANFD_1	32KB	0x1301_0000	0x1301_7FFF	See <a href="#">CAN FD Registers Map</a> table
	CRU	4KB	0x1400_0000	0x1400_0FFF	See <a href="#">CRU Registers Map</a> table
	ADC_0	4KB	0x1400_1000	0x1400_1FFF	See <a href="#">ADC Registers Map</a> table
	ADC_1	4KB	0x1400_2000	0x1400_2FFF	
	ADC_2	4KB	0x1400_3000	0x1400_3FFF	
	PWMA_0	4KB	0x1400_4000	0x1400_4FFF	See <a href="#">PWMA Registers Map</a> table
	PWMA_1	4KB	0x1400_5000	0x1400_5FFF	
	PWMA_2	4KB	0x1400_6000	0x1400_6FFF	
	PWMA_3	4KB	0x1400_7000	0x1400_7FFF	
High-speed Periphery	GPIO_2	4KB	0x1400_8000	0x1400_8FFF	See <a href="#">GPIO Registers Map</a> table
	UART_6	4KB	0x1400_9000	0x1400_9FFF	See <a href="#">UART Registers Map</a> table
	UART_7	4KB	0x1400_A000	0x1400_AFFF	
	USB	1MB	0x1500_0000	0x150F_FFFF	See <a href="#">USB 2.0 Controller Registers Map</a> and <a href="#">USB</a>

**Table 3-1 BE-U1000 Core 0/1 Address Space (continued)**

Region	Block Name	Size	Start Address	End Address	Register Map
System Resources					<a href="#">2.0 PHY Registers Map tables</a>
	DMA_0	4KB	0x1510_0000	0x1510_0FFF	<a href="#">See DMA Registers Map table</a>
	DMA_1	4KB	0x1510_1000	0x1510_1FFF	
	Mailbox	4KB	0x1510_3000	0x1510_3FFF	<a href="#">See Mailbox Registers Map table</a>
	Tracer encoder control interface	4KB	0x1510_4000	0x1510_4FFF	<a href="#">See Encoder Block Registers Map table</a>
	Tracer RAM control interface	4KB	0x1510_5000	0x1510_5FFF	<a href="#">See RAM Control Block Registers Map table</a>
	Tracer PIB control interface	4KB	0x1510_6000	0x1510_6FFF	<a href="#">See PIB Block Registers Map table</a>
	Tracer SRAM interface	4KB	0x1510_7000	0x1510_7FFF	
	Bus matrix	4KB	0x1510_8000	0x1510_8FFF	<a href="#">See Bus matrix Registers Map table</a>
	* ROM0/ROM1	64KB	0x4000_0000	0x4000_FFFF	
System Resources	TCMA	64KB	0x4001_0000	0x4001_FFFF	
	TCMB	96KB	0x4002_0000	0x4003_7FFF	
	Front Port core0	256KB	0x5000_0000	0x5003_FFFF	
	SRAM	32KB	0x7000_0000	0x7000_7FFF	
	TCMA core2	2KB	0x7000_F800	0x7000_FFFF	
	TCMB core2	8KB	0x7001_0000	0x7001_1FFF	
	QSPI0 XIP	16MB	0x8000_0000	0x80FF_FFFF	
	QSPI1 XIP	16MB	0x9000_0000	0x90FF_FFFF	
	eFLASH Memory	256KB	0xA000_0000	0xA003_FFFF	
	ROM MicroPython	192KB	0xA080_0000	0xA082_FFFF	
	eFLASH Controller	4KB	0xA100_0000	0xA100_0FFF	<a href="#">See eFLASH Controller Registers Map table</a>



In the table above asterisk symbol (\*) indicates:



- 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.

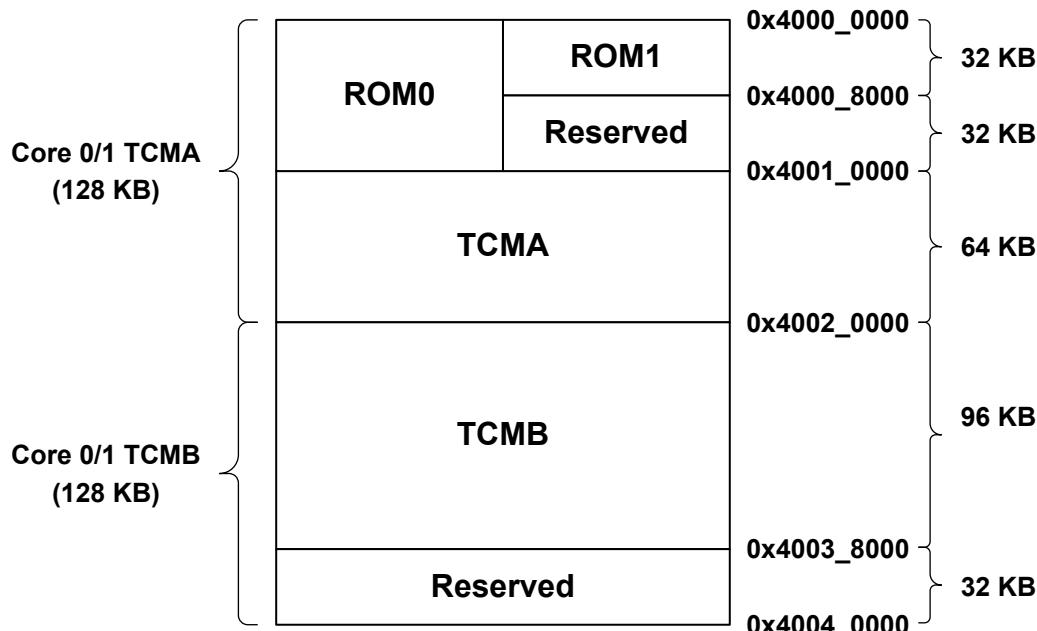


Figure 3-1 Core 0/1 Address Space

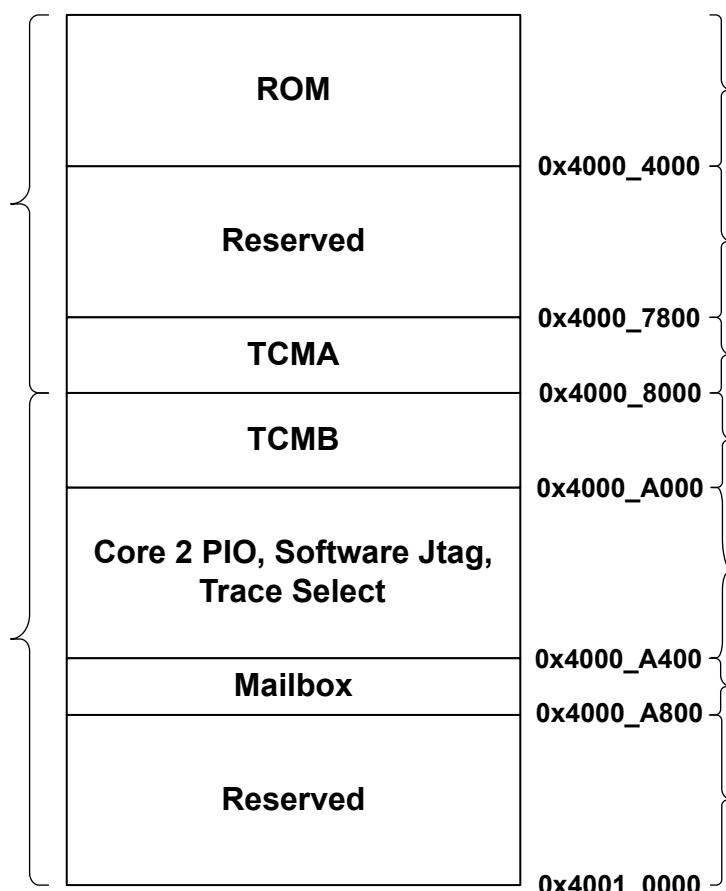
Table 3-2 BE-U1000 Core 2 Address Space

Region	Block Name	Size	Start Address	End Address	Register Map
Core 2 Resources	Core debug	4KB	0x0000_0000	0x0000_0FFF	
	CLINT	64KB	0x0200_0000	0x0200_FFFF	See <a href="#">CLINT Registers Map</a> table
	ROM core2	16KB	0x4000_0000	0x4000_3FFF	
	TCMA core2	2KB	0x4000_7800	0x4000_7FFF	
	TCMB core2	8KB	0x4000_8000	0x4000_9FFF	
	Core 2 PIO, Software Jtag and Trace Select	1KB	0x4000_A000	0x4000_A3FF	For Core 2 PIO registers, see <a href="#">PIO Registers Map</a> table
					Core 2 Software Jtag register, see <a href="#">Software JTAG Register Map</a> table

**Table 3-2 BE-U1000 Core 2 Address Space (continued)**

Region	Block Name	Size	Start Address	End Address	Register Map
					Core 2 Trace Select register, see <a href="#">Trace Select Register Map</a> table
	Mailbox	1KB	0x4000_A400	0x4000_A800	See <a href="#">Mailbox Registers Map</a> table

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

Core 0 and Core 1 of the BE-U1000 microcontroller have a *Core-Local Interrupt Controller (CLIC)* conforming to the RISC-V specification:

- Core 0 CLIC with the 64 interrupt lines (not counting 16 interrupts with Interrupt ID from 0 to 15 that are reserved for legacy interrupts)
- Core 1 CLIC with the 64 interrupt lines (not counting 16 interrupts with Interrupt ID from 0 to 15 that are reserved for legacy interrupts) that are fully identical to the Core 0 CLIC interrupts

Core 0 and Core 1 of the BE-U1000 can process masked (int\_local) and non-masked (int\_nmi) interrupts. Masked interrupt processing is carried out by the CLIC and must guarantee saving the entire BE-U1000 context. Exiting from the processing of these interrupts is always carried out to the entry point. Non-masked interrupts are handled by the command execution subsystem of the corresponding core and require rebooting the BE-U1000 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 ID 71,72 and 73 come directly from the PA, PB and PC pins. They can be used for working in real-time applications. Selecting the source of these interrupts is determined by setting the appropriate 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	<code>usip</code> User software interrupt
1-2	Reserved	
3	Core	<code>msip</code> Machine software interrupt
4	Core	<code>utip</code> User timer interrupt
5-6	Reserved	
7	Core	<code>mtip</code> Machine timer interrupt
8-11	Reserved	
12	Core	<code>csip</code> CLIC software interrupt
13-15	Reserved	
16	QSPI_0	<code>qspi_0_irq</code> QSPI_0 combined interrupt
17	UART_0	<code>uart_0_irq</code> 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 <i>End of Conversion (EOC)</i> interrupt
51	ADC_1	adc_1_eoc_irq ADC_1 <i>End of Conversion (EOC)</i> interrupt
52	ADC_2	adc_2_eoc_irq ADC_2 <i>End of Conversion (EOC)</i> 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 full by the Core 2
70		mailbox_mb_1_irq Indicates that the MB1 is full 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 <b>Core0 Bus Error Unit (BEU)</b> interrupt
76	Core 1 Bus Error Unit	core1_bus_err_intr <b>Core1 Bus Error Unit (BEU)</b> interrupt
77	CRU PLL	pll_nlock Indicates that the PLL unlock (Active high)
78-79	Reserved	



In the table above asterisk symbol (\*) indicates:

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



Interrupts with Interrupt ID 28, 45 from the GPIO\_0 and GPIO\_1 can be generated inside these blocks with or without the glitch filtering option. For more information refer to the [Debounce](#)



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

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	<code>wdt_1_irq</code> 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	<code>mailbox_mb_0_irq</code> Indicates that the MB0 is full by the Core 0, Core 1
1		<code>mailbox_mb_1_irq</code> Indicates that the MB1 is full by the Core 0, Core 1
*2	Port A	<code>pa_irq</code> Interrupt from the Port A
*3	Port B	<code>pb_irq</code> Interrupt from the Port B
*4	Port C	<code>pc_irq</code> Interrupt from the Port C
5	USB	<code>mc_nint</code> USB combined interrupt
6	USB DMA	<code>dma_nint</code> USB DMA interrupt
7-15	Reserved	



In the table above asterisk symbol (\*) indicates:

- 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 BE-U1000 system resources:

- selection the source of the clock signals;
- clock signals 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](#)

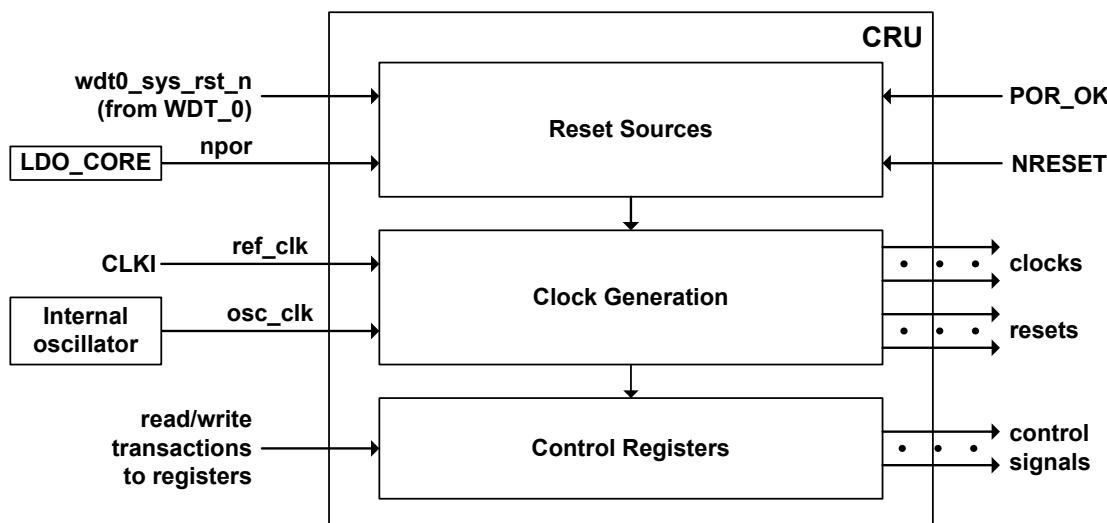


Figure 5-1 CRU Block Diagram

#### 5.1.1. Clock Generation

In this section:

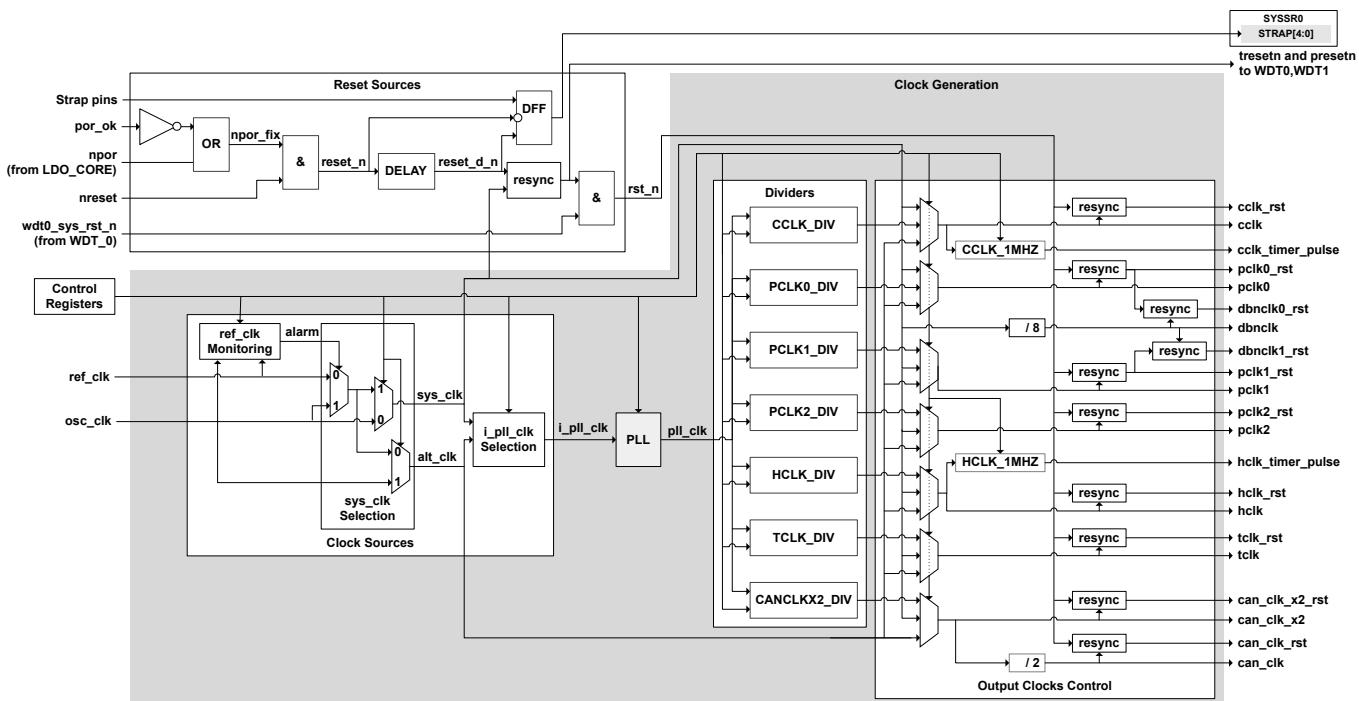
- [Clock Management](#)
- [Clock Domains](#)

##### 5.1.1.1. Clock Management

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

The Clock and Reset contain the following blocks (see the diagram below):

- [Clock Sources](#)
- [Phase-Locked Loop \(PLL\)](#)
- [Dividers](#)
- [Output Clocks](#)



**Figure 5-2 Clock and Reset 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 <code>cclk</code> Clock Domain
*hclk	100	Clock for the <code>hclk</code> Clock Domain
pclk0	100	Clock for the <code>pclk_0</code> Clock Domain
pclk1	100	Clock for the <code>pclk_1</code> Clock Domain
pclk2	100	Clock for the <code>pclk_2</code> Clock Domain
tclk	100	Clock for the <code>tclk</code> 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 asterisk symbol (\*) indicates:

- `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 divided by 2 `can_clk_x2` that is generated in the [Output Clocks](#) block
- `dbnclk` is obtained from the divided by 8 `sys_clk`.



For detailed description of the BE-U1000 system domains and devices they include, refer to [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 selected as `sys_clk`, `alt_clk` and `i_pll_clk` outputs. It also contains the `ref_clk` Monitoring block for tracking the `ref_clk` presence. The figure below shows the Clock Sources block structure.

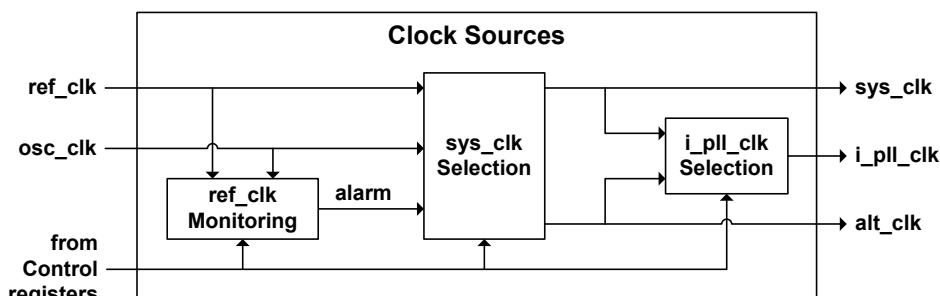


Figure 5-3 Clock Sources Block Diagram

##### 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.

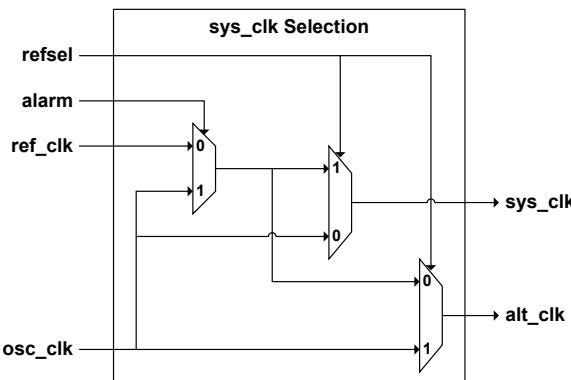


Figure 5-4 sys\_clk Selection Block

As shown in the figure above, the `sys_clk` Selection block consists of several multiplexors 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 for tracking the `ref_clk` signal presence. You should set the `ALARM_EN` bit of the `SYSCR0` Register to `0x1` to allow the block to work. 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

<code>SYSSR0.STRAP[3]</code> value	<code>CLKCR0.BOOTCLKSEL</code> value	<code>refsel</code> value
0x0	0x0	1
0x0	0x1	1
0x1	0x0	0
0x1	0x1	1

Depending on the `alarm` and `refsel` signals values, 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

<code>alarm</code> value	<code>refsel</code> value	<code>sys_clk</code> corresponds to:	<code>alt_clk</code> corresponds to:
0	0	<code>osc_clk</code>	<code>ref_clk</code>
0	1	<code>ref_clk</code>	<code>osc_clk</code>
1	0	<code>osc_clk</code>	<code>osc_clk</code>
1	1	<code>osc_clk</code>	<code>osc_clk</code>



- When `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 `SYSSR0` register to 0x0. Setting of this bit will also stop 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 setting the `I_PLL_CLK_SEL` bit of the `CLKSEL` register, the `i_pll_clk` will corresponds either to `sys_clk` or `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 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 16.2 MHz - 1300 MHz;
- allowed VCO range 260 MHz – 1300 MHz.

#### 5.1.1.1.2.1. PLL Output Frequency

The following figure shows the PLL block diagram.

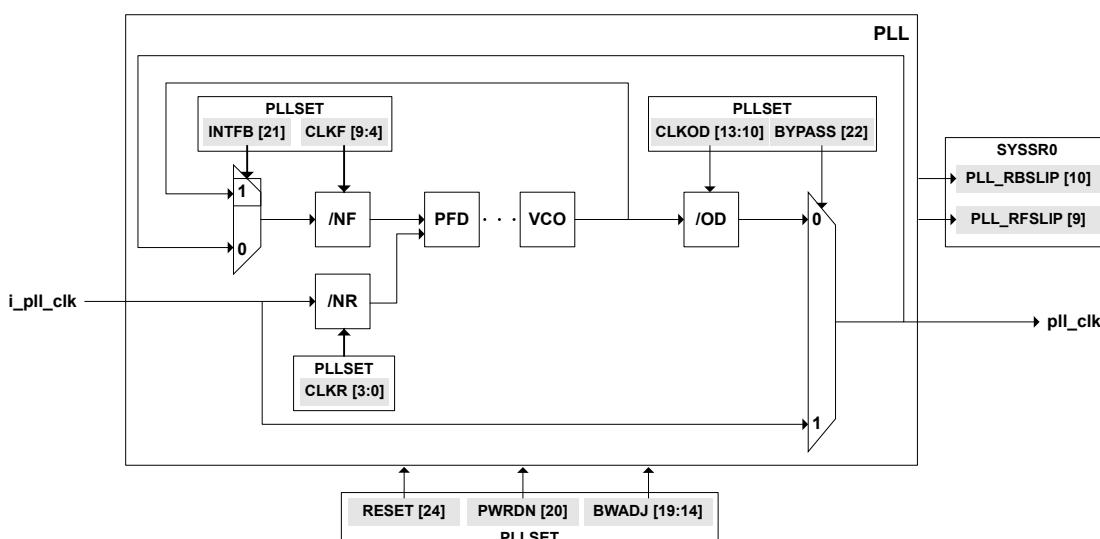


Figure 5-5 PLL Block Diagram

## 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:

$$\begin{aligned} pll\_clk &= \frac{i\_pll\_clk * NF}{NR * OD} \\ F_{VCO} &= \frac{i\_pll\_clk * NF}{NR} \end{aligned}$$



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

The following table contains recommended PLL settings for the `i_pll_clk` frequency of 25 MHz.

**Table 5-5 Recommended PLL Settings for the `i_pll_clk` Frequency of 25 MHz**

NR	NF	OD	NB	<code>pll_clk, MHz</code>
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`).

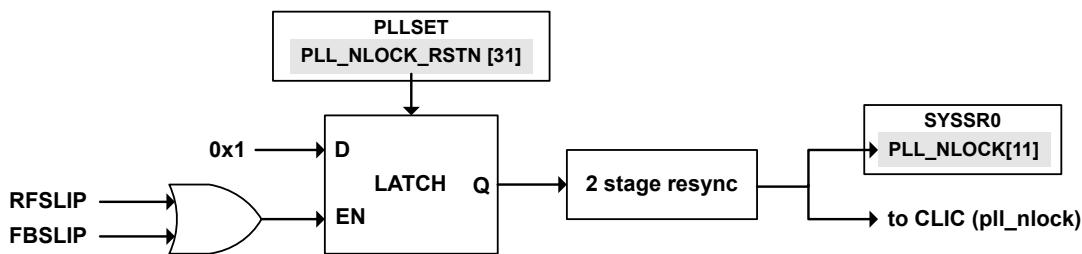
`PLLSET.BYPASS` controls the PLL's output multiplexer that selects either the reference clock or the clock from the PLL's output divider. If the PLL is locked and `BYPASS` is asserted, then the overall power dissipation will be similar to what one would see under functional mode operation.

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 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 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 following figure shows the PLL stability tracking.

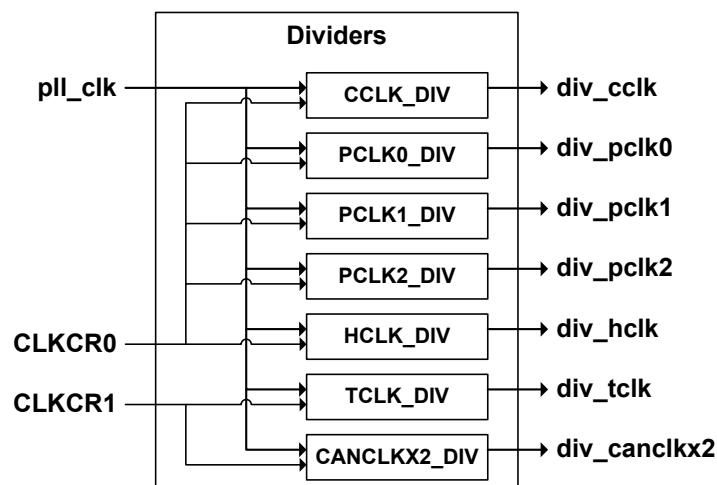


**Figure 5-6 PLL Stability Tracking**

**PLL\_NLOCK** is enabled to read at the **SYSSR0** and it is also the source of the **pll\_nlock** interrupt to Core 0 and Core 1 CLIC.

#### 5.1.1.1.3. Dividers

Dividers block is used to create a clock signals of the required frequency, based on the **clk\_pll** signal. It contains seven programmable dividers that are controlled through the appropriate bit fields of the **CLKCR0** and **CLKCR1** Registers as shown in the following figure.



**Figure 5-7 Dividers Block Diagram**

#### 5.1.1.1.4. Output Clocks

The Output Clocks 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.

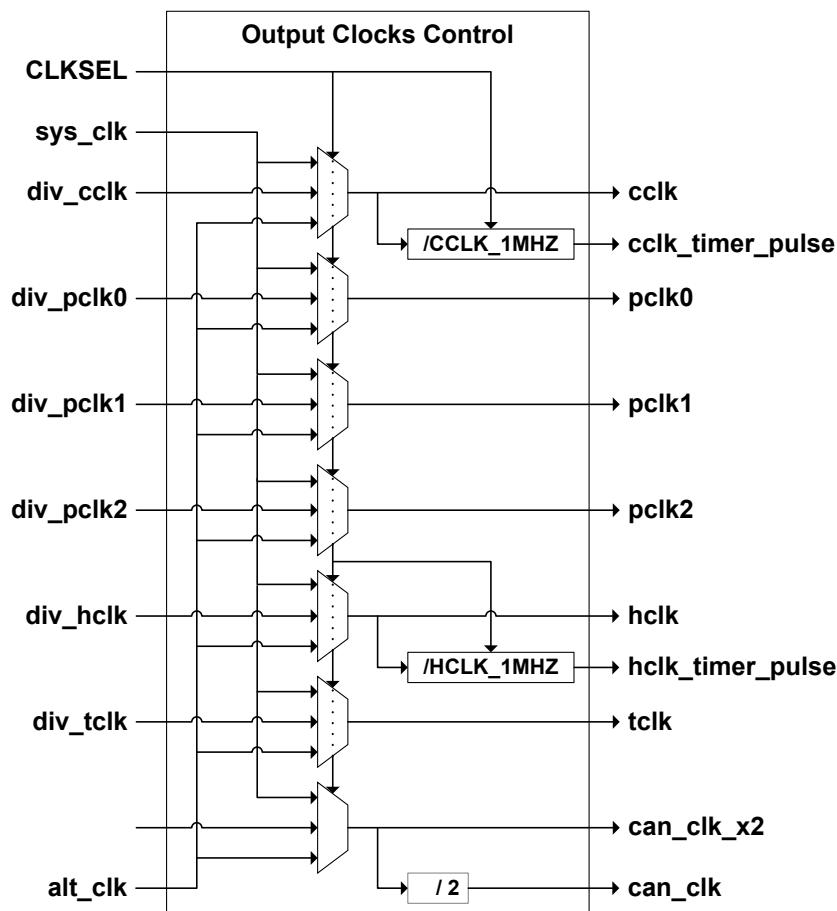


Figure 5-8 Output Clocks Block Diagram

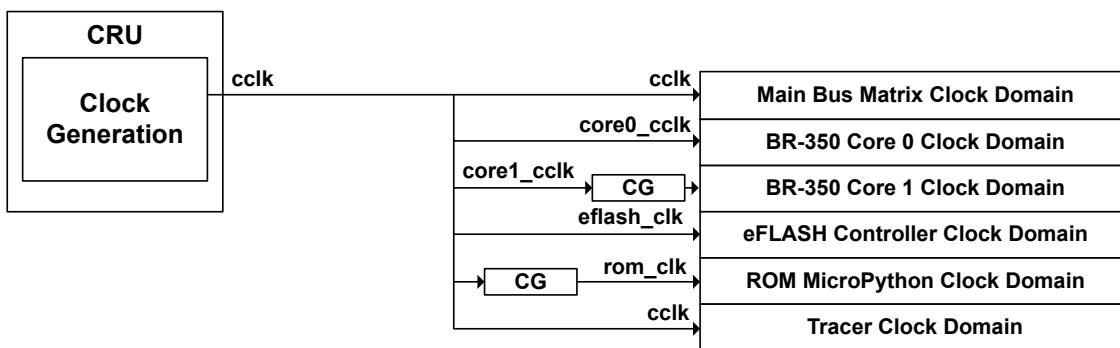
### 5.1.1.2. Clock Domains

In this section:

- [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**

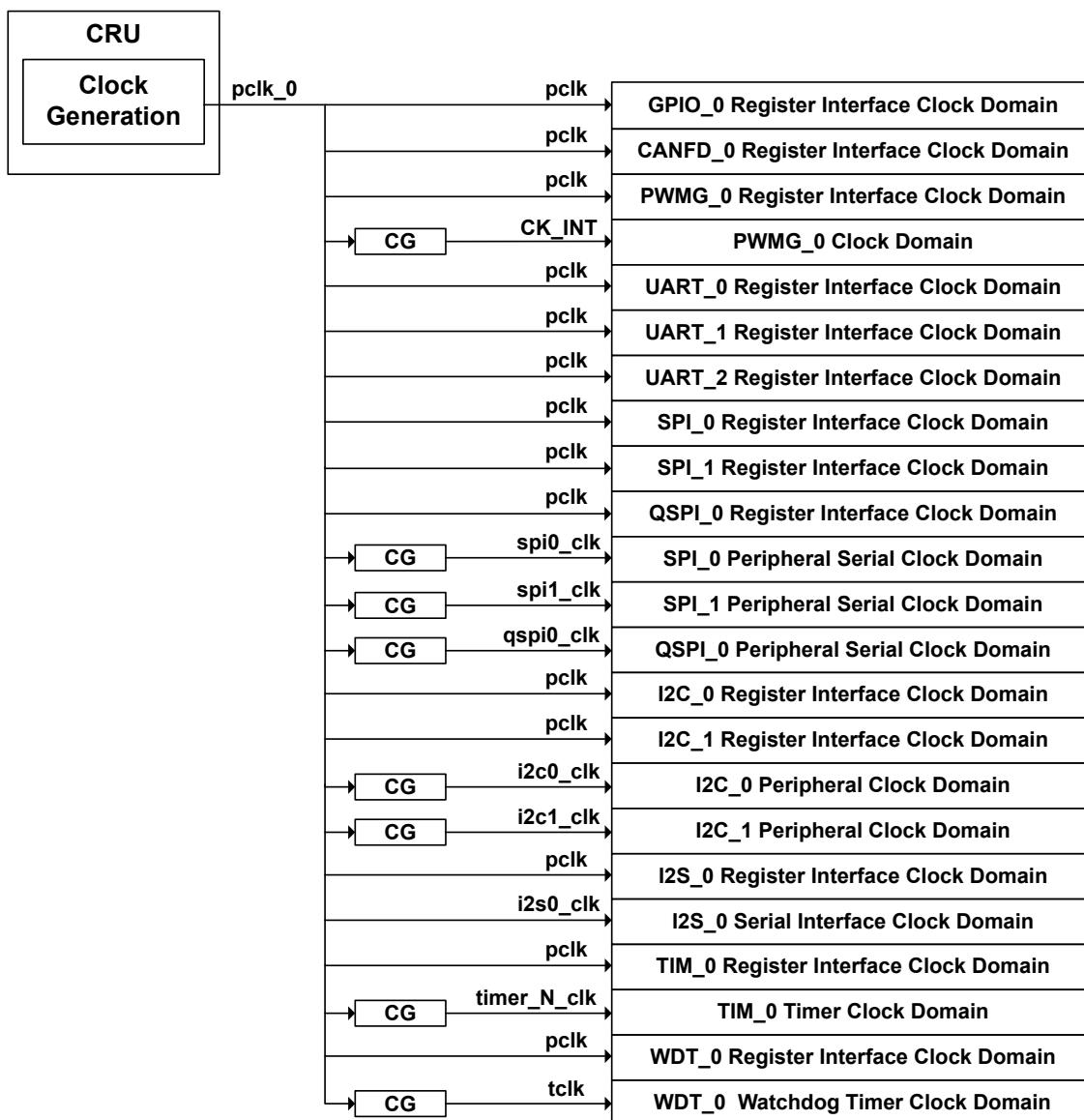
As shown in the figure above, the `rom_clk` and `core1_cclk` are gated.



`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.

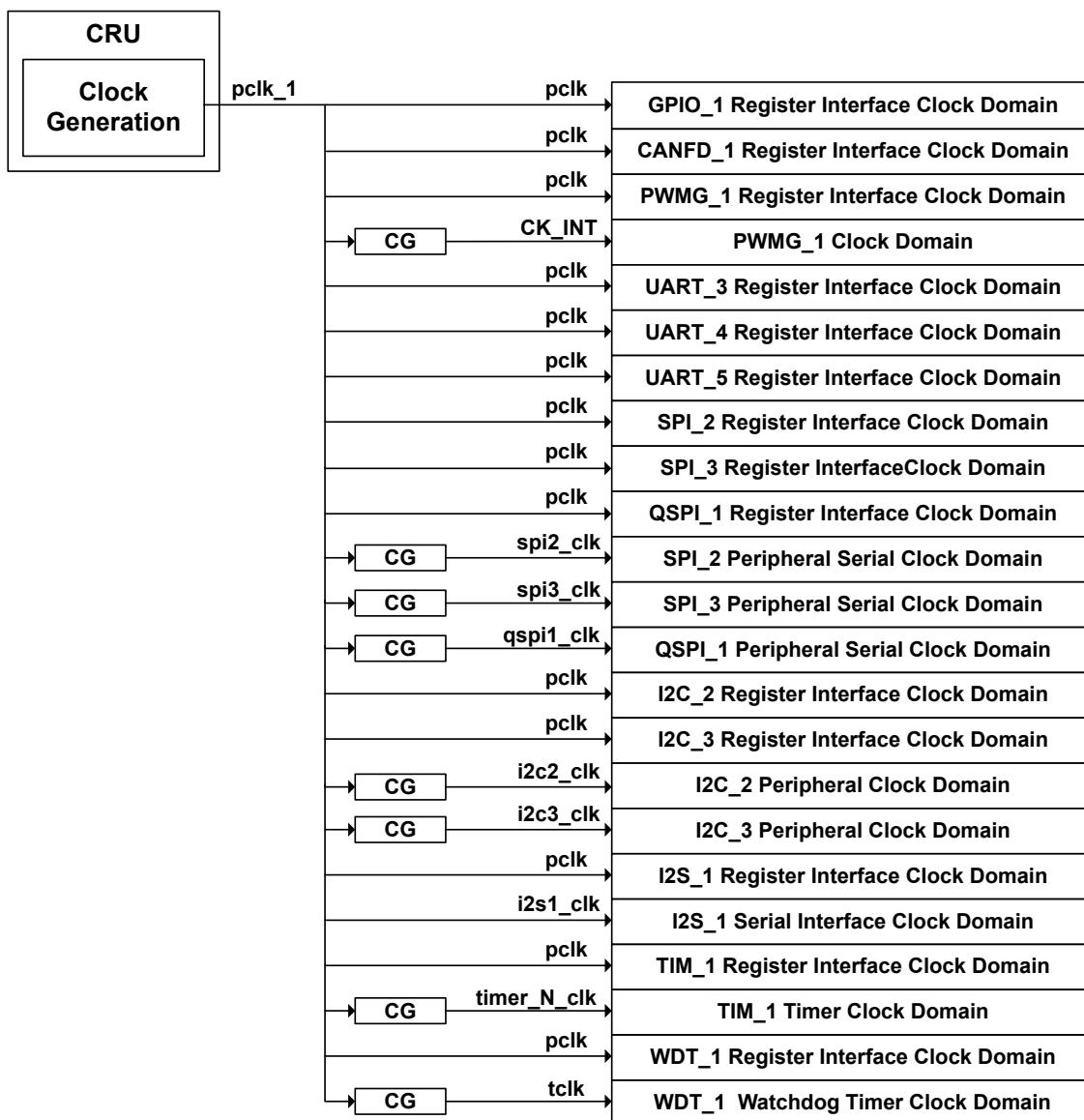


**Figure 5-10 `pclk_0` Clock Domain**

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.

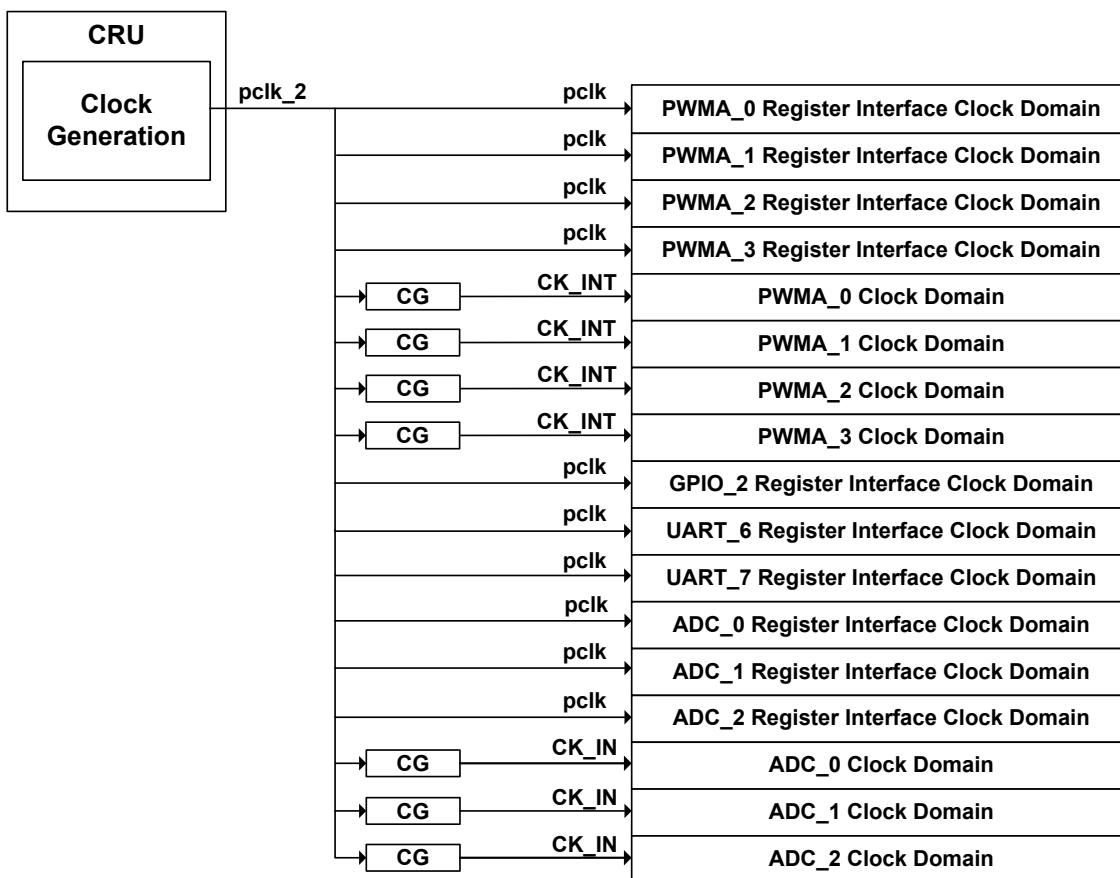


Figure 5-12 `pclk_2` Clock Domain

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.

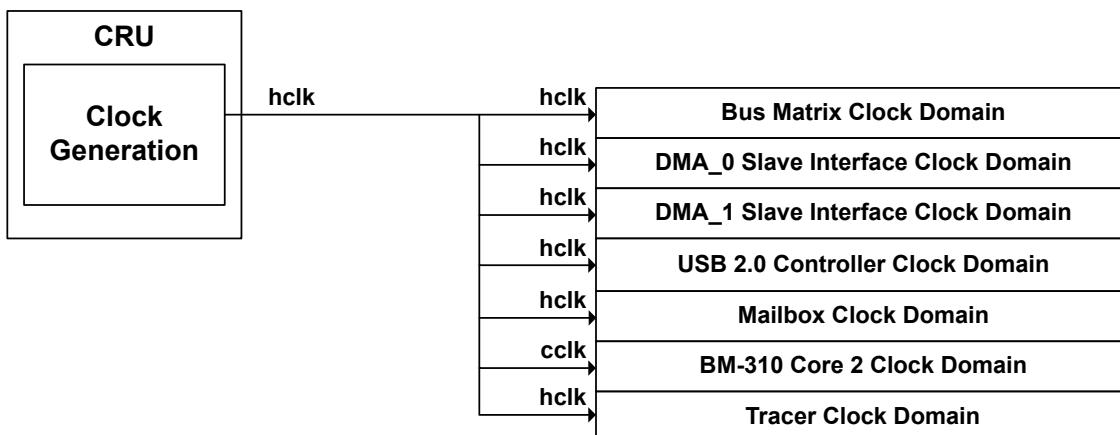
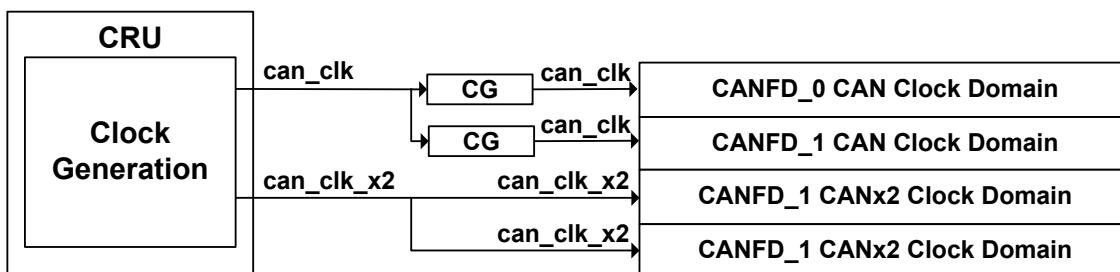


Figure 5-13 `hclk` Clock Domain

#### 5.1.1.2.6. `can_clk` and `can_clk_x2` Clock Domains

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



**Figure 5-14 can\_clk and can\_clk\_x2 Clock Domains**

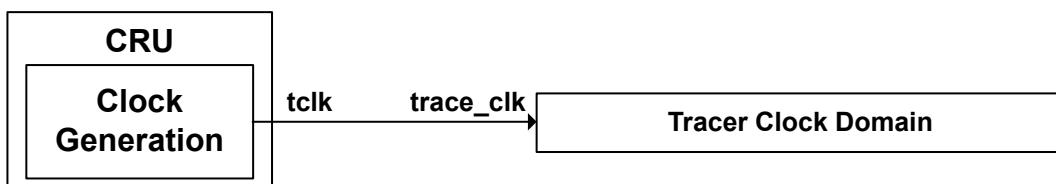
As shown in the figure above, some clocks of the `can_clk` clock domain are gated.



`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.

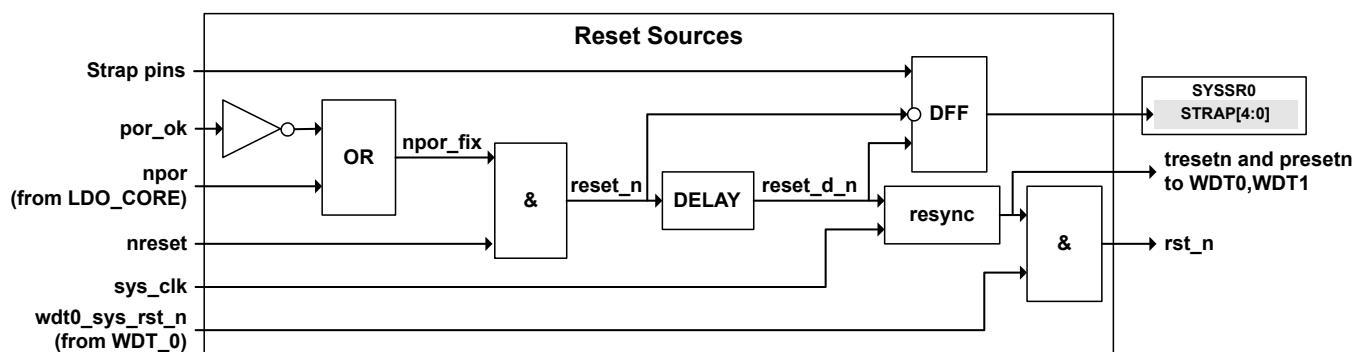


**Figure 5-15 tclk Clock Domain**

## 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 above



**Figure 5-16 Reset Sources Block Diagram**

For more information 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 refer to [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	Reset signal from LDO. 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 <code>npor_fix = npor   !por_ok</code>
*reset_n	Low	Reset signal for triggers that fix the value of the strap pins.
reset_d_n	Low	The <code>reset_n</code> signal delayed on the delay line. It is the clock for triggers that fix the value of the strap pins.
por_ok	Low	Enables the use of the <code>npor</code> signal
nreset	Low	Reset signal from the external NRESET pin. See chapter 5 <b>Pinout and pin description</b> of the <b>BE-U1000 Datasheet</b> for details.
wdt0_sys_RST_n	Low	Reset signal from the WDT_0.



In the table above asterisk symbol (\*) indicates `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	Reset for the WDT_0 and WDT_1. Used to reset the watchdog counter.
*presetn	Low	Reset for the WDT_0 and WDT_1. Used to reset the watchdog counter.
*rst_n	Low	Reset for the Clock Generation. Used for generation the BE-U1000 system domains reset signals. For more information about Clock Generation, refer to <a href="#">Clock Management</a> .



In the table above asterisk symbol (\*) indicates:

- `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, which form the following ports (with 16 I/O pins in each):

- 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 allows to choose one of six functions for each I/O pin. Each I/O pin multiplexor 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 multiplexor as an example.

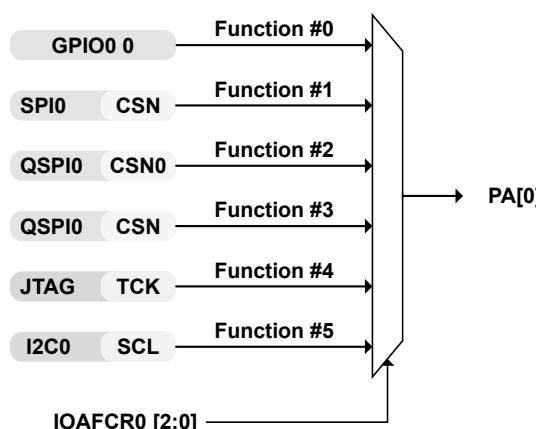


Figure 5-17 PA[0] I/O pin multiplexor

The figures below illustrate the available per I/O pin functions for each port and dependence between the `IOAFCRx` registers settings 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[20:18]	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[20:18]	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[20:18]	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 follows:

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

## 5.2. Control Registers

Registers region of the CRU mapped in the BE-U1000 microcontroller Memory Map as the following table shows.

**Table 5-8 Memory Address Region for Control Registers**

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



For details, refer to [Memory Map](#) chapter.

In this chapter:

- [Registers Map](#)
- [Register Descriptions](#)

### 5.2.1. Registers Map

The following table provides 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 Registers Map**

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

**Table 5-9 CRU Registers 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 of a register description 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	<p><code>hclk</code> clock divider value for generating a 1 MHz pulse that is used as <code>timer_pulse</code> Core 2 input. The resulting frequency is <code>hclk</code> divided by (<code>HCLK1MHZ</code> + 1).</p> <p><b>Value After Reset:</b> 0x18</p>
[23:15]	CCLK1MHZ	R/W	<p><code>cclk</code> clock divider value for generating a 1 MHz pulse that is used as <code>timer_pulse</code> Core 0, Core 1 input. The resulting frequency is <code>cclk</code> divided by (<code>CCLK1MHZ</code> + 1).</p> <p><b>Value After Reset:</b> 0x30</p>
[14:13]	CANX2CLKSEL	R/W	<p>Selects the source of the <code>can_clk_x2</code> signal of the <a href="#">can_clk</a> and <a href="#">can_clk_x2 Clock Domains</a>.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <code>sys_clk</code> signal</li> <li><b>0x1:</b> <code>div_tclk</code> signal</li> <li><b>0x2:</b> <code>alt_clk</code> signal</li> <li><b>0x3:</b> Reserved</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[12:11]	TCLK_SEL	R/W	<p>Selects the source of the <code>tclk</code> signal of the <a href="#">tclk Clock Domain</a>.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <code>sys_clk</code> signal</li> <li><b>0x1:</b> <code>div_tclk</code> signal</li> <li><b>0x2:</b> <code>alt_clk</code> signal</li> <li><b>0x3:</b> Reserved</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[10:9]	HCLK_SEL	R/W	<p>Selects the source of the <code>hclk</code> signal of the <a href="#">hclk Clock Domain</a></p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <code>sys_clk</code> signal</li> <li><b>0x1:</b> <code>div_hclk</code> signal</li> <li><b>0x2:</b> <code>alt_clk</code> signal</li> <li><b>0x3:</b> Reserved</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[8:7]	PCLK2_SEL	R/W	<p>Selects the source of the <code>pclk_2</code> signal of the <a href="#">pclk_2 Clock Domain</a></p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <code>sys_clk</code> signal</li> <li><b>0x1:</b> <code>div_pclk2</code> signal</li> <li><b>0x2:</b> <code>alt_clk</code> signal</li> <li><b>0x3:</b> Reserved</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[6:5]	PCLK1_SEL	R/W	<p>Selects the source of the <code>pclk_1</code> signal of the <a href="#">pclk_1 Clock Domain</a></p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <code>sys_clk</code> signal</li> <li><b>0x1:</b> <code>div_pclk1</code> signal</li> </ul>

**Table 5-10 Fields For Register: CLKSEL (continued)**

Bits	Name	R/W	Description
			<p><b>0x2:</b> alt_clk signal  <b>0x3:</b> Reserved</p> <p><b>Value After Reset:</b> 0x0</p>
[4:3]	PCLK0_SEL	R/W	<p>Selects the source of the <code>pclk_0</code> signal of the <code>pclk_0</code> Clock Domain</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> sys_clk signal</li> <li><b>0x1:</b> div_pclk0 signal</li> <li><b>0x2:</b> alt_clk signal</li> <li><b>0x3:</b> Reserved</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[2:1]	CCLK_SEL	R/W	<p>Selects the source of the <code>cclk</code> signal of the <code>cclk</code> Clock Domain</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> sys_clk signal</li> <li><b>0x1:</b> div_cclk signal</li> <li><b>0x2:</b> alt_clk signal</li> <li><b>0x3:</b> Reserved</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[0]	I_PLL_CLK_SEL	R/W	<p>Selects the source of the <code>i_pll_clk</code> signal.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> sys_clk signal</li> <li><b>0x1:</b> alt_clk signal</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p><code>PLL_NLOCK</code> flag reset signal</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> reset is active</li> <li><b>0x1:</b> reset is inactive</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[30:25]			Reserved
[24]	RESET	R/W	<p>PLL RESET input value</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> reset is active</li> <li><b>0x0:</b> reset is inactive</li> </ul> <p><b>Value After Reset:</b> 0x1</p>

**Table 5-11 Fields For Register: PLLSET (continued)**

Bits	Name	R/W	Description
[23]	TEST	R/W	PLL TEST input value Reference-to-counters-to-output bypass when high
[22]	BYPASS	R/W	PLL BYPASS input value <b>Values:</b> 0x1: bypass mode is enabled 0x0: bypass mode is disabled <b>Value After Reset:</b> 0x1
[21]	INTFB	R/W	Selects the feedback signal. PLL INTFB input value <b>Values:</b> 0x1: internal signal  This bit should always be set to 0x1 value. <b>Value After Reset:</b> 0x0
[20]	PWRDN	R/W	PowerDown mode is enabled. PLL PWRDN input value <b>Values:</b> 0x1: PowerDown mode is enabled 0x0: PowerDown mode is disabled <b>Value After Reset:</b> 0x0
[19:14]	BWADJ	R/W	PLL BWADJ input value NB = BWADJ[19:14] + 1, BWADJ[14] is LSB <b>Value After Reset:</b> 0x0
[13:10]	CLKOD	R/W	PLL CLKOD input value OD = CLKOD[13:10] + 1, CLKOD[10] is LSB <b>Value After Reset:</b> 0x0
[9:4]	CLKF	R/W	PLL CLKF input value NF = CLKF[9:4] + 1, CLKF[4] is LSB <b>Value After Reset:</b> 0x0
[3:0]	CLKR	R/W	PLL CLKR input value NR = CLKR[3:0] + 1, CLKR[0] is LSB <b>Value After Reset:</b> 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	Selects the source of the boot signal. For more information refer to <a href="#">Clock Sources</a> <b>Value After Reset:</b> 0x0

**Table 5-12 Fields For Register: CLKCR0 (continued)**

Bits	Name	R/W	Description
[30]	OSCGENEN	R/W	<p>osc_clk generation enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Internal oscillator is enabled</li> <li><b>0x0:</b> Internal oscillator is disabled</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[29]	HCLKDIVEN	R/W	<p>Enables PLL output frequency divider:</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> divider is disabled</li> <li><b>0x1:</b> divider is enabled</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[28:24]	HCLKDIV	R/W	<p>hclk domain divisor value.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <code>HCLKDIV[0] = 0</code> - integer division. The frequency division ratio is equal to <code>HCLKDIV[4:1]+1</code>. Valid values <code>HCLKDIV[4:1]</code> from 0 to 15. If <code>HCLKDIV[4:0] = 0</code> - bypass</li> <li><b>0x1:</b> <code>HCLKDIV[0] = 1</code> - fractional division. The frequency division ratio is equal to <code>HCLKDIV[4:1]+1.5</code>. Valid values <code>HCLKDIV[4:1]</code> from 0 to 14.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[23]	PCLK2DIVEN	R/W	<p>Enables PLL output frequency divider:</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> divider is disabled</li> <li><b>0x1:</b> divider is enabled</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[22:18]	PCLK2DIV	R/W	<p>pclk_2 domain divisor value.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <code>PCLK2DIV[0] = 0</code> - integer division. The frequency division ratio is equal to <code>PCLK2DIV[4:1]+1</code>. Valid values <code>PCLK2DIV[4:1]</code> from 0 to 15. If <code>PCLK2DIV[4:0] = 0</code> - bypass</li> <li><b>0x1:</b> <code>PCLK2DIV[0] = 1</code> - fractional division. The frequency division ratio is equal to <code>PCLK2DIV[4:1]+1.5</code>. Valid values <code>PCLK2DIV[4:1]</code> from 0 to 14.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[17]	PCLK1DIVEN	R/W	<p>Enables PLL output frequency divider:</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> divider is disabled</li> <li><b>0x1:</b> divider is enabled</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[16:12]	PCLK1DIV	R/W	<p>pclk_1 domain divisor value.</p> <p><b>Values:</b></p>

**Table 5-12 Fields For Register: CLKCR0 (continued)**

Bits	Name	R/W	Description
			<p><b>0x0:</b> <code>PCLK1DIV[0]</code> = 0 - integer division. The frequency division ratio is equal to <code>PCLK1DIV[4:1]+1</code>. Valid values <code>PCLK1DIV[4:1]</code> from 0 to 15. If <code>PCLK1DIV[4:0]</code> = 0 - bypass</p> <p><b>0x1:</b> <code>PCLK1DIV[0]</code> = 1 - fractional division. The frequency division ratio is equal to <code>PCLK1DIV[4:1]+1.5</code>. Valid values <code>PCLK1DIV[4:1]</code> from 0 to 14.</p> <p><b>Value After Reset:</b> 0x0</p>
[11]	PCLK0DIVEN	R/W	<p>Enables PLL output frequency divider:</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> divider is disabled</li> <li><b>0x1:</b> divider is enabled</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[10:6]	PCLK0DIV	R/W	<p><code>pclk_0</code> domain divisor value.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <code>PCLK0DIV[0]</code> = 0 - integer division. The frequency division ratio is equal to <code>PCLK0DIV[4:1]+1</code>. Valid values <code>PCLK0DIV[4:1]</code> from 0 to 15. If <code>PCLK0DIV[4:0]</code> = 0 - bypass</li> <li><b>0x1:</b> <code>PCLK0DIV[0]</code> = 1 - fractional division. The frequency division ratio is equal to <code>PCLK0DIV[4:1]+1.5</code>. Valid values <code>PCLK0DIV[4:1]</code> from 0 to 14.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[5]	CCLKDIVEN	R/W	<p>Enables PLL output frequency divider:</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> divider is disabled</li> <li><b>0x1:</b> divider is enabled</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[4:0]	CCLKDIV	R/W	<p><code>cclk</code> domain divisor value.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <code>CCLKDIV[0]</code> = 0 - integer division. The frequency division ratio is equal to <code>CCLKDIV[4:1]+1</code>. Valid values <code>CCLKDIV[4:1]</code> from 0 to 15. If <code>CCLKDIV[4:0]</code> = 0 - bypass</li> <li><b>0x1:</b> <code>CCLKDIV[0]</code> = 1 - fractional division. The frequency division ratio is equal to <code>CCLKDIV[4:1]+1.5</code>. Valid values <code>CCLKDIV[4:1]</code> from 0 to 14.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

#### 5.2.2.4. `pclk0` Clock Gating and Resets 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	<p>CANFD_0 software reset</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x1: reset is inactive</li> <li>0x0: reset is active</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[28]	PWMG0RSTN	R/W	<p>PWMG_0 software reset</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x1: reset is inactive</li> <li>0x0: reset is active</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[27]	GPIO0RSTN	R/W	<p>GPIO_0 software reset</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x1: reset is inactive</li> <li>0x0: reset is active</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[26]	WDT0RSTN	R/W	<p>WDT_0 software reset</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x1: reset is inactive</li> <li>0x0: reset is active (when <code>WDTCLRKEY</code>=0xd09c1ea7)</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[25]	TIMER0RSTN	R/W	<p>TIM_0 software reset</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x1: reset is inactive</li> <li>0x0: reset is active</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[24]	I2S0RSTN	R/W	<p>I2S_0 software reset</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x1: reset is inactive</li> <li>0x0: reset is active</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[23]	I2C1RSTN	R/W	<p>I2C_1 software reset</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x1: reset is inactive</li> <li>0x0: reset is active</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[22]	I2C0RSTN	R/W	<p>I2C_0 software reset</p> <p><b>Values:</b></p>

**Table 5-13 Fields For Register: PCLK0EN (continued)**

Bits	Name	R/W	Description
			<p><b>0x1:</b> reset is inactive  <b>0x0:</b> reset is active  <b>Value After Reset:</b> 0x1</p>
[21]	UART2RSTN	R/W	<p>UART_2 software reset  <b>Values:</b>  <b>0x1:</b> reset is inactive  <b>0x0:</b> reset is active  <b>Value After Reset:</b> 0x1</p>
[20]	UART1RSTN	R/W	<p>UART_1 software reset  <b>Values:</b>  <b>0x1:</b> reset is inactive  <b>0x0:</b> reset is active  <b>Value After Reset:</b> 0x1</p>
[19]	UART0RSTN	R/W	<p>UART_0 software reset  <b>Values:</b>  <b>0x1:</b> reset is inactive  <b>0x0:</b> reset is active  <b>Value After Reset:</b> 0x1</p>
[18]	SPI1RSTN	R/W	<p>SPI_1 software reset  <b>Values:</b>  <b>0x1:</b> reset is inactive  <b>0x0:</b> reset is active  <b>Value After Reset:</b> 0x1</p>
[17]	SPI0RSTN	R/W	<p>SPI_0 software reset  <b>Values:</b>  <b>0x1:</b> reset is inactive  <b>0x0:</b> reset is active  <b>Value After Reset:</b> 0x1</p>
[16]	QSPI0RSTN	R/W	<p>QSPI_0 software reset  <b>Values:</b>  <b>0x1:</b> reset is inactive  <b>0x0:</b> reset is active  <b>Value After Reset:</b> 0x1</p>
[15:14]			Reserved
[13]	CAN0CLKEN	R/W	<p>Enables the <code>can_clk</code> of the CANFD_0  <b>Values:</b>  <b>0x1:</b> enabled  <b>0x0:</b> disabled  <b>Value After Reset:</b> 0x1</p>

**Table 5-13 Fields For Register: PCLK0EN (continued)**

Bits	Name	R/W	Description
[12]	PWMG0CLKEN	R/W	<p>Enables the <code>CK_INT</code> of the PWMG_0</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> enabled</li> <li><b>0x0:</b> disabled</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[11]			Reserved
[10]	WDT0CLKEN	R/W	<p>Enables the <code>tclk</code> of the WDT_0</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> enabled</li> <li><b>0x0:</b> disabled</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[9]	TIMER0CLKEN	R/W	<p>Enables the <code>timer_N_clk</code> of the TIM_0</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> enabled</li> <li><b>0x0:</b> disabled</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[8]			Reserved
[7]	I2C1CLKEN	R/W	<p>Enables the <code>i2c1_clk</code> of the I2C_1</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> enabled</li> <li><b>0x0:</b> disabled</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[6]	I2C0CLKEN	R/W	<p>Enables the <code>i2c0_clk</code> of the I2C_0</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> enabled</li> <li><b>0x0:</b> disabled</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[5:3]			Reserved
[2]	SPI1CLKEN	R/W	<p>Enables the <code>spi1_clk</code> of the SPI_1</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> enabled</li> <li><b>0x0:</b> disabled</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[1]	SPI0CLKEN	R/W	<p>Enables the <code>spi0_clk</code> of the SPI_0</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> enabled</li> <li><b>0x0:</b> disabled</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[0]	QSPI0CLKEN	R/W	Enables the <code>qspi0_clk</code> of the QSPI_0

**Table 5-13 Fields For Register: PCLK0EN (continued)**

Bits	Name	R/W	Description
			<b>Values:</b> <b>0x1:</b> enabled <b>0x0:</b> disabled <b>Value After Reset:</b> 0x1

### 5.2.2.5. `pclk1` Clock Gating and Resets 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 <b>Values:</b> <b>0x1:</b> reset is inactive <b>0x0:</b> reset is active <b>Value After Reset:</b> 0x1
[28]	PWMG1RSTN	R/W	PWMG_1 software reset <b>Values:</b> <b>0x1:</b> reset is inactive <b>0x0:</b> reset is active <b>Value After Reset:</b> 0x1
[27]	GPIO1RSTN	R/W	GPIO_1 software reset <b>Values:</b> <b>0x1:</b> reset is inactive <b>0x0:</b> reset is active <b>Value After Reset:</b> 0x1
[26]	WDT1RSTN	R/W	WDT_1 software reset <b>Values:</b> <b>0x1:</b> reset is inactive <b>0x0:</b> reset is active (when <code>WDTCLRKEY.WDTCLRKEY=0xd09c1ea7</code> ) <b>Value After Reset:</b> 0x1
[25]	TIMER1RSTN	R/W	TIM_1 software reset <b>Values:</b> <b>0x1:</b> reset is inactive <b>0x0:</b> reset is active <b>Value After Reset:</b> 0x1
[24]	I2S1RSTN	R/W	I2S_1 software reset <b>Values:</b>

**Table 5-14 Fields For Register: PCLK1EN (continued)**

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

**Table 5-14 Fields For Register: PCLK1EN (continued)**

Bits	Name	R/W	Description
			<p><b>0x1:</b> reset is inactive  <b>0x0:</b> reset is active</p> <p><b>Value After Reset:</b> 0x1</p>
[15:14]			Reserved
[13]	CAN1CLKEN	R/W	<p>Enables the <code>can_clk</code> of the CANFD_1</p> <p><b>Values:</b></p> <p><b>0x1:</b> enabled  <b>0x0:</b> disabled</p> <p><b>Value After Reset:</b> 0x1</p>
[12]	PWMG1CLKEN	R/W	<p>Enables the <code>CK_INT</code> of the PWMG_1</p> <p><b>Values:</b></p> <p><b>0x1:</b> enabled  <b>0x0:</b> disabled</p> <p><b>Value After Reset:</b> 0x1</p>
[11]			Reserved
[10]	WDT1CLKEN	R/W	<p>Enables the <code>tclk</code> of the WDT_1</p> <p><b>Values:</b></p> <p><b>0x1:</b> enabled  <b>0x0:</b> disabled</p> <p><b>Value After Reset:</b> 0x1</p>
[9]	TIMER1CLKEN	R/W	<p>Enables the <code>timer_N_clk</code> of the TIM_1</p> <p><b>Values:</b></p> <p><b>0x1:</b> enabled  <b>0x0:</b> disabled</p> <p><b>Value After Reset:</b> 0x1</p>
[8]			Reserved
[7]	I2C3CLKEN	R/W	<p>Enables the <code>i2c3_clk</code> of the I2C_3</p> <p><b>Values:</b></p> <p><b>0x1:</b> enabled  <b>0x0:</b> disabled</p> <p><b>Value After Reset:</b> 0x1</p>
[6]	I2C2CLKEN	R/W	<p>Enables the <code>i2c2_clk</code> of the I2C_2</p> <p><b>Values:</b></p> <p><b>0x1:</b> enabled  <b>0x0:</b> disabled</p> <p><b>Value After Reset:</b> 0x1</p>
[5:3]			Reserved
[2]	SPI3CLKEN	R/W	<p>Enables the <code>spi3_clk</code> of the SPI_3</p> <p><b>Values:</b></p>

**Table 5-14 Fields For Register: PCLK1EN (continued)**

Bits	Name	R/W	Description
			<p><b>0x1:</b> enabled  <b>0x0:</b> disabled</p> <p><b>Value After Reset:</b> 0x1</p>
[1]	SPI2CLKEN	R/W	<p>Enables the <code>spi2_clk</code> of the SPI_2</p> <p><b>Values:</b></p> <p><b>0x1:</b> enabled  <b>0x0:</b> disabled</p> <p><b>Value After Reset:</b> 0x1</p>
[0]	QSPI1CLKEN	R/W	<p>Enables the <code>qspi1_clk</code> of the QSPI_1</p> <p><b>Values:</b></p> <p><b>0x1:</b> enabled  <b>0x0:</b> disabled</p> <p><b>Value After Reset:</b> 0x1</p>

#### 5.2.2.6. `pclk2` Clock Gating and Resets 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	<p>UART_7 software reset</p> <p><b>Values:</b></p> <p><b>0x1:</b> reset is inactive  <b>0x0:</b> reset is active</p> <p><b>Value After Reset:</b> 0x1</p>
[24]	UART6RSTN	R/W	<p>UART_6 software reset</p> <p><b>Values:</b></p> <p><b>0x1:</b> reset is inactive  <b>0x0:</b> reset is active</p> <p><b>Value After Reset:</b> 0x1</p>
[23]	GPIO2RSTN	R/W	<p>GPIO_2 software reset</p> <p><b>Values:</b></p> <p><b>0x1:</b> reset is inactive  <b>0x0:</b> reset is active</p> <p><b>Value After Reset:</b> 0x1</p>
[22]	PWMA3RSTN	R/W	<p>PWMA_3 software reset</p> <p><b>Values:</b></p> <p><b>0x1:</b> reset is inactive  <b>0x0:</b> reset is active</p>

**Table 5-15 Fields For Register: PCLK2EN (continued)**

Bits	Name	R/W	Description
			<b>Value After Reset:</b> 0x1
[21]	PWMA2RSTN	R/W	<p>PWMA_2 software reset</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> reset is inactive</li> <li><b>0x0:</b> reset is active</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[20]	PWMA1RSTN	R/W	<p>PWMA_1 software reset</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> reset is inactive</li> <li><b>0x0:</b> reset is active</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[19]	PWMA0RSTN	R/W	<p>PWMA_0 software reset</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> reset is inactive</li> <li><b>0x0:</b> reset is active</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[18]	ADC2RSTN	R/W	<p>ADC_2 software reset</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> reset is inactive</li> <li><b>0x0:</b> reset is active</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[17]	ADC1RSTN	R/W	<p>ADC_1 software reset</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> reset is inactive</li> <li><b>0x0:</b> reset is active</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[16]	ADC0RSTN	R/W	<p>ADC_0 software reset</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> reset is inactive</li> <li><b>0x0:</b> reset is active</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[15:14]			Reserved
[6]	PWMA3CLKEN	R/W	<p>Enables the CK_INT of the PWMA_3</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> enabled</li> <li><b>0x0:</b> disabled</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[5]	PWMA2CLKEN	R/W	<p>Enables the CK_INT of the PWMA_2</p> <p><b>Values:</b></p>

**Table 5-15 Fields For Register: PCLK2EN (continued)**

Bits	Name	R/W	Description
			<b>0x1:</b> enabled <b>0x0:</b> disabled <b>Value After Reset:</b> 0x1
[4]	PWMA1CLKEN	R/W	Enables the CK_IN of the PWMA_1 <b>Values:</b> <b>0x1:</b> enabled <b>0x0:</b> disabled <b>Value After Reset:</b> 0x1
[3]	PWMA0CLKEN	R/W	Enables the CK_IN of the PWMA_0 <b>Values:</b> <b>0x1:</b> enabled <b>0x0:</b> disabled <b>Value After Reset:</b> 0x1
[2]	ADC2CLKEN	R/W	Enables the CK_IN of the ADC_2 <b>Values:</b> <b>0x1:</b> enabled <b>0x0:</b> disabled <b>Value After Reset:</b> 0x1
[1]	ADC1CLKEN	R/W	Enables the CK_IN of the ADC_1 <b>Values:</b> <b>0x1:</b> enabled <b>0x0:</b> disabled <b>Value After Reset:</b> 0x1
[0]	ADC0CLKEN	R/W	Enables the CK_IN of the ADC_0 <b>Values:</b> <b>0x1:</b> enabled <b>0x0:</b> disabled <b>Value After Reset:</b> 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	DMA protocol lock flag. Disables the modification of the FLASH memory using the DMA protocol: <b>Values:</b>

**Table 5-16 Fields For Register: SYSCR0 (continued)**

Bits	Name	R/W	Description
			<p><b>0x1:</b> means that the controller can't modify (program) the FLASH memory using the DMA protocol</p> <p><b>0x0:</b> means that the controller can modify (program) the FLASH memory using the DMA protocol</p> <p><b>Value After Reset:</b> 0x1</p>
[30]	SOFTDBGEN	R/W	<p>Enables software debugging of the Core 0, Core 1</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> enabled</li> <li><b>0x0:</b> disabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[29:27]			Reserved for internal use
[26]	FLASHNVRDIS	R/W	<p>NVR disable flag. Disables the access to the FLASH NVR array. Can only be set to 1 by the software. Can only be reset upon power-up or via an external <code>nreset</code> signal.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> 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)</li> <li><b>0x0:</b> enabled (means that the controller can perform an operation in the NVR array)</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[25:17]			Reserved
[16]	WDTHITCLR	R/W	<p>WDTHIT flag reset of the <code>SYSSR0</code> register</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> WDTHIT flag is reseten</li> <li><b>0x0:</b> WDTHIT flag is not reseten</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[15:12]	ALARM_LVL	R/W	<p>The <code>alarm</code> signal is generated when the internal counter of the <code>ref_clk</code> Monitoring block reaches the value written to this bit field. For more information about <code>alarm</code> refer to <a href="#">Clock Sources</a></p> <p><b>Value After Reset:</b> 0x0</p>
[11]	ALARM_EN	R/W	<p>Enables the <code>ref_clk</code> Monitoring block</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> enabled</li> <li><b>0x0:</b> disabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[10]	NVR	R/W	<p>NVR enable flag. Enables the access to the FLASH NVR array</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> enabled (operation is performed in the NVR array)</li> <li><b>0x0:</b> disabled (operation is performed in the Main array)</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

**Table 5-16 Fields For Register: SYSCR0 (continued)**

Bits	Name	R/W	Description
[9]	TRACESEL	R/W	<p>Selects the core to work with the trace debugging module</p> <p> Trace source select (from which core - Core 0 or Core 1 collect trace) is done via the multiplexor that is controlled via the logical OR of the <code>TRACE_SEL</code>.ALTTRACESEL bit and <code>TRACESEL</code> bit</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Core 1</li> <li><b>0x0:</b> Core 0</li> </ul>
[8]	XIP1_EN	R/W	<p>Enables the QSPI_1 XIP mode.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> enabled</li> <li><b>0x0:</b> disabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[7]	XIP0_EN	R/W	<p>Enables the QSPI_0 XIP mode.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> enabled</li> <li><b>0x0:</b> disabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[6]	CORE2CXRSTN	R/W	<p>Resets the logic inside the Core 2, except for the processor core and TCM arbiter. Active low level.</p> <p><b>Value After Reset:</b> 0x0</p>
[5]	CORE2FPRSTN	R/W	<p>Resets the Core 2 TCM arbiter. Active low level.</p> <p><b>Value After Reset:</b> 0x0</p>
[4]	CORE2RSTN	R/W	<p>Resets the Core 2 processor core. Active low level.</p> <p><b>Value After Reset:</b> 0x0</p>
[3]	CORE1CXRSTN	R/W	<p>Resets the logic inside the Core 1, except for the processor core and TCM arbiter. Active low level.</p> <p><b>Value After Reset:</b> 0x0</p>
[2]	CORE1FPRSTN	R/W	<p>Resets the Core 1 TCM arbiter. Active low level.</p> <p><b>Value After Reset:</b> 0x0</p>
[1]	CORE1RSTN	R/W	<p>Resets the Core 1 processor core. Active low level.</p> <p><b>Value After Reset:</b> 0x0</p>
[0]	CORE1CLKEN	R/W	<p>Enables the Core 1 processor core clocking.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> enabled</li> <li><b>0x0:</b> disabled</li> </ul> <p><b>Value After Reset:</b> 0x1</p>

### 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 <b>Value After Reset:</b> 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 <b>Value After Reset:</b> 0x0

### 5.2.2.10. Main Bus Matrix Priority 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 <b>Values:</b> 0x0: lowest priority ... 0x7: highest priority <b>Value After Reset:</b> 0x0
[23]			Reserved
[22:20]	DMA1_PRIOR	R/W	Priority of the DMA_1 <b>Values:</b> 0x0: lowest priority ... 0x7: highest priority <b>Value After Reset:</b> 0x0
[19]			Reserved
[18:16]	DMA0_PRIOR	R/W	Priority of the DMA_0 <b>Values:</b> 0x0: lowest priority ... 0x7: highest priority

**Table 5-19 Fields For Register: PRIOR0 (continued)**

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

### 5.2.2.11. Main Bus Matrix Priority 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 <b>Values:</b>

**Table 5-20 Fields For Register: PRIOR1 (continued)**

Bits	Name	R/W	Description
			<p><b>0x0:</b> lowest priority</p> <p>...</p> <p><b>0x7:</b> highest priority</p> <p><b>Value After Reset:</b> 0x0</p>
[23]			Reserved
[22:20]	SRAM_PRIOR	R/W	<p>Priority of the SRAM</p> <p><b>Values:</b></p> <p><b>0x0:</b> lowest priority</p> <p>...</p> <p><b>0x7:</b> highest priority</p> <p><b>Value After Reset:</b> 0x0</p>
[19]			Reserved
[18:16]	CORE0_FP_PRIOR	R/W	<p>Priority of the Core 0 <i>Front Port (FP)</i></p> <p><b>Values:</b></p> <p><b>0x0:</b> lowest priority</p> <p>...</p> <p><b>0x7:</b> highest priority</p> <p><b>Value After Reset:</b> 0x0</p>
[15]			Reserved
[14:12]	BUS_PRIOR	R/W	<p>Priority of the Bus matrix</p> <p><b>Values:</b></p> <p><b>0x0:</b> lowest priority</p> <p>...</p> <p><b>0x7:</b> highest priority</p> <p><b>Value After Reset:</b> 0x0</p>
[11]			Reserved
[10:8]	PERIPH2_PRIOR	R/W	<p>Priority of the Periphery 2</p> <p><b>Values:</b></p> <p><b>0x0:</b> lowest priority</p> <p>...</p> <p><b>0x7:</b> highest priority</p> <p><b>Value After Reset:</b> 0x0</p>
[7]			Reserved
[6:4]	PERIPH1_PRIOR	R/W	<p>Priority of the Periphery 1</p> <p><b>Values:</b></p> <p><b>0x0:</b> lowest priority</p> <p>...</p> <p><b>0x7:</b> highest priority</p> <p><b>Value After Reset:</b> 0x0</p>

**Table 5-20 Fields For Register: PRIOR1 (continued)**

Bits	Name	R/W	Description
[3]			Reserved
[2:0]	PERIPH0_PRIOR	R/W	<p>Priority of the Periphery 0</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x0: lowest priority</li> <li>...</li> <li>0x7: highest priority</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p>Enables Pull-Up resistors for Port B I/O pins.</p> <p>Each bit of the <code>PB_PU_EN</code> bit field controls the Pull-Up resistor of the appropriate Port B I/O pin:</p> <ul style="list-style-type: none"> <li>• <code>PB_PU_EN [31]</code>: Controls Pull-Up resistor of the PB[15] I/O pin</li> <li>• <code>PB_PU_EN [30]</code>: Controls Pull-Up resistor of the PB[14] I/O pin</li> <li>• ...</li> <li>• <code>PB_PU_EN [16]</code>: Controls Pull-Up resistor of the PB[0] I/O pin</li> </ul> <p><b>Per bit values:</b></p> <ul style="list-style-type: none"> <li>0x0: Disable Pull-Up resistor</li> <li>0x1: Enable Pull-Up resistor</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[15:0]	PA_PU_EN	R/W	<p>Enables Pull-Up resistors for Port A I/O pins.</p> <p>Each bit of the <code>PA_PU_EN</code> bit field controls the Pull-Up resistor of the appropriate Port A I/O pin:</p> <ul style="list-style-type: none"> <li>• <code>PA_PU_EN [15]</code>: Controls Pull-Up resistor of the PA[15] I/O pin</li> <li>• <code>PA_PU_EN [14]</code>: Controls Pull-Up resistor of the PA[14] I/O pin</li> <li>• ...</li> <li>• <code>PA_PU_EN [0]</code>: Controls Pull-Up resistor of the PA[0] I/O pin</li> </ul> <p><b>Per bit values:</b></p> <ul style="list-style-type: none"> <li>0x0: Disable Pull-Up resistor</li> <li>0x1: Enable Pull-Up resistor</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

### 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]	<b>PC_PU_EN</b>	R/W	<p>Enables Pull-Up resistors for Port C I/O pins.            Each bit of the <b>PC_PU_EN</b> bit field controls the Pull-Up resistor of the appropriate Port C I/O pin:</p> <ul style="list-style-type: none"> <li>• <b>PC_PU_EN [15]:</b> Controls Pull-Up resistor of the PC[15] I/O pin</li> <li>• <b>PC_PU_EN [14]:</b> Controls Pull-Up resistor of the PC[14] I/O pin</li> <li>• ...</li> <li>• <b>PC_PU_EN [0]:</b> Controls Pull-Up resistor of the PC[0] I/O pin</li> </ul> <p><b>Per bit values:</b></p> <p><b>0x0:</b> Disable Pull-Up resistor  <b>0x1:</b> Enable Pull-Up resistor</p> <p><b>Value After Reset:</b> 0x0</p>

### 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]	<b>PB_PD_EN</b>	R/W	<p>Enables Pull-Down resistors for Port B I/O pins.            Each bit of the <b>PB_PD_EN</b> bit field controls the Pull-Down resistor of the appropriate Port B I/O pin:</p> <ul style="list-style-type: none"> <li>• <b>PB_PD_EN [31]:</b> Controls Pull-Down resistor of the PB[15] I/O pin</li> <li>• <b>PB_PD_EN [30]:</b> Controls Pull-Down resistor of the PB[14] I/O pin</li> <li>• ...</li> <li>• <b>PB_PD_EN [16]:</b> Controls Pull-Down resistor of the PB[0] I/O pin</li> </ul> <p><b>Per bit values:</b></p> <p><b>0x0:</b> Disable Pull-Down resistor  <b>0x1:</b> Enable Pull-Down resistor</p> <p><b>Value After Reset:</b> 0xFFFF</p>
[15:0]	<b>PA_PD_EN</b>	R/W	<p>Enables Pull-Down resistors for Port A I/O pins.            Each bit of the <b>PA_PD_EN</b> bit field controls the Pull-Down resistor of the appropriate Port A I/O pin:</p>

**Table 5-23 Fields For Register: IOPDCR0 (continued)**

Bits	Name	R/W	Description
			<ul style="list-style-type: none"> <li>PA_PD_EN [15]: Controls Pull-Down resistor of the PA[15] I/O pin</li> <li>PA_PD_EN [14]: Controls Pull-Down resistor of the PA[14] I/O pin</li> <li>...</li> <li>PA_PD_EN [0]: Controls Pull-Down resistor of the PA[0] I/O pin</li> </ul> <p><b>Per bit values:</b></p> <p>0x0: Disable Pull-Down resistor 0x1: Enable Pull-Down resistor</p> <p><b>Value After Reset:</b> 0xFFFF</p>

### 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	<p>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:</p> <ul style="list-style-type: none"> <li>PC_PD_EN [15]: Controls Pull-Down resistor of the PC[15] I/O pin</li> <li>PC_PD_EN [14]: Controls Pull-Down resistor of the PC[14] I/O pin</li> <li>...</li> <li>PC_PD_EN [0]: Controls Pull-Down resistor of the PC[0] I/O pin</li> </ul> <p><b>Per bit values:</b></p> <p>0x0: Disable Pull-Down resistor 0x1: Enable Pull-Down resistor</p> <p><b>Value After Reset:</b> 0xFFFF</p>

### 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	<p>PA[7] I/O pin input enable</p> <p><b>Values:</b></p>

**Table 5-25 Fields For Register: IOAFCR0 (continued)**

Bits	Name	R/W	Description
			<p><b>0x0:</b> Disable  <b>0x1:</b> Enable</p> <p><b>Value After Reset:</b> 0x1</p>
[30:28]	PA7_AF	R/W	<p>Selects the PA[7] I/O pin alternate function.  See <b>I/O pins functions and control for Port A</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting.</p> <p><b>Value After Reset:</b> 0x0</p>
[27]	PA6_IE	R/W	<p>PA[6] I/O pin input enable</p> <p><b>Values:</b></p> <p><b>0x0:</b> Disable  <b>0x1:</b> Enable</p> <p><b>Value After Reset:</b> 0x1</p>
[26:24]	PA6_AF	R/W	<p>Selects the PA[6] I/O pin alternate function.  See <b>I/O pins functions and control for Port A</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting.</p> <p><b>Value After Reset:</b> 0x0</p>
[23]	PA5_IE	R/W	<p>PA[5] I/O pin input enable</p> <p><b>Values:</b></p> <p><b>0x0:</b> Disable  <b>0x1:</b> Enable</p> <p><b>Value After Reset:</b> 0x1</p>
[22:20]	PA5_AF	R/W	<p>Selects the PA[5] I/O pin alternate function.  See <b>I/O pins functions and control for Port A</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting.</p> <p><b>Value After Reset:</b> 0x0</p>
[19]	PA4_IE	R/W	<p>PA[4] I/O pin input enable</p> <p><b>Values:</b></p> <p><b>0x0:</b> Disable  <b>0x1:</b> Enable</p> <p><b>Value After Reset:</b> 0x1</p>
[18:16]	PA4_AF	R/W	<p>Selects the PA[4] I/O pin alternate function.  See <b>I/O pins functions and control for Port A</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting.</p> <p><b>Value After Reset:</b> 0x0</p>
[15]	PA3_IE	R/W	<p>PA[3] I/O pin input enable</p> <p><b>Values:</b></p> <p><b>0x0:</b> Disable  <b>0x1:</b> Enable</p> <p><b>Value After Reset:</b> 0x1</p>
[14:12]	PA3_AF	R/W	<p>Selects the PA[3] I/O pin alternate function.  See <b>I/O pins functions and control for Port A</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting.</p>

**Table 5-25 Fields For Register: IOAFCR0 (continued)**

Bits	Name	R/W	Description
			<b>Value After Reset:</b> 0x0
[11]	PA2_IE	R/W	PA[2] I/O pin input enable <b>Values:</b> 0x0: Disable 0x1: Enable <b>Value After Reset:</b> 0x1
[10:8]	PA2_AF	R/W	Selects the PA[2] I/O pin alternate function. See <b>I/O pins functions and control for Port A</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting. <b>Value After Reset:</b> 0x0
[7]	PA1_IE	R/W	PA[1] I/O pin input enable <b>Values:</b> 0x0: Disable 0x1: Enable <b>Value After Reset:</b> 0x1
[6:4]	PA1_AF	R/W	Selects the PA[1] I/O pin alternate function. See <b>I/O pins functions and control for Port A</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting. <b>Value After Reset:</b> 0x0
[3]	PA0_IE	R/W	PA[0] I/O pin input enable <b>Values:</b> 0x0: Disable 0x1: Enable <b>Value After Reset:</b> 0x1
[2:0]	PA0_AF	R/W	Selects the PA[0] I/O pin alternate function. See <b>I/O pins functions and control for Port A</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting. <b>Value After Reset:</b> 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	PA[15] I/O pin input enable <b>Values:</b> 0x0: Disable 0x1: Enable <b>Value After Reset:</b> 0x1

**Table 5-26 Fields For Register: IOAFCR1 (continued)**

Bits	Name	R/W	Description
[30:28]	PA15_AF	R/W	Selects the PA[15] I/O pin alternate function. See <b>I/O pins functions and control for Port A</b> figure in <b>Alternate Functions</b> section for details on bits setting. <b>Value After Reset:</b> 0x0
[27]	PA14_IE	R/W	PA[14] I/O pin input enable <b>Values:</b> 0x0: Disable 0x1: Enable <b>Value After Reset:</b> 0x1
[26:24]	PA14_AF	R/W	Selects the PA[14] I/O pin alternate function. See <b>I/O pins functions and control for Port A</b> figure in <b>Alternate Functions</b> section for details on bits setting. <b>Value After Reset:</b> 0x0
[23]	PA13_IE	R/W	PA[13] I/O pin input enable <b>Values:</b> 0x0: Disable 0x1: Enable <b>Value After Reset:</b> 0x1
[22:20]	PA13_AF	R/W	Selects the PA[13] I/O pin alternate function. See <b>I/O pins functions and control for Port A</b> figure in <b>Alternate Functions</b> section for details on bits setting. <b>Value After Reset:</b> 0x0
[19]	PA12_IE	R/W	PA[12] I/O pin input enable <b>Values:</b> 0x0: Disable 0x1: Enable <b>Value After Reset:</b> 0x1
[18:16]	PA12_AF	R/W	Selects the PA[12] I/O pin alternate function. See <b>I/O pins functions and control for Port A</b> figure in <b>Alternate Functions</b> section for details on bits setting. <b>Value After Reset:</b> 0x0
[15]	PA11_IE	R/W	PA[11] I/O pin input enable <b>Values:</b> 0x0: Disable 0x1: Enable <b>Value After Reset:</b> 0x1
[14:12]	PA11_AF	R/W	Selects the PA[11] I/O pin alternate function. See <b>I/O pins functions and control for Port A</b> figure in <b>Alternate Functions</b> section for details on bits setting. <b>Value After Reset:</b> 0x0
[11]	PA10_IE	R/W	PA[10] I/O pin input enable <b>Values:</b>

**Table 5-26 Fields For Register: IOAFCR1 (continued)**

Bits	Name	R/W	Description
			<b>0x0:</b> Disable <b>0x1:</b> Enable <b>Value After Reset:</b> 0x1
[10:8]	PA10_AF	R/W	Selects the PA[10] I/O pin alternate function. See <b>I/O pins functions and control for Port A</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting. <b>Value After Reset:</b> 0x0
[7]	PA9_IE	R/W	PA[9] I/O pin input enable <b>Values:</b> <b>0x0:</b> Disable <b>0x1:</b> Enable <b>Value After Reset:</b> 0x1
[6:4]	PA9_AF	R/W	Selects the PA[9] I/O pin alternate function. See <b>I/O pins functions and control for Port A</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting. <b>Value After Reset:</b> 0x0
[3]	PA8_IE	R/W	PA[8] I/O pin input enable <b>Values:</b> <b>0x0:</b> Disable <b>0x1:</b> Enable <b>Value After Reset:</b> 0x1
[2:0]	PA8_AF	R/W	Selects the PA[8] I/O pin alternate function. See <b>I/O pins functions and control for Port A</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting. <b>Value After Reset:</b> 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	PB[7] I/O pin input enable <b>Values:</b> <b>0x0:</b> Disable <b>0x1:</b> Enable <b>Value After Reset:</b> 0x1
[30:28]	PB7_AF	R/W	Selects the PB[7] I/O pin alternate function. See <b>I/O pins functions and control for Port B</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting. <b>Value After Reset:</b> 0x0

**Table 5-27 Fields For Register: IOAFCR2 (continued)**

Bits	Name	R/W	Description
[27]	PB6_IE	R/W	PB[6] I/O pin input enable <b>Values:</b> <b>0x0:</b> Disable <b>0x1:</b> Enable <b>Value After Reset:</b> 0x1
[26:24]	PB6_AF	R/W	Selects the PB[6] I/O pin alternate function. See <b>I/O pins functions and control for Port B</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting. <b>Value After Reset:</b> 0x0
[23]	PB5_IE	R/W	PB[5] I/O pin input enable <b>Values:</b> <b>0x0:</b> Disable <b>0x1:</b> Enable <b>Value After Reset:</b> 0x1
[22:20]	PB5_AF	R/W	Selects the PB[5] I/O pin alternate function. See <b>I/O pins functions and control for Port B</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting. <b>Value After Reset:</b> 0x0
[19]	PB4_IE	R/W	PB[4] I/O pin input enable <b>Values:</b> <b>0x0:</b> Disable <b>0x1:</b> Enable <b>Value After Reset:</b> 0x1
[18:16]	PB4_AF	R/W	Selects the PB[4] I/O pin alternate function. See <b>I/O pins functions and control for Port B</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting. <b>Value After Reset:</b> 0x0
[15]	PB3_IE	R/W	PB[3] I/O pin input enable <b>Values:</b> <b>0x0:</b> Disable <b>0x1:</b> Enable <b>Value After Reset:</b> 0x1
[14:12]	PB3_AF	R/W	Selects the PB[3] I/O pin alternate function. See <b>I/O pins functions and control for Port B</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting. <b>Value After Reset:</b> 0x0
[11]	PB2_IE	R/W	PB[2] I/O pin input enable <b>Values:</b> <b>0x0:</b> Disable <b>0x1:</b> Enable <b>Value After Reset:</b> 0x1

**Table 5-27 Fields For Register: IOAFCR2 (continued)**

Bits	Name	R/W	Description
[10:8]	PB2_AF	R/W	Selects the PB[2] I/O pin alternate function. See <b>I/O pins functions and control for Port B</b> figure in <b>Alternate Functions</b> section for details on bits setting. <b>Value After Reset:</b> 0x0
[7]	PB1_IE	R/W	PB[1] I/O pin input enable <b>Values:</b> 0x0: Disable 0x1: Enable <b>Value After Reset:</b> 0x1
[6:4]	PB1_AF	R/W	Selects the PB[1] I/O pin alternate function. See <b>I/O pins functions and control for Port B</b> figure in <b>Alternate Functions</b> section for details on bits setting. <b>Value After Reset:</b> 0x0
[3]	PB0_IE	R/W	PB[0] I/O pin input enable <b>Values:</b> 0x0: Disable 0x1: Enable <b>Value After Reset:</b> 0x1
[2:0]	PB0_AF	R/W	Selects the PB[0] I/O pin alternate function. See <b>I/O pins functions and control for Port B</b> figure in <b>Alternate Functions</b> section for details on bits setting. <b>Value After Reset:</b> 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	PB[15] I/O pin input enable <b>Values:</b> 0x0: Disable 0x1: Enable <b>Value After Reset:</b> 0x1
[30:28]	PB15_AF	R/W	Selects the PB[15] I/O pin alternate function. See <b>I/O pins functions and control for Port B</b> figure in <b>Alternate Functions</b> section for details on bits setting. <b>Value After Reset:</b> 0x0
[27]	PB14_IE	R/W	PB[14] I/O pin input enable <b>Values:</b>

**Table 5-28 Fields For Register: IOAFCR3 (continued)**

Bits	Name	R/W	Description
			<p><b>0x0:</b> Disable  <b>0x1:</b> Enable</p> <p><b>Value After Reset:</b> 0x1</p>
[26:24]	PB14_AF	R/W	<p>Selects the PB[14] I/O pin alternate function.            See <b>I/O pins functions and control for Port B</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting.</p> <p><b>Value After Reset:</b> 0x0</p>
[23]	PB13_IE	R/W	<p>PB[13] I/O pin input enable</p> <p><b>Values:</b></p> <p><b>0x0:</b> Disable  <b>0x1:</b> Enable</p> <p><b>Value After Reset:</b> 0x1</p>
[22:20]	PB13_AF	R/W	<p>Selects the PB[13] I/O pin alternate function.            See <b>I/O pins functions and control for Port B</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting.</p> <p><b>Value After Reset:</b> 0x0</p>
[19]	PB12_IE	R/W	<p>PB[12] I/O pin input enable</p> <p><b>Values:</b></p> <p><b>0x0:</b> Disable  <b>0x1:</b> Enable</p> <p><b>Value After Reset:</b> 0x1</p>
[18:16]	PB12_AF	R/W	<p>Selects the PB[12] I/O pin alternate function.            See <b>I/O pins functions and control for Port B</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting.</p> <p><b>Value After Reset:</b> 0x0</p>
[15]	PB11_IE	R/W	<p>PB[11] I/O pin input enable</p> <p><b>Values:</b></p> <p><b>0x0:</b> Disable  <b>0x1:</b> Enable</p> <p><b>Value After Reset:</b> 0x1</p>
[14:12]	PB11_AF	R/W	<p>Selects the PB[11] I/O pin alternate function.            See <b>I/O pins functions and control for Port B</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting.</p> <p><b>Value After Reset:</b> 0x0</p>
[11]	PB10_IE	R/W	<p>PB[10] I/O pin input enable</p> <p><b>Values:</b></p> <p><b>0x0:</b> Disable  <b>0x1:</b> Enable</p> <p><b>Value After Reset:</b> 0x1</p>
[10:8]	PB10_AF	R/W	<p>Selects the PB[10] I/O pin alternate function.            See <b>I/O pins functions and control for Port B</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting.</p>

**Table 5-28 Fields For Register: IOAFCR3 (continued)**

Bits	Name	R/W	Description
			<b>Value After Reset:</b> 0x0
[7]	PB9_IE	R/W	PB[9] I/O pin input enable <b>Values:</b> 0x0: Disable 0x1: Enable <b>Value After Reset:</b> 0x1
[6:4]	PB9_AF	R/W	Selects the PB[9] I/O pin alternate function. See <b>I/O pins functions and control for Port B</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting. <b>Value After Reset:</b> 0x0
[3]	PB8_IE	R/W	PB[8] I/O pin input enable <b>Values:</b> 0x0: Disable 0x1: Enable <b>Value After Reset:</b> 0x1
[2:0]	PB8_AF	R/W	Selects the PB[8] I/O pin alternate function. See <b>I/O pins functions and control for Port B</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting. <b>Value After Reset:</b> 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	PC[7] I/O pin input enable <b>Values:</b> 0x0: Disable 0x1: Enable <b>Value After Reset:</b> 0x1
[30:28]	PC7_AF	R/W	Selects the PC[7] I/O pin alternate function. See <b>I/O pins functions and control for Port C</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting. <b>Value After Reset:</b> 0x0
[27]	PC6_IE	R/W	PC[6] I/O pin input enable <b>Values:</b> 0x0: Disable 0x1: Enable <b>Value After Reset:</b> 0x1

**Table 5-29 Fields For Register: IOAFCR4 (continued)**

Bits	Name	R/W	Description
[26:24]	PC6_AF	R/W	Selects the PC[6] I/O pin alternate function. See <b>I/O pins functions and control for Port C</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting. <b>Value After Reset:</b> 0x0
[23]	PC5_IE	R/W	PC[5] I/O pin input enable <b>Values:</b> 0x0: Disable 0x1: Enable <b>Value After Reset:</b> 0x1
[22:20]	PC5_AF	R/W	Selects the PC[5] I/O pin alternate function. See <b>I/O pins functions and control for Port C</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting. <b>Value After Reset:</b> 0x0
[19]	PC4_IE	R/W	PC[4] I/O pin input enable <b>Values:</b> 0x0: Disable 0x1: Enable <b>Value After Reset:</b> 0x1
[18:16]	PC4_AF	R/W	Selects the PC[4] I/O pin alternate function. See <b>I/O pins functions and control for Port C</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting. <b>Value After Reset:</b> 0x0
[15]	PC3_IE	R/W	PC[3] I/O pin input enable <b>Values:</b> 0x0: Disable 0x1: Enable <b>Value After Reset:</b> 0x1
[14:12]	PC3_AF	R/W	Selects the PC[3] I/O pin alternate function. See <b>I/O pins functions and control for Port C</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting. <b>Value After Reset:</b> 0x0
[11]	PC2_IE	R/W	PC[2] I/O pin input enable <b>Values:</b> 0x0: Disable 0x1: Enable <b>Value After Reset:</b> 0x1
[10:8]	PC2_AF	R/W	Selects the PC[2] I/O pin alternate function. See <b>I/O pins functions and control for Port C</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting. <b>Value After Reset:</b> 0x0
[7]	PC1_IE	R/W	PC[1] I/O pin input enable <b>Values:</b>

**Table 5-29 Fields For Register: IOAFCR4 (continued)**

Bits	Name	R/W	Description
			<b>0x0:</b> Disable <b>0x1:</b> Enable <b>Value After Reset:</b> 0x1
[6:4]	PC1_AF	R/W	Selects the PC[1] I/O pin alternate function. See <b>I/O pins functions and control for Port C</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting. <b>Value After Reset:</b> 0x0
[3]	PC0_IE	R/W	PC[0] I/O pin input enable <b>Values:</b> <b>0x0:</b> Disable <b>0x1:</b> Enable <b>Value After Reset:</b> 0x1
[2:0]	PC0_AF	R/W	Selects the PC[0] I/O pin alternate function. See <b>I/O pins functions and control for Port C</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting. <b>Value After Reset:</b> 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	PC[15] I/O pin input enable <b>Values:</b> <b>0x0:</b> Disable <b>0x1:</b> Enable <b>Value After Reset:</b> 0x1
[30:28]	PC15_AF	R/W	Selects the PC[15] I/O pin alternate function. See <b>I/O pins functions and control for Port C</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting. <b>Value After Reset:</b> 0x0
[27]	PC14_IE	R/W	PC[14] I/O pin input enable <b>Values:</b> <b>0x0:</b> Disable <b>0x1:</b> Enable <b>Value After Reset:</b> 0x1
[26:24]	PC14_AF	R/W	Selects the PC[14] I/O pin alternate function. See <b>I/O pins functions and control for Port C</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting. <b>Value After Reset:</b> 0x0

**Table 5-30 Fields For Register: IOAFCR5 (continued)**

Bits	Name	R/W	Description
[23]	PC13_IE	R/W	<p>PC[13] I/O pin input enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Disable</li> <li><b>0x1:</b> Enable</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[22:20]	PC13_AF	R/W	<p>Selects the PC[13] I/O pin alternate function.</p> <p>See <b>I/O pins functions and control for Port C</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting.</p> <p><b>Value After Reset:</b> 0x0</p>
[19]	PC12_IE	R/W	<p>PC[12] I/O pin input enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Disable</li> <li><b>0x1:</b> Enable</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[18:16]	PC12_AF	R/W	<p>Selects the PC[12] I/O pin alternate function.</p> <p>See <b>I/O pins functions and control for Port C</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting.</p> <p><b>Value After Reset:</b> 0x0</p>
[15]	PC11_IE	R/W	<p>PC[11] I/O pin input enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Disable</li> <li><b>0x1:</b> Enable</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[14:12]	PC11_AF	R/W	<p>Selects the PC[11] I/O pin alternate function.</p> <p>See <b>I/O pins functions and control for Port C</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting.</p> <p><b>Value After Reset:</b> 0x0</p>
[11]	PC10_IE	R/W	<p>PC[10] I/O pin input enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Disable</li> <li><b>0x1:</b> Enable</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[10:8]	PC10_AF	R/W	<p>Selects the PC[10] I/O pin alternate function.</p> <p>See <b>I/O pins functions and control for Port C</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting.</p> <p><b>Value After Reset:</b> 0x0</p>
[7]	PC9_IE	R/W	<p>PC[9] I/O pin input enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Disable</li> <li><b>0x1:</b> Enable</li> </ul> <p><b>Value After Reset:</b> 0x1</p>

**Table 5-30 Fields For Register: `IOAFCR5` (continued)**

Bits	Name	R/W	Description
[6:4]	PC9_AF	R/W	Selects the PC[9] I/O pin alternate function. See <b>I/O pins functions and control for Port C</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting. <b>Value After Reset:</b> 0x0
[3]	PC8_IE	R/W	PC[8] I/O pin input enable <b>Values:</b> 0x0: Disable 0x1: Enable <b>Value After Reset:</b> 0x1
[2:0]	PC8_AF	R/W	Selects the PC[8] I/O pin alternate function. See <b>I/O pins functions and control for Port C</b> figure in <a href="#">Alternate Functions</a> section for details on bits setting. <b>Value After Reset:</b> 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	Controls the load function of the PA[15] output driver
[29:28]	PADS14	R/W	Controls the load function of the PA[14] output driver
[27:26]	PADS13	R/W	Controls the load function of the PA[13] output driver
[25:24]	PADS12	R/W	Controls the load function of the PA[12] output driver
[23:22]	PADS11	R/W	Controls the load function of the PA[11] output driver
[21:20]	PADS10	R/W	Controls the load function of the PA[10] output driver
[19:18]	PADS9	R/W	Controls the load function of the PA[9] output driver
[17:16]	PADS8	R/W	Controls the load function of the PA[8] output driver
[15:14]	PADS7	R/W	Controls the load function of the PA[7] output driver
[13:12]	PADS6	R/W	Controls the load function of the PA[6] output driver
[11:10]	PADS5	R/W	Controls the load function of the PA[5] output driver
[9:8]	PADS4	R/W	Controls the load function of the PA[4] output driver
[7:6]	PADS3	R/W	Controls the load function of the PA[3] output driver
[5:4]	PADS2	R/W	Controls the load function of the PA[2] output driver

**Table 5-31 Fields For Register: IODSCR0 (continued)**

Bits	Name	R/W	Description
[3:2]	PADS1	R/W	Controls the load function of the PA[1] output driver
[1:0]	PADS0	R/W	Controls the load function of the PA[0] output driver  11:3 (maximum) (18mA) 10: 2 (13,5mA) 01: 1 (9mA) 00: 0 (4,5mA) <b>Value After Reset:</b> 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	Controls the load function of the PB[15] output driver
[29:28]	PBDS14	R/W	Controls the load function of the PB[14] output driver
[27:26]	PBDS13	R/W	Controls the load function of the PB[13] output driver
[25:24]	PBDS12	R/W	Controls the load function of the PB[12] output driver
[23:22]	PBDS11	R/W	Controls the load function of the PB[11] output driver
[21:20]	PBDS10	R/W	Controls the load function of the PB[10] output driver
[19:18]	PBDS9	R/W	Controls the load function of the PB[9] output driver
[17:16]	PBDS8	R/W	Controls the load function of the PB[8] output driver
[15:14]	PBDS7	R/W	Controls the load function of the PB[7] output driver
[13:12]	PBDS6	R/W	Controls the load function of the PB[6] output driver
[11:10]	PBDS5	R/W	Controls the load function of the PB[5] output driver
[9:8]	PBDS4	R/W	Controls the load function of the PB[4] output driver
[7:6]	PBDS3	R/W	Controls the load function of the PB[3] output driver
[5:4]	PBDS2	R/W	Controls the load function of the PB[2] output driver
[3:2]	PBDS1	R/W	Controls the load function of the PB[1] output driver
[1:0]	PBDS0	R/W	Controls the load function of the PB[0] output driver  11:3 (maximum) (18mA) 10: 2 (13,5mA)

**Table 5-32 Fields For Register: IODSCR1 (continued)**

Bits	Name	R/W	Description
			<b>01:</b> 1 (9mA) <b>00:</b> 0 (4,5mA) <b>Value After Reset:</b> 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	Controls the load function of the PC[15] output driver
[29:28]	PCDS14	R/W	Controls the load function of the PC[14] output driver
[27:26]	PCDS13	R/W	Controls the load function of the PC[13] output driver
[25:24]	PCDS12	R/W	Controls the load function of the PC[12] output driver
[23:22]	PCDS11	R/W	Controls the load function of the PC[11] output driver
[21:20]	PCDS10	R/W	Controls the load function of the PC[10] output driver
[19:18]	PCDS9	R/W	Controls the load function of the PC[9] output driver
[17:16]	PCDS8	R/W	Controls the load function of the PC[8] output driver
[15:14]	PCDS7	R/W	Controls the load function of the PC[7] output driver
[13:12]	PCDS6	R/W	Controls the load function of the PC[6] output driver
[11:10]	PCDS5	R/W	Controls the load function of the PC[5] output driver
[9:8]	PCDS4	R/W	Controls the load function of the PC[4] output driver
[7:6]	PCDS3	R/W	Controls the load function of the PC[3] output driver
[5:4]	PCDS2	R/W	Controls the load function of the PC[2] output driver
[3:2]	PCDS1	R/W	Controls the load function of the PC[1] output driver
[1:0]	PCDS0	R/W	Controls the load function of the PC[0] output driver <b>11:3</b> (maximum) (18mA) <b>10:</b> 2 (13,5mA) <b>01:</b> 1 (9mA) <b>00:</b> 0 (4,5mA) <b>Value After Reset:</b> 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	<p>Controls the LDO FLASH mode</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x1: Power Down mode</li> <li>0x0: Normal mode</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[13:11]	LDOF_LDO_PI	R/W	<p>Controls the LDO FLASH protective current</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x0: 290mA</li> <li>0x1: 260mA</li> <li>0x2: 225mA</li> <li>0x3: 195mA</li> <li>0x4: 410mA</li> <li>0x5: 380mA</li> <li>0x6: 350mA</li> <li>0x7: 320mA</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[10:8]	LDOF_ADJ	R/W	<p>Adjusts the LDO FLASH output voltage</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x0: 1.70V</li> <li>0x1: 1.75V</li> <li>0x2: 1.60V</li> <li>0x3: 1.65V</li> <li>0x4: 1.90V</li> <li>0x5: 1.95V</li> <li>0x6: 1.85V</li> <li>0x7: 1.80V</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[7:6]			Reserved
[5:3]	LDOC_LDO_PI	R/W	<p>Controls the LDO CORE protective current</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x0: 500mA</li> <li>0x1: 460mA</li> <li>0x2: 420mA</li> <li>0x3: 380mA</li> <li>0x4: 660mA</li> </ul>

**Table 5-34 Fields For Register: LDOCR (continued)**

Bits	Name	R/W	Description
			<p><b>0x5:</b> 620mA  <b>0x6:</b> 580mA</p> <p><b>Value After Reset:</b> 0x0</p>
[2:0]	LDOC_ADJ	R/W	<p>Adjusts the LDO CORE output voltage</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> 1.10V</li> <li><b>0x1:</b> 1.15V</li> <li><b>0x2:</b> 1.00V</li> <li><b>0x3:</b> 1.05V</li> <li><b>0x4:</b> 1.30V</li> <li><b>0x5:</b> 1.35V</li> <li><b>0x6:</b> 1.20V</li> <li><b>0x7:</b> 1.25V</li> </ul> <p><b>Value After Reset:</b> 0x6</p>

### 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]			<p>Reserved</p> <p><b>!</b> This field should not be changed</p> <p><b>Value After Reset:</b> 0x1</p>
[27]	DRVVBUSGPR	R/W	<p>Alternative control of the I/O cell driver control signal when the alternative DRVVBUS function is selected</p> <p><b>Value After Reset:</b> 0x0</p>
[26]	DRVVBUSSRC	R/W	<p>Selects the source of the I/O cell driver control signal when the alternative DRVVBUS function is selected</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> DRVVBUS output</li> <li><b>0x1:</b> DRVVBUSGPR</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[25]	DRVVBUSPOL	R/W	<p>Selects the polarity of the I/O cell driver control signal when the alternative DRVVBUS function is selected</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> corresponds to the DRVVBUS output</li> <li><b>0x1:</b> inversion of the DRVVBUS output</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[24]	DRVVBUSVAL	R/W	I/O cell value when the alternative DRVVBUS function is selected

**Table 5-35 Fields For Register: USBCR (continued)**

Bits	Name	R/W	Description
			<b>Values:</b> <b>0x0:</b> logical 0 <b>0x1:</b> logical 1 <b>Value After Reset:</b> 0x0
[23:17]			Reserved
[16]	BIST_STAT	R	Result of the USB PHY BIST operation <b>Values:</b> <b>0x0:</b> BIST completed with an error <b>0x1:</b> BIST completed without errors <b>Value After Reset:</b> 0x0
[15]	IDPULLUPGPR	R/W	Alternative control of the USB PHY <code>id_pullup</code> signal <b>Value After Reset:</b> 0x0
[14]	IDPULLUPSRC	R/W	USB PHY ID_PULLUP input source: <b>Values:</b> <b>0x0:</b> microcontroller IDPULLUP output <b>0x1:</b> <code>IDPULLUPGPR</code> <b>Value After Reset:</b> 0x0
[13]	IDPULLUPPOL	R/W	USB PHY ID_PULLUP input polarity <b>Values:</b> <b>0x0:</b> corresponds to microcontroller IDPULLUP output <b>0x1:</b> inversion of the microcontroller IDPULLUP output <b>Value After Reset:</b> 0x0
[12:10]			Reserved
[9]	TX_SE0	R/W	Controls the TX_SE0 input of the USB PHY. Enables the SE0 signal: <b>0x0:</b> normal operation <b>0x1:</b> SE0 is set on D+/D-
[8]	TX_DAT	R/W	Controls the TX_DAT input of the USB PHY. <b>Value After Reset:</b> 0x0
[7]	TX_ENABLE_N	R/W	Controls the TX_ENABLE_N input of the USB PHY. <b>Value After Reset:</b> 0x0
[6]	FSLS_SERIALMODE	R/W	Controls the FSLS_SERIALMODE input of the USB PHY. Enables serial packet transmission: <b>0x0:</b> FS packets are transmitted over the parallel interface <b>0x1:</b> FS and LS packets are transmitted over the serial interface <b>Value After Reset:</b> 0x0
[5]	OTG_SUSPENDM	R/W	Controls the OTG_SUSPENDM input of the USB PHY. Enables OTG/HOST functions:

**Table 5-35 Fields For Register: `USBCR` (continued)**

Bits	Name	R/W	Description
			<p><b>0x0:</b> OTG is disabled  <b>0x1:</b> OTG is enabled</p> <p><b>Value After Reset:</b> 0x0</p>
[4]	REFCLK_MODE	R/W	<p>Controls the REFCLK_MODE input of the USB PHY.  Selects the USP PHY reference clock frequency:</p> <p><b>0x0:</b> 25MHz  <b>0x1:</b> 12MHz</p> <p> It is recommended to use the 25MHz reference clock frequency.</p> <p><b>Value After Reset:</b> 0x0</p>
[3]	BIST_EN	R/W	<p>Enables the USB PHY BIST operation:</p> <p><b>0x0:</b> BIST is disabled  <b>0x1:</b> BIST is enabled</p> <p><b>Value After Reset:</b> 0x0</p>
[2]	PLL_EN	R/W	<p>Enables the USB PHY PLL:</p> <p><b>0x0:</b> PLL is disabled  <b>0x1:</b> PLL is enabled</p> <p><b>Value After Reset:</b> 0x0</p>
[1]	RESET	R/W	<p>Controls the RESET input of the USB PHY:</p> <p><b>Values:</b></p> <p><b>0x0:</b> RESET is inactive  <b>0x1:</b> RESET is active</p> <p><b>Value After Reset:</b> 0x1</p>
[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	<p>Event flag that the PLL exited the steady state</p> <p><b>Values:</b></p> <p><b>0x1:</b> event  <b>0x0:</b> no event</p> <p><b>Value After Reset:</b> 0x0</p>

**Table 5-36 Fields For Register: SYSSR0 (continued)**

Bits	Name	R/W	Description
[10]	PLL_FBSLIP	R	FBSLIP PLL output
[9]	PLL_RFSLIP	R	RFSLIP PLL output
[8]	WDTHIT	R	Indicates that <code>wdt0_sys_rst_n</code> is active <b>Values:</b> 0x1: active 0x0: non active <code>WDTHIT</code> is reset to 0x0 by writing 1 to the <code>SYSCR0.WDTHITCLR</code> <b>Value After Reset:</b> 0x0
[7]	POR_OK	R	Indicates POR_OK input status <b>Value After Reset:</b> N/A
[6:5]			Reserved
[4:0]	STRAP	R	The values of triggers that stores the state of the strap pins {PC[8], PC[6], PC[4], PB[10], PA[10]} For more information about strap pins refer to the <b>3.9.3 Strap pins</b> section of the <b>BE-U1000 Datasheet</b> <b>Value After Reset:</b> 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	Code word to enable WDT_0 reset by set 0 to the <code>PCLK0EN.WDT0RSTN</code> and WDT_1 reset by set 0 to the <code>PCLK1EN.WDT1RSTN</code> . WDT_0 is reset by the <code>PCLK0EN.WDT0RSTN</code> and WDT_1 is reset by the <code>PCLK1EN.WDT1RSTN</code> if the <code>WDTCLRKEY</code> is set to 0xD09C1EA7. <b>Value After Reset:</b> 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	Selects the <code>int_local[4]</code> interrupt ( <code>pc_irq</code> ) source of the Core 2 <b>Value After Reset:</b> N/A
[23:20]	PADBINTSEL2	R/W	Selects the <code>int_local[3]</code> interrupt ( <code>pb_irq</code> ) source of the Core 2

**Table 5-38 Fields For Register: INTCR0 (continued)**

Bits	Name	R/W	Description
			<b>Value After Reset:</b> N/A
[19:16]	PADAINTSEL2	R/W	Selects the int_local[2] interrupt ( <code>pa_irq</code> ) source of the Core 2 <b>Value After Reset:</b> N/A
[15:12]	PADCINTSEL0	R/W	Selects the int_local[73] interrupt ( <code>pc_irq</code> ) source of the Core 0, Core 1 <b>Value After Reset:</b> N/A
[11:8]	PADBINTSEL0	R/W	Selects the int_local[72] interrupt ( <code>pb_irq</code> ) source of the Core 0, Core 1 <b>Value After Reset:</b> N/A
[7:4]	PADAINTSEL0	R/W	Selects the int_local[71] interrupt ( <code>pa_irq</code> ) source of the Core 0, Core 1 <b>Value After Reset:</b> 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: <b>Values:</b> 0x0: divider is disabled 0x1: divider is enabled <b>Value After Reset:</b> 0x1
[10:6]	CANCLKDIV	R/W	CANCLKX2_DIV divisor value. <b>Values:</b> 0x0: <code>CANCLKDIV[0] = 0</code> - integer division. The frequency division ratio is equal to <code>CANCLKDIV[4:1]+1</code> . Valid values <code>CANCLKDIV[4:1]</code> from 0 to 15. If <code>CANCLKDIV[4:0] = 0</code> - bypass 0x1: <code>CANCLKDIV[0] = 1</code> - fractional division. The frequency division ratio is equal to <code>CANCLKDIV[4:1]+1.5</code> . Valid values <code>CANCLKDIV[4:1]</code> from 0 to 14. <b>Value After Reset:</b> 0x0
[5]	TCLKDIVEN	R/W	Enables the TCLK_DIV divider: <b>Values:</b> 0x0: divider is disabled 0x1: divider is enabled <b>Value After Reset:</b> 0x1

**Table 5-39 Fields For Register: CLKCR1 (continued)**

Bits	Name	R/W	Description
[4:0]	TCLKDIV	R/W	<p>TCLK_DIV divisor value.  <b>Values:</b></p> <p><b>0x0:</b> TCLKDIV[0] = 0 - integer division. The frequency division ratio is equal to TCLKDIV[4:1]+1. Valid values TCLKDIV[4:1] from 0 to 15. If TCLKDIV[4:0] = 0 - bypass</p> <p><b>0x1:</b> TCLKDIV[0] = 1 - fractional division. The frequency division ratio is equal to TCLKDIV[4:1]+1.5. Valid values TCLKDIV[4:1] from 0 to 14.</p> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p>FLASH NVR 1 lock flag. Disables the modification of the 1st block of the NVR array (sectors 0, 1)  <b>Values:</b></p> <p><b>0x1:</b> modification of the 1st NVR block is disabled  <b>0x0:</b> modification of the 1st NVR block is enabled</p> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p>Enables the Core 1:  <b>Values:</b></p> <p><b>0x1:</b> enabled  <b>0x0:</b> disabled</p> <p><b>Value After Reset:</b> 0x1</p>

## 6. RISC-V Cores

In this chapter:

- [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.

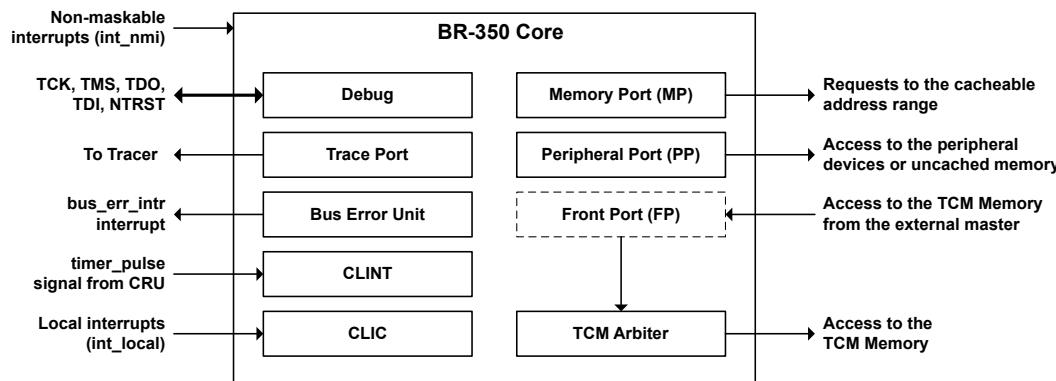


Figure 6-1 BR-350 Core Block Diagram

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

- [CLIC](#) and [CLINT](#) that are used to handle the interrupts
- Memory Port is used when the BR-350 Core makes requests to the cacheable address range
- Peripheral Port is used for access the BR-350 Core to the peripheral devices or uncached memory
- Trace Port is used to provide the information about executed instructions to the external [Tracer](#)
- Front Port is used to access the TCM Memory (through the TCM Arbiter) from the external master device



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

- TCM Arbiter is used for arbitration of the requests to the TCM Memory
- [Bus Error Unit](#) causes an interrupt (`bus_err_intr`) if an error occurs
- [Debug](#) is used for external core debug via 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).

If there is an active NMI, the execution flow switches to the address stored in the `NMITVEC` register, regardless of the other CSRs state. The following actions will be performed:

1. The privilege mode changes to the M-mode;
2. The `MEPC` CSR is written with the virtual address of the instruction that was interrupted;
3. The active NMI number is written to 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 that prohibit of taking active interrupts from the `int_nmi`. `NMIE` can be set to 1 only after reset or after execution the `MRET` command.



Exiting from an NMI handler using the `MRET` instruction returns the flow of execution to the address written to the `MEPC` register. However, NMIs are fatal because the state of the CSRs may be overwritten (for example, if an NMI occurs while an M-mode interrupt handler is executing). 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 eight 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.



Processor core implements four event counters. It means that `MHPMEVENT3 - MHPMEVENT6` and `MHPMCOUNTER3 - MHPMCOUNTER6` CSRs are implemented.

The following table lists the events supported by the core. Writing an unsupported event will set the `MHPMEVENTx` register to 0 (corresponding 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
<b>Frontend unit events</b>		
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
<b>Instruction cache unit events</b>		
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.

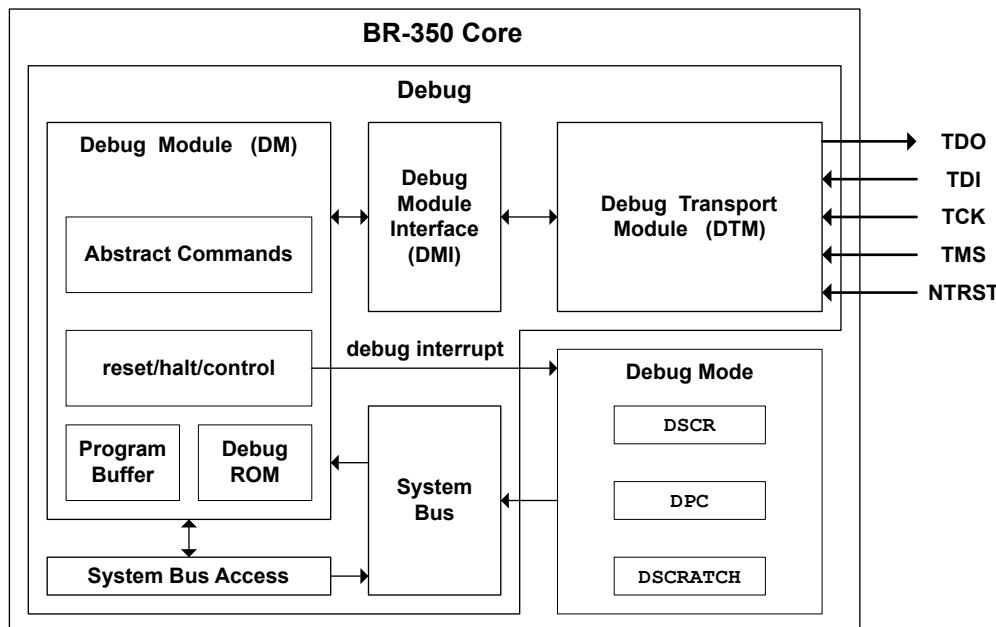


Figure 6-2 Debug Subsystem



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.

Core processor interacts with DM only in Debug Mode. Core reads and writes the 4KB 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	Access
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 - 0xFFFF	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, control flow redirects execution to the exception handler inside the Debug ROM. This handler perform 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 <code>dret</code> instruction and continue execution from the virtual address stored in <code>DPC</code> 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

To use hardware breakpoints, the Trigger Module is implemented in the core. The Trigger Module contains four mach triggers with the ability to stop when the instruction address or memory access address matches. The triggers can also be chained together to stop when multiple 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 core should provide machine software and timer interrupts support. The *Core Local Interruptor (CLINT)* is responsible for generating requests (signals) for these interrupts. The CLINT implementation corresponds to the [RISC-V Advanced Core Local Interruptor Specification](#).



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 causes an interrupt (`bus_err_intr`) if an error occurs in the processor core module (when writing to the I/O region, the core received a response signalling an error).



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 accessed via csr instructions and are described in the [CLIC CSRs](#) section.

### 6.1.2. BR-350 Core Registers

In this section:

- [Control and Status Registers](#)
- [DTM Registers](#)
- [DM Registers \(access via DMI\)](#)
- [CLINT Registers](#)

- Bus Error Unit Registers
- CLIC Registers

### 6.1.2.1. Control and Status Registers

This section describes the *Control and Status Registers (CSRs)* that can be accessed 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 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
<b>User Mode (U-Mode)</b>		
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
<b>Machine Mode (M-Mode)</b>		
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 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 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	<a href="#">Machine NMI Trap Handler Address</a>
XCCFG	0xBC8	<a href="#">Core Configuration</a>
XICFG	0xBD8	<a href="#">Instruction Cache Configuration</a>

### 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 and reading 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]	<b>NMITVEC</b>	WARL	NMI handler address This address must be 4-byte aligned. <b>Value After Reset:</b> 0x0

#### 6.1.2.1.4.2.2. Core Configuration (**XCCFG**)

The **XCCFG** register is at offset 0xBC8.



This register is only accessible 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]	<b>WBCLR</b>	R/W	When this bit is 1, validities of branch predictor structures will be cleared after wfi commit. <b>Value After Reset:</b> 0x0
[1]	<b>FBCLR</b>	R/W	When this bit is 1, validities of branch predictor structures will be cleared after fence.i commit. <b>Value After Reset:</b> 0x0
[0]	<b>WFI</b>	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. <b>Value After Reset:</b> 0x0

#### 6.1.2.1.4.2.3. Instruction Cache Configuration (**XICFG**)

The **XICFG** register is at offset 0xBD8.



This register is only accessible 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]	<b>WFIINV</b>	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
			<b>Value After Reset:</b> 0x0

### 6.1.2.2. DTM Registers

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

#### 6.1.2.2.1. Registers 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. Registers 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
DATA <i>i</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

**Table 6-12 DM Registers (access via DMI) (continued)**

Register	Offset	Description
ABSTRACTAUTO	0x18	Abstract Command Autoexec
PROGBUF <sub>i</sub>	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

Registers region of the *Core Local Interruptor (CLINT)* 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	64KB	0x0200_0000	0x0200_FFFF



For details, refer to [Memory Map](#) chapter.

##### 6.1.2.4.1. Registers Map

The following table provides 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 Registers Map**

Register	Offset	Description
MSIPO	0x0	<a href="#">Machine Software Interrupt Pending</a>
MTIMECMP0	0x4000	<a href="#">Machine Timer Compare</a>
MTIME	0xBFF8	<a href="#">Machine Timer Value</a>

**Table 6-14 CLINT Registers Map (continued)**

Register	Offset	Description
		 Timer value is increased by the <code>timer_pulse</code> 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	Access	Description
[31:1]			These bits are wired to zero
[0]	<code>MSIP0</code>	R/W	Machine software interrupt pending <b>Value After Reset:</b> 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	Access	Description
[63:32]	<code>MTIMECMP0_HI</code>	R/W	High part of timer compare value <b>Value After Reset:</b> 0x0
[31:0]	<code>MTIMECMP0_LO</code>	R/W	Low part of timer compare value <b>Value After Reset:</b> 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	Access	Description
[63:32]	MTIME_HI	R	High part of timer value <b>Value After Reset:</b> 0x0
[31:0]	MTIME_LO	R	Low part of timer value <b>Value After Reset:</b> 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	4KB	0x0202_0000	0x0202_0FFF



For details, refer to [Memory Map](#) chapter.

#### 6.1.2.5.1. Registers Map

The following table provides 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 accessible from the Machine privilege mode (M-mode).

**Table 6-19 Bus Error Unit Registers Map**

Register	Offset	Description
STATUS_CPU	0x0	<a href="#">CPU Status Register</a>
CONTROL_CPU	0x8	<a href="#">CPU Interrupt Control Register</a>
ADDR_LO_CPU	0x10	<a href="#">CPU Error Address Low Register</a>
ADDR_HI_CPU	0x14	<a href="#">CPU Error Address High Register</a>

#### 6.1.2.5.2. 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.

##### 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 <b>Values:</b> 0x0: An error while recording 0x1: Reserved <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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 <code>IRQ</code> bit of the <code>STATUS_CPU</code> register is set to 0x0.  <b>Value After Reset:</b> 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 <code>IRQ</code> bit of the <code>STATUS_CPU</code> register is set to 0x0. <b>Value After Reset:</b> N/A

### 6.1.2.6. CLIC Registers

Registers 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 accessed 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	20KB	0x0D00_0000	0x0D00_4FFF



For details, refer to [Memory Map](#) chapter.

#### 6.1.2.6.1. Registers Map

The following table provides 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 Registers 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.

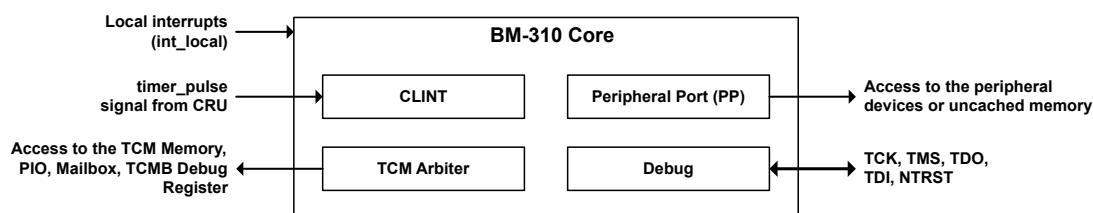


Figure 6-3 BM-310 Core Block Diagram

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

- [CLINT](#) is used to handle the interrupts
- Peripheral Port is used for access the BM-310 Core to the peripheral devices or uncached memory
- TCM Arbiter is used for arbitration of the requests to the TCM Memory. It also used for access to Mailbox, PIO and [Software JTAG Register](#).
- [Debug](#) is used for external core debug via 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.

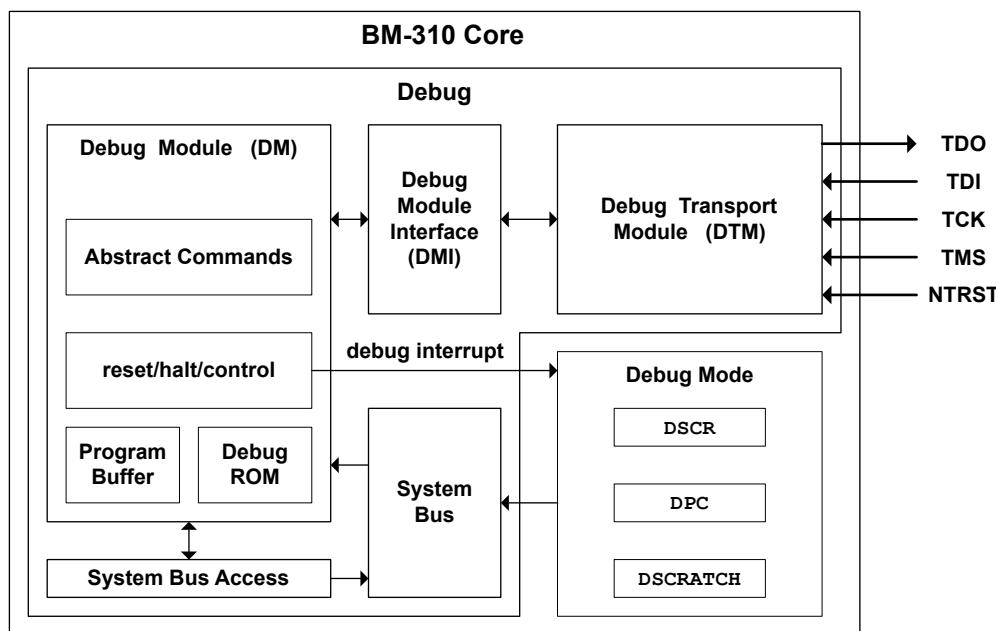


Figure 6-4 Debug Subsystem



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 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.

#### 6.2.1.1. 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 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.

Core processor interacts with DM only in Debug Mode. Core reads and writes the 4KB 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	Access
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	Access
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 - 0xFFFF	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, control flow redirects execution to the exception handler inside the Debug ROM. This handler perform 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 <code>dret</code> instruction and continue execution from the virtual address stored in <code>DPC</code> 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

To use hardware breakpoints, the Trigger Module is implemented in the core. The Trigger Module contains four mach triggers with the ability to stop when the instruction address or memory access address matches. The triggers can also be chained together to stop when multiple 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 core should provide machine software and timer interrupts support. The *Core Local Interruptor (CLINT)* is responsible for generating requests (signals) for these interrupts. The CLINT implementation corresponds to the [RISC-V Advanced Core Local Interruptor Specification](#).



The CLINT includes memory-mapped registers that are described in the [CLINT Registers](#) section.

### 6.2.2. BM-310 Core Registers

In this section:

- [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 Status Registers (CSRs)* that can be accessed 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 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 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 6-29 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 accessible 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 wfi commit. <b>Value After Reset:</b> 0x0
[1]	FBCLR	R/W	When this bit is 1, validities of branch predictor structures will be cleared after fence.i commit.

**Table 6-31 Fields For Register: xccfg (continued)**

Bits	Name	R/W	Description
			<b>Value After Reset:</b> 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. <b>Value After Reset:</b> 0x0

## 6.2.2.2. DTM Registers

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

### 6.2.2.2.1. Registers 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 6-32 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. Registers 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 6-33 DM Registers (access via DMI)**

Register	Offset	Description
DATA <i>i</i>	0x4 + (i*0x1)	Abstract Data <i>i</i> (for <i>i</i> = 0; <i>i</i> <= 3)
DMCONTROL	0x10	Debug Module Control
DMSTATUS	0x11	Debug Module Status
HARTINFO	0x12	Hart Info

**Table 6-33 DM Registers (access via DMI) (continued)**

Register	Offset	Description
ABSTRACTCS	0x16	Abstract Control and Status
COMMAND	0x17	Abstract Command
ABSTRACTAUTO	0x18	Abstract Command Autoexec
PROGBUF <sub>i</sub>	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]

#### 6.2.2.4. CLINT Registers

Registers region of the *Core Local Interruptor (CLINT)* 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	64KB	0x0200_0000	0x0200_FFFF



For details, refer to [Memory Map](#) chapter.

##### 6.2.2.4.1. Registers Map

The following table provides 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 Registers Map**

Register	Offset	Description
MSIPO	0x0	<a href="#">Machine Software Interrupt Pending</a>
MTIMECMP0	0x4000	<a href="#">Machine Timer Compare</a>

**Table 6-35 CLINT Registers Map (continued)**

Register	Offset	Description
MTIME	0xBFF8	<p>Machine Timer Value</p> <p><span style="color: yellow;">!</span> Timer value is increased by the <code>timer_pulse</code> signal from the Control Registers Unit</p>

#### 6.2.2.4.2. Register Descriptions

##### 6.2.2.4.2.1. Machine Software Interrupt Pending (`MSIPO`)

The `MSIPO` register is at offset 0x0.

Machine software interrupt pending (`MSIPO`) 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: `MSIPO`**

Bits	Name	Access	Description
[31:1]			These bits are wired to zero
[0]	<code>MSIPO</code>	R/W	Machine software interrupt pending <b>Value After Reset:</b> 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	Access	Description
[63:32]	<code>MTIMECMP0_HI</code>	R/W	High part of timer compare value <b>Value After Reset:</b> 0x0
[31:0]	<code>MTIMECMP0_LO</code>	R/W	Low part of timer compare value <b>Value After Reset:</b> 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	Access	Description
[63:32]	MTIME_HI	R	High part of timer value <b>Value After Reset:</b> 0x0
[31:0]	MTIME_LO	R	Low part of timer value <b>Value After Reset:</b> 0x0

## 7. Core 2 PIO

The BE-U1000 microcontroller contains a feature that allows user to implement its own additional interfaces via the **Programmable Input/Output (PIO)** and this feature supported by the PIO Block and Core 2 - where the core specially intended to support additional programmable interfaces of the BE-U1000, that has not supported by the appropriate internal controllers in the BE-U1000.

The PIO Block has a several registers that used to data exchange with I/O pins and you can find it in [PIO Registers](#) section.

The diagram below gives a common overview on the PIO Block location in the BE-U1000.

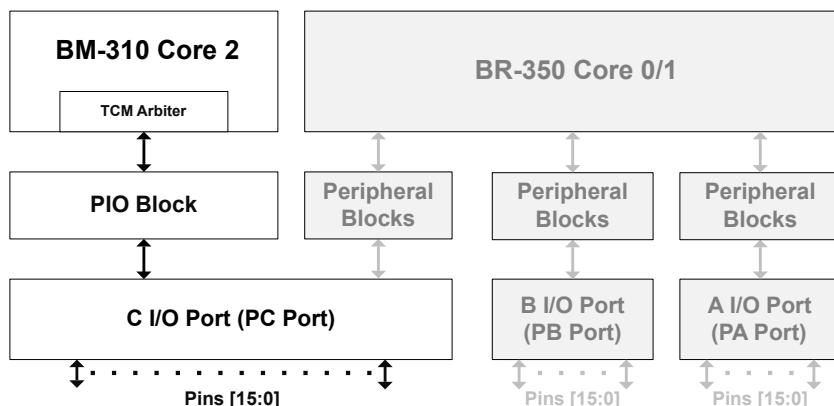


Figure 7-1 PIO Block in the BE-U1000

### 7.1. PIO Registers

In this section:

- [Registers Map](#)
- [Register Descriptions](#)

The following table describes microcontroller 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 [Registers Map](#) and [Register Descriptions](#) sections below.

#### 7.1.1. Registers Map

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

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

Table 7-2 PIO Registers Map

Name	Offset	Description	Default Value
INT	0x0	<a href="#">Interrupt Register</a>	0x0

**Table 7-2 PIO Registers Map (continued)**

Name	Offset	Description	Default Value
PIO_IN	0x4	Input Data Register	0x0
PIO_EN	0x8	Enable Register	0x0
PIO_OUT	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. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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
			<p><code>PIO_EN[i]</code> = <b>1</b>: <code>PIO_EN[i]</code> - configured as output.</p> <p><code>PIO_EN[i]</code> = <b>0</b>: <code>PIO_EN[i]</code> - configured as input.</p> <p><b>Value After Reset:</b> 0x0</p>

#### 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.

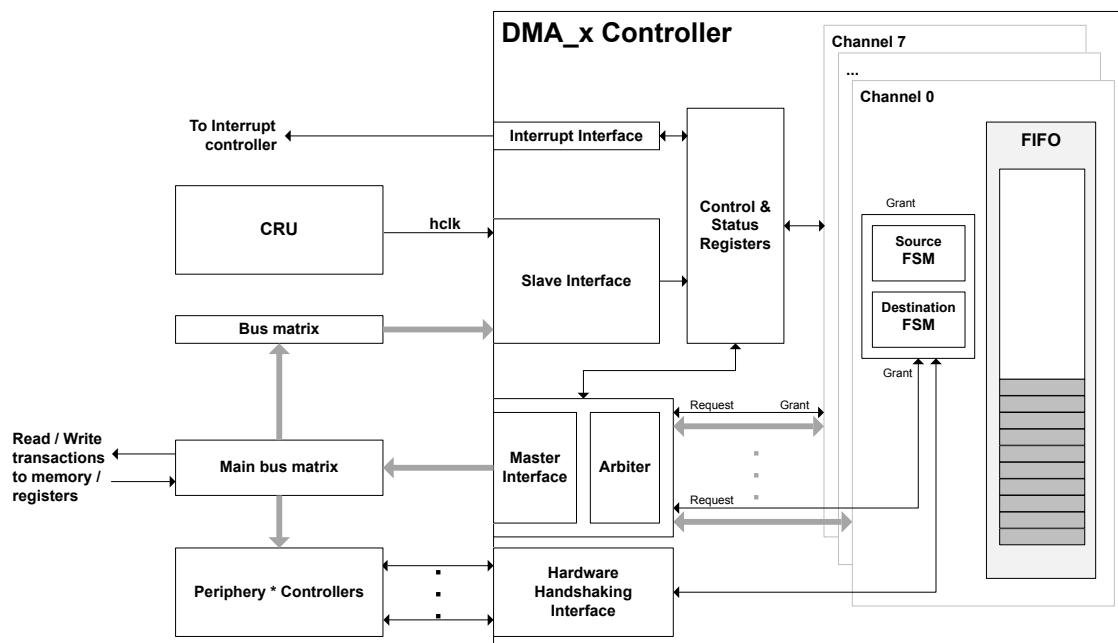
**Table 7-6 Fields For Register: `PIO_OU`**

Bits	Name	R/W	Description
[31:16]			Reserved
[15:0]	<code>PIO_OU</code>	R/W	<p>The data, that be transferred to the output.</p> <p><b>Value After Reset:</b> 0x0</p>

## 8. DMA Controller

This chapter describes the *Direct Memory Access (DMA)* controller of the BE-U1000 microcontroller. There are two DMA controllers in BE-U1000 microcontroller located in the High-speed Periphery block.

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



**Figure 8-1 DMA Controller Functional Block Diagram**



- $\times$  indicates 0 for DMA\_0 and 1 for DMA\_1
- Asterisk symbol (\*) indicates the number of an appropriate Periphery

The DMA Controller component provides the following features:

- General:
  - Slave Interface – used to program the DMA Controller
  - 8 Channels, one per source and 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 which are controlled via CRU when selecting the [Alternate Functions](#) using a multiplexer
  - 2 software handshaking interface
  - Handshaking interface supports single or burst DMA transactions
  - Enabling and disabling of 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 Channel Configuration register, `CFGx`). 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 all 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_CS, 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 QSPI0_1 I/O pins (QSPI0_CS0, QSPI0_SCK, QSPI0_MOSI, QSPI0_MISO, QSPI0_CS1, QSPI0_CS2 or QSPI0_CS, QSPI0_SCK, QSPI0_IO0, QSPI0_IO1, QSPI0_IO2, QSPI0_IO3)	SPI_0 TX	* Use all SPI_0 I/O pins (SPI0_CS, 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 all SPI_3 I/O pins (SPI3_CS, 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 QSPI1_1 I/O pins (QSPI1_CS0, QSPI1_SCK,	SPI_2 TX		

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

DMA HS channel	Peripheral block 0 (high priority)	Peripheral block 1 (low priority)	
7	QSPI_1 RX	QSPI1_MOSI, QSPI1_MISO, QSPI1_CSN1, QSPI1_CSN2 or QSPI1_CSN, QSPI1_SCK, QSPI1_IO0, QSPI1_IO1, QSPI1_IO2, QSPI1_IO3)	SPI_2 RX * Use all SPI_2 I/O pins (SPI2_CSN, SPI2_SCK, SPI2_MOSI, SPI2_MISO)
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 can be seen from the table above, some of the DMA Handshake channels are connected to the pair of peripheral blocks. There are some peculiarities of using such DMA Handshake channels:

- It is necessary to program all I/O pins, that corresponds to the peripheral block, to the appropriate alternate function. For example, to use the DMA Handshake channels for UART\_6, Port C I/O pins from PC[6] to PC[9] should be programmed to Function #2 via the appropriate `IOAFCRx` CRU registers. If one of the mentioned Port C I/O pins (from PC[6] to PC[9]) will be programmed for other alternate function, DMA Handshake channels can not be used for the UART\_6. Thus, it is not possible to configure the DMA Handshake channel 0 (TX) to work only with the I2C\_0 and DMA Handshake channel 1 (RX) to work only with the UART\_6 at the same time.
- While using both peripheral blocks of the DMA Handshake channels, peripheral block 0 will have the higher priority than peripheral block 1. For example, if both I2C\_1 and UART\_2 peripheral blocks are used (pay attention that I/O pins alternate functions should be programmed so that all physical interface signals of the I2C\_1 and UART\_2 blocks are used, as it was described above), I2C\_1 block will have higher priority than UART\_2 block. Priorities of the blocks are shown in the table above.

For DMA Handshake channels connected to a single peripheral block, it is not necessary to program the I/O pins alternate functions to use all physical interface signals of the corresponding block. For example, it is possible to set the PA[6] I/O pin with the alternate Function #1 and use UART\_0 with the DMA Handshake channel 8 (TX), while the PA[7] can be set to the 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	<code>IOAFCR0[2:0] = 0x5 &amp;&amp; IOAFCR0[6:4] = 0x5    IOAFCR0[20:18]</code>
	I2C0_SDA	<code>= 0x1 &amp;&amp; IOAFCR0[22:20] = 0x1    IOAFCR0[26:24] = 0x3 &amp;&amp;</code>

**Table 8-2 Programming the Alternate Functions (continued)**

Peripheral block	I/O pins	Registers settings
		<code>IOAFCR0[30:28] = 0x3    IOAFCR1[2:0] = 0x2 &amp;&amp; IOAFCR1[6:4] = 0x2</code>
I2C_1	I2C1_SCL	<code>IOAFCR1[18:16] = 0x1 &amp;&amp; IOAFCR1[22:20] = 0x1   </code>
	I2C1_SDA	<code>IOAFCR1[26:24] = 0x2 &amp;&amp; IOAFCR1[30:28] = 0x2</code>
SPI_1	SPI1_CSN	<code>IOAFCR1[2:0] = 0x1 &amp;&amp; IOAFCR1[6:4] = 0x1 &amp;&amp; IOAFCR1[10:8] = 0x1 &amp;&amp; IOAFCR1[14:12] = 0x1</code>
	SPI1_SCK	
	SPI1_MOSI	
	SPI1_MISO	
QSPI_0	QSPI0_CSNO	<code>IOAFCR0[2:0] = 0x2 &amp;&amp; IOAFCR0[6:4] = 0x2 &amp;&amp; IOAFCR0[10:8] = 0x2 &amp;&amp; IOAFCR0[14:12] = 0x2 &amp;&amp; IOAFCR0[20:18] = 0x2 &amp;&amp; IOAFCR0[22:20] = 0x2    IOAFCR0[2:0] = 0x3 &amp;&amp; IOAFCR0[6:4] = 0x3 &amp;&amp; IOAFCR0[10:8] = 0x3 IOAFCR0[14:12] = 0x3 &amp;&amp; IOAFCR0[20:18] = 0x3 &amp;&amp; IOAFCR0[22:20] = 0x3</code>
	QSPI0_SCK	
	QSPI0_MOSI	
	QSPI0_MISO	
	QSPI0_CSNI	
	QSPI0_CSN2	
	QSPI0_CSN	
	QSPI0_SCK	
	QSPI0_IO0	
	QSPI0_IO1	
UART_6	UART6_TX	<code>IOAFCR4[26:24] = 0x2 &amp;&amp; IOAFCR4[30:28] = 0x2 &amp;&amp; IOAFCR5[2:0] = 0x2 &amp;&amp; IOAFCR5[6:4] = 0x2</code>
	UART6_RX	
	UART6_DE	
	UART6_RE	
UART_2	UART2_TX	<code>IOAFCR1[18:16] = 0x2 &amp;&amp; IOAFCR1[22:20] = 0x2</code>
	UART2_RX	
I2S_0	I2S0_SDO	<code>IOAFCR1[10:8] = 0x3 &amp;&amp; IOAFCR1[14:12] = 0x3 &amp;&amp; IOAFCR1[18:16] = 0x3 &amp;&amp; IOAFCR1[22:20] = 0x3</code>
	I2S0_SDI	
	I2S0_WS	
	I2S0_SCLK	
SPI_0	SPI0_CSN	<code>IOAFCR0[2:0] = 0x1 &amp;&amp; IOAFCR0[6:4] = 0x1 &amp;&amp; IOAFCR0[10:8] = 0x1 &amp;&amp; IOAFCR0[14:12] = 0x1</code>
	SPI0_SCK	

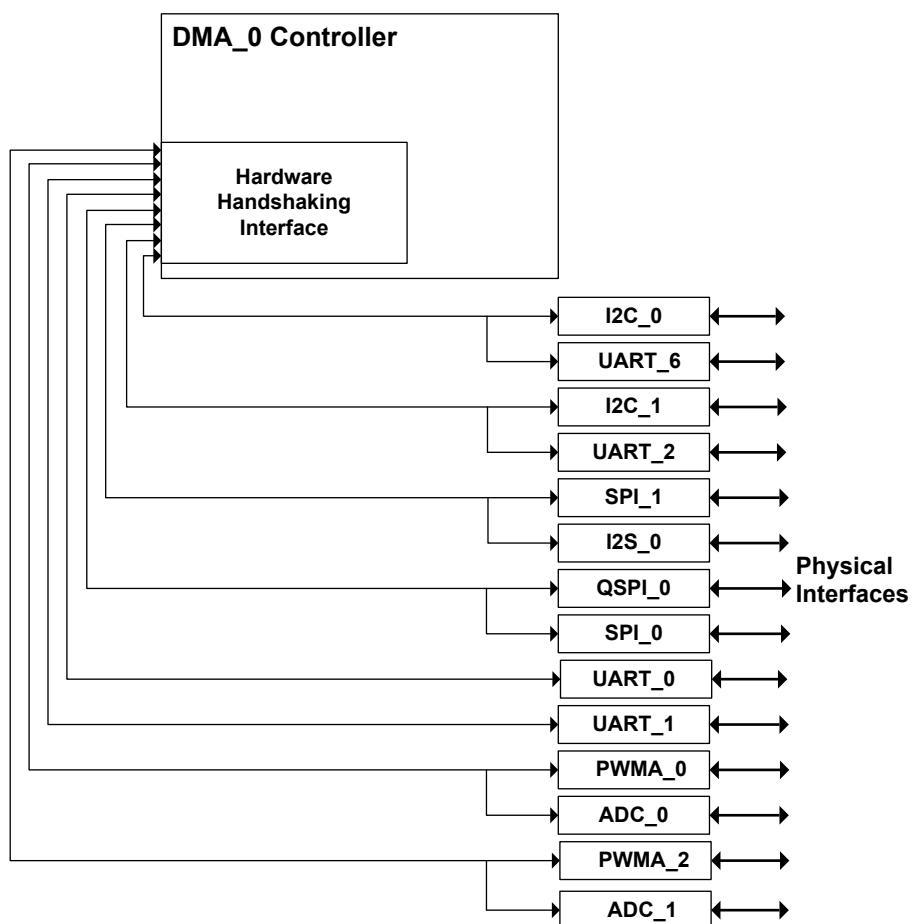
**Table 8-2 Programming the Alternate Functions (continued)**

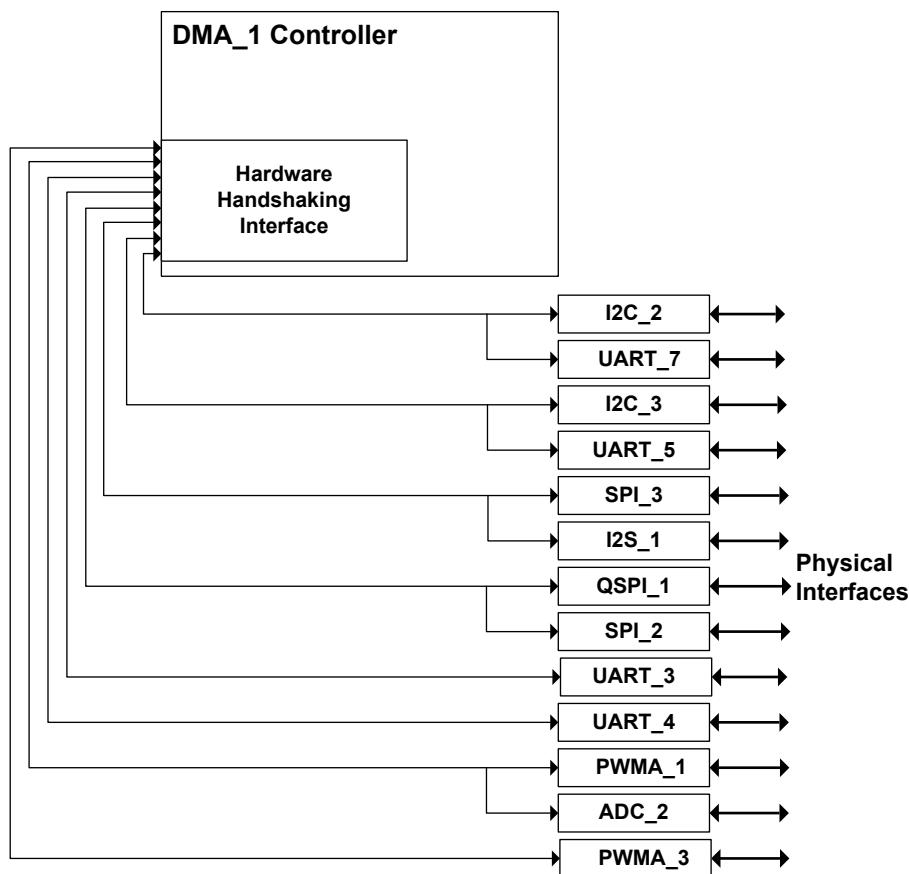
Peripheral block	I/O pins	Registers settings
	SPI0_MOSI	
	SPI0_MISO	
<b>DMA_1</b>		
I2C_2	I2C2_SCL	IOAFCR2[2:0] = 0x5 && IOAFCR2[6:4] = 0x5    IOAFCR2[20:18] = 0x1 && IOAFCR2[22:20] = 0x1    IOAFCR2[26:24] = 0x3 && IOAFCR2[30:28] = 0x3    IOAFCR3[2:0] = 0x2 && IOAFCR3[6:4] = 0x2
	I2C2_SDA	
I2C_3	I2C3_SCL	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_CSNO	IOAFCR2[2:0] = 0x2 && IOAFCR2[6:4] = 0x2 && IOAFCR2[10:8] = 0x2 && IOAFCR2[14:12] = 0x2 && IOAFCR2[20:18] = 0x2
	QSPI1_SCK	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_MOSI	
	QSPI1_MISO	
	QSPI1_CSNI	
	QSPI1_CSN2	
	QSPI1_CSN	
	QSPI1_SCK	
	QSPI1_IO0	
	QSPI1_IO1	
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_SD1	

**Table 8-2 Programming the Alternate Functions (continued)**

Peripheral block	I/O pins	Registers settings
	I2S1_WS	
	I2S1_SCLK	
SPI_2	SPI2_CSN	<code>IOAFCR2[2:0] = 0x1 &amp;&amp; IOAFCR2[6:4] = 0x1 &amp;&amp; IOAFCR2[10:8] = 0x1 &amp;&amp; IOAFCR2[14:12] = 0x1</code>
	SPI2_SCK	
	SPI2_MOSI	
	SPI2_MISO	

The following figures show the connectivity of handshaking channels for DMA\_0 and DMA\_1 respectively (pay attention that tx and rx handshaking channels are shown as a single line).

**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.

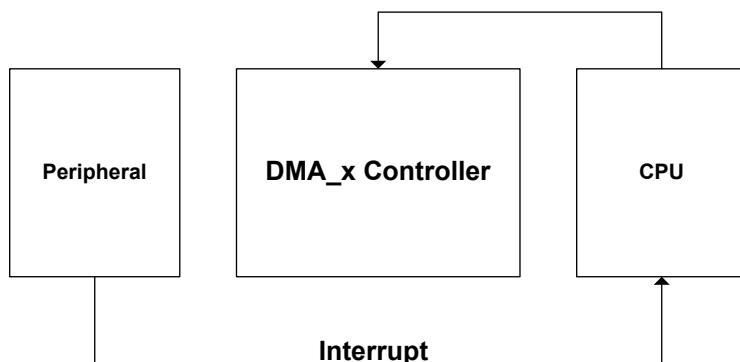
Therefore:

- 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; for more information. 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 only a peripheral:

- 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 may 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
<code>CTLx.TT_FC</code>	Transfer type and flow control

**Table 8-3 Parameters Used in Transfer Examples (continued)**

<b>Parameter</b>	<b>Description</b>
<code>CTLx.BLOCK_TS</code>	Block transfer size
<code>CTLx.SRC_TR_WIDTH</code>	Source transfer width
<code>CTLx.DST_TR_WIDTH</code>	Destination transfer width
<code>CTLx.SRC_MSIZE</code>	Source burst transaction length
<code>CTLx.DEST_MSIZE</code>	Destination burst transaction length
<code>CFGx.FIFO_MODE</code>	FIFO mode select
<code>CFGx.FCMODE</code>	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_MSIZEx = 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_MSIZEx` and Transmit FIFO Overflow

Programming `CTLx.DEST_MSIZEx` 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_MSIZEx <= SPI*_TX_FIFO_DEPTH - DMATDLR
```

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

```
CTLx.DEST_MSIZEx = 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_MSIZEx`.

---



`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_MSIZEx` 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_MSIZEx` – the \*SPI Controller receive FIFO may be emptied, but not underflowed, at the completion of the source burst transaction. For optimal operation, `CTLx.SRC_MSIZEx` should be set at the watermark level; that is:

$$\text{CTLx.SRC_MSIZEx} = \text{DMARDLR} + 1$$

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.

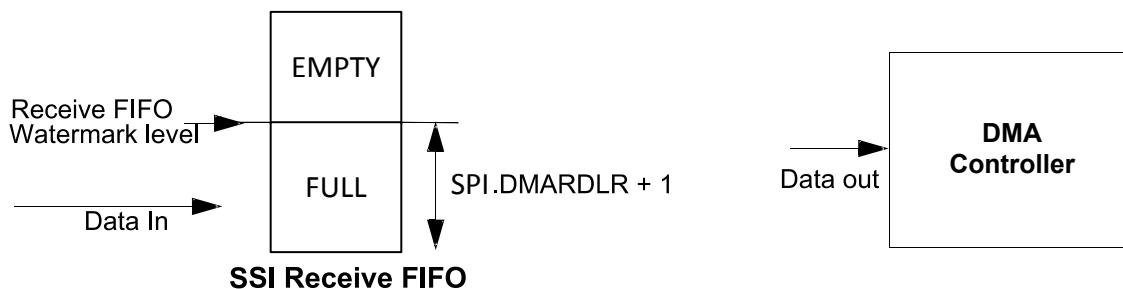


Figure 8-5 SPI Receive FIFO

### 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 `CFGx.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 a 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

There are two DMA controllers in BE-U1000 microcontroller located in the High-speed Periphery. The following table describes address regions for the DMA controllers in the BE-U1000 microcontroller.

**Table 8-4 Memory Address Regions for DMA Controllers**

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

### 8.2.1. Registers Map

This section contains information about the register memory map.

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.



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 Registers Map**

Name	Address Offset	Bits	Memory Access	Description
<b>Channel 0 Registers</b>				
SAR0	0x0	32	R/W	<a href="#">Source Address for Channel 0</a>
DAR0	0x8	32	R/W	<a href="#">Destination Address Register for Channel 0</a>
CTL0	0x18	64	R/W	<a href="#">Control Register for Channel 0</a>
CFG0	0x40	64	R/W	<a href="#">Configuration Register for Channel 0</a>
<b>Channel 1 Registers</b>				
SAR1	0x58	32	R/W	<a href="#">Source Address for Channel 1</a>
DAR1	0x60	32	R/W	<a href="#">Destination Address Register for Channel 1</a>
CTL1	0x70	64	R/W	<a href="#">Control Register for Channel 1</a>
CFG1	0x98	64	R/W	<a href="#">Configuration Register for Channel 1</a>
<b>Channel 2 Registers</b>				
SAR2	0xb0	32	R/W	<a href="#">Source Address for Channel 2</a>
DAR2	0xb8	32	R/W	<a href="#">Destination Address Register for Channel 2</a>
LLP2	0xc0	32	R/W	<a href="#">Linked List Pointer Register for Channel 2</a>
CTL2	0xc8	64	R/W	<a href="#">Control Register for Channel 2</a>

**Table 8-5 DMA Registers Map (continued)**

Name	Address Offset	Bits	Memory Access	Description
CFG2	0xf0	64	R/W	Configuration Register for Channel 2
<b>Channel 3 Registers</b>				
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
<b>Channel 4 Registers</b>				
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
<b>Channel 5 Registers</b>				
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
<b>Channel 6 Registers</b>				
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
<b>Channel 7 Registers</b>				
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 Registers 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
<b>Interrupt Registers</b>				
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
<b>Software Handshaking Registers</b>				

**Table 8-5 DMA Registers 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
<b>Miscellaneous Registers</b>				
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

The `SARx`, `DARx`, `LLPx`, `CTLx`, and `CFGx` channel registers should be programmed prior to enabling the channel. It is an Illegal Register Access when a write to the `SARx`, `DARx`, `LLPx`, `CTLx` registers occurs when the channel is enabled.



It is necessary to take into consideration that LLP channel registers exist only in channel 2 and channel 3.

#### 8.2.2.1.1. Source Address Register (`SARx`) for Channel x

The `SARx` register is at offset: for  $x=0$  to  $7$ :  $0x000+x*0x058$ . Its characteristics are:

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	<p>Current Source Address of DMA transfer</p> <p>Updated after each source transfer. The <code>SINC</code> field in the <code>CTLx</code> register determines whether the address increments, decrements, or is left unchanged on every source transfer throughout the block transfer.</p> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p>Current Destination address of DMA transfer</p> <p>Updated after each destination transfer. The <code>DINC</code> field in the <code>CTLx</code> register determines whether the address increments, decrements, or is left unchanged on every destination transfer throughout the block transfer.</p> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p>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.</p> <p>LLI accesses are always 32-bit accesses (<code>Hsize = 2</code>) aligned to 32-bit boundaries and cannot be changed or programmed to anything other than 32-bit.</p> <p><b>Value After Reset:</b> 0x0</p>
[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 (linked list item – 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	<p>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 <i>Linked List Item (LLI)</i> 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. <b>Value After Reset:</b> 0x0</p>
[43:40]			Reserved
[39:32]	BLOCK_TS	R/W	<p>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. <b>Width:</b> 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. <b>Value After Reset:</b> 0x2</p>
[31:29]			Reserved
[28]	LLP_SRC_EN	R/W	<p>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 <a href="#">Block Chaining using Linked Lists</a>. <b>Values:</b> <b>0x0 (LLP_SRC_DISABLE):</b> Block chaining using Linked List is disabled on the Source side <b>0x1 (LLP_SRC_ENABLE):</b> Block chaining using Linked List is enabled on the Source side <b>Value After Reset:</b> 0x0</p>
[27]	LLP_DST_EN	R/W	<p>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 <a href="#">Block Chaining using Linked Lists</a>. <b>Values:</b></p>

**Table 8-9 Fields For Register: CTLx (continued)**

Bits	Name	R/W	Description
			<p><b>0x0 (LLP_DST_DISABLE):</b> Block chaining using Linked List is disabled on the Destination side</p> <p><b>0x1 (LLP_DST_ENABLE):</b> Block chaining using Linked List is enabled on the Destination side</p> <p><b>Value After Reset:</b> 0x0</p>
[26:22]			Reserved
[21:20]	TT_FC	R/W	<p><b>Transfer Type and Flow Control.</b> Flow control can be assigned to the DMA controller, the source peripheral, or the destination peripheral.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0 (TT_FC_0):</b> Transfer type is Memory to Memory and Flow Controller is DMA Controller</li> <li><b>0x1 (TT_FC_1):</b> Transfer type is Memory to Peripheral and Flow Controller is DMA Controller</li> <li><b>0x2 (TT_FC_2):</b> Transfer type is Peripheral to Memory and Flow Controller is DMA Controller</li> <li><b>0x3 (TT_FC_3):</b> Transfer type is Peripheral to Peripheral and Flow Controller is DMA Controller</li> </ul> <p><b>Value After Reset:</b> 0x3</p>
[19:17]			Reserved
[16:14]	SRC_MSIZE	R/W	<p>Source Burst Transaction Length</p> <p>Number of data items, each of width <code>SRC_TR_WIDTH</code>, 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 <a href="#">the decoding</a> for this field.</p> <p><b>Value After Reset:</b> 0x1</p>
[13:11]	DEST_MSIZE	R/W	<p>Destination Burst Transaction Length</p> <p>Number of data items, each of width <code>DST_TR_WIDTH</code>, 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 <a href="#">the decoding</a> for this field.</p> <p><b>Value After Reset:</b> 0x1</p>
[10:9]	SINC	R/W	<p>Source Address Increment</p> <p>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.”</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0 (SINC_0):</b> Increments the source address</li> <li><b>0x1 (SINC_1):</b> Decrement the source address</li> <li><b>0x2 (SINC_2):</b> No change in the source address</li> <li><b>0x3 (SINC_3):</b> No change in the source address</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

**Table 8-9 Fields For Register: CTLx (continued)**

Bits	Name	R/W	Description
[8:7]	DINC	R/W	<p>Destination Address Increment 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.”</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0 (DINC_0):</b> Increments the destination address</li> <li><b>0x1 (DINC_1):</b> Decrements the destination address</li> <li><b>0x2 (DINC_2):</b> No change in the destination address</li> <li><b>0x3 (DINC_3):</b> No change in the destination address</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[6:4]	SRC_TR_WIDTH	R/W	<p>Source Transfer Width. (See the decoding for this field) Mapped to bus <code>hsize</code>. For a non-memory peripheral, typically the peripheral (source) FIFO width.</p> <p><b>Value After Reset:</b> 0x0</p>
[3:1]	DST_TR_WIDTH	R/W	<p>Destination Transfer Width. See the decoding for this field. Mapped to bus <code>hsize</code>. For a non-memory peripheral, typically <code>xgw</code> peripheral (destination) FIFO width.</p> <p><b>Value After Reset:</b> 0x0</p>
[0]	INT_EN	R/W	<p>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 <code>INT_EN</code> = 0.</p> <p><b>Value After Reset:</b> 0x1</p>

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**

<code>CTLx.SRC_MSIZE</code> / <code>CTLx.DEST_MSIZE</code>	Number of data items to be transferred (of width <code>CTLx.SRC_TR_WIDTH</code> or <code>CTLx.DST_TR_WIDTH</code> )
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 need to 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	<p>Destination hardware interface 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.</p> <p> For correct DMA operation, only one peripheral (source or destination) should be assigned to the same handshaking interface.</p> <p><b>Value After Reset:</b> 0x0</p>

**Table 8-13 Fields For Register: CFGx (continued)**

Bits	Name	R/W	Description
[42:39]	SRC_PER	R/W	<p>Source Hardware Interface Assigns a hardware handshaking interface (0 - 15) to the source of channel <math>x</math> 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.</p> <p><b>Value After Reset:</b> 0x0</p>
[38:37]			Reserved
[36:34]	PROTCTL	R/W	<p>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.</p> <p>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.</p> <p><b>Value After Reset:</b> 0x1</p>
[33]	FIFO_MODE	R/W	<p>FIFO Mode Select Determines how much space or data needs to be available in the FIFO before a burst transaction request is serviced.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Space/data available for single transfer of the specified transfer width.</li> <li><b>0x1:</b> 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.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[32]	FCMODE	R/W	<p>Flow Control Mode Determines when source transaction requests are serviced when the Destination Peripheral is the flow controller.</p> <p> This bit should be always set to 0x0 due to DMA is always the flow controller.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Source transaction requests are serviced when they occur. Data pre-fetching is enabled.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[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
			<p><b>Automatic Destination Reload.</b> The <a href="#">DARx</a> 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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0 (DISABLE):</b> Destination Reload Disabled.</li> <li><b>0x1 (ENABLE):</b> Destination Reload Enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[30]	RELOAD_SRC	R/W	<p>This bit field is available only for channels 2 and 3.</p> <p><b>Automatic Source Reload.</b> The <a href="#">SARx</a> 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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0 (DISABLE):</b> Source Reload Disabled</li> <li><b>0x1 (ENABLE):</b> Source Reload Enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[29:20]			Reserved
[19]	SRC_HS_POL	R/W	<p>Source Handshaking Interface Polarity</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Active high</li> <li><b>0x1:</b> Active low</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[18]	DST_HS_PL	R/W	<p>Destination Handshaking Interface Polarity</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Active high</li> <li><b>0x1:</b> Active low</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[17:12]			Reserved
[11]	HS_SEL_SRC	R/W	<p>Source Software or Hardware Handshaking Select</p> <p>This register field selects which of the handshaking interfaces – hardware or software – is active for source requests on this channel.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Hardware handshaking interface. Software-initiated transaction requests are ignored.</li> <li><b>0x1:</b> Software handshaking interface. Hardware-initiated transaction requests are ignored.</li> </ul> <p>If the source peripheral is memory, then this bit is ignored.</p> <p><b>Value After Reset:</b> 0x1</p>
[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
			<p>This register selects which of the handshaking interfaces – hardware or software – is active for destination requests on this channel.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Hardware handshaking interface. Software-initiated transaction requests are ignored.</li> <li><b>0x1:</b> Software handshaking interface. Hardware- initiated transaction requests are ignored.</li> </ul> <p>If the destination peripheral is memory, then this bit is ignored.</p> <p><b>Value After Reset:</b> 0x1</p>
[9]	FIFO_EMPTY	R	<p>Channel FIFO status</p> <p>Indicates if there is data left in the channel FIFO. Can be used in conjunction with <code>CH_SUSP</code> to cleanly disable a channel.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0 (NOT_EMPTY):</b> Channel FIFO is not empty</li> <li><b>0x1 (EMPTY):</b> Channel FIFO is empty</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[8]	CH_SUSP	R/W	<p>Channel Suspend</p> <p>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 <code>FIFO_EMPTY</code> to cleanly disable a channel without losing any data.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0 (NOT_SUSPENDED):</b> DMA transfer from the source is not suspended</li> <li><b>0x1 (SUSPENDED):</b> Suspend DMA transfer from the source</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[7:5]	CH_PRIOR	R/W	<p>Channel priority</p> <p>A priority of 7 is the highest priority, and 0 is the lowest. This field must be programmed within the following range: 0:7</p> <p>A programmed value outside this range will cause erroneous behavior.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0 (<code>CH_PRIOR_0</code>):</b> Channel priority is 0</li> <li><b>0x1 (<code>CH_PRIOR_1</code>):</b> Channel priority is 1</li> <li><b>0x2 (<code>CH_PRIOR_2</code>):</b> Channel priority is 2</li> <li><b>0x3 (<code>CH_PRIOR_3</code>):</b> Channel priority is 3</li> <li><b>0x4 (<code>CH_PRIOR_4</code>):</b> Channel priority is 4</li> <li><b>0x5 (<code>CH_PRIOR_5</code>):</b> Channel priority is 5</li> <li><b>0x6 (<code>CH_PRIOR_6</code>):</b> Channel priority is 6</li> <li><b>0x7 (<code>CH_PRIOR_7</code>):</b> Channel priority is 7</li> </ul> <p><b>Value After Reset:</b> Channel Number (x)</p>

**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 <code>IntTfr</code> Interrupt <b>Values:</b> 0x0 (INACTIVE): Inactive Raw Interrupt Status 0x1 (ACTIVE): Active Raw Interrupt Status <b>Value After Reset :</b> 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 <code>IntBlock</code> Interrupt <b>Values:</b> 0x0 (INACTIVE): Inactive Raw Interrupt Status 0x1 (ACTIVE): Active Raw Interrupt Status <b>Value After Reset:</b> 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 <code>IntSrcTran</code> Interrupt <b>Values:</b> 0x0 (INACTIVE): Inactive Raw Interrupt Status 0x1 (ACTIVE): Active Raw Interrupt Status <b>Value After Reset:</b> 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 <code>IntDstTran</code> Interrupt <b>Values:</b>

**Table 8-18 Fields For Register: RAW\_DST\_TRAN (continued)**

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

#### 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	<p>Raw Status for <b>IntDstTran</b> Interrupt</p> <p><b>Values:</b></p> <p><b>0x0 (INACTIVE):</b> Inactive Raw Interrupt Status  <b>0x1 (ACTIVE):</b> Active Raw Interrupt Status</p> <p><b>Value After Reset:</b> 0x0</p>

#### 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** or **int\_n** bus, depending on interrupt polarity) 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	<p>Status for <b>IntTfr</b> Interrupt</p> <p><b>Values:</b></p> <p><b>0x0 (INACTIVE):</b> Inactive Interrupt Status  <b>0x1 (ACTIVE):</b> Active Interrupt Status</p> <p><b>Value After Reset:</b> 0x0</p>

### 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 or int\_n bus, depending on interrupt polarity) 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 <b>Values:</b> 0x0 (INACTIVE): Inactive Interrupt Status 0x1 (ACTIVE): Active Interrupt Status <b>Value After Reset:</b> 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 or int\_n bus, depending on interrupt polarity) 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 <b>Values:</b> 0x0 (INACTIVE): Inactive Interrupt Status 0x1 (ACTIVE): Active Interrupt Status <b>Value After Reset:</b> 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 or int\_n bus, depending on interrupt polarity) 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 <b>Values:</b> 0x0 (INACTIVE): Inactive Interrupt Status 0x1 (ACTIVE): Active Interrupt Status <b>Value After Reset:</b> 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 or int\_n bus, depending on interrupt polarity) 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 <b>Values:</b> 0x0 (INACTIVE): Inactive Interrupt Status 0x1 (ACTIVE): Active Interrupt Status <b>Value After Reset:</b> 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 <b>Values:</b> 0x0 (DISABLED): Interrupt mask write disable 0x1 (ENABLED): Interrupt mask write enable <b>Value After Reset:</b> 0x0

**Table 8-25 Fields For Register: MASK\_TFR (continued)**

Bits	Name	R/W	Description
[7:0]	INT_MASK	R/W	<p>Mask for <code>IntTfr</code> Interrupt</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x0 (MASK): Mask the interrupts</li> <li>0x1 (UNMASK): Unmask the interrupts</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p>Interrupt Mask Write Enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x0 (DISABLED): Interrupt mask write disable</li> <li>0x1 (ENABLED): Interrupt mask write enable</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[7:0]	INT_MASK	R/W	<p>Mask for <code>IntBlock</code> Interrupt</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x0 (MASK): Mask the interrupts</li> <li>0x1 (UNMASK): Unmask the interrupts</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

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

The `MASK_SRC_TRAN` register is at offset 0x320. Its characteristics are:

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	<p>Interrupt Mask Write Enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x0 (DISABLED): Interrupt mask write disable</li> <li>0x1 (ENABLED): Interrupt mask write enable</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[7:0]	INT_MASK	R/W	Mask for <code>IntSrcTran</code> Interrupt

**Table 8-27 Fields For Register: MASK\_SRC\_TRAN (continued)**

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

#### 8.2.2.2.14. Mask for IntDstTran Interrupt (**MASK\_DST\_TRAN**)

The **MASK\_DST\_TRAN** register is at offset 0x328. Its characteristics are:

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 <b>Values:</b> <b>0x0 (DISABLED):</b> Interrupt mask write disable <b>0x1 (ENABLED):</b> Interrupt mask write enable <b>Value After Reset:</b> 0x0
[7:0]	INT_MASK	R/W	Mask for IntDstTran Interrupt <b>Values:</b> <b>0x0 (MASK):</b> Mask the interrupts <b>0x1 (UNMASK):</b> Unmask the interrupts <b>Value After Reset:</b> 0x0

#### 8.2.2.2.15. Mask for IntErr Interrupt (**MASK\_ERR**)

The **MASK\_ERR** register is at offset 0x330. Its characteristics are:

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 <b>Values:</b> <b>0x0 (DISABLED):</b> Interrupt mask write disable <b>0x1 (ENABLED):</b> Interrupt mask write enable <b>Value After Reset:</b> 0x0
[7:0]	INT_MASK	R/W	Mask for IntErr Interrupt <b>Values:</b>

**Table 8-29 Fields For Register: MASK\_ERR (continued)**

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

**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	<p>Clear for IntTfr Interrupt</p> <p><b>Values:</b></p> <p><b>0x0 (NOT_CLEAR):</b> No effect  <b>0x1 (CLEAR):</b> Clears interrupts</p> <p><b>Value After Reset:</b> 0x0</p>

**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	<p>Clear for IntBlock Interrupt</p> <p><b>Value After Reset :</b> 0x0</p>

**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	<p>Clear for IntSrcTran Interrupt</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x0 (NOT_CLEAR): No effect</li> <li>0x1 (CLEAR): Clears interrupts</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

#### 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	<p>Clear for IntDstTran Interrupt</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x0 (NOT_CLEAR): No effect</li> <li>0x1 (CLEAR): Clears interrupts</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

#### 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	<p>Clear for IntErr Interrupt</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x0 (NOT_CLEAR): No effect</li> <li>0x1 (CLEAR): Clears interrupts</li> </ul>

**Table 8-34 Fields For Register: CLR\_ERR (continued)**

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

### 8.2.2.2.21. Status for each interrupt type (STAT)

The STAT register is at offset 0x360.

The 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 <b>Values:</b> 0x0 (INACTIVE): OR of the contents of STAT_ERR register is 0 0x1 (ACTIVE): OR of the contents of STAT_ERR register is 1 <b>Value After Reset:</b> 0x0
[3]	DSTT	R	OR of the contents of STAT_DST_TRAN <b>Values:</b> 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 <b>Value After Reset:</b> 0x0
[2]	SRCT	R	OR of the contents of STAT_SRC_TRAN <b>Values:</b> 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 <b>Value After Reset:</b> 0x0
[1]	BLOCK	R	OR of the contents of STAT_BLOCK register <b>Values:</b> 0x0 (INACTIVE): OR of the contents of STAT_BLOCK register is 0 0x1 (ACTIVE): OR of the contents of STAT_BLOCK register is 1 <b>Value After Reset:</b> 0x0
[0]	TFR	R	OR of the contents of STAT_TFR register <b>Values:</b> 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
			<b>Value After Reset:</b> 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]	<code>SRC_REQ_WE</code>	W	<p>Source Software Transaction Request write enable  <b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0 (DISABLED):</b> Source request write Disable</li> <li><b>0x1 (ENABLED):</b> Source request write Enable</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[7:0]	<code>SRC_REQ</code>	R/W	<p>Source Software Transaction Request  <b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0 (INACTIVE):</b> Source request is not active</li> <li><b>0x1 (ACTIVE):</b> Source request is active</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

#### 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	<p>Destination Software Transaction Request write enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x0 (DISABLED): Destination request write Disable</li> <li>0x1 (ENABLED): Destination request write Enable</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[7:0]	DST_REQ	R/W	<p>Destination Software Transaction Request</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x0 (INACTIVE): Destination request is not active</li> <li>0x1 (ACTIVE): Destination request is active</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

#### 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	<p>Source Single Transaction Request write enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x0 (DISABLED): Single write Disable</li> <li>0x1 (ENABLED): Single write Enable</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[7:0]	SRC_SGLREQ	R/W	<p>Source Single Transaction Request</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x0 (INACTIVE): Source request is not active</li> <li>0x1 (ACTIVE): Source request is active</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

#### 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]	<code>DST_SGLREQ_WE</code>	W	Destination Single Transaction Request write enable <b>Values:</b> 0x0 (DISABLED): Destination write Disable 0x1 (ENABLED): Destination write Enable <b>Value After Reset:</b> 0x0
[7:0]	<code>DST_SGLREQ</code>	R/W	Destination Single Transaction Request <b>Values:</b> 0x0 (INACTIVE): Destination Single or burst request is not active 0x1 (ACTIVE): Destination Single or burst request is active <b>Value After Reset:</b> 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]	<code>LSTSRC_WE</code>	W	Source Last Transaction Request write enable <b>Values:</b> 0x0 (DISABLED): Source last transaction request write disable 0x1 (ENABLED): Source last transaction request write enable <b>Value After Reset:</b> 0x0
[7:0]	<code>LSTSRC</code>	R/W	Source Last Transaction Request register <b>Values:</b>

**Table 8-40 Fields For Register: LST\_SRC (continued)**

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

### 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	<p>Source Last Transaction Request write enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0</b> (DISABLED): Destination last transaction request write disable</li> <li><b>0x1</b> (ENABLED): Destination last transaction request write enable</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[7:0]	LSTDST	R/W	<p>Destination Last Transaction Request</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0</b> (NOT_LAST): Not last transaction in current block</li> <li><b>0x1</b> (LAST): Last transaction in current block</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

### 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: <b>0x0:</b> DMA Controller Disabled <b>0x1:</b> DMA Controller Enabled <b>Reset value:</b> 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 <b>Value After Reset :</b> 0x0

[7:0]	CH_EN	R/W	<b>Channel Enable</b> Valid values: <b>0x0:</b> Disable the Channel <b>0x1:</b> Enable the Channel
			The <code>CH_EN</code> 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. <b>Value After Reset :</b> 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	<p>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.</p> <p>Valid values:</p> <ul style="list-style-type: none"> <li><b>0x0 (NORMAL_MODE):</b> Puts the Slave Interface into Normal mode</li> <li><b>0x1 (TEST_MODE):</b> Puts the Slave Interface into Test mode. In this mode, the readback value of the writable registers always matches the values written.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

## 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 [SAR<sub>x</sub>/DAR<sub>x</sub>](#) 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 [CTL<sub>x</sub>](#) 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 [LLP<sub>x</sub>](#) register in the DMA Controller is reprogrammed using block chaining with linked lists. A

block descriptor consists of six registers: [SAR<sub>x</sub>](#), [DAR<sub>x</sub>](#), [LLP<sub>x</sub>](#), [CTL<sub>x</sub>](#). The first four registers, along with the [CFG<sub>x</sub>](#) 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 ([LLP<sub>x</sub>](#)) that stores the address in memory of the next linked list item. Each LLI contains the corresponding block descriptors:

1. [SAR<sub>x</sub>](#)
2. [DAR<sub>x</sub>](#)
3. [LLP<sub>x</sub>](#)
4. [CTL<sub>x</sub>](#)

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 [SAR<sub>x</sub>](#), [DAR<sub>x</sub>](#), [LLP<sub>x</sub>](#), and [CTL<sub>x</sub>](#) registers are fetched from system memory on an LLI update. The updated contents of the [CTL<sub>x</sub>](#) register is written back to memory on block completion.

Rows 6 through 10 of the table below show the required values of [LLP<sub>x</sub>](#), [CTL<sub>x</sub>](#), and [CFG<sub>x</sub>](#) for multi-block DMA transfers using block chaining.



For rows 6 through 10 of the table below, the [LLI.CTL<sub>x</sub>](#), [LLI.LLP<sub>x</sub>](#), [LLI.SAR<sub>x</sub>](#), and [LLI.DAR<sub>x</sub>](#) register locations of the [LLI](#) are always affected at the start of every block transfer. The [LLI.LLP<sub>x</sub>](#) and [LLI.CTL<sub>x</sub>](#) locations are always used to reprogram the DMA Controller [LLP<sub>x</sub>](#) and [CTL<sub>x</sub>](#) registers. However, depending on the row number in the table below, the [LLI.SAR<sub>x</sub>/LLI.DAR<sub>x</sub>](#) address may or may not be used to reprogram the DMA Controller [SAR<sub>x</sub>/DAR<sub>x</sub>](#) registers.

**Table 8-45 Transfer Types**

Transfer Type	<a href="#">LLP<sub>x</sub>[31:2] = 0</a>	<a href="#">CTL<sub>x</sub> [28]</a>	<a href="#">CFG<sub>x</sub> [30]</a>	<a href="#">CTL<sub>x</sub> [27]</a>	<a href="#">CFG<sub>x</sub> [31]</a>	<a href="#">CTL<sub>x</sub>, LLP<sub>x</sub> Update Method</a>	<a href="#">SAR<sub>x</sub> Update Method</a>	<a href="#">DAR<sub>x</sub> Update Method</a>	<a href="#">Write Back</a>
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	CTL <sub>x</sub> , LLP <sub>x</sub> are reloaded from initial values.	Contiguous	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	Contiguous	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 Linked List item.	Contiguous	Linked List	Yes
7. Linked list multi-block transfer with auto-reload SAR	No	0	1	1	0	CTLx, LLPx loaded from next Linked List item.	Auto-Reload	Linked List	Yes
8. Linked list multi-block transfer with contiguous DAR	No	1	0	0	0	CTLx, LLPx loaded from next Linked List item.	Linked List	Contiguous	Yes
9. Linked list multi-block transfer with auto-reload DAR	No	1	0	0	1	CTLx, LLPx loaded from next Linked List item.	Linked List	Auto-Reload	Yes
10. Linked list multi-block transfer	No	1	0	1	0	CTLx, LLPx loaded from next Linked List item.	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 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.

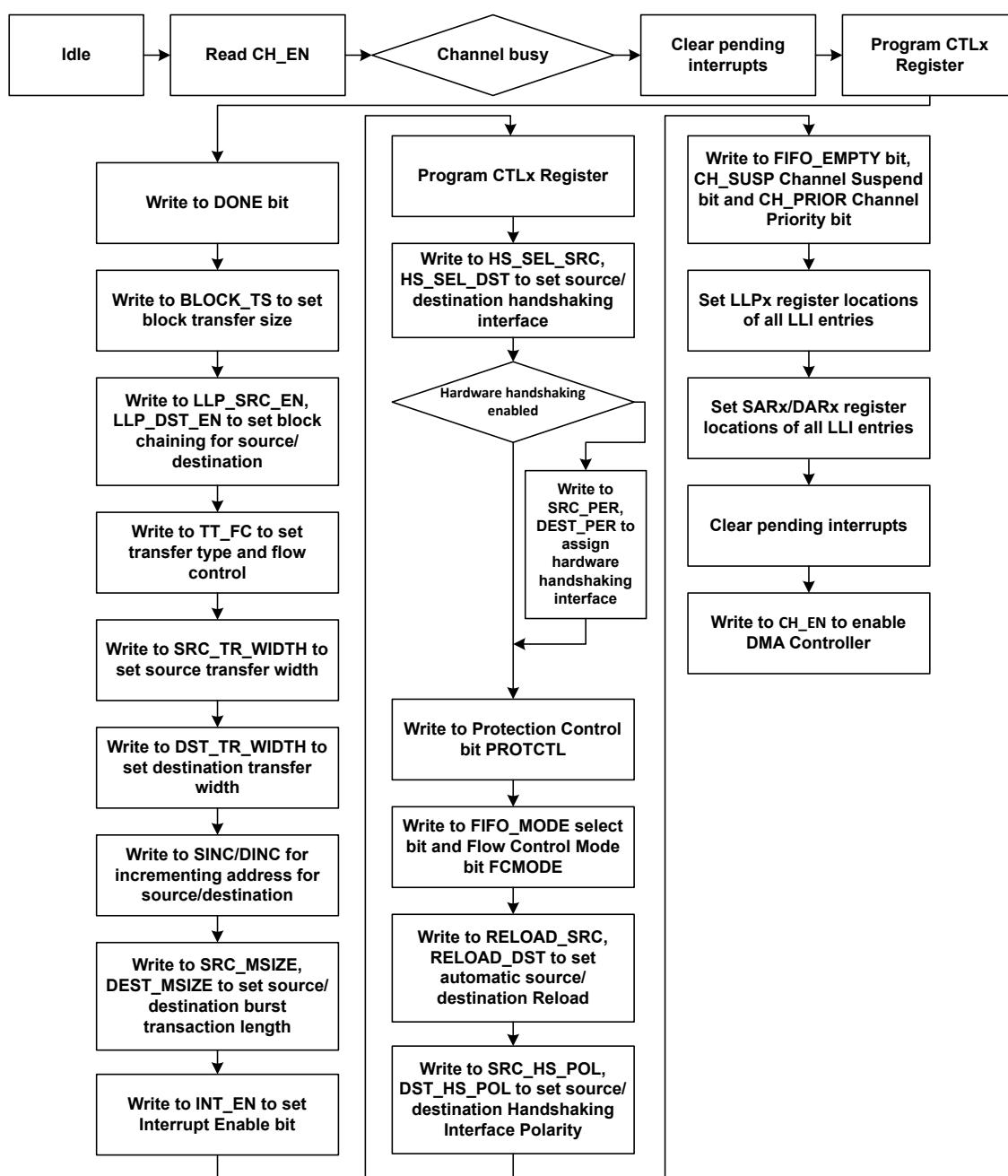


Figure 8-6 Flowchart for DMA Programming Example

### 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 [table of transfer modes](#), 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 ands Destination**

Transfer Type	<code>LLPx[31 :2] = 0</code>	<code>CTLx [28]</code>	<code>CFGx [30]</code>	<code>CTLx [27]</code>	<code>CFGx [31]</code>	<code>CTLx, LLPx</code> Update Method	<code>SARx</code> Update Method	<code>DARx</code> Update Method	Write Back
Linked list multi-block transfer	No	1	0	1	0	<code>CTLx, LLPx</code> 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.

---

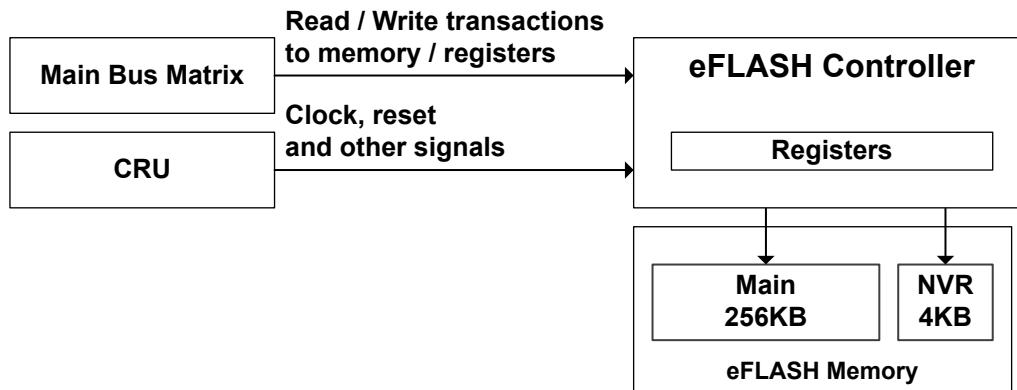
If a channel is disabled by software, an active single or burst transaction is not guaranteed to receive an acknowledgement.

### 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 functional block diagram**

As shown in the figure above, the eFLASH Memory consists of the Main array (256KB) and NVR array (4KB).

The eFLASH Memory can be accessed in the following ways:

- Registers access: access to the eFLASH Memory occurs through the eFLASH Controller registers
- Direct access: access 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)	<a href="#">SYSCR0.NVR</a>
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)	<a href="#">SYSCR0.FLASHNVRDIS</a>
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)	<a href="#">FLASHNVRCR.NVR1DIS</a>

**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	<code>SYSCR0.DAMODDIS</code>



Pay attention that NVR 1 is forbidden to modify because it contains the information (identification, configuration and reference information) which is recorded during the eFLASH memory factory tasting. NVR 1 modification will lead to the unpredictable behaviour of the eFLASH memory.

## 9.1. eFLASH Controller Registers

Registers 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	4KB	0xA100_0000	0xA100_0FFF



For details, refer to [Memory Map](#) chapter.

In this chapter:

- [Registers Map](#)
- [Register Descriptions](#)

### 9.1.1. Registers Map

The following table provides 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	<a href="#">eFLASH Controller Control Register</a>	0x0
EFLASH_ADDR	0x4	<a href="#">eFLASH Controller Address Register</a>	0x0
EFLASH_WDATA	0x8	<a href="#">eFLASH Controller Write Data Register</a>	0x0
EFLASH_RDATA	0xC	<a href="#">eFLASH Controller Read Data Register</a>	0x0
EFLASH_SR	0x10	<a href="#">eFLASH Controller Status Register</a>	0x1
EFLASH_TIME0	0x14	<a href="#">eFLASH Controller Time Intervals Register 0</a>	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 <code>OP_CODE</code> field will be performed in the NVR array if this bit is set to 0x1. <b>Value After Reset:</b> 0x0.
[3:1]	OP_CODE	R/W	Operation Code. <b>Values:</b> <b>0x0:</b> (CFG) Memory configuration <b>0x1:</b> (WRITE) Writes a 32-bit word to the eFLASH memory <b>0x2:</b> (READ) Reads a 32-bit word from the eFLASH memory <b>0x3:</b> (CHIP_ERASE) Chip erase <b>0x4:</b> (BLOCK_ERASE) Block erase <b>0x5:</b> (SECTOR_ERASE) Sector erase <b>0x6:</b> (ENTER_DPD) Enters the <i>Deep Power Down (DPD)</i> Mode <b>0x7:</b> (EXIT_DPD) Exit DPD Mode

**Table 9-4 Fields For Register: `EFLASH_CR` (continued)**

Bits	Name	R/W	Description
			<b>Value After Reset:</b> 0x0.
[0]	RUN	R/W	Write 0x1 to this bit to execute the command from the <code>OP_CODE</code> field. This bit is automatically set to 0x0 after the command execution begins. <b>Value After Reset:</b> 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.  <b>Value After Reset:</b> 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 wrtten to memory when executing the WRITE command.  <b>Value After Reset:</b> 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.  <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 0x1

### 9.1.2.6. eFLASH Controller Time Intervals Register 0 (`EFLASH_TIME0`)

The `EFLASH_TIME0` register is at offset 0x14.



Pay attention that **Values After Reset**, mentioned for the bit fields in the table below, may be differ from the actual values due to their changes 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 50MHz 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 <sub>AA</sub> and t <sub>RC</sub> .  The default value of this bit field corresponds to the 25ns time interval at the 50MHz frequency.  <b>Value After Reset:</b> 0x2
[26:4]	SCE_T	R/W	Configures the coefficient for the t <sub>SCE</sub> .  The default value of this bit field corresponds to the 8ms time interval at the 50MHz frequency.  <b>Value After Reset:</b> 0x61A80
[3:0]	DS_T	R/W	Configures the coefficient for the t <sub>DS</sub> , t <sub>APS</sub> , t <sub>APH</sub> , t <sub>DH</sub> , t <sub>PGH</sub> , t <sub>PREP</sub> , t <sub>CFS</sub> , t <sub>CFH</sub> , t <sub>CONFEN</sub> .  The default value of this bit field corresponds to the 15ns time interval at the 50MHz frequency.  <b>Value After Reset:</b> 0x1

### 9.1.2.7. eFLASH Controller Time Intervals Register 1 (`EFLASH_TIME1`)

The `EFLASH_TIME1` register is at offset 0x18.



Pay attention that **Values After Reset**, mentioned for the bit fields in the table below, may be differ from the actual values due to their changes 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 50MHz 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	<p>Configures the coefficient for the <math>t_{RCV}</math> (ERASE).</p> <p> The default value of this bit field corresponds to the 50us time interval at the 50MHz frequency.</p> <p><b>Value After Reset:</b> 0x9C4</p>
[15:2]	RHR_T	R/W	<p>Configures the coefficient for the <math>t_{RHR}</math>, <math>t_{NVS}</math> (CHIP ERASE), <math>t_{DPDH}</math>, <math>t_{CFL}</math>.</p> <p> The default value of this bit field corresponds to the 10us time interval at the 50MHz frequency.</p> <p><b>Value After Reset:</b> 0x1F4</p>
[1:0]	AS_T	R/W	<p>Configures the coefficient for the <math>t_{AS}</math>, <math>t_{CS}</math>, <math>t_{AH}</math>.</p> <p> The default value of this bit field corresponds to the 2ns time interval at the 50MHz frequency.</p> <p><b>Value After Reset:</b> 0x1</p>

#### 9.1.2.8. eFLASH Controller Time Intervals Register 2 (`EFLASH_TIME2`)

The `EFLASH_TIME2` register is at offset 0x1C.



Pay attention that **Values After Reset**, mentioned for the bit fields in the table below, may be differ from the actual values due to their changes 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 50MHz 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	<p>Configures the coefficient for the <math>t_{NVS}</math> (PROGRAM, SECTOR ERASE, BLOCK ERASE).</p> <p> The default value of this bit field corresponds to the 4us time interval at the 50MHz frequency.</p> <p><b>Value After Reset:</b> 0xC8</p>
[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 8us time interval at the 50MHz frequency. <b>Value After Reset:</b> 0x190
[6:0]	RW_T	R/W	Configures the coefficient for the tRW.  The default value of this bit field corresponds to the 100ns time interval at the 50MHz frequency. <b>Value After Reset:</b> 0x5

### 9.1.2.9. eFLASH Controller Time Intervals Register 3 (EFLASH\_TIME3)

The EFLASH\_TIME3 register is at offset 0x20.



Pay attention that **Values After Reset**, mentioned for the bit fields in the table below, may be differ from the actual values due to their changes 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 50MHz 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 tPGS.  The default value of this bit field corresponds to the 20us time interval at the 50MHz frequency. <b>Value After Reset:</b> 0x3E8
[16:12]	CBD_T	R/W	Configures the coefficient for the tCBD.  The default value of this bit field corresponds to the 20ns time interval at the 50MHz frequency. <b>Value After Reset:</b> 0x1
[11:0]	PREPROG_T	R/W	Configures the coefficient for the tPREPROG.  The default value of this bit field corresponds to the 2.5us time interval at the 50MHz frequency. <b>Value After Reset:</b> 0x7D

### 9.1.2.10. eFLASH Controller Time Intervals Register 4 (EFLASH\_TIME4)

The EFLASH\_TIME4 register is at offset 0x24.



Pay attention that **Values After Reset**, mentioned for the bit fields in the table below, may be differ from the actual values due to their changes 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 50MHz 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 50ns time interval at the 50MHz frequency. <b>Value After Reset:</b> 0x3
[25:4]	SBE_T	R/W	Configures the coefficient for the $t_{SBE}$ , $t_{HV\_pg}$ , $t_{HV\_prep}$ . The default value of this bit field corresponds to the 4ms time interval at the 50MHz frequency. <b>Value After Reset:</b> 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 10ns time interval at the 50MHz frequency. <b>Value After Reset:</b> 0x1

### 9.1.2.11. eFLASH Controller Time Intervals Register 5 (`EFLASH_TIME5`)

The `EFLASH_TIME5` register is at offset 0x28.



Pay attention that **Values After Reset**, mentioned for the bit fields in the table below, may be differ from the actual values due to their changes 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 50MHz 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 200ns time interval at the 50MHz frequency. <b>Value After Reset:</b> 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 3.2ms time interval at the 50MHz frequency. <b>Value After Reset:</b> 0x27100

### 9.1.2.12. eFLASH Controller Time Intervals Register 6 (EFLASH\_TIME6)

The EFLASH\_TIME6 register is at offset 0x2C.



Pay attention that **Value After Reset**, mentioned for the bit field in the table below, may be differ from the actual value due to it modification 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 50MHz 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 tPREPGS, tRCV (PROGRAM).  The default value of this bit field corresponds to the 5us time interval at the 50MHz frequency. <b>Value After Reset:</b> 0xFA

## 9.2. Programming the eFLASH Controller

In this section:

- [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 \(EXIT\\_DPD\)](#)
- [Direct 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 50MHz 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 ( $t_{DS}=15\text{ns}$ ) at the 100MHz frequency ( $T_f=10\text{ns}$ ) 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
$t_{AA}, t_{RC}$	25ns
$t_{SCE}$	8ms
$t_{DS}, t_{APS}, t_{APH}, t_{DH}, t_{PGH}, t_{PREPGH}, t_{CFS}, t_{CFH}, t_{CONFEN}$	15ns
$t_{RCV}$ (ERASE)	50us
$t_{RHR}, t_{NVS}$ (CHIP ERASE), $t_{DPDH}, t_{CFL}$	10us
$t_{AS}, t_{CS}, t_{AH}$	2ns
$t_{NVS}$ (PROGRAM, SECTOR ERASE, BLOCK ERASE)	4us
$t_{PROG}$	8us
$t_{RW}$	100ns
$t_{PGS}$	20us
$t_{CBD}$	20ns
$t_{PREPROG}$	2.5us
$t_{TEST}$	50ns
$t_{SBE}, t_{HV\_pg}, t_{HV\_prep}$	4ms
$t_{RS}$ (RECALL), $t_{RR}$ (RECALL)	10ns
$t_{RCR}, t_{AAR}$	200ns
$t_{ERASE}$	3.2ms
$t_{PREPGS}, t_{RCV}$ (PROGRAM)	5us

## 9.2.2. Memory Configuration Operation (CFG)

Execute the following steps to perform the eFLASH memory configuration process:

1. Write 0x1 to the `EFLASH_CR.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. Direct access to the eFLASH Memory

To enable the direct access to the eFLASH memory, set the `SYSCRO.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 is used for the message exchanging between the Core 0/1 and Core 2. Each core mentioned above can leave its own data messages in this space and can read messages from the rest of cores. The access address to this space differs from the direction of access - by Core 0/1 side or Core 2 side.



The Mailbox intended for message exchanging only between the Core 0/1 and Core 2.

The Mailbox contains two similar modules - MB0...1 with its own set of registers.

For details on described above, refer to [Mailbox Registers](#) section.

The following diagram gives an overview to the Mailbox in the BE-U1000 microcontroller.



Due to architectural features, the Core 2 has two equivalent way access to the Mailbox: via the TCM Arbiter and Peripheral Port and access from the each of them need to use the different offsets for the same Mailbox registers. For details, refer to [Mailbox Registers](#) section.

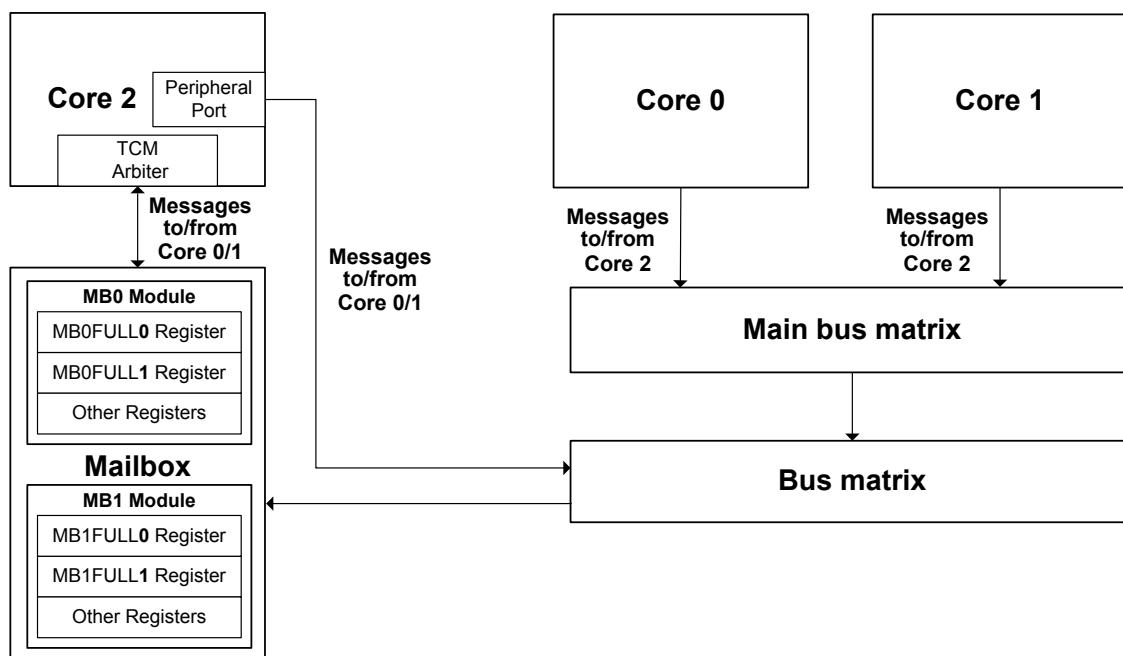


Figure 10-1 Mailbox in Block Diagram

### 10.1. Mailbox Registers

In this section:

- [Registers 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*	4KB	0x1510_3000	0x1510_3FFF
The Mailbox registers, accessed from the Core 2 side	1KB	0x4000_A400	0x4000_A800



\*Due to architectural features, the Core 2 has two equivalent way access to the Mailbox: via the TCM Arbiter and Peripheral Port and access from the each of them need to use the 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.

You can find list and description of Mailbox registers in the [Registers Map](#) and [Register Descriptions](#) sections below.

### 10.1.1. Registers Map

This section tables contain information about the Mailbox registers memory map.

Since the Mailbox contains two similar modules **MB0...1**, the table below shows memory mapping for registers of both modules.

The following table provides 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 Registers Map**

Name	Offset	Description	Default Value
<b>MB0 Module of Mailbox Registers</b>			
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
<b>MB1 Module of Mailbox Registers</b>			
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 Registers 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. <b>Value After Reset:</b> 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	<p>This bit acts as a readiness flag for the message in the <b>MBnDATA0</b> register</p> <ul style="list-style-type: none"> <li>• This bit is set to 0x1 when the <b>MBnDATA0</b> register is written</li> <li>• This bit is set to 0x0 when the <b>MBnDATA0</b> register is read from the Core 2 side</li> </ul> <p>This bit can be set to any value by writing from the Core 0/1 side <b>Value After Reset:</b> 0x0</p>

### 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]	<a href="#">MB_N_EMPTY0</a>	R	R	<p>This bit acts as message not ready flag in the <a href="#">MBnDATA0</a> register</p> <ul style="list-style-type: none"> <li>• This bit is set to 0x0 when the <a href="#">MBnDATA0</a> register is written</li> <li>• This bit is set to 0x1 when the <a href="#">MBnDATA0</a> register is read from the Core 2 side</li> </ul> <p><b>Value After Reset:</b> 0x1</p>

### 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]	<a href="#">MB_N_DATA1</a>	R	R/W	<p>Message data.</p> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p>This bit acts as a readiness flag for the message in the <a href="#">MBnDATA1</a> register</p> <ul style="list-style-type: none"> <li>• This bit is set to 0x1 when the <a href="#">MBnDATA1</a> register is written</li> <li>• This bit is set to 0x0 when the <a href="#">MBnDATA1</a> register is read from the Core 0/1 side</li> </ul> <p>This bit can be set to any value by writing from the Core 2 side</p> <p><b>Value After Reset:</b> 0x0</p>

**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.

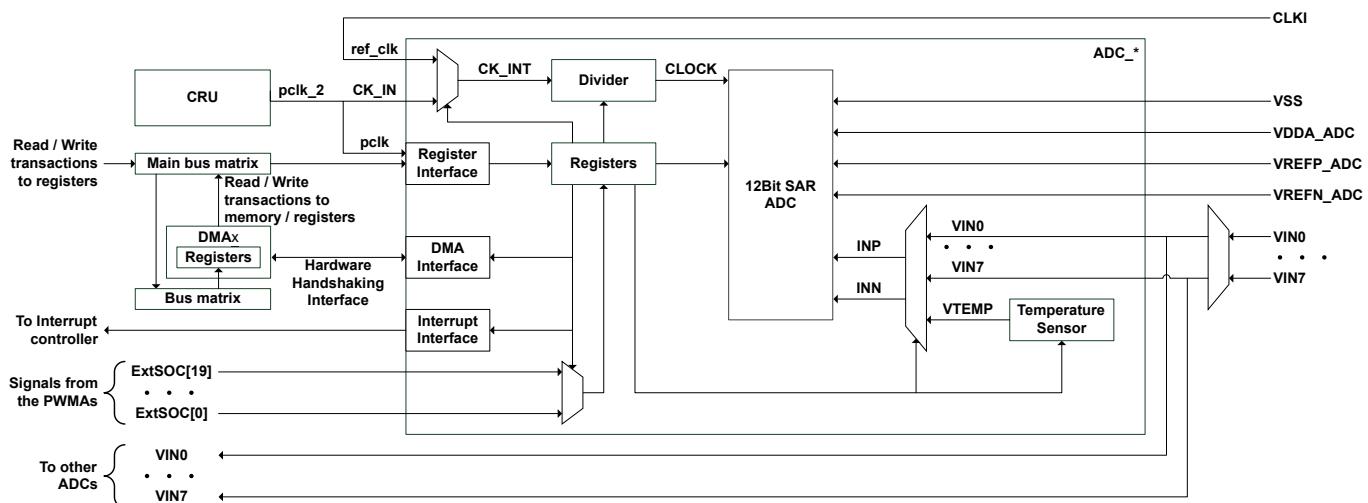
**Table 10-8 Fields For Register: MBnEMPTY1**

Bits	Name	Access		Description
		Core 0/1	Core 2	
[0]	MB_N_EMPTY1	R	R	<p>This bit acts as message not ready flag in the <a href="#">MBnDATA1</a> register</p> <ul style="list-style-type: none"> <li>• This bit is set to 0x0 when the <a href="#">MBnDATA1</a> register is written</li> <li>• This bit is set to 0x1 when the <a href="#">MBnDATA1</a> register is read from the Core 0/1 side</li> </ul> <p><b>Value After Reset:</b> 0x1</p>

## 11. ADC

This section describes the *Analog-to-Digital Converter (ADC)* of the BE-U1000 microcontroller. The BE-U1000 contains three ADC.

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



**Figure 11-1 ADC Functional Block Diagram**



In the figure above, asterisk symbol (\*) indicates the ADC number and  $\underline{x}$  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

In this section:

- 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
<code>VREFP_ADC</code>	Input, positive reference voltage
<code>VREFN_ADC</code>	Input, negative reference voltage
<code>VDDA_ADC</code>	Input, analog power supply
<code>VSS</code>	Input, analog ground
<code>VIN[7:0]</code>	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	
		<code>VIN</code> connection to <code>INP</code>	<code>VIN</code> connection to <code>INN</code>
Channel 0	<code>VIN0</code>	<code>VIN0</code>	<code>VIN1</code>
Channel 1	<code>VIN1</code>	<code>VIN1</code>	<code>VIN0</code>
Channel 2	<code>VIN2</code>	<code>VIN2</code>	<code>VIN3</code>
Channel 3	<code>VIN3</code>	<code>VIN3</code>	<code>VIN2</code>
Channel 4	<code>VIN4</code>	<code>VIN4</code>	<code>VIN5</code>
Channel 5	<code>VIN5</code>	<code>VIN5</code>	<code>VIN4</code>
Channel 6	<code>VIN6</code>	<code>VIN6</code>	<code>VIN7</code>
Channel 7	<code>VIN7</code>	<code>VIN7</code>	<code>VIN6</code>

### 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.

In this section:

- [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 ( $x \leq 8$ ) 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`, `SQ1` and `SQ2` fields of the `ASQR` register;
- second short sequence: conversion of channels specified in the `SQ3`, `SQ4` and `SQ5` fields of the `ASQR` register;
- third short sequence: conversion of channel specified in the `SQ6` field 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  $T_L = 0^\circ\text{C}$  and  $T_H = 50^\circ\text{C}$

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 detection mode.
2. Record the `ADR.DATA` at  $T_L$  ( $0^\circ\text{C}$ ) as  $D_{TL}$
3. Record the `ADR.DATA` at  $T_H$  ( $50^\circ\text{C}$ ) 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 <code>oc0</code> signal	PWMA_1 <code>oc0</code> signal	PWMA_2 <code>oc0</code> signal
ADC_CR1.EXTSEL = 0x12	PWMA_0 <code>oc1</code> signal	PWMA_1 <code>oc1</code> signal	PWMA_2 <code>oc1</code> signal
ADC_CR1.EXTSEL = 0x11	PWMA_0 <code>oc2</code> signal	PWMA_1 <code>oc2</code> signal	PWMA_2 <code>oc2</code> signal
ADC_CR1.EXTSEL = 0x10	PWMA_0 <code>oc3</code> signal	PWMA_1 <code>oc3</code> signal	PWMA_2 <code>oc3</code> signal
ADC_CR1.EXTSEL = 0xF	PWMA_0 <code>TRGO</code> signal	PWMA_1 <code>TRGO</code> signal	PWMA_2 <code>TRGO</code> signal
ADC_CR1.EXTSEL = 0xE	PWMA_0 <code>pwma_0_tim_irq</code> signal	PWMA_1 <code>pwma_1_tim_irq</code> signal	PWMA_2 <code>pwma_2_tim_irq</code> signal
ADC_CR1.EXTSEL = 0xD	PWMA_1 <code>oc0</code> signal	PWMA_2 <code>oc0</code> signal	PWMA_3 <code>oc0</code> signal
ADC_CR1.EXTSEL = 0xC	PWMA_1 <code>oc1</code> signal	PWMA_2 <code>oc1</code> signal	PWMA_3 <code>oc1</code> signal
ADC_CR1.EXTSEL = 0xB	PWMA_1 <code>oc2</code> signal	PWMA_2 <code>oc2</code> signal	PWMA_3 <code>oc2</code> signal
ADC_CR1.EXTSEL = 0xA	PWMA_1 <code>oc3</code> signal	PWMA_2 <code>oc3</code> signal	PWMA_3 <code>oc3</code> signal
ADC_CR1.EXTSEL = 0x9	PWMA_1 <code>TRGO</code> signal	PWMA_2 <code>TRGO</code> signal	PWMA_3 <code>TRGO</code> signal
ADC_CR1.EXTSEL = 0x8	PWMA_1 <code>pwma_1_tim_irq</code> signal	PWMA_2 <code>pwma_2_tim_irq</code> signal	PWMA_3 <code>pwma_3_tim_irq</code> signal
ADC_CR1.EXTSEL = 0x7	PWMA_2 <code>oc0</code> signal	PWMA_3 <code>oc0</code> signal	PWMA_0 <code>oc0</code> signal
ADC_CR1.EXTSEL = 0x6	PWMA_2 <code>oc1</code> signal	PWMA_3 <code>oc1</code> signal	PWMA_0 <code>oc1</code> signal
ADC_CR1.EXTSEL = 0x5	PWMA_2 <code>oc2</code> signal	PWMA_3 <code>oc2</code> signal	PWMA_0 <code>oc2</code> signal
ADC_CR1.EXTSEL = 0x4	PWMA_2 <code>oc3</code> signal	PWMA_3 <code>oc3</code> signal	PWMA_0 <code>oc3</code> signal
ADC_CR1.EXTSEL = 0x3	PWMA_2 <code>TRGO</code> signal	PWMA_3 <code>TRGO</code> signal	PWMA_0 <code>TRGO</code> signal
ADC_CR1.EXTSEL = 0x2	PWMA_2 <code>pwma_2_tim_irq</code> signal	PWMA_3 <code>pwma_3_tim_irq</code> signal	PWMA_0 <code>pwma_0_tim_irq</code> signal
ADC_CR1.EXTSEL = 0x1	PWMA_3 <code>TRGO</code> signal	PWMA_0 <code>TRGO</code> signal	PWMA_1 <code>TRGO</code> signal
ADC_CR1.EXTSEL = 0x0	PWMA_3 <code>oc0</code> signal	PWMA_0 <code>oc0</code> signal	PWMA_1 <code>oc0</code> 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	4KB	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	4KB	0x1400_2000	0x1400_2FFF
	ADC_2	4KB	0x1400_3000	0x1400_3FFF



For details, refer to [Memory Map](#) chapter.

In this chapter:

- [Registers Map](#)
- [Register Descriptions](#)

### 11.2.1. Registers Map

The following table provides 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 Registers Map**

Register	Offset	Description	Default Value
ADC_CR1	0x00	<a href="#">ADC Control Register 1</a>	0x0
ASER	0x04	<a href="#">ADC Set Register</a>	0x346
ASTR	0x08	<a href="#">ADC Status Register</a>	0x0
ASMPR	0x0C	<a href="#">ADC Sample Time Register</a>	0x0
ASQR	0x10	<a href="#">ADC Sequence Register</a>	0x0
ADR	0x14	<a href="#">ADC Data Register</a>	0x0

### 11.2.2. 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.

#### 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 <a href="#">CLKI</a> pin <b>Values:</b> 0x0: <a href="#">pclk_2</a> 0x1: <a href="#">CLKI</a>

**Table 11-7 Fields For Register: ADC\_CRI (continued)**

Bits	Name	R/W	Description
			<b>Value After Reset:</b> 0x0
[21]	TEMPSENS	R/W	<p>Temperature Sensor Enables the <a href="#">Temperature Sensor Mode</a></p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Disabled</li> <li><b>0x1:</b> Enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[20]	CLRINTRPT	W	<p>Interrupt clear Write 0x1 to this bit to reset the setted interrupt. Writing 0x0 has no effect.</p> <p><b>Value After Reset:</b> 0x0</p>
[19:16]	CKD	R/W	<p>Clock division This bit-field indicates the division ratio between the input clock (<a href="#">CK_INT</a>) and ADC clock (<a href="#">CLOCK</a>).</p> <p> This bit field can be set to 0x0 only in <a href="#">Single Mode</a>.</p> <p> When setting this bit field, note that the maximum ADC clock (<a href="#">CLOCK</a>) frequency is 25 MHz.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> tCLOCK=tCK_INT</li> <li><b>0x1:</b> tCLOCK=2*tCK_INT</li> <li><b>0x2:</b> tCLOCK=3*tCK_INT</li> <li><b>0x3:</b> tCLOCK=4*tCK_INT</li> <li><b>0x4:</b> tCLOCK=5*tCK_INT</li> <li><b>0x5:</b> tCLOCK=6*tCK_INT</li> <li><b>0x6:</b> tCLOCK=7*tCK_INT</li> <li><b>0x7:</b> tCLOCK=8*tCK_INT</li> <li><b>0x8:</b> tCLOCK=9*tCK_INT</li> <li><b>0x9:</b> tCLOCK=10*tCK_INT</li> <li><b>0xA:</b> tCLOCK=11*tCK_INT</li> <li><b>0xB:</b> tCLOCK=12*tCK_INT</li> <li><b>0xC:</b> tCLOCK=13*tCK_INT</li> <li><b>0xD:</b> tCLOCK=14*tCK_INT</li> <li><b>0xE:</b> tCLOCK=15*tCK_INT</li> <li><b>0xF:</b> tCLOCK=16*tCK_INT</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[15]	SWSTART	R/W	<p>Start of Conversion This bit is used by software to start the conversion. Setting this bit to 0x1 starts the conversion.</p> <p><b>Value After Reset:</b> 0x0</p>
[14]	EXTRIG	R/W	<p>PWMA event conversion mode Enables the PWMA event to be used to start the conversion.</p> <p><b>Values:</b></p>

**Table 11-7 Fields For Register: ADC\_CRI (continued)**

Bits	Name	R/W	Description
			<p><b>0x0:</b> PWMA event conversion mode disabled  <b>0x1:</b> PWMA event conversion mode enabled</p> <p><b>Value After Reset:</b> 0x0</p>
[13:9]	EXTSEL	R/W	<p>External event select      Selects the PWMA event source.</p> <p> For details on the possible field values, refer to the PWMA Event Selection table in the <a href="#">Conversion on PWMA Event</a> section.</p> <p><b>Value After Reset:</b> 0x0</p>
[8:6]	DISCNUM	R/W	<p>Discontinuous mode short sequence length      This bit field define the number of the length of short sequence to be converted in <a href="#">Discontinuous Mode</a> minus 1</p> <p><b>Value After Reset:</b> 0x0 (means 1 sample)</p>
[5]	DMAEN	R/W	<p>DMA transfer enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> DMA transfer is disabled</li> <li><b>0x1:</b> DMA transfer is enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[4]	IEEOC	R/W	<p>Interrupt enable EOC      Enables the interrupt on the EOC signal edge</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Disabled</li> <li><b>0x1:</b> Enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[3]	DISCEN	R/W	<p>Discontinuous mode      Enables the <a href="#">Discontinuous Mode</a></p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Disabled</li> <li><b>0x1:</b> Enabled</li> </ul> <p> If both SCANEN and DISCEN bits are set to 0x1, discontinuous mode will be used</p> <p><b>Value After Reset:</b> 0x0</p>
[2]	SCANEN	R/W	<p>Scan mode      Enables the <a href="#">Scan Mode</a></p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Disabled</li> <li><b>0x1:</b> Enabled</li> </ul> <p> If both SCANEN and DISCEN bits are set to 0x1, discontinuous mode will be used</p> <p><b>Value After Reset:</b> 0x0</p>

**Table 11-7 Fields For Register: ADC\_CRI (continued)**

Bits	Name	R/W	Description
[1]	CONT	R/W	Continuous conversion <b>Values:</b> 0x0: Disabled 0x1: Enabled <b>Value After Reset:</b> 0x0
[0]	ADON	R/W	ADC on/off <b>Values:</b> 0x0: ADC Off 0x1: ADC On <b>Value After Reset:</b> 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 <b>Values:</b> 0: measurement 1: zero calibration <b>Value After Reset:</b> 0x0
[17:12]			Reserved (return 0 when read)
[11:8]	ADJ_TD_GA	R/W	Temperature sensor gain adjustment signal. <b>Values:</b> Xx0x: disable temperature sensor offset adjustment Xx1x: enable temperature sensor offset adjustment <b>Value After Reset:</b> 0x3
[7:4]	ADJ_TD_OS	R/W	Temperature sensor offset adjustment signal. <b>Values:</b> 000x: for CLOCK period >160ns 001x: for CLOCK period 80ns to 160ns 010x: for CLOCK period 40ns to 80ns <b>Value After Reset:</b> 0x4
[3]	SELDO_HS_LU	R/W	Output data type selection <b>Values:</b> 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
			<b>Value After Reset:</b> 0x0
[2:1]	TS_MODE	R/W	<p>Filler mode of temperature sensor.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>00: by pass internal low pass filter</li> <li>01: 64 times average</li> <li>10: 128 times average</li> <li>11: 256 times average</li> </ul> <p><b>Value After Reset:</b> 0x3</p>
[0]	SELVI_HD_LS	R/W	<p>Input type selection</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0: Single ended</li> <li>1: Differential</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p><i>Start of conversion (SOC)</i> complete</p> <p> Any write clears this register field.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x0: Conversion not started</li> <li>0x1: Conversion started</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[0]	EOC	R/WC	<p><i>End of Conversion (EOC)</i> complete</p> <p> Any write clears this register field.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x0: Conversion not completed</li> <li>0x1: Conversion completed</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

### 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.SMP<sub>x</sub> bit field corresponds to a channel specified in the ASQR.SQ<sub>x</sub> bit field, where <sub>x</sub> is the



conversion number. The possible values are calculated from the ADC input capacity (30pF) and input resistance (300ohm). 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 <b>SMP0</b> bit. <b>Value After Reset:</b> 0x0
[20:18]	SMP6	R/W	6th conversion sample start delay selection  The possible values of this bit are the same as the <b>SMP0</b> bit. <b>Value After Reset:</b> 0x0
[17:15]	SMP5	R/W	5th conversion sample start delay selection  The possible values of this bit are the same as the <b>SMP0</b> bit. <b>Value After Reset:</b> 0x0
[14:12]	SMP4	R/W	4th conversion sample start delay selection  The possible values of this bit are the same as the <b>SMP0</b> bit. <b>Value After Reset:</b> 0x0
[11:9]	SMP3	R/W	3th conversion sample start delay selection  The possible values of this bit are the same as the <b>SMP0</b> bit. <b>Value After Reset:</b> 0x0
[8:6]	SMP2	R/W	2th conversion sample start delay selection  The possible values of this bit are the same as the <b>SMP0</b> bit. <b>Value After Reset:</b> 0x0
[5:3]	SMP1	R/W	1th conversion sample delay selection  The possible values of this bit are the same as the <b>SMP0</b> bit. <b>Value After Reset:</b> 0x0
[2:0]	SMP0	R/W	0th conversion sample start delay selection <b>Values:</b> <b>0x0:</b> 0 ADC clock periods (0* tCLOCK) <b>0x1:</b> 25 ADC clock periods (25* tCLOCK) <b>0x2:</b> 50 ADC clock periods (50* tCLOCK) <b>0x3:</b> 75 ADC clock periods (75* tCLOCK) <b>0x4:</b> 125 ADC clock periods (125* tCLOCK)

**Table 11-10 Fields For Register: ASMPR (continued)**

Bits	Name	R/W	Description
			<p><b>0x5:</b> 175 ADC clock periods (<math>175 * t_{CLOCK}</math>)</p> <p><b>0x6:</b> 225 ADC clock periods (<math>225 * t_{CLOCK}</math>)</p> <p><b>0x7:</b> 275 ADC clock periods (<math>275 * t_{CLOCK}</math>)</p> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p>Full sequence length</p> <p>Defines the total number of the conversions in the conversion sequence minus 1.</p> <p><b>Value After Reset:</b> 0x0</p>
[23:21]	SQ7	R/W	<p>7th conversion in sequence</p> <p>Defines the channel number (0..7), that will be assigned as the 7th in the conversion sequence minus 1.</p> <p><b>Value After Reset:</b> 0x0</p>
[20:18]	SQ6	R/W	<p>6th conversion in sequence</p> <p>Defines the channel number (0..7), that will be assigned as the 6th in the conversion sequence minus 1.</p> <p><b>Value After Reset:</b> 0x0</p>
[17:15]	SQ5	R/W	<p>5th conversion in sequence</p> <p>Defines the channel number (0..7), that will be assigned as the 5th in the conversion sequence minus 1.</p> <p><b>Value After Reset:</b> 0x0</p>
[14:12]	SQ4	R/W	<p>4th conversion in sequence</p> <p>Defines the channel number (0..7), that will be assigned as the 4th in the conversion sequence minus 1.</p> <p><b>Value After Reset:</b> 0x0</p>
[11:9]	SQ3	R/W	<p>3th conversion in sequence</p> <p>Defines the channel number (0..7), that will be assigned as the 3th in the conversion sequence minus 1.</p> <p><b>Value After Reset:</b> 0x0</p>
[8:6]	SQ2	R/W	<p>2th conversion in sequence</p> <p>Defines the channel number (0..7), that will be assigned as the 2th in the conversion sequence minus 1.</p> <p><b>Value After Reset:</b> 0x0</p>
[5:3]	SQ1	R/W	1th 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 minus 1. <b>Value After Reset:</b> 0x0
[2:0]	SQ0	R/W	0th conversion in sequence Defines the channel number (0..7), that will be assigned as the 0th in the conversion sequence minus 1. <b>Value After Reset:</b> 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 <b>Value After Reset:</b> 0x0

## 12. Timers

In this chapter:

- [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 PWMA.

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

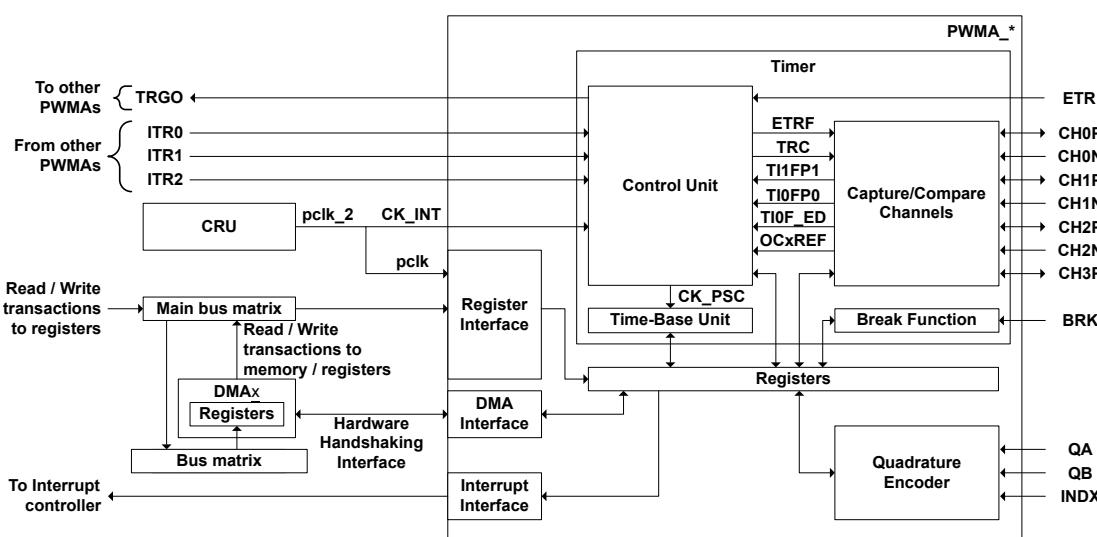


Figure 12-1 PWMA Functional Block Diagram



In the figure above, asterisk symbol (\*) indicates the PWMA number, and  $\times$  indicates the DMA number.



The Quadrature Encoder is a completely independent from the PWMA Timer block. Its description and recommendations for its programming are given in the sections [Quadrature Encoder](#).

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 controller supports the following features:

- 4 independent channels: 3 complimentary channels and one single channel
- Programmable prescaler for the counter clock ( $\text{CK\_CNT}$ ) 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 generation on the following events:
  - 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

In this section:

- [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 PWMA 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).



Refer to the [Master Mode Controller](#) section for details on `TRGO` generation.

**Table 12-1 ITR to TRGO connection**

ITR signal	TRGO signal
<b>PWMA_0</b>	
ITR2	TRGO_1
ITR1	TRGO_2
ITR0	TRGO_3
<b>PWMA_1</b>	
ITR2	TRGO_0
ITR1	TRGO_2
ITR0	TRGO_3

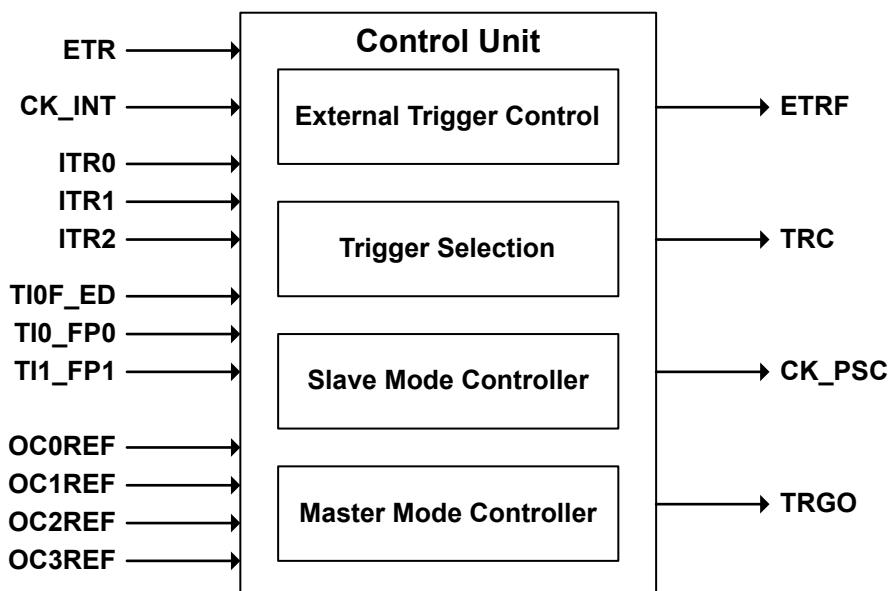
**Table 12-1 ITR to TRGO connection (continued)**

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

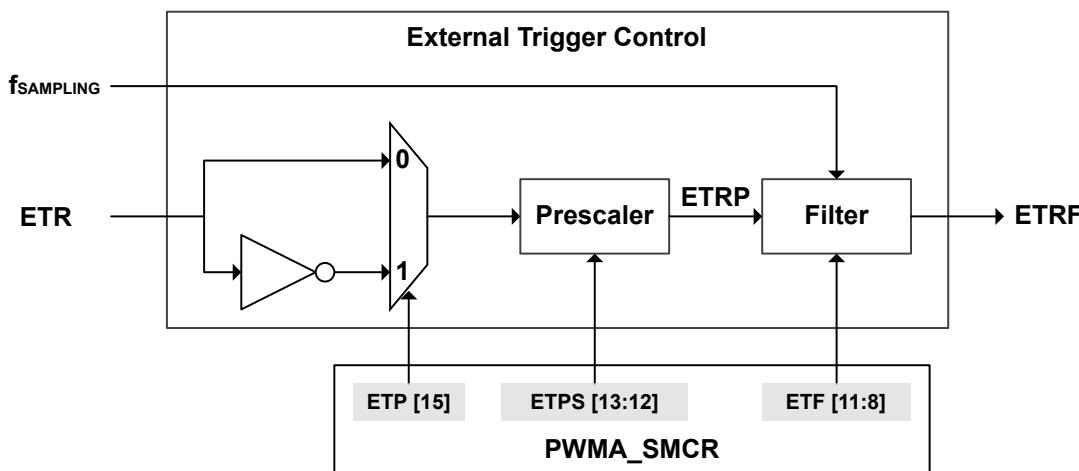
Control Unit consists of the following blocks, as the figure below shows:

- External Trigger Control
- Trigger Selection
- Slave Mode Controller
- Master Mode Controller

**Figure 12-2 Control Unit**

#### 12.1.1.2.1. External Trigger Control

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 **ETP** input signal
- **PWMA\_SMCR.ETPS** bit controls the prescaler that is used to divide the **ETP** input signal to produce the **ETRP** signal
- **PWMA\_SMCR.ETF** is used to select the sampling frequency (**f<sub>SAMPLING</sub>**) of the **ETRF** signal



ETRP frequency must be at most 1/4 of Timer clock (**CK\_INT**) frequency. It means that the maximum ETP frequency can be two times higher than the Timer clock frequency if the **ETPS** bit is set to 0x3 (ETP 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 **IC<sub>x</sub>** signal, where **x** is the channel number. As shown in the figure below, **TRGI** source is selected by setting the **PWMA\_SMCR.TS** bit.

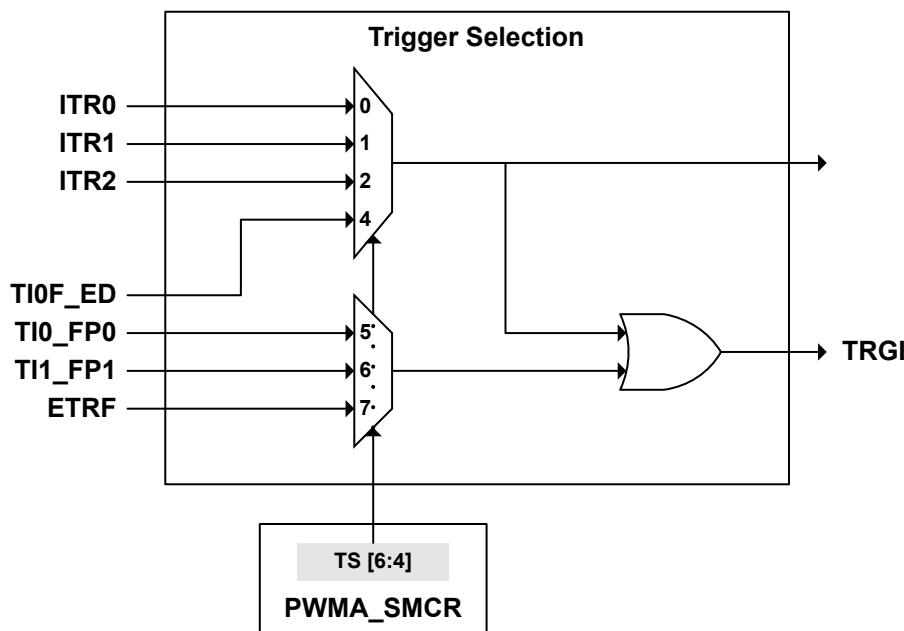


Figure 12-4 Trigger Selection block



The `TRG` 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 PWMA. Refer to the [Internal Trigger x Connection](#) section for more information.
- `TI0F_ED`, `TI0_FP0` signals from the Channel 0 Input Stage and `TI1_FP1` signal from the Channel 1 Input Stage. Refer to the [Input Stage](#) section for more information.
- `ETRF` signal from the [External Trigger Control](#) block.

#### 12.1.1.2.3. Slave Mode Controller

Slave Mode Controller is used to select the `CK_PSC` signal source and set the operation of the Timer 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 operation of the Timer slave mode are controlled via the `SMS` and `ECE` bits of the `PWMA_SMCR` register.

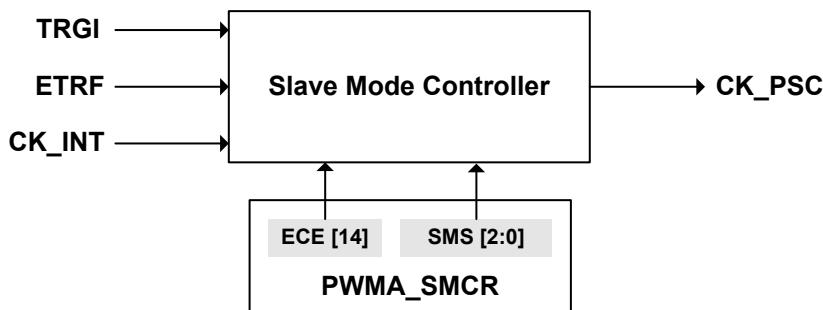


Figure 12-5 Slave Mode Controller

The following table describes the relationships between the settings of the `SMS` and `ECE` bits of the `PWMA_SMCR` register and the slave operation modes of the Timer.

Table 12-2 Timer slave operation modes

<code>PWMA_SMCR</code>		<b>Slave Operation Mode</b>	<b>CK_PSC signal source</b>
<code>SMS</code>	<code>ECE</code>		
0x0	0x0	Internal Clock Mode	CK_INT
	0x1	External Clock Mode 2	ETRF
0x4	0x0	Reset Mode The counter is controlled through the <code>TRGI</code> signal.	CK_INT
	0x1	External Clock Mode 2 + Reset Mode The counter is controlled through the <code>TRGI</code> signal	ETRF
0x5	0x0	Gated Mode The counter is controlled through the <code>TRGI</code> signal	CK_INT
	0x1	External Clock Mode 2 + Gated Mode The counter is controlled through the <code>TRGI</code> signal	ETRF
0x6	0x0	Trigger Mode The counter is controlled through the <code>TRGI</code> signal	CK_INT
	0x1	External Clock Mode 2 + Trigger Mode The counter is controlled through the <code>TRGI</code> 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

Master Mode Controller is used to generate the `TRGO` signal, where the `TRGO` signal generation is controlled through the `MMS` bit of the `PWMA_CR2` register. Simplified diagram of the Master Mode Controller is shown in the following figure.

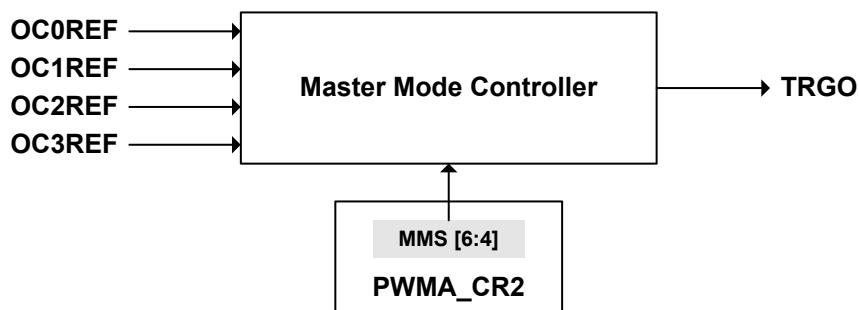


Figure 12-6 Master Mode Controller

The `TRGO` signal is used to control the Timers of the other PWMA, where each `TRGO` signal from the Timer is connected to the corresponding `ITR` input of the other Timer (refer to the [Internal Trigger x Connection](#) section for details). 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.

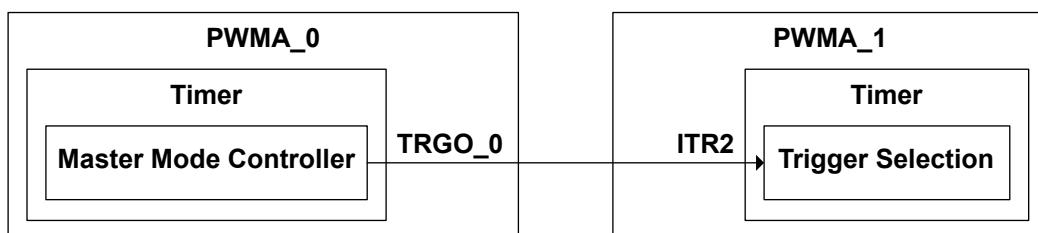


Figure 12-7 TRGO to ITR connection

### 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

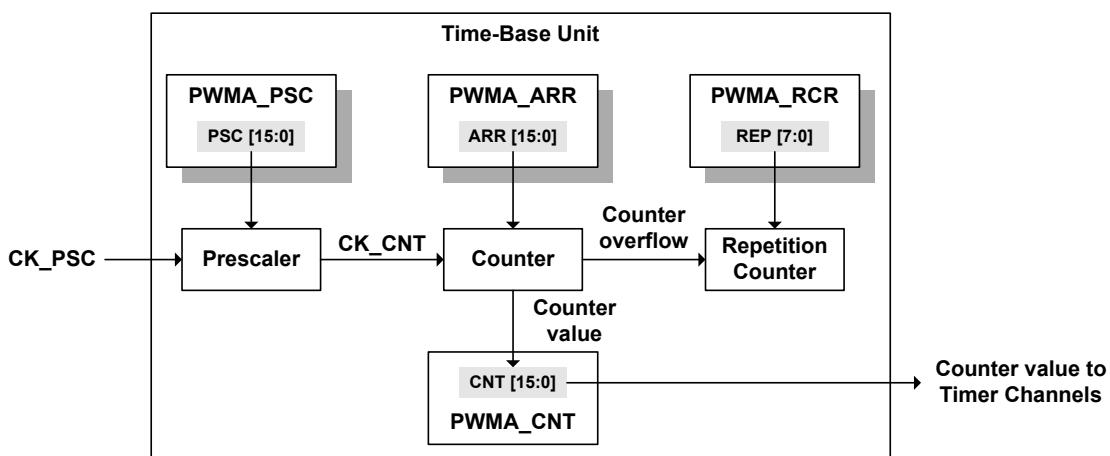


Figure 12-8 Time-base Unit

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 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 1 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. Refer to the [Counter Modes](#) section for detailed description of the UEV generation and [PWMA\\_ARR](#) usage.

#### 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 either on the overflow or on the underflow depending on when the [PWMA\\_RCR](#) register was written and when 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 starting the counter, the UEV occurs on the underflow. For example for [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, the **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`). Else the Update Event is generated at each counter overflow.

Setting the `UG` bit in the `PWMA_EGR` register (by software or by hardware in the Timer slave operation mode) also generates an Update Event.

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 `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 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 `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 buffer of the prescaler is reloaded with the preload value (content of the `PWMA_PSC` register)

Depending on the Timer settings, the occurrence 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, the **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`). Else the Update Event is generated at each counter underflow.

Setting the `UG` bit in the `PWMA_EGR` register (by software or by hardware in the Timer slave operation mode) also generates an Update Event.

When the UEV is generated, the counter restarts from the current auto-reload value, whereas 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 `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 buffer of the prescaler 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 may lead to the interrupt generation or 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.AR-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, the **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`). Else the Update Event 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 operation mode).



If the `UG` bit of the `PWMA_EGR` register is set, the counter restarts counting from 0, as well as the counter of the prescaler.

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 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 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 buffer of the prescaler 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.

Capture/Compare Channel combines two completely different devices, one of which is used when the channel operates in input mode (to capture a signal), the other in output mode (to generate a signal). Each Capture/Compare Channel is built around a **Capture/Compare Stage** with PWMA Capture/Compare Register (including a shadow register), an **Input Stage** for capture (with digital filter, multiplexing and 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.

The following figure shows the simplified diagram of the Timer Capture/Compare Channels.

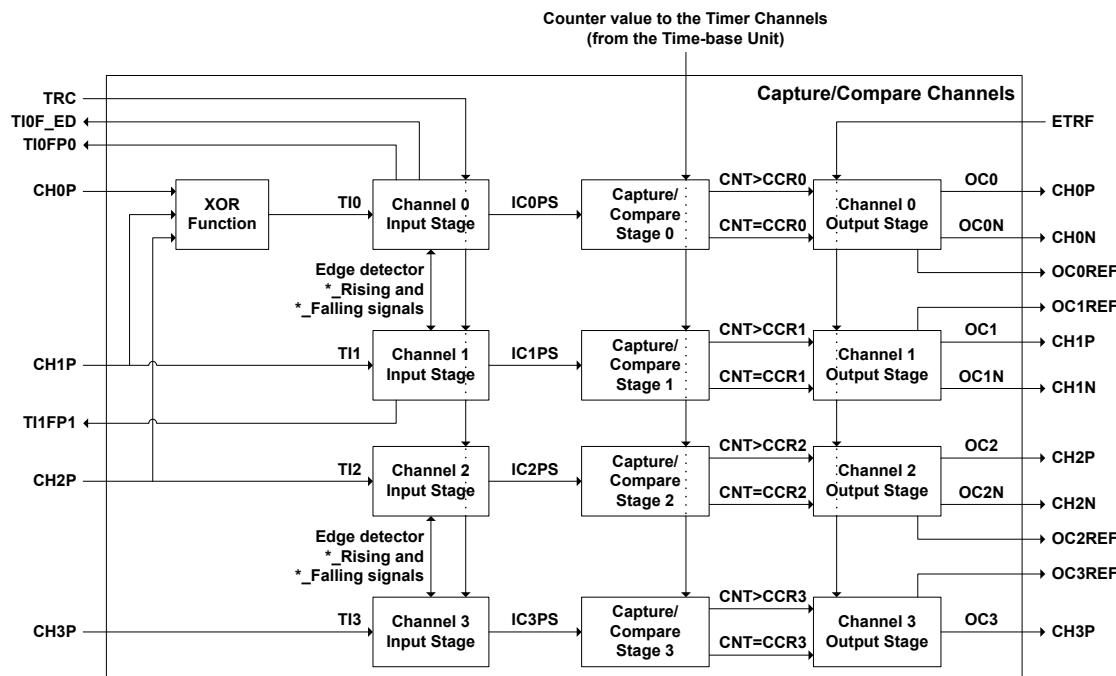


Figure 12-9 Capture/Compare Channels

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. The following figure shows the diagram of the Channel 0 Input Stage.



The main difference of 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`.

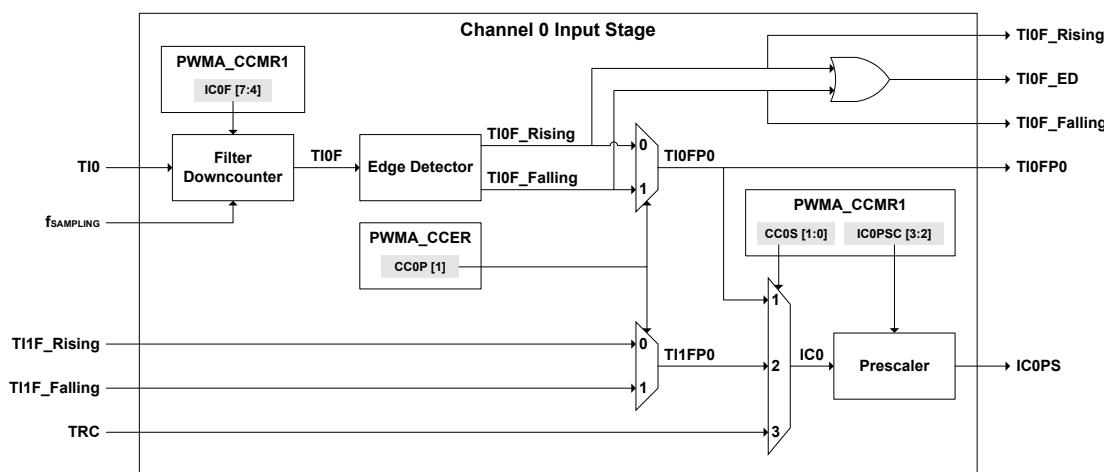


Figure 12-10 Channel 0 Input Stage

As the figure above shows, channel 0 input signal (`TI0`) is resynchronized and passed through the digital filter, which is enabled and configured through the `IC0F` bit field of the `PWMA_CCMR1` register. In this case, `TI0` signal is sampled with the sampling frequency (`fSAMPLING`) 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 `PWMA_CCER` register, allows one of the `TI0F_Rising` and `TI0F_Falling` signals to be used as the source of the `TI0FP0` signal.



It should be noted that 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 multiplexor, 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 reflect how they were received, where:



- **TI0** means that the signal was received on the channel 0 input, while **TI1** means that the signal was received on the channel 1 input;
- **F** means that the signal was resynchronized and passed through the digital filter
- **P0** shows that the active level of the signal is controlled through the channel 0 settings, i.e. **CC0P** bit of the **PWMA\_CCER** register

As shown in the figure above, one of the **TI0FP0**, **TI1FP0** and **TRC** signals can be selected as the **IC0** signal source (for more information about the **TRC** signal, refer to the [Trigger Selection](#) section). This selection provided by the **CC0S** bit field of the **PWMA\_CCMR1** register. **IC0** signal then is connected to the programmable prescaler that is controlled through the **IC0PSC** bit field of the **PWMA\_CCMR1** register. **IC0PS** signal on the prescaler output is connected to the Capture/Compare Stage 0.

### 12.1.1.5.2. Output Stage

In this section, the operation of the Output Stage will be described using the Channel 0 Output Stage as an example. The following figure shows the diagram of the Channel 0 Output Stage.



Channel 3 Output Stage differs from the Output Stages of the other channels. The main difference is the absence of the complimentary output (**OC0N**) and the Dead-time Generator. Channel 3 Output Stage diagram is given in the end of this section.

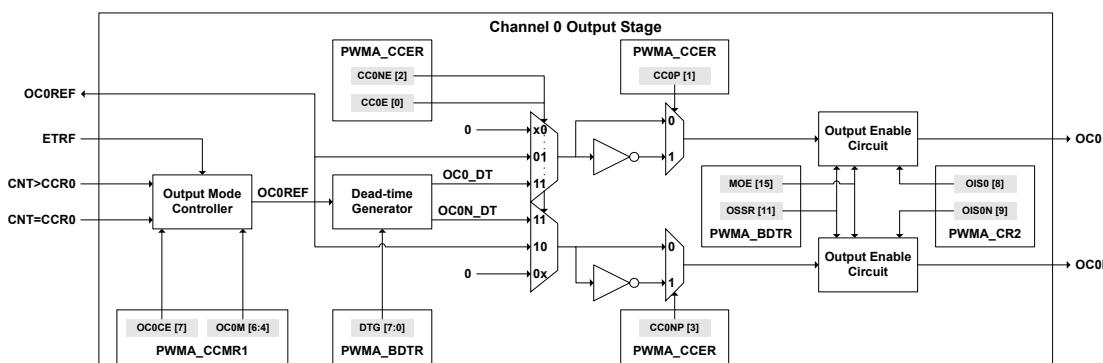


Figure 12-11 Channel 0 Output Stage

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 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 has to be adjust it depending on the devices connected to the outputs and their characteristics. For more information, refer to the [Dead-time Insertion](#) section.



**OC0\_DT** and **OC0N\_DT** signals are only used when both **CC0E** and **CC0NE** bits of the **PWMA\_CCER** register are set to **0x1**. This can be seen in the figure above by considering how the **CC0E** and **CC0NE** bits settings affect the two multiplexors 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.

You can also select the polarity of the outputs (main output `OC0` or complementary `OC0N`) independently for each output. This is done by writing to the `CC0P` and `CC0NP` bits of the `PWMA_CCER` register. After the multiplexors that control selection of polarity, signals are connected to the Output Enable Circuit.

Output Enable Circuit provides the ability to turn channel outputs on and off, as well as control the level of the disabled outputs. Channel outputs are turned on and off through the `MOE` bit of the `PWMA_BDTR` register.



- `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, what is used to turn the channel outputs to a known state even if the Timer clock is unavailable (see [Break Function](#) section for details).
- `PWMA_BDTR.MOE` bit can be set by software or, if the `PWMA_BDTR.AOE` bit is set, automatically at the next Update Event.



It should be noted that the `MOE` bit affects the state of all Timer outputs (`OCx` and `OCxN`, where `x` is the channel number).

In dependence of the `MOE` bit state, the level of the disabled outputs is controlled as follows:

- `PWMA_BDTR.MOE` = `0x1` (level is controlled through the `OSSR` bit of the `PWMA_BDTR` register and through the `CC0P` and `CC0NP` bits of 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` (level is controlled through the `CC0P` and `CC0NP` bits of the `PWMA_CCER` register and through the `OIS0` and `OISON` bits of 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, after the dead time outputs are set to their idle state that is equal to the `PWMA_CR2.OIS0` bit value for the `OC0` output and to the `PWMA_CR2.OISON` bit value for the `OC0N` output.



If the values of both bits `OIS0` and `OISON` correspond to active levels of the outputs (`OIS0 != CC0P` and `OISON != CC0NP`), then both outputs are set to inactive state that is equal to the `CC0P` and `CC0NP` bits values.



When both outputs of a channel are not used (`CC0E` = `CC0NE` = `0`), the `OIS0`, `OISON`, `CC0P` and `CC0NP` bits must be kept cleared.

The table below shows how the output control bits affect the channels with complimentary outputs (`OCx` and `OCxN`, where `x` is the channel number).

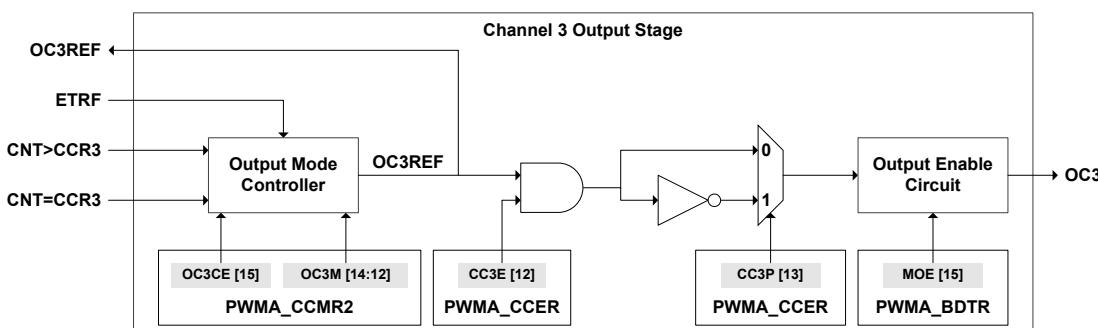
**Table 12-3 Output control bits for complementary `OCx` and `OCxN` channels**

Control Bits				Output States	
MOE	OSSR	CCxE	CCxNE	OCx output state	OCxN output state
0x1	0x0	0x0	0x0	Output Disabled ( <code>OCx=0</code> )	Output Disabled ( <code>OCxN=0</code> )

**Table 12-3 Output control bits for complementary  $OCx$  and  $OCxN$  channels (continued)**

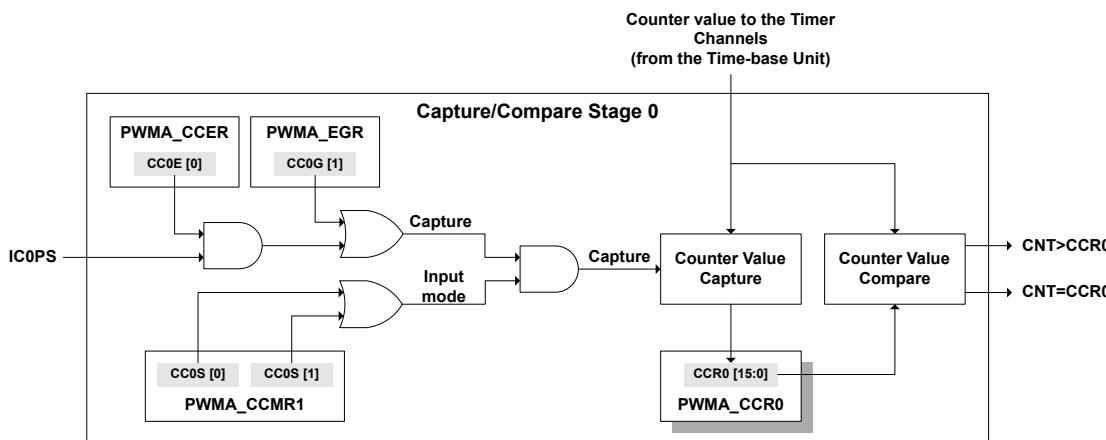
Control Bits				Output States	
0x1	0x0	0x0	0x1	Output Disabled ( $OCx=0$ )	$OCxREF + \text{Polarity}$ $OCxN = OCxREF \text{ xor } CCxNP$
		0x1	0x0	$OCxREF + \text{Polarity}$ $OCx = OCxREF \text{ xor } CCxP$	Output Disabled ( $OCxN=0$ )
		0x1	0x1	$OCxREF + \text{Polarity} + \text{dead-time}$	Complementary to $OCxREF$ (not $OCxREF$ ) + Polarity + dead-time
	0x0	0x0	0x0	Output Disabled and set to inactive state ( $OCx = CCxP$ )	Output Disabled and set to inactive state ( $OCxN = CCxNP$ )
		0x0	0x1	Output Disabled and set to inactive state ( $OCx = CCxP$ )	$OCxREF + \text{Polarity}$ $OCxN = OCxREF \text{ xor } CCxNP$
		0x1	0x0	$OCxREF + \text{Polarity}$ $OCx = OCxREF \text{ xor } CCxP$	Output Disabled and set to inactive state ( $OCxN = CCxNP$ )
		0x1	0x1	$OCREF + \text{Polarity} + \text{dead-time}$	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 $OCxN = OISxN$ )	
		0x0	0x1	 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.	
		0x1	0x0	 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.	
		0x1	0x1	 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.	

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

Capture/Compare Stage is used both when channel operates in input mode and when channel operates in output mode. The figure below shows the 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.



Channel works 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 channel works 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 to the `PWMA_CCR0` register.

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 compare with the `PWMA_CNT` register value in the Counter Value Compare block. As a result, Counter Value Compare block generates the `CNT=CCR0` and `CNT>CCR0` output signals, where:

- `CNT=CCR0` signal is generated when the `PWMA_CCR0` register value is equal to the `PWMA_CNT` register value
- `CNT>CCR0` signal is generated when the `PWMA_CNT` register value is greater than the `PWMA_CCR0` register value

The signals `CNT=CCR0` and `CNT>CCR0` are then used in the Channel 0 Output Stage.

#### 12.1.1.6. XOR Function

The `T10S` bit in the `PWMA_CR2` register allows the input of channel 0 (`T10`) to be connected to the output of a XOR gate, combining the three input pins `CH0P`, `CH1P` and `CH2P`.

Depending on the `PWMA_CR2.T10S` bit setting, the `T10` is connected as follows:

- `TIO = CH0P` if the `TI0S` bit of the `PWMA_CR2` is set to `0x0`
- `TIO = 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`, Dead-time Generator produces `OCx_DT` and `OCxN_DT` signals (here `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

Break function is designed to prevent damage of the controlled power cascades in case of the emergency situations. The break source is the BE-U1000 break input pin (`BRK`). When the break function is enabled, the channel outputs are turned to a known state even if the Timer clock is unavailable. In this case, 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 failure of the clocking system lead 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 mentioned above bits affect on the channel outputs, refer to the **Output control bits for complementary `OCx` and `OCxN` channels** table in the [Output Stage](#) section.

When exiting from reset, the break circuit is disabled and the `MOE` bit is low. 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. `BKE` and `BKP` can be modified at the same time. When the `BKE` and `BKP` bits are written, a delay of 1 Register Interface clock cycle is applied before the writing is effective. Consequently, it is necessary to wait 1 Register Interface clock period 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 status 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. Such registers are a bunch of:

- software accessible register (can also be called as preload register or buffer register)
- software inaccessible register (can also be called as shadow register or active register)

Active registers are used during the timer operations. They are updated with the values of the appropriate preload registers when the *Update Event (UEV)* occurs.

In the figures, the buffered registers are shown as rectangles with the gray background.

### 12.1.1.10. Quadrature Encoder

Quadrature Encoder is used to measure the shaft rotation angle. Quadrature Encoder is enabled by setting the `QEN` bit of the `PWMA_QESET` register to 0x1. The following figure shows the diagram of the Quadrature Encoder.

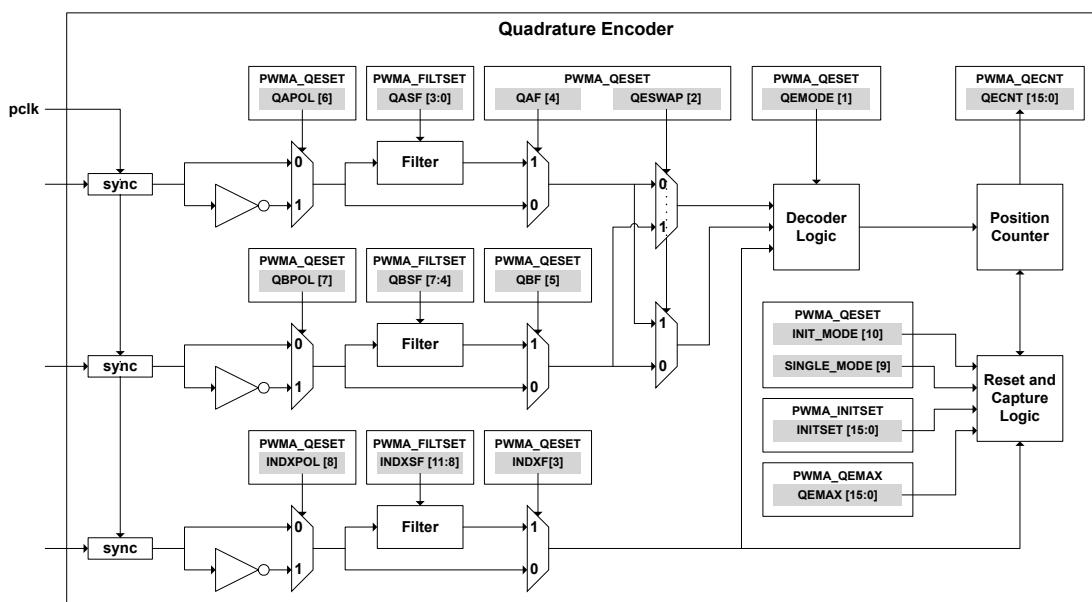


Figure 12-14 Quadrature Encoder

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 through the `PWMA_FILTSET` register. It is also possible to perform an inversion of the input signal, "bypass" the filtering and swap the `qa` and `qb` signals through the appropriate bits of the `PWMA_QESET` register.

### 12.1.1.10.1. Operation Modes

Quadrature Encoder can works 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 in dependence of the `qa` and `qb` signal phases.

Quadrature mode supports the following features:

- Can be used with single or multiple mode, where:
  - In single mode (`PWMA_QESET.SINGLE_MODE` = 0x1), the counter can count up to `PWMA_QEMAX` value and then stop, or count down to 0 and then stop.
  - In multiple mode (`PWMA_QESET.SINGLE_MODE` = 0x0), 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.
- Counter reset or counter initialization on the `indx` rising edge (see [Initialization and Reset](#) for details).

#### 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 with single or multiple mode, where:

- In single mode (`PWMA_QESET.SINGLE_MODE` = 0x1), the counter can count up to `PWMA_QEMAX` value and then stop, or count down to 0 and then stop.
- In multiple mode (`PWMA_QESET.SINGLE_MODE` = 0x0), 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.

#### 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, the following operation 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

Register regions of the PWMA controllers are mapped in the BE-U1000 microcontroller Memory Map as the following table shows.

**Table 12-4 Memory Address Regions for PWMA Controller Registers**

Region	Block Name	Size	Start Address	End Address
Periphery 2	PWMA_0	4KB	0x1400_4000	0x1400_4FFF
	PWMA_1	4KB	0x1400_5000	0x1400_5FFF
	PWMA_2	4KB	0x1400_6000	0x1400_6FFF
	PWMA_3	4KB	0x1400_7000	0x1400_7FFF



For details, refer to [Memory Map](#) chapter.

In this chapter:

- [Register Map](#)
- [Register and Field Descriptions](#)

### 12.1.2.1. Registers Map

The following table provides 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 Registers Map**

Register	Offset	Description	Default Value
PWMA Timer Registers			
PWMA_CR1	0x00	<a href="#">PWMA Control Register 1</a>	0x0
PWMA_CR2	0x04	<a href="#">PWMA Control Register 2</a>	0x0
PWMA_SMCR	0x08	<a href="#">PWMA Slave Mode Control Register</a>	0x0
PWMA_DIER	0x0C	<a href="#">PWMA DMA/Interrupt Enable Register</a>	0x0
PWMA_SR	0x10	<a href="#">PWMA Status Register</a>	0x0
PWMA_EGR	0x14	<a href="#">PWMA Event Generation Register</a>	0x0
PWMA_CCMR1	0x18	<a href="#">PWMA Capture/Compare Mode Register 1</a>	0x0
PWMA_CCMR2	0x1C	<a href="#">PWMA Capture/Compare Mode Register 2</a>	0x0
PWMA_CCER	0x20	<a href="#">PWMA Capture/Compare Enable Register</a>	0x0
PWMA_CNT	0x24	<a href="#">PWMA Counter</a>	0x0
PWMA_PSC	0x28	<a href="#">PWMA Prescaler</a>	0x0
PWMA_ARR	0x2C	<a href="#">PWMA Auto-reload Register</a>	0x0

**Table 12-5 PWMA Registers Map (continued)**

Register	Offset	Description	Default Value
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
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 of a register description 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	<p>Clock division            This bit-field indicates the division ratio between the timer clock period (<math>t_{CK\_INT}</math>) and period of the dead-time and sampling clock (<math>t_{DTS}</math>). The dead-time and sampling clock is used by the dead-time generators and the digital filters (ETR, TIx).</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <math>t_{DTS}=t_{CK\_INT}</math></li> <li><b>0x1:</b> <math>t_{DTS}=2*t_{CK\_INT}</math></li> </ul>

**Table 12-6 Fields For Register: PWMA\_CR1 (continued)**

Bits	Name	R/W	Description
			<p><b>0x2:</b> <math>t_{DTS}=4*t_{CK\_INT}</math></p> <p><b>0x3:</b> Reserved</p> <p><b>Value After Reset:</b> 0x0</p>
[7]	ARPE	R/W	<p>Auto-reload preload enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> PWMA_ARR register is not buffered</li> <li><b>0x1:</b> PWMA_ARR register is buffered</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[6:5]	CMS	R/W	<p>Center-aligned mode selection</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Edge-aligned mode. The counter counts up or down depending on the direction bit (DIR).</li> <li><b>0x1:</b> Center-aligned mode 1. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (<math>CCxS=0x0</math> in PWMA_CCMRx registers) are set only when the counter is counting down.</li> <li><b>0x2:</b> Center-aligned mode 2. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (<math>CCxS=0x0</math> in PWMA_CCMRx registers) are set only when the counter is counting up.</li> <li><b>0x3:</b> Center-aligned mode 3. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (<math>CCxS=0x0</math> in PWMA_CCMRx registers) are set both when the counter is counting up or down.</li> </ul> <p> It is not allowed to switch from Edge-aligned mode to Center-aligned mode as long as the counter is enabled (<math>CEN=0x1</math>).</p> <p><b>Value After Reset:</b> 0x0</p>
[4]	DIR	R/W	<p>Direction</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Counter used as upcounter</li> <li><b>0x1:</b> Counter used as downcounter</li> </ul> <p> This bit is read only when the timer is configured in Center-aligned mode or Encoder mode.</p> <p><b>Value After Reset:</b> 0x0</p>
[3]	OPM	R/W	<p>One pulse mode</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Counter is not stopped at the Update Event</li> <li><b>0x1:</b> Counter stops counting at the next Update Event (clearing the CEN bit)</li> </ul>

**Table 12-6 Fields For Register: PWMA\_CR1 (continued)**

Bits	Name	R/W	Description
			<b>Value After Reset:</b> 0x0
[2]			Reserved (return 0 when read)
[1]	UDIS	R/W	<p>Update disable This bit is set and cleared by software to enable/disable <i>Update Event (UEV)</i> generation.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> UEV enabled. The Update event is generated by one of the following events: <ul style="list-style-type: none"> <li>◦ counter overflow/underflow;</li> <li>◦ setting the <code>PWMA_EGR.UG</code> bit;</li> <li>◦ update generation through the slave mode controller.</li> </ul> </li> <li><b>0x1:</b> UEV disabled. The Update event is not generated, shadow registers keep their value (<code>ARR</code>, <code>PSC</code>, <code>CCRx</code>). However the counter and the prescaler are reinitialized if the <code>PWMA_EGR.UG</code> bit is set or if a hardware reset is received from the slave mode controller.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[0]	CEN	R/W	<p>Counter enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Counter disabled</li> <li><b>0x1:</b> Counter enabled</li> </ul> <p> • External clock, gated mode and encoder mode can work only if the <code>CEN</code> bit has been previously set by software. However trigger mode can set the <code>CEN</code> bit automatically by hardware.</p> <ul style="list-style-type: none"> <li>• <code>CEN</code> is reset by hardware in one-pulse mode.</li> <li>• Counter starts counting 1 clock cycle after setting the <code>CEN</code> bit.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

#### 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	<p>Output Idle state Channel 2 (<code>OC2N</code> output)</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <code>OC2N=0</code> after a dead-time when <code>PWMA_BDTR.MOE=0x0</code></li> <li><b>0x1:</b> <code>OC2N=1</code> after a dead-time when <code>PWMA_BDTR.MOE=0x0</code></li> </ul>

**Table 12-7 Fields For Register: PWMA\_CR2 (continued)**

Bits	Name	R/W	Description
			<p><b>!</b> If the settings of the <code>OIS2</code> and <code>OIS2N</code> bits correspond to the active levels at the outputs (i.e. <code>OIS2</code> = not <code>CC2P</code> and <code>OIS2N</code> = not <code>CC2NP</code>), then in the idle state both specified outputs will be set to inactive level (i.e. <code>OC2</code> = <code>CC2P</code> and <code>OC2N</code> = <code>CC2NP</code>). It means that the outputs state may not be equal to the given ones.</p> <p><b>Value After Reset:</b> 0x0</p>
[12]	<code>OIS2</code>	R/W	<p>Output Idle state Channel 2 (<code>OC2</code> output)</p> <p><b>Values:</b></p> <p><b>0x0:</b> <code>OC2=0</code> (after a dead-time if <code>OC2N</code> is implemented) when <code>PWMA_BDTR.MOE=0x0</code></p> <p><b>0x1:</b> <code>OC2=1</code> (after a dead-time if <code>OC2N</code> is implemented) when <code>PWMA_BDTR.MOE=0x0</code></p> <p><b>!</b> Output state in the idle mode may not match the <code>OIS2</code> bit setting. See <code>OIS2N</code> bit note for details.</p> <p><b>Value After Reset:</b> 0x0</p>
[11]	<code>OIS1N</code>	R/W	<p>Output Idle state Channel 1 (<code>OC1N</code> output)</p> <p><b>Values:</b></p> <p><b>0x0:</b> <code>OC1N=0</code> after a dead-time when <code>PWMA_BDTR.MOE=0x0</code></p> <p><b>0x1:</b> <code>OC1N=1</code> after a dead-time when <code>PWMA_BDTR.MOE=0x0</code></p> <p><b>!</b> If the settings of the <code>OIS1</code> and <code>OIS1N</code> bits correspond to the active levels at the outputs (i.e. <code>OIS1</code> = not <code>CC1P</code> and <code>OIS1N</code> = not <code>CC1NP</code>), then in the idle state both specified outputs will be set to inactive level (i.e. <code>OC1</code> = <code>CC1P</code> and <code>OC1N</code> = <code>CC1NP</code>). It means that the outputs state may not be equal to the given ones.</p> <p><b>Value After Reset:</b> 0x0</p>
[10]	<code>OIS1</code>	R/W	<p>Output Idle state Channel 1 (<code>OC1</code> output)</p> <p><b>Values:</b></p> <p><b>0x0:</b> <code>OC1=0</code> (after a dead-time if <code>OC1N</code> is implemented) when <code>PWMA_BDTR.MOE=0x0</code></p> <p><b>0x1:</b> <code>OC1=1</code> (after a dead-time if <code>OC1N</code> is implemented) when <code>PWMA_BDTR.MOE=0x0</code></p> <p><b>!</b> Output state in the idle mode may not match the <code>OIS1</code> bit setting. See <code>OIS1N</code> bit note for details.</p> <p><b>Value After Reset:</b> 0x0</p>
[9]	<code>OIS0N</code>	R/W	<p>Output Idle state Channel 0 (<code>OC0N</code> output)</p> <p><b>Values:</b></p> <p><b>0x0:</b> <code>OC0N=0</code> after a dead-time when <code>PWMA_BDTR.MOE=0x0</code></p> <p><b>0x1:</b> <code>OC0N=1</code> after a dead-time when <code>PWMA_BDTR.MOE=0x0</code></p> <p><b>!</b> If the settings of the <code>OIS0</code> and <code>OIS0N</code> bits correspond to the active levels at the outputs (i.e. <code>OIS0</code> = not <code>CC0P</code> and <code>OIS0N</code> = not <code>CC0NP</code>), then in the idle state both specified outputs will be set to inactive level (i.e. <code>OC0</code> = <code>CC0P</code> and <code>OC0N</code> = <code>CC0NP</code>). It means that the outputs state may not be equal to the given ones.</p>

**Table 12-7 Fields For Register: PWMA\_CR2 (continued)**

Bits	Name	R/W	Description
			<p><b>⚠</b> 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 be equal to the given ones.</p> <p><b>Value After Reset:</b> 0x0</p>
[8]	OIS0	R/W	<p>Output Idle state Channel 0 (OC0 output)</p> <p><b>Values:</b></p> <p><b>0x0:</b> OC0=0 (after a dead-time if OC0N is implemented) when PWMA_BDTR.MOE=0x0</p> <p><b>0x1:</b> OC1=1 (after a dead-time if OC1N is implemented) when PWMA_BDTR.MOE=0x0</p> <p><b>⚠</b> Output state in the idle mode may not match the OIS0 bit setting. See OISON bit note for details.</p> <p><b>Value After Reset:</b> 0x0</p>
[7]	TIO0	R/W	<p>TIO selection</p> <p><b>Values:</b></p> <p><b>0x0:</b> The CH0P pin is connected to the TIO input</p> <p><b>0x1:</b> The CH0P, CH1P and CH2P pins are connected to the TIO input (XOR combination; TIO = CH0P xor CH1P xor CH2P)</p> <p><b>⚠</b> 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.</p> <p><b>Value After Reset:</b> 0x0</p>
[6:4]	MMS	R/W	<p>Master mode selection</p> <p>These bits allow to select the information to be sent in master mode to slave timers for synchronization (TRGO). The combination is as follows:</p> <p><b>Values:</b></p> <p><b>0x0:</b> Reset - the UG bit from the PWMA_EGR register is used as trigger output (TRGO). If the reset is generated by the trigger input (slave mode controller configured in reset mode) then the signal on TRGO is delayed compared to the actual reset.</p> <p><b>0x1:</b> 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 a window in which a slave timer is enable.</p> <p><b>0x2:</b> Update - The update event is selected as trigger output (TRGO). For instance a master timer can then be used as a prescaler for a slave timer.</p> <p><b>0x3:</b> Compare Pulse - The trigger output send a positive pulse when the PWMA_SR.CC0IF flag is to be set (even if it was already high), as soon as a capture or a compare match occurred.</p> <p><b>0x4:</b> Compare - OC0REF signal is used as trigger output (TRGO)</p> <p><b>0x5:</b> Compare - OC1REF signal is used as trigger output (TRGO)</p>

**Table 12-7 Fields For Register: PWMA\_CR2 (continued)**

Bits	Name	R/W	Description
			<p><b>0x6:</b> Compare - OC2REF signal is used as trigger output (TRGO)  <b>0x7:</b> Compare - OC3REF signal is used as trigger output (TRGO)</p>  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.
			<b>Value After Reset:</b> 0x0
[3]			Reserved (return 0 when read)
[2]	CCUS	R/W	<p>Capture/compare control update selection</p> <p><b>Values:</b></p> <p><b>0x0:</b> When capture/compare control bits (CCxE, CCxNE and OCxM) are preloaded (CCPC=0x1), they are updated by setting the COMG bit only</p> <p><b>0x1:</b> 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</p>  This bit acts only on channels that have a complementary output.
			<b>Value After Reset:</b> 0x0
[1]			Reserved (return 0 when read)
[0]	CCPC	R/W	<p>Capture/compare preloaded control</p> <p><b>Values:</b></p> <p><b>0x0:</b> CCxE, CCxNE and OCxM bits are not preloaded</p> <p><b>0x1:</b> CCxE, CCxNE and OCxM bits are preloaded, after having been written, they are updated only when a <i>Commutation Event (COM)</i> occurs (PWMA_EGR.COMG bit set or rising edge detected on TRGI, depending on the CCUS bit).</p>  This bit acts only on channels that have a complementary output.
			<b>Value After Reset:</b> 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	<p>External trigger polarity</p> <p>This bit selects whether ETR or ETR (inverted ETR) is used for trigger operations</p>

**Table 12-8 Fields For Register: PWMA\_SMCR (continued)**

Bits	Name	R/W	Description
			<p><b>Values:</b></p> <p><b>0x0:</b> ETR is non-inverted, active at high level or rising edge</p> <p><b>0x1:</b> ETR is inverted, active at low level or falling edge</p> <p><b>Value After Reset:</b> 0x0</p>
[14]	ECE	R/W	<p>External clock enable</p> <p>This bit enables External clock mode 2.</p> <p><b>Values:</b></p> <p><b>0x0:</b> External clock mode 2 disabled.</p> <p><b>0x1:</b> External clock mode 2 enabled. The counter is clocked by any active edge on the ETRF signal.</p> <p> 1. Setting the ECE bit has the same effect as selecting external clock mode 1 with TRGI connected to ETRF (SMS=0x7 and TS=0x7). 2. It is possible to simultaneously use external clock mode 2 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 external clock mode 1 and external clock mode 2 are enabled at the same time, the external clock input is ETRF.</p> <p><b>Value After Reset:</b> 0x0</p>
[13:12]	ETPS	R/W	<p>External trigger prescaler</p> <p>External trigger signal ETRP frequency must be at most 1/4 of timer clock frequency (CK_INT). A prescaler can be enabled to reduce ETRP frequency. It is useful when inputting fast external clocks.</p> <p><b>Values:</b></p> <p><b>0x0:</b> Prescaler OFF</p> <p><b>0x1:</b> ETRP frequency divided by 2</p> <p><b>0x2:</b> ETRP frequency divided by 4</p> <p><b>0x3:</b> ETRP frequency divided by 8</p> <p><b>Value After Reset:</b> 0x0</p>
[11:8]	ETF	R/W	<p>External trigger filter</p> <p>This bit-field then defines the frequency used to sample ETRP signal and the length of the digital filter applied to ETRP. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output.</p> <p><b>Values:</b></p> <p><b>0x0:</b> No filter, sampling is done at fDTS</p> <p><b>0x1:</b> fSAMPLING=fCK_INT, N=2</p> <p><b>0x2:</b> fSAMPLING=fCK_INT, N=4</p> <p><b>0x3:</b> fSAMPLING=fCK_INT, N=8</p> <p><b>0x4:</b> fSAMPLING=fDTS/2, N=6</p> <p><b>0x5:</b> fSAMPLING=fDTS/2, N=8</p>

**Table 12-8 Fields For Register: PWMA\_SMCR (continued)**

Bits	Name	R/W	Description
			<p><b>0x6:</b> fSAMPLING=fDTS/4, N=6  <b>0x7:</b> fSAMPLING=fDTS/4, N=8  <b>0x8:</b> fSAMPLING=fDTS/8, N=6  <b>0x9:</b> fSAMPLING=fDTS/8, N=8  <b>0xA:</b> fSAMPLING=fDTS/16, N=5  <b>0xB:</b> fSAMPLING=fDTS/16, N=6  <b>0xC:</b> fSAMPLING=fDTS/16, N=8  <b>0xD:</b> fSAMPLING=fDTS/32, N=5  <b>0xE:</b> fSAMPLING=fDTS/32, N=6  <b>0xF:</b> fSAMPLING=fDTS/32, N=8</p> <p><b>Value After Reset:</b> 0x0</p>
[7]			Reserved
[6:4]	TS	R/W	<p>Trigger selection  This bit-field selects the trigger input to be used to synchronize the counter.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Internal Trigger 0 (ITR0)</li> <li><b>0x1:</b> Internal Trigger 1 (ITR1)</li> <li><b>0x2:</b> Internal Trigger 2 (ITR2)</li> <li><b>0x3:</b> Reserved</li> <li><b>0x4:</b> TI1 Edge Detector (TI0F_ED)</li> <li><b>0x5:</b> Filtered Timer Input 1 (TI0FP0)</li> <li><b>0x6:</b> Filtered Timer Input 2 (TI1FP1)</li> <li><b>0x7:</b> External Trigger input (ETRF)</li> </ul> <p> These bits must be changed only when they are not used (e.g. when SMS=0x0) to avoid wrong edge detections at the transition.</p> <p><b>Value After Reset:</b> 0x0</p>
[3]			Reserved
[2:0]	SMS	R/W	<p>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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Slave mode disabled - if PWMA_CR1.CEN = 0x1 then the prescaler is clocked directly by the internal clock.</li> <li><b>0x1:</b> Reserved</li> <li><b>0x2:</b> Reserved</li> <li><b>0x3:</b> Reserved</li> <li><b>0x4:</b> Reset Mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter and generates an update of the registers.</li> </ul>

**Table 12-8 Fields For Register: PWMA\_SMCR (continued)**

Bits	Name	R/W	Description
			<p><b>0x5:</b> Gated Mode - The counter clock is enabled when the trigger input (<code>TRGI</code>) 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.</p> <p><b>0x6:</b> Trigger Mode - The counter starts at a rising edge of the trigger <code>TRGI</code> (but it is not reset). Only the start of the counter is controlled.</p> <p><b>0x7:</b> External Clock Mode 1 - Rising edges of the selected trigger (<code>TRGI</code>) clock the counter.</p> <p> The gated mode must not be used if <code>TIOF_ED</code> is selected as the trigger input (<code>TS=0x4</code>). Indeed, <code>TIOF_ED</code> outputs 1 pulse for each transition on <code>TIOF</code>, whereas the gated mode checks the level of the trigger signal.</p> <p><b>Value After Reset:</b> 0x0</p>

#### 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	<p>Interrupt clear</p> <p>Write 0x1 to this bit to reset the setted interrupt. Writing 0x0 has no effect.</p> <p><b>Value After Reset:</b> 0x0</p>
[14]	TDE	R/W	<p>Trigger DMA request enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Trigger DMA request disabled</li> <li><b>0x1:</b> Trigger DMA request enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[13]	COMDE	R/W	<p>COM DMA request enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> COM DMA request disabled</li> <li><b>0x1:</b> COM DMA request enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[12]	CC3DE	R/W	<p>Capture/Compare Channel 3 DMA request enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> CC3 DMA request disabled</li> <li><b>0x1:</b> CC3 DMA request enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

**Table 12-9 Fields For Register: PWMA\_DIER (continued)**

Bits	Name	R/W	Description
[11]	CC2DE	R/W	<p>Capture/Compare Channel 2 DMA request enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x0: CC2 DMA request disabled</li> <li>0x1: CC2 DMA request enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[10]	CC1DE	R/W	<p>Capture/Compare Channel 1 DMA request enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x0: CC1 DMA request disabled</li> <li>0x1: CC1 DMA request enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[9]	CC0DE	R/W	<p>Capture/Compare Channel 0 DMA request enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x0: CC0 DMA request disabled</li> <li>0x1: CC0 DMA request enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[8]	UDE	R/W	<p>Update DMA request enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x0: Update DMA request disabled</li> <li>0x1: Update DMA request enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[7]	BIE	R/W	<p>Break interrupt enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x0: Break interrupt disabled</li> <li>0x1: Break interrupt enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[6]	TIE	R/W	<p>Trigger interrupt enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x0: Trigger interrupt disabled</li> <li>0x1: Trigger interrupt enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[5]	COMIE	R/W	<p>COM interrupt enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x0: COM interrupt disabled</li> <li>0x1: COM interrupt enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[4]	CC3IE	R/W	<p>Capture/Compare Channel 3 interrupt enable</p> <p><b>Values:</b></p>

**Table 12-9 Fields For Register: PWMA\_DIER (continued)**

Bits	Name	R/W	Description
			<p><b>0x0:</b> CC3 interrupt disabled  <b>0x1:</b> CC3 interrupt enabled</p> <p><b>Value After Reset:</b> 0x0</p>
[3]	CC2IE	R/W	<p>Capture/Compare Channel 2 interrupt enable</p> <p><b>Values:</b></p> <p><b>0x0:</b> CC2 interrupt disabled  <b>0x1:</b> CC2 interrupt enabled</p> <p><b>Value After Reset:</b> 0x0</p>
[2]	CC1IE	R/W	<p>Capture/Compare Channel 1 interrupt enable</p> <p><b>Values:</b></p> <p><b>0x0:</b> CC1 interrupt disabled  <b>0x1:</b> CC1 interrupt enabled</p> <p><b>Value After Reset:</b> 0x0</p>
[1]	CC0IE	R/W	<p>Capture/Compare Channel 0 interrupt enable</p> <p><b>Values:</b></p> <p><b>0x0:</b> CC0 interrupt disabled  <b>0x1:</b> CC0 interrupt enabled</p> <p><b>Value After Reset:</b> 0x0</p>
[0]	UIE	R/W	<p>Update interrupt enable</p> <p><b>Values:</b></p> <p><b>0x0:</b> Update interrupt disabled  <b>0x1:</b> Update interrupt enabled</p> <p><b>Value After Reset:</b> 0x0</p>

#### 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	<p>Capture/Compare Channel 3 overcapture flag</p> <p>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.</p> <p><b>Values:</b></p>

**Table 12-10 Fields For Register: PWMA\_SR (continued)**

Bits	Name	R/W	Description
			<p><b>0x0:</b> No overcapture has been detected</p> <p><b>0x1:</b> The counter value has been captured in <a href="#">PWMA_CCR3</a> register while <a href="#">CC3IF</a> flag was already set</p> <p><b>Value After Reset:</b> 0x0</p>
[11]	CC2OF	RW0C	<p>Capture/Compare Channel 2 overcapture flag</p> <p>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.</p> <p><b>Values:</b></p> <p><b>0x0:</b> No overcapture has been detected</p> <p><b>0x1:</b> The counter value has been captured in <a href="#">PWMA_CCR2</a> register while <a href="#">CC2IF</a> flag was already set</p> <p><b>Value After Reset:</b> 0x0</p>
[10]	CC1OF	RW0C	<p>Capture/Compare Channel 1 overcapture flag</p> <p>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.</p> <p><b>Values:</b></p> <p><b>0x0:</b> No overcapture has been detected</p> <p><b>0x1:</b> The counter value has been captured in <a href="#">PWMA_CCR1</a> register while <a href="#">CC1IF</a> flag was already set</p> <p><b>Value After Reset:</b> 0x0</p>
[9]	CC0OF	RW0C	<p>Capture/Compare Channel 0 overcapture flag</p> <p>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.</p> <p><b>Values:</b></p> <p><b>0x0:</b> No overcapture has been detected</p> <p><b>0x1:</b> The counter value has been captured in <a href="#">PWMA_CCR0</a> register while <a href="#">CC0IF</a> flag was already set</p> <p><b>Value After Reset:</b> 0x0</p>
[8]			Reserved
[7]	BIF	RW0C	<p>Break interrupt flag</p> <p>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.</p> <p><b>Values:</b></p> <p><b>0x0:</b> No break event occurred</p> <p><b>0x1:</b> An active level has been detected on the break input</p> <p><b>Value After Reset:</b> 0x0</p>
[6]	TIF	RW0C	Trigger interrupt flag

**Table 12-10 Fields For Register: [PWMA\\_SR](#) (continued)**

Bits	Name	R/W	Description
			<p>This flag is set by hardware on trigger event (active edge detected on <a href="#">TRGI</a> input) when the slave mode controller is enabled in all modes but gated mode. It is set when the counter starts or stops when gated mode is selected. It is cleared by software.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> No trigger event occurred</li> <li><b>0x1:</b> Trigger interrupt pending</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[5]	<a href="#">COMIF</a>	RW0C	<p>COM interrupt flag</p> <p>This flag is set by hardware on COM event (when Capture/compare Control bits (<a href="#">CCxE</a>, <a href="#">CCxNE</a>, <a href="#">OCxM</a>) have been updated). It is cleared by software.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> No COM event occurred</li> <li><b>0x1:</b> COM interrupt pending</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[4]	<a href="#">CC3IF</a>	RW0C	<p>Capture/Compare Channel 3 interrupt flag</p> <p><b>If channel CC3 is configured as output</b> (<a href="#">PWMA_CCMR2.CC3S</a> = 0x0):</p> <p>This flag is set by hardware when the counter matches the compare value, with some exception in center-aligned mode (refer to the <a href="#">CMS</a> bits in the <a href="#">PWMA_CR1</a> register description). It is cleared by software.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> No match</li> <li><b>0x1:</b> The content of the counter <a href="#">PWMA_CNT</a> matches the content of the <a href="#">PWMA_CCR3</a> register. When the contents of <a href="#">PWMA_CCR3</a> are greater than the contents of <a href="#">PWMA_ARR</a>, the <a href="#">CC3IF</a> bit goes high on the counter overflow (in upcounting and up/down-counting modes) or underflow (in downcounting mode)</li> </ul> <p><b>If channel CC3 is configured as input</b> (<a href="#">PWMA_CCMR2.CC3S</a> != 0x0):</p> <p>This bit is set by hardware on a capture. It is cleared by software or by reading the <a href="#">PWMA_CCR3</a> register.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> No input capture occurred</li> <li><b>0x1:</b> The counter value has been captured in <a href="#">PWMA_CCR3</a> register (An edge has been detected on <a href="#">IC3</a> which matches the selected polarity)</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[3]	<a href="#">CC2IF</a>	RW0C	<p>Capture/Compare Channel 2 interrupt flag</p> <p><b>If channel CC2 is configured as output</b> (<a href="#">PWMA_CCMR2.CC2S</a> = 0x0):</p> <p>This flag is set by hardware when the counter matches the compare value, with some exception in center-aligned mode (refer to the <a href="#">CMS</a> bits in the <a href="#">PWMA_CR1</a> register description). It is cleared by software.</p>

**Table 12-10 Fields For Register: PWMA\_SR (continued)**

Bits	Name	R/W	Description
			<p><b>Values:</b></p> <p><b>0x0:</b> No match</p> <p><b>0x1:</b> The content of the counter <code>PWMA_CNT</code> matches the content of the <code>PWMA_CCR2</code> register. When the contents of <code>PWMA_CCR2</code> are greater than the contents of <code>PWMA_ARR</code>, the <code>CC2IF</code> bit goes high on the counter overflow (in upcounting and up/down-counting modes) or underflow (in downcounting mode)</p> <p><b>If channel CC2 is configured as input (<code>PWMA_CCMR2.cc2s != 0x0</code>):</b> This bit is set by hardware on a capture. It is cleared by software or by reading the <code>PWMA_CCR2</code> register.</p> <p><b>Values:</b></p> <p><b>0x0:</b> No input capture occurred</p> <p><b>0x1:</b> The counter value has been captured in <code>PWMA_CCR2</code> register (An edge has been detected on <code>IC2</code> which matches the selected polarity)</p> <p><b>Value After Reset:</b> 0x0</p>
[2]	CC1IF	RW0C	<p>Capture/Compare Channel 1 interrupt flag</p> <p><b>If channel CC1 is configured as output (<code>PWMA_CCMR1.cc1s = 0x0</code>):</b> This flag is set by hardware when the counter matches the compare value, with some exception in center-aligned mode (refer to the <code>CMS</code> bits in the <code>PWMA_CR1</code> register description). It is cleared by software.</p> <p><b>Values:</b></p> <p><b>0x0:</b> No match</p> <p><b>0x1:</b> The content of the counter <code>PWMA_CNT</code> matches the content of the <code>PWMA_CCR1</code> register. When the contents of <code>PWMA_CCR1</code> are greater than the contents of <code>PWMA_ARR</code>, the <code>CC1IF</code> bit goes high on the counter overflow (in upcounting and up/down-counting modes) or underflow (in downcounting mode)</p> <p><b>If channel CC1 is configured as input (<code>PWMA_CCMR1.cc1s != 0x0</code>):</b> This bit is set by hardware on a capture. It is cleared by software or by reading the <code>PWMA_CCR1</code> register.</p> <p><b>Values:</b></p> <p><b>0x0:</b> No input capture occurred</p> <p><b>0x1:</b> The counter value has been captured in <code>PWMA_CCR1</code> register (An edge has been detected on <code>IC1</code> which matches the selected polarity)</p> <p><b>Value After Reset:</b> 0x0</p>
[1]	CC0IF	RW0C	<p>Capture/Compare Channel 0 interrupt flag</p> <p><b>If channel CC0 is configured as output (<code>PWMA_CCMR1.cc0s = 0x0</code>):</b> This flag is set by hardware when the counter matches the compare value, with some exception in center-aligned mode (refer to the <code>CMS</code> bits in the <code>PWMA_CR1</code> register description). It is cleared by software.</p> <p><b>Values:</b></p>

**Table 12-10 Fields For Register: PWMA\_SR (continued)**

Bits	Name	R/W	Description
			<p><b>0x0:</b> No match</p> <p><b>0x1:</b> The content of the counter <a href="#">PWMA_CNT</a> matches the content of the <a href="#">PWMA_CCR0</a> register. When the contents of <a href="#">PWMA_CCR0</a> are greater than the contents of <a href="#">PWMA_ARR</a>, the <a href="#">CC0IF</a> bit goes high on the counter overflow (in upcounting and up/down-counting modes) or underflow (in downcounting mode)</p> <p><b>If channel CC0 is configured as input (<a href="#">PWMA_CCMR1.cc0s</a> != 0x0):</b> This bit is set by hardware on a capture. It is cleared by software or by reading the <a href="#">PWMA_CCR0</a> register.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> No input capture occurred</li> <li><b>0x1:</b> The counter value has been captured in <a href="#">PWMA_CCR0</a> register (An edge has been detected on <a href="#">IC0</a> which matches the selected polarity)</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[0]	UIF	RW0C	<p>Update interrupt flag This bit is set by hardware on an update event. It is cleared by software.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> No update occurred</li> <li><b>0x1:</b> Update interrupt pending. This bit is set by hardware when the registers are updated:           <ul style="list-style-type: none"> <li>◦ at overflow or underflow regarding the repetition counter value (update if repetition counter = 0), if <a href="#">UDIS</a>=0x0 in the <a href="#">PWMA_CR1</a> register;</li> <li>◦ when CNT is reinitialized by software using the <a href="#">UG</a> bit in <a href="#">PWMA_EGR</a> register, if <a href="#">UDIS</a>=0x0 in the <a href="#">PWMA_CR1</a> register;</li> <li>◦ when CNT is reinitialized by a trigger event (see also <a href="#">PWMA_SMCR</a>), if <a href="#">UDIS</a>=0x0 in the <a href="#">PWMA_CR1</a> register.</li> </ul> </li> </ul> <p><b>Value After Reset:</b> 0x0</p>

#### 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	<p>Break generation This bit is set by software in order to generate an event, it is automatically cleared by hardware.</p> <p><b>Values:</b></p>

**Table 12-11 Fields For Register: PWMA\_EGR (continued)**

Bits	Name	R/W	Description
			<p><b>0x0:</b> No action</p> <p><b>0x1:</b> A break event is generated. <a href="#">PWMA_BDTR.MOE</a> bit is cleared and <a href="#">PWMA_SR.BIF</a> flag is set. Related interrupt or DMA transfer can occur if enabled.</p> <p><b>Value After Reset:</b> 0x0</p>
[6]	TG	W	<p>Trigger generation This bit is set by software in order to generate an event, it is automatically cleared by hardware.</p> <p><b>Values:</b></p> <p><b>0x0:</b> No action</p> <p><b>0x1:</b> The <a href="#">TIF</a> flag is set in <a href="#">PWMA_SR</a> register. Related interrupt or DMA transfer can occur if enabled.</p> <p><b>Value After Reset:</b> 0x0</p>
[5]	COMG	W	<p>Capture/Compare control update generation This bit can be set by software, it is automatically cleared by hardware</p> <p><b>Values:</b></p> <p><b>0x0:</b> No action</p> <p><b>0x1:</b> When <a href="#">PWMA_CR2.CCPC</a> bit is set, it allows to update <a href="#">CCxE</a>, <a href="#">CCxNE</a> and <a href="#">OCxM</a> bits</p> <p> This bit acts only on channels having a complementary output.</p> <p><b>Value After Reset:</b> 0x0</p>
[4]	CC3G	W	<p>Capture/Compare 3 generation This bit is set by software in order to generate an event, it is automatically cleared by hardware.</p> <p><b>Values:</b></p> <p><b>0x0:</b> No action</p> <p><b>0x1:</b> A capture/compare event is generated on channel 3: <b>If channel CC3 is configured as output</b> (<a href="#">PWMA_CCMR2.CC3S</a> = 0x0): <a href="#">PWMA_SR.CC3IF</a> flag is set. Corresponding interrupt or DMA request is sent if enabled. <b>If channel CC3 is configured as input</b> (<a href="#">PWMA_CCMR2.CC3S</a> != 0x0): The current value of the counter is captured in <a href="#">PWMA_CCR3</a> register. The <a href="#">PWMA_SR.CC3IF</a> flag is set, the corresponding interrupt or DMA request is sent if enabled. The <a href="#">PWMA_SR.CC3OF</a> flag is set if the <a href="#">CC3IF</a> flag was already high.</p> <p><b>Value After Reset:</b> 0x0</p>
[3]	CC2G	W	<p>Capture/Compare 2 generation This bit is set by software in order to generate an event, it is automatically cleared by hardware.</p>

**Table 12-11 Fields For Register: PWMA\_EGR (continued)**

Bits	Name	R/W	Description
			<p><b>Values:</b></p> <p><b>0x0:</b> No action</p> <p><b>0x1:</b> A capture/compare event is generated on channel 2:  <b>If channel CC2 is configured as output</b> (<code>PWMA_CCMR2.CC2S</code> = <code>0x0</code>):  <code>PWMA_SR.CC2IF</code> flag is set. Corresponding interrupt or DMA request is sent if enabled.</p> <p><b>If channel CC2 is configured as input</b> (<code>PWMA_CCMR2.CC2S</code> != <code>0x0</code>):  The current value of the counter is captured in <code>PWMA_CCR2</code> register.  The <code>PWMA_SR.CC2IF</code> flag is set, the corresponding interrupt or DMA request is sent if enabled. The <code>PWMA_SR.CC2OF</code> flag is set if the <code>CC2IF</code> flag was already high.</p> <p><b>Value After Reset:</b> 0x0</p>
[2]	CC1G	W	<p>Capture/Compare 1 generation  This bit is set by software in order to generate an event, it is automatically cleared by hardware.</p> <p><b>Values:</b></p> <p><b>0x0:</b> No action</p> <p><b>0x1:</b> A capture/compare event is generated on channel 1:  <b>If channel CC1 is configured as output</b> (<code>PWMA_CCMR1.CC1S</code> = <code>0x0</code>):  <code>PWMA_SR.CC1IF</code> flag is set. Corresponding interrupt or DMA request is sent if enabled.</p> <p><b>If channel CC1 is configured as input</b> (<code>PWMA_CCMR1.CC1S</code> != <code>0x0</code>):  The current value of the counter is captured in <code>PWMA_CCR1</code> register.  The <code>PWMA_SR.CC1IF</code> flag is set, the corresponding interrupt or DMA request is sent if enabled. The <code>PWMA_SR.CC1OF</code> flag is set if the <code>CC1IF</code> flag was already high.</p> <p><b>Value After Reset:</b> 0x0</p>
[1]	CC0G	W	<p>Capture/Compare 0 generation  This bit is set by software in order to generate an event, it is automatically cleared by hardware.</p> <p><b>Values:</b></p> <p><b>0x0:</b> No action</p> <p><b>0x1:</b> A capture/compare event is generated on channel 0:  <b>If channel CC0 is configured as output</b> (<code>PWMA_CCMR1.CC0S</code> = <code>0x0</code>):  <code>PWMA_SR.CC0IF</code> flag is set. Corresponding interrupt or DMA request is sent if enabled.</p> <p><b>If channel CC0 is configured as input</b> (<code>PWMA_CCMR1.CC0S</code> != <code>0x0</code>):</p>

**Table 12-11 Fields For Register: PWMA\_EGR (continued)**

Bits	Name	R/W	Description
			<p>The current value of the counter is captured in <a href="#">PWMA_CCR0</a> register. The <a href="#">PWMA_SR.CC0IF</a> flag is set, the corresponding interrupt or DMA request is sent if enabled. The <a href="#">PWMA_SR.CC0OF</a> flag is set if the <a href="#">CC0IF</a> flag was already high.</p> <p><b>Value After Reset:</b> 0x0</p>
[0]	UG	W	<p>Update generation This bit can be set by software, it is automatically cleared by hardware.</p> <p><b>Values:</b></p> <p><b>0x0:</b> No action</p> <p><b>0x1:</b> 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 <a href="#">PWMA_CR1.DIR</a>=0x0 (upcounting), else it takes the auto-reload value (<a href="#">PWMA_ARR</a>) if <a href="#">PWMA_CR1.DIR</a>=0x1 (downcounting).</p> <p><b>Value After Reset:</b> 0x0</p>

#### 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 used in input (capture mode) or in output (compare mode). The direction of a channel is defined by configuring the corresponding [CCxS](#) bits. All the other bits of this register have a different function in input and in output mode. For a given bit, [OCxx](#) describes its function when the channel is configured in output, [ICxx](#) describes its function when the channel is configured in input. So the user must take care 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]	<a href="#">IC1*</a> or <a href="#">OC1*</a>	R/W	<p> The functions of these bits depend on the value of the <a href="#">CC1S</a> bit field. For details, see the table <a href="#">PWMA_CCMR1[15:10] bits functions for different CC1S values</a> below.</p>
[9:8]	<a href="#">CC1S</a>	R/W	<p>Capture/Compare Channel 1 selection This bit-field defines the direction of the channel 1 (input/output) as well as the used input.</p> <p><b>Values:</b></p> <p><b>0x0:</b> CC1 channel is configured as output</p> <p><b>0x1:</b> CC1 channel is configured as input, <a href="#">IC1</a> is mapped on <a href="#">TII</a></p> <p><b>0x2:</b> CC1 channel is configured as input, <a href="#">IC1</a> is mapped on <a href="#">TIO</a></p> <p><b>0x3:</b> CC1 channel is configured as input, <a href="#">IC1</a> is mapped on <a href="#">TRC</a>. This mode is working only if an internal trigger input is selected through the <a href="#">TS</a> bit (<a href="#">PWMA_SMCR</a> register)</p>

**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) <b>Value After Reset:</b> 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 <b>PWMA_CCMR1[7:2] bits functions for different cc0s values</b> below.
[1:0]	cc0s	R/W	<p>Capture/Compare Channel 0 selection        This bit-field defines the direction of the channel 0 (input/output) as well as the used input.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> CC0 channel is configured as output</li> <li><b>0x1:</b> CC0 channel is configured as input, IC0 is mapped on TIO</li> <li><b>0x2:</b> CC0 channel is configured as input, IC0 is mapped on TII</li> <li><b>0x3:</b> CC0 channel is configured as input, IC0 is mapped on TRC.        This mode is working only if an internal trigger input is selected through the TS bit (PWMA_SMCR register)</li> </ul>  cc0s bits are writable only when the channel is OFF (CC0E = 0x0 in PWMA_CCER) <b>Value After Reset:</b> 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
<b>Channel 1 Output Compare mode (CC1S = 0x0)</b>			
[15]	OC1CE	R/W	<p>Output Compare Channel 1 clear enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> OC1REF is not affected by the ETRF Input</li> <li><b>0x1:</b> OC1REF is cleared as soon as a High level is detected on ETRF input</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[14:12]	OC1M	R/W	<p>Output Compare Channel 1 mode</p> <p>These bits define the behavior of the output reference signal OC1REF from which OC1 and OC1N are derived. OC1REF is active high whereas OC1 and OC1N active level depends on CC1P and CC1NP bits.</p>  The possible values of this bit are the same as the OC0M bit.

- 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

**Table 12-13 PWMA\_CCMR1[15:10] bits functions for different CC1S 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 OC1M bit to 0x4 or 0x5 to get a 100% duty cycle.  <b>Value After Reset:</b> 0x0
[11]	OC1PE	R/W	<p>Output Compare Channel 1 preload enable  <b>Values:</b></p> <p><b>0x0:</b> Preload register on <a href="#">PWMA_CCR1</a> disabled. <a href="#">PWMA_CCR1</a> can be written at anytime, the new value is taken in account immediately.</p> <p><b>0x1:</b> Preload register on <a href="#">PWMA_CCR1</a> enabled. Read/Write operations access the preload register. <a href="#">PWMA_CCR1</a> preload value is loaded in the active register at each update event.</p> <p> The PWM mode can be used without validating the preload register only in one pulse mode (<a href="#">OPM</a> bit set in <a href="#">PWMA_CR1</a> register). Else the behavior is not guaranteed.</p> <p><b>Value After Reset:</b> 0x0</p>
[10]			Reserved

**Channel 1 Input Capture mode (CC1S != 0x0)**

[15:12]	IC1F	R/W	<p>Input Capture Channel 1 filter  This bit-field defines the frequency used to sample <a href="#">TI1</a> input and the length of the digital filter applied to <a href="#">TI1</a>. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output.</p> <p> The possible values of this bit are the same as the <a href="#">IC0F</a> bit.</p> <p><b>Value After Reset:</b> 0x0</p>
[11:10]	IC1PSC	R/W	<p>Input Capture Channel 1 prescaler  This bit-field defines the ratio of the prescaler acting on CC1 input (<a href="#">IC1</a>). The prescaler is reset as soon as <a href="#">CC1E</a>=0x0 (<a href="#">PWMA_CCER</a> register).</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> no prescaler, capture is done each time an edge is detected on the capture input</li> <li><b>0x1:</b> capture is done once every 2 events</li> <li><b>0x2:</b> capture is done once every 4 events</li> <li><b>0x3:</b> capture is done once every 8 events</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

The table below describes the [PWMA\\_CCMR1\[7:2\]](#) bits functions for different [PWMA\\_CCMR1.CC0S](#) bit values

**Table 12-14 PWMA\_CCMR1[7:2] bits functions for different CC0S values**

Bits	Name	R/W	Description
<b>Channel 0 Output Compare mode (CC0S = 0x0)</b>			

**Table 12-14 PWMA\_CCMR1[7:2] bits functions for different CC0S values (continued)**

Bits	Name	R/W	Description
[7]	OC0CE	R/W	<p>Output Compare Channel 0 clear enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> OC0REF is not affected by the ETRF Input</li> <li><b>0x1:</b> OC0REF is cleared as soon as a High level is detected on ETRF input</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[6:4]	OC0M	R/W	<p>Output Compare Channel 0 mode</p> <p>These bits define the behavior of the output reference signal OC0REF from which OC0 and OC0N are derived. OC0REF is active high whereas OC0 and OC0N active level depends on CC0P and CC0NP bits.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> 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).</li> <li><b>0x1:</b> Set channel 0 to active level on match. OC0REF signal is forced high when the counter PWMA_CNT matches the PWMA_CCR0 register.</li> <li><b>0x2:</b> Set channel 0 to inactive level on match. OC0REF signal is forced low when the counter PWMA_CNT matches the PWMA_CCR0 register..</li> <li><b>0x3:</b> Toggle - OC0REF toggles when PWMA_CNT=PWMA_CCR0.</li> <li><b>0x4:</b> Force inactive level - OC0REF is forced low.</li> <li><b>0x5:</b> Force active level - OC0REF is forced high.</li> <li><b>0x6:</b> PWM mode 1 - In upcounting, channel 0 is active as long as PWMA_CNT&lt;PWMA_CCR0 else inactive. In downcounting, channel 0 is inactive (OC0REF='0') as long as PWMA_CNT&gt;PWMA_CCR0 else active (OC0REF='1').</li> <li><b>0x7:</b> PWM mode 2 - In upcounting, channel 0 is inactive as long as PWMA_CNT&lt;PWMA_CCR0 else active. In downcounting, channel 0 is active as long as PWMA_CNT&gt;PWMA_CCR0 else inactive.</li> </ul> <p> 1. In PWM mode 1 or 2, the OC0REF 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.</p> <p> • 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 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.</p> <p><b>Value After Reset:</b> 0x0</p>

**Table 12-14 PWMA\_CCMR1[7:2] bits functions for different CC0S values (continued)**

Bits	Name	R/W	Description
[3]	OC0PE	R/W	<p>Output Compare Channel 0 preload enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Preload register on <a href="#">PWMA_CCR0</a> disabled. <a href="#">PWMA_CCR0</a> can be written at anytime, the new value is taken in account immediately.</li> <li><b>0x1:</b> Preload register on <a href="#">PWMA_CCR0</a> enabled. Read/Write operations access the preload register. <a href="#">PWMA_CCR0</a> preload value is loaded in the active register at each update event.</li> </ul> <p> The PWM mode can be used without validating the preload register only in one pulse mode (<a href="#">OPM</a> bit set in <a href="#">PWMA_CR1</a> register). Else the behavior is not guaranteed.</p> <p><b>Value After Reset:</b> 0x0</p>
[2]			Reserved
<b>Channel 0 Input Capture mode (CC0S != 0x0)</b>			
[7:4]	IC0F	R/W	<p>Input Capture Channel 0 filter</p> <p>This bit-field defines the frequency used to sample <a href="#">TI0</a> input and the length of the digital filter applied to <a href="#">TI0</a>. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> No filter, sampling is done at <math>f_{DTS}</math> (<math>f_{SAMPLING}=f_{DTS}</math>)</li> <li><b>0x1:</b> <math>f_{SAMPLING}=f_{CK\_INT}</math>, N=2</li> <li><b>0x2:</b> <math>f_{SAMPLING}=f_{CK\_INT}</math>, N=4</li> <li><b>0x3:</b> <math>f_{SAMPLING}=f_{CK\_INT}</math>, N=8</li> <li><b>0x4:</b> <math>f_{SAMPLING}=f_{DTS}/2</math>, N=6</li> <li><b>0x5:</b> <math>f_{SAMPLING}=f_{DTS}/2</math>, N=8</li> <li><b>0x6:</b> <math>f_{SAMPLING}=f_{DTS}/4</math>, N=6</li> <li><b>0x7:</b> <math>f_{SAMPLING}=f_{DTS}/4</math>, N=8</li> <li><b>0x8:</b> <math>f_{SAMPLING}=f_{DTS}/8</math>, N=6</li> <li><b>0x9:</b> <math>f_{SAMPLING}=f_{DTS}/8</math>, N=8</li> <li><b>0xA:</b> <math>f_{SAMPLING}=f_{DTS}/16</math>, N=5</li> <li><b>0xB:</b> <math>f_{SAMPLING}=f_{DTS}/16</math>, N=6</li> <li><b>0xC:</b> <math>f_{SAMPLING}=f_{DTS}/16</math>, N=8</li> <li><b>0xD:</b> <math>f_{SAMPLING}=f_{DTS}/32</math>, N=5</li> <li><b>0xE:</b> <math>f_{SAMPLING}=f_{DTS}/32</math>, N=6</li> <li><b>0xF</b> <math>f_{SAMPLING}=f_{DTS}/32</math>, N=8</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[3:2]	IC0PSC	R/W	<p>Input Capture Channel 0 prescaler</p> <p>This bit-field defines the ratio of the prescaler acting on CC0 input (<a href="#">IC0</a>). The prescaler is reset as soon as <a href="#">CC0E</a>=0x0 (<a href="#">PWMA_CCER</a> register).</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> no prescaler, capture is done each time an edge is detected on the capture input</li> <li><b>0x1:</b> capture is done once every 2 events</li> </ul>

**Table 12-14 PWMA\_CCMR1[7:2] bits functions for different cc0s values (continued)**

Bits	Name	R/W	Description
			<p><b>0x2:</b> capture is done once every 4 events  <b>0x3:</b> capture is done once every 8 events</p> <p><b>Value After Reset:</b> 0x0</p>

### 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 used in input (capture mode) or in output (compare mode). The direction of a channel is defined by configuring the corresponding CC<sub>xs</sub> bits. All the other bits of this register have a different function in input and in output mode. For a given bit, OC<sub>xx</sub> describes its function when the channel is configured in output, IC<sub>xx</sub> describes its function when the channel is configured in input. So the user must take care 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	<p></p> <p>The functions of these bits depend on the value of the CC3S bit field. For details, see the table <a href="#">PWMA_CCMR2[15:10] bits functions for different cc3s values</a> below.</p>
[9:8]	CC3S	R/W	<p>Capture/Compare Channel 3 selection  This bit-field defines the direction of the channel 3 (input/output) as well as the used input.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> CC3 channel is configured as output</li> <li><b>0x1:</b> CC3 channel is configured as input, IC3 is mapped on TI3</li> <li><b>0x2:</b> CC3 channel is configured as input, IC3 is mapped on TI2</li> <li><b>0x3:</b> CC3 channel is configured as input, IC3 is mapped on TRC.  This mode is working only if an internal trigger input is selected through the TS bit (<a href="#">PWMA_SMCR</a> register)</li> </ul> <p> CC3S bits are writable only when the channel is OFF (CC3E = 0x0 in <a href="#">PWMA_CCER</a>)</p> <p><b>Value After Reset:</b> 0x0</p>
[7:2]	IC2* or OC2*	R/W	<p></p> <p>The functions of these bits depend on the value of the CC2S bit field. For details, see the table <a href="#">PWMA_CCMR2[7:2] bits functions for different cc2s values</a> below.</p>
[1:0]	CC2S	R/W	<p>Capture/Compare Channel 2 selection  This bit-field defines the direction of the channel 2 (input/output) as well as the used input.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> CC2 channel is configured as output</li> <li><b>0x1:</b> CC2 channel is configured as input, IC2 is mapped on TI2</li> </ul>

**Table 12-15 Fields For Register: PWMA\_CCMR2 (continued)**

Bits	Name	R/W	Description
			<p><b>0x2:</b> CC2 channel is configured as input, <code>IC2</code> is mapped on <code>TI3</code></p> <p><b>0x3:</b> CC2 channel is configured as input, <code>IC2</code> is mapped on <code>TRC</code>. This mode is working only if an internal trigger input is selected through the <code>TS</code> bit (<code>PWMA_SMCR</code> register)</p> <p> <code>CC2S</code> bits are writable only when the channel is OFF (<code>CC2E = 0x0</code> in <code>PWMA_CCER</code>)</p> <p><b>Value After Reset:</b> 0x0</p>

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
<b>Channel 3 Output Compare mode (CC3S = 0x0)</b>			
[15]	OC3CE	R/W	<p>Output Compare Channel 3 clear enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <code>OC3REF</code> is not affected by the <code>ETRF</code> Input</li> <li><b>0x1:</b> <code>OC3REF</code> is cleared as soon as a High level is detected on <code>ETRF</code> input</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[14:12]	OC3M	R/W	<p>Output Compare Channel 3 mode</p> <p>These bits define the behavior of the output reference signal <code>OC3REF</code> from which <code>OC3</code> is derived. <code>OC3REF</code> is active high whereas <code>OC3</code> active level depends on <code>CC3P</code> bit.</p> <p> The possible values of this bit are the same as the <code>OC2M</code> bit.</p> <p> • If <code>OC3M</code> bit is set to 0x6 or 0x7, it is not recommended to set the <code>PWMA_CCR3</code> 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.</p> <p>• It is recommended to set the <code>OC3M</code> bit to 0x4 or 0x5 to get a 100% duty cycle.</p> <p><b>Value After Reset:</b> 0x0</p>
[11]	OC3PE	R/W	<p>Output Compare Channel 3 preload enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Preload register on <code>PWMA_CCR3</code> disabled. <code>PWMA_CCR3</code> can be written at anytime, the new value is taken in account immediately.</li> <li><b>0x1:</b> Preload register on <code>PWMA_CCR3</code> enabled. Read/Write operations access the preload register. <code>PWMA_CCR3</code> preload value is loaded in the active register at each update event.</li> </ul>

**Table 12-16 PWMA\_CCMR2[15:10] bits functions for different CC3S values (continued)**

Bits	Name	R/W	Description
			 The PWM mode can be used without validating the preload register only in one pulse mode ( <code>OPM</code> bit set in <a href="#">PWMA_CR1</a> register). Else the behavior is not guaranteed.  <b>Value After Reset:</b> 0x0
[10]			Reserved
<b>Channel 3 Input Capture mode (CC3S != 0x0)</b>			
[15:12]	IC3F	R/W	<p>Input Capture Channel 3 filter          This bit-field defines the frequency used to sample <code>TI3</code> input and the length of the digital filter applied to <code>TI3</code>. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output.</p> <p> The possible values of this bit are the same as the <code>IC2F</code> bit.</p> <p><b>Value After Reset:</b> 0x0</p>
[11:10]	IC3PSC	R/W	<p>Input Capture Channel 3 prescaler          This bit-field defines the ratio of the prescaler acting on CC3 input (<code>IC3</code>). The prescaler is reset as soon as <code>CC3E=0x0</code> (<a href="#">PWMA_CCER</a> register).</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> no prescaler, capture is done each time an edge is detected on the capture input</li> <li><b>0x1:</b> capture is done once every 2 events</li> <li><b>0x2:</b> capture is done once every 4 events</li> <li><b>0x3:</b> capture is done once every 8 events</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

The table below describes the `PWMA_CCMR2[7:2]` bits functions for different `PWMA_CCMR2.CC2S` bit values

**Table 12-17 PWMA\_CCMR2[7:2] bits functions for different CC2S values**

Bits	Name	R/W	Description
<b>Channel 2 Output Compare mode (CC2S = 0x0)</b>			
[7]	OC2CE	R/W	<p>Output Compare Channel 2 clear enable  <b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <code>OC2REF</code> is not affected by the <code>ETRF</code> Input</li> <li><b>0x1:</b> <code>OC2REF</code> is cleared as soon as a High level is detected on <code>ETRF</code> input</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[6:4]	OC2M	R/W	<p>Output Compare Channel 2 mode          These bits define the behavior of the output reference signal <code>OC2REF</code> from which <code>OC2</code> and <code>OC2N</code> are derived. <code>OC2REF</code> is active high whereas <code>OC2</code> and <code>OC2N</code> active level depends on <code>CC2P</code> and <code>CC2NP</code> bits.  <b>Values:</b></p>

**Table 12-17 PWMA\_CCMR2[7:2] bits functions for different cc2s values (continued)**

Bits	Name	R/W	Description
			<p><b>0x0:</b> Frozen - The comparison between the output compare register <code>PWMA_CCR2</code> and the counter <code>PWMA_CNT</code> has no effect on the outputs.(this mode is used to generate a timing base).</p> <p><b>0x1:</b> Set channel 2 to active level on match. <code>OC2REF</code> signal is forced high when the counter <code>PWMA_CNT</code> matches the <code>PWMA_CCR2</code> register.</p> <p><b>0x2:</b> Set channel 2 to inactive level on match. <code>OC2REF</code> signal is forced low when the counter <code>PWMA_CNT</code> matches the <code>PWMA_CCR2</code> register..</p> <p><b>0x3:</b> Toggle - <code>OC2REF</code> toggles when <code>PWMA_CNT=PWMA_CCR2</code>.</p> <p><b>0x4:</b> Force inactive level - <code>OC2REF</code> is forced low.</p> <p><b>0x5:</b> Force active level - <code>OC2REF</code> is forced high.</p> <p><b>0x6:</b> PWM mode 1 - In upcounting, channel 2 is active as long as <code>PWMA_CNT&lt;PWMA_CCR2</code> else inactive. In downcounting, channel 2 is inactive (<code>OC2REF='0'</code>) as long as <code>PWMA_CNT&gt;PWMA_CCR2</code> else active (<code>OC2REF='1'</code>).</p> <p><b>0x7:</b> PWM mode 2 - In upcounting, channel 2 is inactive as long as <code>PWMA_CNT&lt;PWMA_CCR2</code> else active. In downcounting, channel 2 is active as long as <code>PWMA_CNT&gt;PWMA_CCR2</code> else inactive.</p> <p> 1. In PWM mode 1 or 2, the <code>OCR2EF</code> 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 <code>CCPC</code> bit is set in the <code>PWMA_CR2</code> register then the <code>OC2M</code> active bits take the new value from the preloaded bits only when a COM event is generated.</p> <p> • If <code>OC2M</code> bit is set to <code>0x6</code> or <code>0x7</code>, it is not recommended to set the <code>PWMA_CCR2</code> register to <code>0x0</code>. 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 <code>OC2M</code> bit to <code>0x4</code> or <code>0x5</code> to get a 100% duty cycle.</p> <p><b>Value After Reset:</b> 0x0</p>
[3]	OC2PE	R/W	<p>Output Compare Channel 2 preload enable</p> <p><b>Values:</b></p> <p><b>0x0:</b> Preload register on <code>PWMA_CCR2</code> disabled. <code>PWMA_CCR2</code> can be written at anytime, the new value is taken in account immediately.</p> <p><b>0x1:</b> Preload register on <code>PWMA_CCR2</code> enabled. Read/Write operations access the preload register. <code>PWMA_CCR2</code> preload value is loaded in the active register at each update event.</p> <p> The PWM mode can be used without validating the preload register only in one pulse mode (<code>OPM</code> bit set in <code>PWMA_CR1</code> register). Else the behavior is not guaranteed.</p> <p><b>Value After Reset:</b> 0x0</p>

**Table 12-17 PWMA\_CCMR2[7:2] bits functions for different CC2S values (continued)**

Bits	Name	R/W	Description
[2]			Reserved
<b>Channel 2 Input Capture mode (CC2S != 0x0)</b>			
[7:4]	IC2F	R/W	<p>Input Capture Channel 2 filter            This bit-field defines the frequency used to sample <math>\text{TI}_2</math> input and the length of the digital filter applied to <math>\text{TI}_2</math>. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> No filter, sampling is done at <math>f_{DTS}</math> (<math>f_{SAMPLING}=f_{DTS}</math>)</li> <li><b>0x1:</b> <math>f_{SAMPLING}=f_{CK\_INT}</math>, N=2</li> <li><b>0x2:</b> <math>f_{SAMPLING}=f_{CK\_INT}</math>, N=4</li> <li><b>0x3:</b> <math>f_{SAMPLING}=f_{CK\_INT}</math>, N=8</li> <li><b>0x4:</b> <math>f_{SAMPLING}=f_{DTS}/2</math>, N=6</li> <li><b>0x5:</b> <math>f_{SAMPLING}=f_{DTS}/2</math>, N=8</li> <li><b>0x6:</b> <math>f_{SAMPLING}=f_{DTS}/4</math>, N=6</li> <li><b>0x7:</b> <math>f_{SAMPLING}=f_{DTS}/4</math>, N=8</li> <li><b>0x8:</b> <math>f_{SAMPLING}=f_{DTS}/8</math>, N=6</li> <li><b>0x9:</b> <math>f_{SAMPLING}=f_{DTS}/8</math>, N=8</li> <li><b>0xA:</b> <math>f_{SAMPLING}=f_{DTS}/16</math>, N=5</li> <li><b>0xB:</b> <math>f_{SAMPLING}=f_{DTS}/16</math>, N=6</li> <li><b>0xC:</b> <math>f_{SAMPLING}=f_{DTS}/16</math>, N=8</li> <li><b>0xD:</b> <math>f_{SAMPLING}=f_{DTS}/32</math>, N=5</li> <li><b>0xE:</b> <math>f_{SAMPLING}=f_{DTS}/32</math>, N=6</li> <li><b>0xF</b> <math>f_{SAMPLING}=f_{DTS}/32</math>, N=8</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[3:2]	IC2PSC	R/W	<p>Input Capture Channel 2 prescaler            This bit-field defines the ratio of the prescaler acting on CC2 input (IC20).            The prescaler is reset as soon as CC2E=0x0 (<a href="#">PWMA_CCER</a> register).</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> no prescaler, capture is done each time an edge is detected on the capture input</li> <li><b>0x1:</b> capture is done once every 2 events</li> <li><b>0x2:</b> capture is done once every 4 events</li> <li><b>0x3:</b> capture is done once every 8 events</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

### 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. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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.

**Table 12-18 Fields For Register: PWMA\_CCER (continued)**

Bits	Name	R/W	Description
			<b>Value After Reset:</b> 0x0
[3]	CC0NP	R/W	<p>Capture/Compare Channel 0 complementary output polarity</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> OC0N active high</li> <li><b>0x1:</b> OC0N active low</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[2]	CC0NE	R/W	<p>Capture/Compare Channel 0 complementary output enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Off - OC0N is not active. OC0N level is then function of MOE, OSSR, OISO, OIS0N and CC0E bits.</li> <li><b>0x1:</b> On - OC0N signal is output on the corresponding output pin depending on MOE, OSSR, OISO, OIS0N and CC0E bits.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[1]	CC0P	R/W	<p>Capture/Compare Channel 0 output polarity</p> <p><b>If channel CC0 is configured as output (PWMA_CCMR1.CC0S = 0x0):</b></p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> OC0 active high</li> <li><b>0x1:</b> OC0 active low</li> </ul> <p><b>If channel CC0 is configured as input (PWMA_CCMR1.CC0S != 0x0):</b></p> <p>Selects the active polarity of TI0FP0 and TI1FP0.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Non-inverted / rising edge</li> <li><b>0x1:</b> Inverted / falling edge</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[0]	CC0E	R/W	<p>Capture/Compare Channel 0 output enable</p> <p><b>If channel CC0 is configured as output (PWMA_CCMR1.CC0S = 0x0):</b></p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Off - OC0 is not active. OC0 level is then function of MOE, OSSR, OISO, OIS0N and CC0NE bits.</li> <li><b>0x1:</b> On - OC0 signal is output on the corresponding output pin depending on MOE, OSSR, OISO, OIS0N and CC0NE bits.</li> </ul> <p><b>If channel CC0 is configured as input (PWMA_CCMR1.CC0S != 0x0):</b></p> <p>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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Capture disabled.</li> <li><b>0x1:</b> Capture enabled.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

### 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 <b>Value After Reset:</b> 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 ( <a href="#">CK_CNT</a> ) is equal to $f_{CK\_PSC} / (PSC + 1)$ . PSC contains the value to be loaded in the active prescaler register at each Update Event (including when the counter is cleared through the <a href="#">UG</a> bit of the <a href="#">PWMA_EGR</a> register or through trigger controller when configured in “reset mode”) <b>Value After Reset:</b> 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 <a href="#">ARR</a> is the value to be loaded in the actual auto-reload register.  This register can be buffered (see <a href="#">ARPE</a> bit description in the <a href="#">PWMA_CR1</a> register). The counter is blocked while the Auto-reload value ( <a href="#">ARR</a> ) is null. <b>Value After Reset:</b> 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	<p>Repetition counter value</p> <p>These bits allow the user to set-up the update rate of the compare registers (i.e. periodic transfers from preload to active registers) when preload registers are enable, as well as the update interrupt generation rate, if this interrupt is enable.</p> <p>Each time the <b>Repetition Counter</b> related downcounter reaches zero, an Update Event is generated and it restarts counting from REP value. As the Repetition Counter is reloaded with REP value only at the Update Event, any write to the PWMA_RCR register is not taken in account until the next Update Event.</p> <p>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. It means in PWM mode (REP+1) corresponds to:</p> <ul style="list-style-type: none"> <li>• the number of PWM periods in edge-aligned mode</li> <li>• the number of half PWM period in center-aligned mode.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

#### 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	<p>Capture/Compare Channel 0 value</p> <p>If channel CC0 is configured as output (PWMA_CCMR1.CC0S = 0x0): CCR0 is the value to be loaded in the actual capture/compare 0 register (preload value).</p> <p>It is loaded permanently if the preload feature is not selected in the PWMA_CCMR1 register (bit OC0PE). Else the preload value is copied in the active capture/compare 0 register when an Update Event occurs.</p> <p>The active capture/compare register contains the value to be compared to the counter PWMA_CNT and signaled on OC0 output.</p> <p> 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.</p> <p>If channel CC0 is configured as input (PWMA_CCMR1.CC0S != 0x0): CCR0 is the counter value transferred by the last input capture 0 event (IC0). The PWMA_CCR0 register is read-only and cannot be programmed.</p> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p>Capture/Compare Channel 1 value</p> <p><b>If channel CC1 is configured as output</b> (<a href="#">PWMA_CCMR1.CC1S</a> = 0x0):  <code>CCR1</code> is the value to be loaded in the actual capture/compare 1 register (preload value). It is loaded permanently if the preload feature is not selected in the <a href="#">PWMA_CCMR1</a> register (bit <code>OC1PE</code>). Else the preload value is copied in 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 <a href="#">PWMA_CNT</a> and signaled on OC1 output.</p> <p> If <a href="#">PWMA_CCMR1.OC1M</a> bit is set to 0x6 or 0x7, it is not recommended to set the <code>CCR1</code> 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.</p> <p><b>If channel CC1 is configured as input</b> (<a href="#">PWMA_CCMR1.CC1S</a> != 0x0):  <code>CCR1</code> is the counter value transferred by the last input capture 1 event (<code>IC1</code>). The <a href="#">PWMA_CCR1</a> register is read-only and cannot be programmed.</p> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p>Capture/Compare Channel 2 value</p> <p><b>If channel CC2 is configured as output</b> (<a href="#">PWMA_CCMR2.CC2S</a> = 0x0):  <code>CCR2</code> is the value to be loaded in the actual capture/compare 2 register (preload value). It is loaded permanently if the preload feature is not selected in the <a href="#">PWMA_CCMR2</a> register (bit <code>OC2PE</code>). Else the preload value is copied in 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 <a href="#">PWMA_CNT</a> and signaled on OC2 output.</p> <p> If <a href="#">PWMA_CCMR2.OC2M</a> bit is set to 0x6 or 0x7, it is not recommended to set the <code>CCR2</code> 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.</p>

**Table 12-25 Fields For Register: PWMA\_CCR2 (continued)**

Bits	Name	R/W	Description
			<p>If channel CC2 is configured as input (<a href="#">PWMA_CCMR2.CC2S</a> != 0x0): CCR2 is the counter value transferred by the last input capture 2 event (IC2). The PWMA_CCR2 register is read-only and cannot be programmed.</p> <p><b>Value After Reset:</b> 0x0</p>

#### 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	<p>Capture/Compare Channel 3 value</p> <p>If channel CC3 is configured as output (<a href="#">PWMA_CCMR2.CC3S</a> = 0x0): CCR3 is the value to be loaded in the actual capture/compare 3 register (preload value).</p> <p>It is loaded permanently if the preload feature is not selected in the PWMA_CCMR2 register (bit OC3PE). Else the preload value is copied in the active capture/compare 3 register when an update event occurs.</p> <p>The active capture/compare register contains the value to be compared to the counter <a href="#">PWMA_CNT</a> and signaled on OC3 output.</p> <p> If <a href="#">PWMA_CCMR2.OC3M</a> bit is set to 0x6 or 0x7, it is not recommended to set the CCR3 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.</p> <p>If channel CC3 is configured as input (<a href="#">PWMA_CCMR2.CC3S</a> != 0x0): CCR3 is the counter value transferred by the last input capture 3 event (IC3). The PWMA_CCR3 register is read-only and cannot be programmed.</p> <p><b>Value After Reset:</b> 0x0</p>

#### 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	<p>Main output enable</p> <p>This bit is cleared asynchronously by hardware as soon as the break input is active (if break is enabled, see <a href="#">BKE</a> for details). It is set by software or automatically depending on the <a href="#">AOE</a> bit. It is acting only on the channels which are configured in output.</p> <p><b>Values:</b></p>

**Table 12-27 Fields For Register: PWMA\_BDTR (continued)**

Bits	Name	R/W	Description
			<p><b>0x0:</b> <math>\text{OC}_x</math> and <math>\text{OC}_{xN}</math> outputs are disabled or forced to idle state.</p> <p><b>0x1:</b> <math>\text{OC}_x</math> and <math>\text{OC}_{xN}</math> outputs are enabled if their respective enable bits are set (<math>\text{CC}_{xE}</math>, <math>\text{CC}_{xNE}</math> in <a href="#">PWMA_CCER</a> register).</p> <p><b>Value After Reset:</b> 0x0</p>
[14]	AOE	R/W	<p>Automatic output enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <math>\text{MOE}</math> can be set only by software</li> <li><b>0x1:</b> <math>\text{MOE}</math> can be set by software or automatically at the next update event (if the break input is not active)</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[13]	BKP	R/W	<p>Break polarity</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Break input (<math>\text{BRK}</math>) is active low</li> <li><b>0x1:</b> Break input (<math>\text{BRK}</math>) is active high</li> </ul> <p> Any write operation to this bit takes a delay of one Register Interface clock cycle to become effective.</p> <p><b>Value After Reset:</b> 0x0</p>
[12]	BKE	R/W	<p>Break enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Break input (<math>\text{BRK}</math>) disabled</li> <li><b>0x1:</b> Break input (<math>\text{BRK}</math>) enabled. An active signal level on input resets the <math>\text{MOE}</math> bit.</li> </ul> <p> Any write operation to this bit takes a delay of one Register Interface clock cycle to become effective.</p> <p><b>Value After Reset:</b> 0x0</p>
[11]	OSSR	R/W	<p>Off-state selection for Run mode</p> <p>This bit determines the state of disabled outputs when <math>\text{MOE}=0x1</math> on channels having a complementary output which are configured as outputs. <math>\text{OSSR}</math> is not implemented if no complementary output is implemented in the timer.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> disabled outputs <math>\text{OC}_x/\text{OC}_{xN}</math> are set to 0</li> <li><b>0x1:</b> disabled outputs <math>\text{OC}_x/\text{OC}_{xN}</math> are set to their inactive level (inactive level depends on settings of the <math>\text{CC}_{xP}</math> and <math>\text{CC}_{xNP}</math> bits in the <a href="#">PWMA_CCER</a> register)</li> </ul> <p></p> <ul style="list-style-type: none"> <li>• Output is considered to be disabled when the appropriate <math>\text{CC}_{xE}</math> or <math>\text{CC}_{xNE}</math> bit of the <a href="#">PWMA_CCER</a> is set to 0x0.</li> <li>• The disabled output level can be set to 1 if that level is specified as inactive.</li> </ul>

**Table 12-27 Fields For Register: PWMA\_BDTR (continued)**

Bits	Name	R/W	Description
			<p><b>⚠</b> Avoid configuration when both outputs of the one channel are disabled (appropriate <code>CCxE</code> and <code>CCxNE</code> bits in the <code>PWMA_CCER</code> register are set to 0x0), and break function is enabled (protective shutdown of timer outputs). In this case, reset of the <code>MOE</code> bit can lead to the unpredictable behaviour of the channel outputs.</p> <p><b>Value After Reset:</b> 0x0</p>
[10:8]			Reserved
[7:0]	DTG	R/W	<p>Dead-time generator setup This bit-field defines the duration of the dead-time inserted between the complementary outputs. DT correspond to this duration.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0xxxxxx:</b> DT=DTG[ 7 : 0 ] * t<sub>dtg</sub> with t<sub>dtg</sub>=t<sub>DTS</sub></li> <li><b>10xxxxxx:</b> DT=(64+DTG[ 5 : 0 ]) * t<sub>dtg</sub> with t<sub>dtg</sub>=2*t<sub>DTS</sub></li> <li><b>110xxxxx:</b> DT=(32+DTG[ 4 : 0 ]) * t<sub>dtg</sub> with t<sub>dtg</sub>=8*t<sub>DTS</sub></li> <li><b>111xxxxx:</b> DT=(32+DTG[ 4 : 0 ]) * t<sub>dtg</sub> with t<sub>dtg</sub>=16*t<sub>DTS</sub></li> </ul> <p><b>Example:</b> If t<sub>DTS</sub>=125ns (8MHz), dead-time possible values are:</p> <ul style="list-style-type: none"> <li>• 0 to 15875ns by 125ns steps;</li> <li>• 16us to 31750ns by 250ns steps;</li> <li>• 32us to 63us by 1us steps;</li> <li>• 64us to 126us by 2us steps.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

#### 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	<p>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 <code>PWMA_DMAR</code> register address is performed).</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0 or 0x1:</b> 1 transfer</li> <li><b>0x2:</b> 2 transfers</li> <li><b>0x3:</b> 3 transfers</li> <li>...</li> <li><b>0x11:</b> 17 transfers</li> </ul>

**Table 12-28 Fields For Register: PWMA\_DCR (continued)**

Bits	Name	R/W	Description
			<b>Value After Reset:</b> 0x0
[7:5]			Reserved (return 0 when read)
[4:0]	DBA	R/W	<p>DMA base address This 5-bits vector defines the base-address for DMA transfers (when read/write access are done through the <a href="#">PWMA_DMAR</a> address). <a href="#">DBA</a> is defined as an offset starting from the address of the <a href="#">PWMA_CR1</a> register.</p> <p><b>Values example:</b></p> <ul style="list-style-type: none"> <li>• 0x0: offset corresponds to the <a href="#">PWMA_CR1</a> register</li> <li>• 0x1: offset corresponds to the <a href="#">PWMA_CR2</a> register</li> <li>• 0x2: offset corresponds to the <a href="#">PWMA_SMCR</a> register</li> </ul> <p><b>Usage example:</b> Let us consider the following transfer: <a href="#">DBL</a> = 0x7 (7 transfers) and <a href="#">DBA</a> = 0x0 (offset corresponds to the <a href="#">PWMA_CR1</a> register). In this case the transfer is done to/from 7 registers starting from the <a href="#">PWMA_CR1</a> address.</p> <p><b>Value After Reset:</b> 0x0</p>

**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	<p>DMA register for burst accesses A read or write operation to the <a href="#">PWMA_DMAR</a> register accesses the register located at the address: (<a href="#">PWMA_CR1</a> address) + (<a href="#">DBA</a> + DMA index) * 4, where <a href="#">PWMA_CR1</a> address is the offset of the PWMA Control Register 1, <a href="#">DBA</a> is the DMA base address configured in <a href="#">PWMA_DCR</a> register, DMA index is automatically controlled by the DMA transfer, and ranges from 0 to <a href="#">DBL</a> (<a href="#">DBL</a> configured in <a href="#">PWMA_DCR</a>).</p> <p><b>Value After Reset:</b> 0x0</p>

**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

**Table 12-30 Fields For Register: PWMA\_QESET (continued)**

Bits	Name	R/W	Description
			<p>Write 0x1 to this bit to reset the setted interrupt. Writing 0x0 has no effect.</p> <p><b>Value After Reset:</b> 0x0</p>
[12]	INIE	R/W	<p><code>indx</code> interrupt enable Enables the interrupt on the <code>indx</code> signal edge</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <code>indx</code> interrupt disabled</li> <li><b>0x1:</b> <code>indx</code> interrupt enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[11]	OFIE	R/W	<p>Overflow interrupt enable Enables the interrupt on the counter overflow and counter underflow</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Overflow interrupt disabled</li> <li><b>0x1:</b> Overflow interrupt enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[10]	INIT_MODE	R/W	<p>Initialization mode</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> rising edge of the <code>indx</code> signal resets the counter</li> <li><b>0x1:</b> rising edge of the <code>indx</code> signal will cause the counter initialization with the value of the <code>PWMA_INITSET</code> register</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[9]	SINGLE_MODE	R/W	<p>Single mode</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> (Multiple mode) The counter can count up to <code>PWMA_QEMAX</code> value and, if the <code>PWMA_QECNT</code> = <code>PWMA_QEMAX</code>, the counter restarts with 0. The counter can also count down to 0 and, if the <code>PWMA_QECNT</code> = 0, the counter restarts with <code>PWMA_QEMAX</code> value.</li> <li><b>0x1:</b> (Single mode) The counter can count up to <code>PWMA_QEMAX</code> value and then stop, or count up to 0 and then stop.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[8]	INDXPOL	R/W	<p><code>indx</code> polarity</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <code>indx</code> signal is not inverted</li> <li><b>0x1:</b> <code>indx</code> signal is inverted</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[7]	QBPOL	R/W	<p><code>qb</code> polarity</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <code>qb</code> signal is not inverted</li> <li><b>0x1:</b> <code>qb</code> signal is inverted</li> </ul>

**Table 12-30 Fields For Register: PWMA\_QESET (continued)**

Bits	Name	R/W	Description
			<b>Value After Reset:</b> 0x0
[6]	QAPOL	R/W	<p><code>qa</code> polarity</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <code>qa</code> signal is not inverted</li> <li><b>0x1:</b> <code>qa</code> signal is inverted</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[5]	QBF	R/W	<p><code>qb</code> Filtering</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Disable the <code>qb</code> filtering</li> <li><b>0x1:</b> Enable the <code>qb</code> filtering</li> </ul> <p> Filter settings are controlled through the <code>QBSF</code> bit field of the <a href="#">PWMA_FILTSET</a> register</p> <p><b>Value After Reset:</b> 0x0</p>
[4]	QAF	R/W	<p><code>qa</code> Filtering</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Disable the <code>qa</code> filtering</li> <li><b>0x1:</b> Enable the <code>qa</code> filtering</li> </ul> <p> Filter settings are controlled through the <code>QASF</code> bit field of the <a href="#">PWMA_FILTSET</a> register</p> <p><b>Value After Reset:</b> 0x0</p>
[3]	INDXF	R/W	<p><code>indx</code> Filtering</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Disable the <code>indx</code> filtering</li> <li><b>0x1:</b> Enable the <code>indx</code> filtering</li> </ul> <p> Filter settings are controlled through the <code>INDXSF</code> bit field of the <a href="#">PWMA_FILTSET</a> register</p> <p><b>Value After Reset:</b> 0x0</p>
[2]	QESWAP	R/W	<p>Swap <code>qa</code> and <code>qb</code> signals</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> No swap</li> <li><b>0x1:</b> <code>qa</code> is replaced by <code>qb</code>, and <code>qb</code> is replaced by <code>qa</code></li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[1]	QEMODE	R/W	<p>Selection of counting mode</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Quadrature Mode</li> <li><b>0x1:</b> <code>qa</code> Rising/Falling Edge Mode</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

**Table 12-30 Fields For Register: PWMA\_QESET (continued)**

Bits	Name	R/W	Description
[0]	QEN	R/W	<p>Quadrature Encoder enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x0: Disabled</li> <li>0x1: Enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p><code>indx</code> filter setting</p> <p>This bit-field defines the frequency used to sample the <code>indx</code> signal and the length of the digital filter applied to <code>indx</code>. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output.</p> <p> The possible values of this bit are the same as the QASF bit</p> <p><b>Value After Reset:</b> 0x0</p>
[7:4]	QBSF	R/W	<p><code>qb</code> filter setting</p> <p>This bit-field defines the frequency used to sample the <code>qb</code> signal and the length of the digital filter applied to <code>qb</code>. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output.</p> <p> The possible values of this bit are the same as the QASF bit</p> <p><b>Value After Reset:</b> 0x0</p>
[3:0]	QASF	R/W	<p><code>qa</code> filter setting</p> <p>This bit-field defines the frequency used to sample the <code>qa</code> signal and the length of the digital filter applied to <code>qa</code>. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x0: No filter, sampling is done at fDTS</li> <li>0x1: fSAMPLING=fCK_INT, N=2</li> <li>0x2: fSAMPLING=fCK_INT, N=4</li> <li>0x3: fSAMPLING=fCK_INT, N=8</li> <li>0x4: fSAMPLING=fDTS/2, N=6</li> <li>0x5: fSAMPLING=fDTS/2, N=8</li> </ul>

**Table 12-31 Fields For Register: `PWMA_FILTSET` (continued)**

Bits	Name	R/W	Description
			<p><b>0x6:</b> fSAMPLING=fDTS/4, N=6</p> <p><b>0x7:</b> fSAMPLING=fDTS/4, N=8</p> <p><b>0x8:</b> fSAMPLING=fDTS/8, N=6</p> <p><b>0x9:</b> fSAMPLING=fDTS/8, N=8</p> <p><b>0xA:</b> fSAMPLING=fDTS/16, N=5</p> <p><b>0xB:</b> fSAMPLING=fDTS/16, N=6</p> <p><b>0xC:</b> fSAMPLING=fDTS/16, N=8</p> <p><b>0xD:</b> fSAMPLING=fDTS/32, N=5</p> <p><b>0xE:</b> fSAMPLING=fDTS/32, N=6</p> <p><b>0xF:</b> fSAMPLING=fDTS/32, N=8</p> <p><b>Value After Reset:</b> 0x0</p>

**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]	<code>QEMAX</code>	R/W	<p>Max count value</p> <p>Defines the max count value after which the counter will stop in single mode (<code>PWMA_QESET.SINGLE_MODE</code> = 0x1) or reset in multiple mode (<code>PWMA_QESET.SINGLE_MODE</code> = 0x0)</p> <p><b>Value After Reset:</b> 0xFFFF</p>

**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]	<code>INITSET</code>	R/W	<p>Initialization value for the counter</p> <p>If the <code>PWMA_QESET.INIT_MODE</code> = 0x1, the rising edge of the <code>indx</code> signal will cause the counter initialization with the value of this register.</p> <p><b>Value After Reset:</b> 0xFFFF</p>

**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.

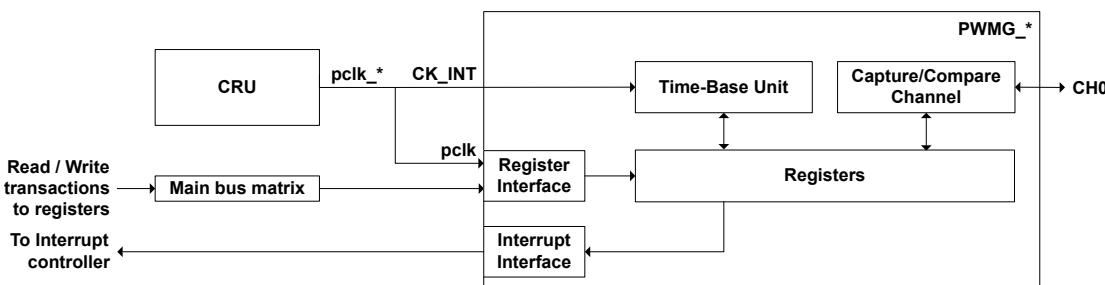
**Table 12-34 Fields For Register: PWMA\_QECNT**

Bits	Name	R/W	Description
[15:0]	QECNT	R/W	Quadrature Encoder Counter value <b>Value After Reset:</b> 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 PWMG.

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

**Figure 12-15 PWMG Functional Block Diagram**

In the figure above, 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**.

The PWMG controller supports the following features:

- Single channel for input signal capture and output signal generation
- Programmable prescaler for the counter clock ( $\text{CK_CNT}$ ) frequency
- Interrupt generation on the following events:
  - Update event (counter overflow, counter initialization)
  - Input capture
  - Output compare

### 12.2.1. PWMG Functional Description

In this section:

- [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](#)

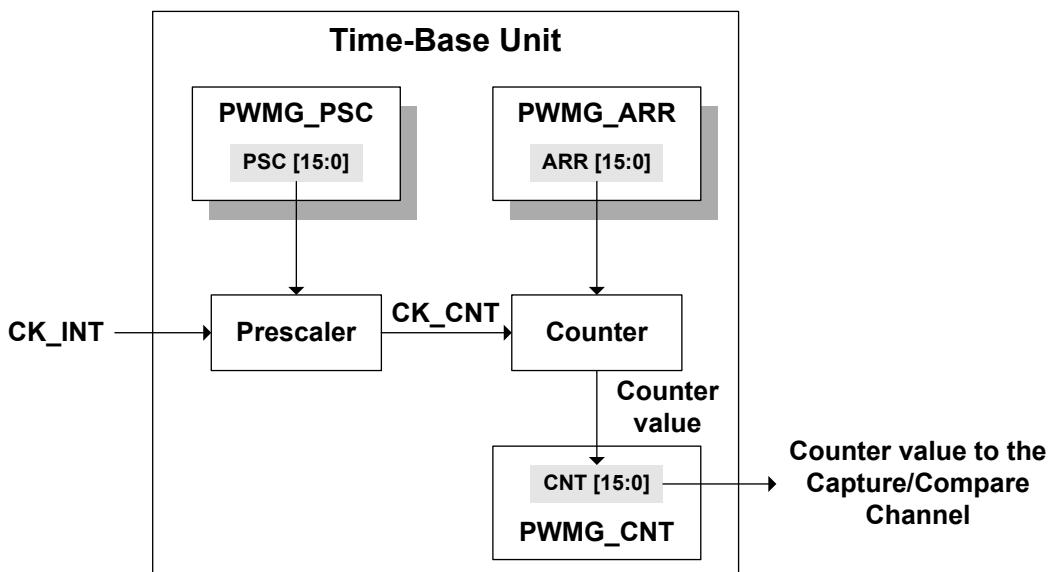


Figure 12-16 Time-base Unit

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 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 a **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 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. Refer to the [Counter Upcounting Mode](#) section for detailed description of the UEV generation and [PWMG\\_ARR](#) usage.

#### 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 PWMG counter can operate in upcounting mode. In this 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.

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 is to avoid updating the shadow registers while writing new values in 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 occurrence 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.

Capture/Compare Channel combines two completely different devices, one of which is used when the channel operates in input mode (to capture a signal), the other in output mode (to generate a signal). Capture/Compare Channel is built around a **Capture/Compare Stage** with PWMG Capture/Compare Register (including a shadow register), an **Input Stage** for capture (with digital filter and 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 output of the channel are physically combined into one `CH0` I/O pin.

The following figure shows the simplified diagram of the PWMG Capture/Compare Channel.

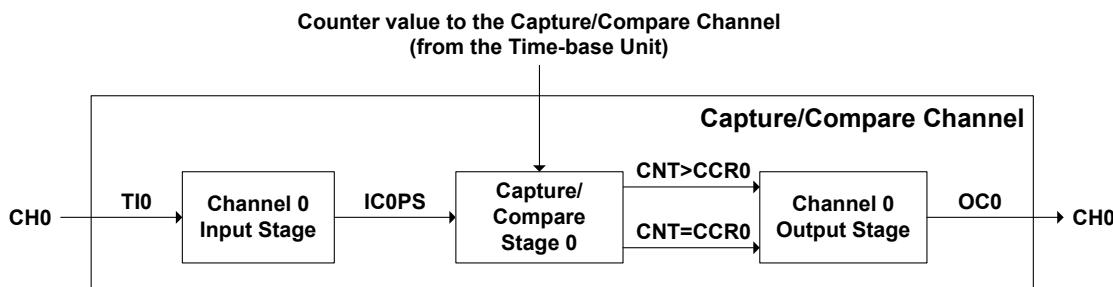


Figure 12-17 Capture/Compare Channel

### 12.2.1.3.1. Input Stage

The following figure shows the diagram of the Channel 0 Input Stage.

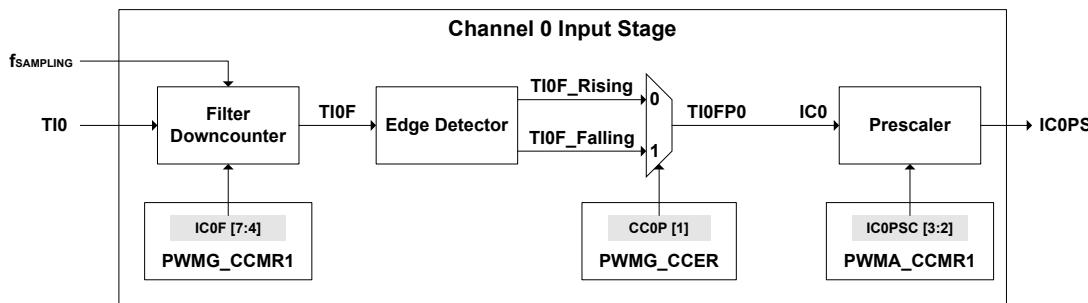


Figure 12-18 Channel 0 Input Stage

As the figure above shows, channel 0 input signal (`TIO`) 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, `TIO` signal is sampled with the sampling frequency (`fSAMPLING`) 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.



`TI0FP0` signal name reflects how it was received, where:

- `TIO` means that the signal was received on the channel 0 input;
- `F` means that the signal was resynchronized and passed through the digital filter
- `P0` shows that the active level of the signal is controlled through 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.

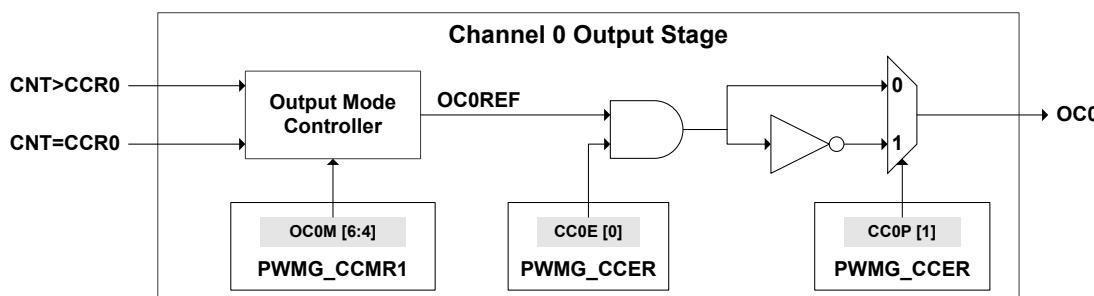


Figure 12-19 Channel 0 Output Stage

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.cc0m` 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 that 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

Capture/Compare Stage is used both when the channel operates in input mode and when the channel operates in output mode.



Channel works 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 works as input.

The figure below shows the diagram of the Capture/Compare Stage 0.

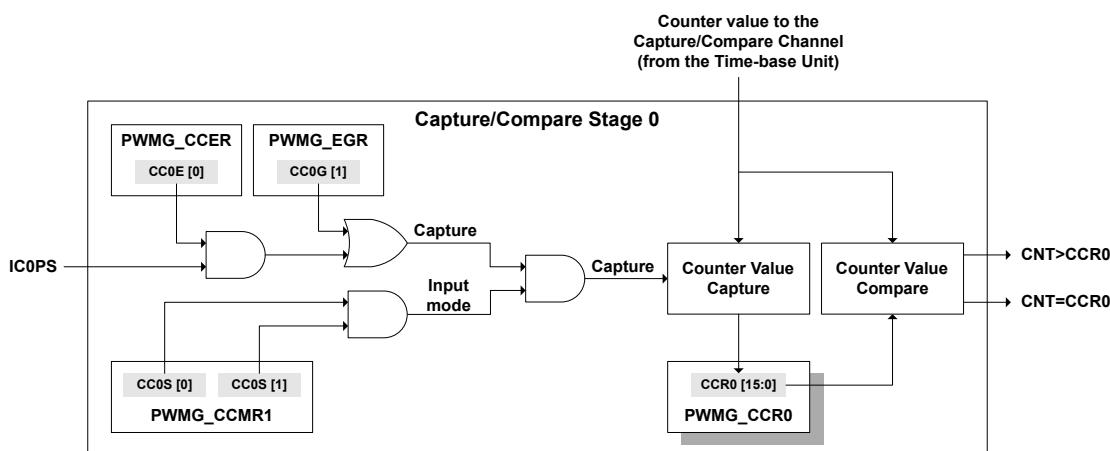


Figure 12-20 Capture/Compare Stage 0

#### 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 compare with the `PWMG_CNT` register value in the Counter Value Compare block. As a result, Counter Value Compare block generates the `CNT=CCR0` and `CNT>CCR0` output signals, where:

- `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 signals `CNT=CCR0` and `CNT>CCR0` are then used in the Channel 0 Output Stage.

#### 12.2.1.4. Shadow Registers

Some of the PWMG registers are buffered. Such registers are a bunch of:

- software accessible register (can also be called as preload register or buffer register)
- software inaccessible register (can also be called as shadow register or active register)

Active registers are used during the PWMG operations. They are updated with the values of the appropriate preload registers when the *Update Event (UEV)* occurs.

In the figures, the buffered registers are shown as rectangles with the gray background.

#### 12.2.2. PWMG Registers

Register regions of the PWMG controllers are mapped in the BE-U1000 microcontroller Memory Map as the following table shows.

**Table 12-35 Memory Address Regions for PWMG Controller Registers**

Region	Block Name	Size	Start Address	End Address
Periphery 0	PWMG_0	4KB	0x1100_A000	0x1100_AFFF
Periphery 1	PWMG_1	4KB	0x1300_A000	0x1300_AFFF



For details, refer to [Memory Map](#) chapter.

In this chapter:

- [Register Map](#)
- [Register and Field Descriptions](#)

#### 12.2.2.1. Registers Map

The following table provides 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 Registers Map**

Register	Offset	Description	Default Value
PWMG_CR1	0x00	<a href="#">PWMG Control Register 1</a>	0x0
PWMG_IER	0x0C	<a href="#">PWMG Interrupt Enable Register</a>	0x0
PWMG_SR	0x10	<a href="#">PWMG Status Register</a>	0x0
PWMG_EGR	0x14	<a href="#">PWMG Event Generation Register</a>	0x0
PWMG_CCMR1	0x18	<a href="#">PWMG Capture/Compare Mode Register 1</a>	0x0

**Table 12-36 PWMG Registers Map (continued)**

Register	Offset	Description	Default Value
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 of a register description 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 period of the sampling clock ( $t_S$ ). The sampling clock is used by the digital filter ( $TIO$ ). <b>Values:</b> 0x0: $t_S=t_{CK\_INT}$ 0x1: $t_S=2*t_{CK\_INT}$ 0x2: $t_S=4*t_{CK\_INT}$ 0x3: Reserved <b>Value After Reset:</b> 0x0
[7]	ARPE	R/W	Auto-reload preload enable <b>Values:</b> 0x0: <a href="#">PWMG_ARR</a> register is not buffered 0x1: <a href="#">PWMG_ARR</a> register is buffered <b>Value After Reset:</b> 0x0
[6:4]			Reserved (return 0 when read)
[3]	OPM	R/W	One pulse mode <b>Values:</b>

**Table 12-37 Fields For Register: `PWMG_CR1` (continued)**

Bits	Name	R/W	Description
			<p><b>0x0:</b> Counter is not stopped at the Update Event</p> <p><b>0x1:</b> Counter stops counting at the next Update Event (clearing the <code>CEN</code> bit)</p> <p><b>Value After Reset:</b> 0x0</p>
[2]			Reserved (return 0 when read)
[1]	UDIS	R/W	<p>Update disable</p> <p>This bit is set and cleared by software to enable/disable <i>Update Event (UEV)</i> generation.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> UEV enabled. The Update event is generated by one of the following events:           <ul style="list-style-type: none"> <li>◦ counter overflow;</li> <li>◦ setting the <code>PWMG_EGR.UG</code> bit.</li> </ul> </li> <li><b>0x1:</b> UEV disabled. The Update event is not generated, shadow registers keep their value (<code>ARR</code>, <code>PSC</code>, <code>CCR0</code>). However the counter and the prescaler are reinitialized if the <code>PWMG_EGR.UG</code> bit is set.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[0]	CEN	R/W	<p>Counter enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Counter disabled</li> <li><b>0x1:</b> Counter enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

#### 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	<p>Interrupt clear</p> <p>Write 0x1 to this bit to reset the setted interrupt. Writing 0x0 has no effect.</p> <p><b>Value After Reset:</b> 0x0</p>
[14:2]			Reserved
[1]	CC0IE	R/W	<p>Capture/Compare Channel 0 interrupt enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> CC0 interrupt disabled</li> <li><b>0x1:</b> CC0 interrupt enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

**Table 12-38 Fields For Register: `PWMG_IER` (continued)**

Bits	Name	R/W	Description
[0]	<code>UIE</code>	R/W	Update interrupt enable <b>Values:</b> <b>0x0:</b> Update interrupt disabled <b>0x1:</b> Update interrupt enabled <b>Value After Reset:</b> 0x0

### 12.2.2.3. 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]	<code>CC0OF</code>	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. <b>Values:</b> <b>0x0:</b> No overcapture has been detected <b>0x1:</b> The counter value has been captured in <code>PWMG_CCR0</code> register while <code>CC0IF</code> flag was already set <b>Value After Reset:</b> 0x0
[8:2]			Reserved
[1]	<code>CC0IF</code>	RW0C	Capture/Compare Channel 0 interrupt flag <b>If channel CC0 is configured as output</b> ( <code>PWMG_CCMR1.CC0S = 0x0</code> ): This flag is set by hardware when the counter matches the compare value. It is cleared by software. <b>Values:</b> <b>0x0:</b> No match <b>0x1:</b> The content of the counter <code>PWMG_CNT</code> matches the content of the <code>PWMG_CCR0</code> register. When the contents of <code>PWMG_CCR0</code> are greater than the contents of <code>PWMG_ARR</code> , the <code>CC0IF</code> bit goes high on the counter overflow <b>If channel CC0 is configured as input</b> ( <code>PWMG_CCMR1.CC0S = 0x3</code> ): This bit is set by hardware on a capture. It is cleared by software or by reading the <code>PWMG_CCR0</code> register. <b>Values:</b>

**Table 12-39 Fields For Register: [PWMG\\_SR](#) (continued)**

Bits	Name	R/W	Description
			<p><b>0x0:</b> No input capture occurred</p> <p><b>0x1:</b> The counter value has been captured in <a href="#">PWMG_CCR0</a> register (An edge has been detected on <a href="#">IC0</a> which matches the selected polarity)</p> <p><b>Value After Reset:</b> 0x0</p>
[0]	UIF	RW0C	<p>Update interrupt flag This bit is set by hardware on an Update Event. It is cleared by software.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> No update occurred</li> <li><b>0x1:</b> Update interrupt pending. This bit is set by hardware when the registers are updated:           <ul style="list-style-type: none"> <li>◦ at overflow, if <a href="#">UDIS</a>=0x0 in the <a href="#">PWMG_CRI</a> register;</li> <li>◦ when counter is reinitialized by software using the <a href="#">UG</a> bit in <a href="#">PWMG_EGR</a> register, if <a href="#">UDIS</a>=0x0 in the <a href="#">PWMG_CRI</a> register;</li> </ul> </li> </ul> <p><b>Value After Reset:</b> 0x0</p>

#### 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	<p>Capture/Compare 0 generation This bit is set by software in order to generate an event, it is automatically cleared by hardware.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> No action</li> <li><b>0x1:</b> A capture/compare event is generated on channel 0: <b>If channel CC0 is configured as output</b> (<a href="#">PWMG_CCMR1.CC0S</a> = 0x0): The <a href="#">PWMG_SR.CC0IF</a> flag is set. Corresponding interrupt is sent if enabled. <b>If channel CC0 is configured as input</b> (<a href="#">PWMG_CCMR1.CC0S</a> = 0x3): The current value of the counter is captured in <a href="#">PWMG_CCR0</a> register. The <a href="#">PWMG_SR.CC0IF</a> flag is set, the corresponding interrupt is sent if enabled. The <a href="#">PWMG_SR.CC0OF</a> flag is set if the <a href="#">CC0IF</a> flag was already high.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

**Table 12-40 Fields For Register: PWMG\_EGR (continued)**

Bits	Name	R/W	Description
[0]	UG	W	<p>Update generation This bit can be set by software, it is automatically cleared by hardware.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> No action</li> <li><b>0x1:</b> 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.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

#### 12.2.2.5. PWMG Capture/Compare Mode Register 1 (PWMG\_CCMR1)

The `PWMG_CCMR1` register is at offset 0x18.



The channel 0 can be used in input (capture mode) or in output (compare mode). The direction of a channel is defined by configuring the corresponding `CC0S` bit field. All the other bits of this register have a different function in input and in output mode. For a given bit, `OC0x` describes its function when the channel 0 is configured in output, `IC0x` describes its function when the channel 0 is configured in input. So the user must take care 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
[7:2]	<code>IC0*</code> or <code>OC0*</code>	R/W	<p> The functions of these bits depend on the value of the <code>CC0S</code> bit field. For details, see the table <code>PWMG_CCMR1[7:2] bits functions for different CC0S values</code> below.</p>
[1:0]	<code>CC0S</code>	R/W	<p>Capture/Compare Channel 0 selection This bit-field defines the direction of the channel 0 (input/output) as well as the used input.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> CC0 channel is configured as output</li> <li><b>0x1:</b> Reserved</li> <li><b>0x2:</b> Reserved</li> <li><b>0x3:</b> CC0 channel is configured as input, <code>IC0</code> is mapped on <code>TIO</code></li> </ul> <p> <code>CC0S</code> bits are writable only when the channel is OFF (<code>CC0E = 0x0</code> in <code>PWMG_CCER</code>)</p> <p><b>Value After Reset:</b> 0x0</p>

The table below describes the `PWMG_CCMR1[7:2]` bits functions for different `PWMG_CCMR1.CC0S` bit values

**Table 12-42 `PWMG_CCMR1[7:2]` bits functions for different `CC0S` values**

Bits	Name	R/W	Description
<b>Channel 0 Output Compare mode (<code>CC0S</code> = 0x0)</b>			
[7]			Reserved
[6:4]	OC0M	R/W	<p>Output Compare Channel 0 mode These bits define the behavior of the output reference signal <code>OC0REF</code> from which <code>OC0</code> is derived. <code>OC0REF</code> is active high whereas <code>OC0</code> active level depends on <code>CC0P</code> bit.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Frozen - The comparison between the output compare register <code>PWMG_CCR0</code> and the counter <code>PWMG_CNT</code> has no effect on the outputs (this mode is used to generate a timing base).</li> <li><b>0x1:</b> Set channel 0 to active level on match. <code>OC0REF</code> signal is forced high when the counter <code>PWMG_CNT</code> matches the <code>PWMG_CCR0</code> register.</li> <li><b>0x2:</b> Set channel 0 to inactive level on match. <code>OC0REF</code> signal is forced low when the counter <code>PWMG_CNT</code> matches the <code>PWMG_CCR0</code> register..</li> <li><b>0x3:</b> Toggle - <code>OC0REF</code> toggles when <code>PWMG_CNT=PWMG_CCR0</code>.</li> <li><b>0x4:</b> Force inactive level - <code>OC0REF</code> is forced low.</li> <li><b>0x5:</b> Force active level - <code>OC0REF</code> is forced high.</li> <li><b>0x6:</b> PWM mode 1 - Channel 0 is active as long as <code>PWMG_CNT&lt;PWMG_CCR0</code> else inactive.</li> <li><b>0x7:</b> PWM mode 2 - Channel 0 is inactive as long as <code>PWMG_CNT&lt;PWMG_CCR0</code> else active.</li> </ul> <p> In PWM mode 1 or 2, the <code>OC0REF</code> level changes only when the result of the comparison changes or when the output compare mode switches from “frozen” mode to “PWM” mode.</p> <p><b>Value After Reset:</b> 0x0</p>
[3]	OC0PE	R/W	<p>Output Compare Channel 0 preload enable</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Preload register on <code>PWMG_CCR0</code> disabled. <code>PWMG_CCR0</code> can be written at anytime, the new value is taken in account immediately.</li> <li><b>0x1:</b> Preload register on <code>PWMG_CCR0</code> enabled. Read/Write operations access the preload register. <code>PWMG_CCR0</code> preload value is loaded in the active register at each Update Event.</li> </ul> <p> The PWM mode can be used without validating the preload register only in one pulse mode (<code>OPM</code> bit set in <code>PWMG_CR1</code> register). Else the behavior is not guaranteed.</p> <p><b>Value After Reset:</b> 0x0</p>
[2]			Reserved
<b>Channel 0 Input Capture mode (<code>CC0S</code> = 0x3)</b>			
[7:4]	IC0F	R/W	<p>Input Capture Channel 0 filter This bit-field defines the frequency used to sample <code>TIO</code> input and the length of the digital filter applied to <code>TIO</code>. The digital filter is made of</p>

**Table 12-42 PWMG\_CCMR1[7:2] bits functions for different CC0S values (continued)**

Bits	Name	R/W	Description
			<p>an event counter in which N consecutive events are needed to validate a transition on the output.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> No filter, sampling is done at <math>f_S</math> (<math>f_{SAMPLING}=f_S</math>)</li> <li><b>0x1:</b> <math>f_{SAMPLING}=f_{CK\_INT}</math>, <math>N=2</math></li> <li><b>0x2:</b> <math>f_{SAMPLING}=f_{CK\_INT}</math>, <math>N=4</math></li> <li><b>0x3:</b> <math>f_{SAMPLING}=f_{CK\_INT}</math>, <math>N=8</math></li> <li><b>0x4:</b> <math>f_{SAMPLING}=f_S/2</math>, <math>N=6</math></li> <li><b>0x5:</b> <math>f_{SAMPLING}=f_S/2</math>, <math>N=8</math></li> <li><b>0x6:</b> <math>f_{SAMPLING}=f_S/4</math>, <math>N=6</math></li> <li><b>0x7:</b> <math>f_{SAMPLING}=f_S/4</math>, <math>N=8</math></li> <li><b>0x8:</b> <math>f_{SAMPLING}=f_S/8</math>, <math>N=6</math></li> <li><b>0x9:</b> <math>f_{SAMPLING}=f_S/8</math>, <math>N=8</math></li> <li><b>0xA:</b> <math>f_{SAMPLING}=f_S/16</math>, <math>N=5</math></li> <li><b>0xB:</b> <math>f_{SAMPLING}=f_S/16</math>, <math>N=6</math></li> <li><b>0xC:</b> <math>f_{SAMPLING}=f_S/16</math>, <math>N=8</math></li> <li><b>0xD:</b> <math>f_{SAMPLING}=f_S/32</math>, <math>N=5</math></li> <li><b>0xE:</b> <math>f_{SAMPLING}=f_S/32</math>, <math>N=6</math></li> <li><b>0xF</b> <math>f_{SAMPLING}=f_S/32</math>, <math>N=8</math></li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[3:2]	IC0PSC	R/W	<p>Input Capture Channel 0 prescaler</p> <p>This bit-field defines the ratio of the prescaler acting on CC0 input (IC0). The prescaler is reset as soon as CC0E=0x0 (<a href="#">PWMG_CCER</a> register).</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> no prescaler, capture is done each time an edge is detected on the capture input</li> <li><b>0x1:</b> capture is done once every 2 events</li> <li><b>0x2:</b> capture is done once every 4 events</li> <li><b>0x3:</b> capture is done once every 8 events</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

#### 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	<p>Capture/Compare Channel 0 output polarity</p> <p>If channel CC0 is configured as output (<a href="#">PWMG_CCMR1</a>.CC0S = 0x0):</p> <p><b>Values:</b></p>

**Table 12-43 Fields For Register: PWMG\_CCER (continued)**

Bits	Name	R/W	Description
			<p><b>0x0:</b> OC0 active high  <b>0x1:</b> OC0 active low</p> <p><b>If channel CC0 is configured as input (PWMG_CCMR1.CC0S = 0x3):</b>  Selects the TI0FPO active polarity.</p> <p><b>Values:</b></p> <p><b>0x0:</b> Non-inverted / rising edge  <b>0x1:</b> Inverted / falling edge</p> <p><b>Value After Reset:</b> 0x0</p>
[0]	CC0E	R/W	<p>Capture/Compare Channel 0 output enable</p> <p><b>If channel CC0 is configured as output (PWMG_CCMR1.CC0S = 0x0):</b></p> <p><b>Values:</b></p> <p><b>0x0:</b> Off - OC0 is not active.  <b>0x1:</b> On - OC0 signal is output on the corresponding output pin.</p> <p><b>If channel CC0 is configured as input (PWMG_CCMR1.CC0S = 0x3):</b>  This bit determines if a capture of the counter value can actually be done into the input capture/compare register 0 (PWMG_CCRO) or not.</p> <p><b>Values:</b></p> <p><b>0x0:</b> Capture disabled.  <b>0x1:</b> Capture enabled.</p> <p><b>Value After Reset:</b> 0x0</p>

#### 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	<p>Counter value</p> <p><b>Value After Reset:</b> 0x0</p>

#### 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	<p>Prescaler value</p> <p>The counter clock frequency (CK_CNT) is equal to fCK_INT / (PSC + 1).</p>

**Table 12-45 Fields For Register: `PWMG_PSC` (continued)**

Bits	Name	R/W	Description
			<p>PSC contains the value to be loaded in the active prescaler register at each Update Event (including when the counter is cleared through the <code>UG</code> bit of the <code>PWMG_EGR</code> register)</p> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p>Auto-reload value</p> <p>ARR is the value to be loaded in the actual auto-reload register.</p> <p> This register can be buffered (see <code>ARPE</code> bit description in the <code>PWMG_CR1</code> register).</p> <p>The counter is blocked while the Auto-reload value (ARR) is null.</p> <p><b>Value After Reset:</b> 0x0</p>

### 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`**

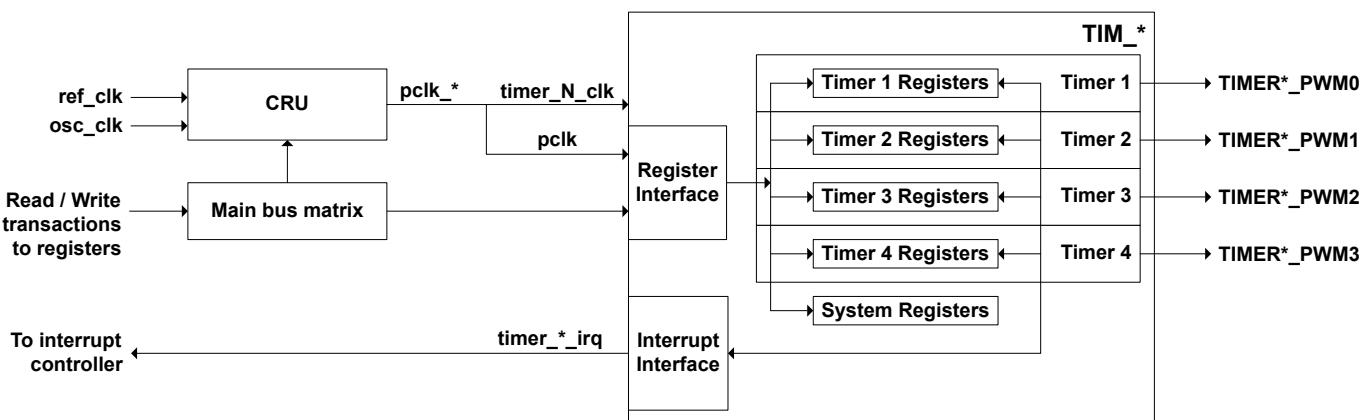
Bits	Name	R/W	Description
[15:0]	CCR0	R/W	<p>Capture/Compare Channel 0 value</p> <p><b>If channel CC0 is configured as output</b> (<code>PWMG_CCMR1.CC0S = 0x0</code>): CCR0 is the value to be loaded in the actual capture/compare 0 register (preload value). It is loaded permanently if the preload feature is not selected in the <code>PWMG_CCMR1</code> register (bit <code>OC0PE</code>). Else the preload value is copied in 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 <code>PWMG_CNT</code> and signaled on <code>OC0</code> output.</p> <p><b>If channel CC0 is configured as input</b> (<code>PWMG_CCMR1.CC0S = 0x3</code>): CCR0 is the counter value transferred by the last input capture 0 event (<code>IC0</code>). The <code>PWMG_CCR0</code> register is read-only and cannot be programmed.</p> <p><b>Value After Reset:</b> 0x0</p>

## 12.3. TIM

This chapter describes the TIM of the BE-U1000 microcontroller and provides information that is required to develop an application, such as a driver for the subsystem or bare-metal application.

The BE-U1000 microcontroller contains two TIMs.

Each TIM implements four identical but separately-programmable timers, which are accessed through the Register interface as the following diagram shows.



**Figure 12-21 TIM Functional block diagram**



In the figure above asterisk symbol (\*) indicates 0 for TIM\_0 and 1 for TIM\_1.

The TIM has the following features:

- The width of each timer is 32 bits
- Two operation modes: free-running and user-defined count
- Support independent clocking
- The width of LOW and HIGH periods of the TIM can be separately programmed via the TIM registers
- Include timer toggle output, which toggles whenever timer counter reloads
- Enable programmable pulse-width modulation of timer toggle outputs
- Include pulse width modulation of timer toggle output with 0% and 100% duty cycle.

### 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:

- Timer is enabled after being reset or disabled
- 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. To Use The Timer

The procedure illustrated in the following figure is a basic flow to follow when programming the Timer.

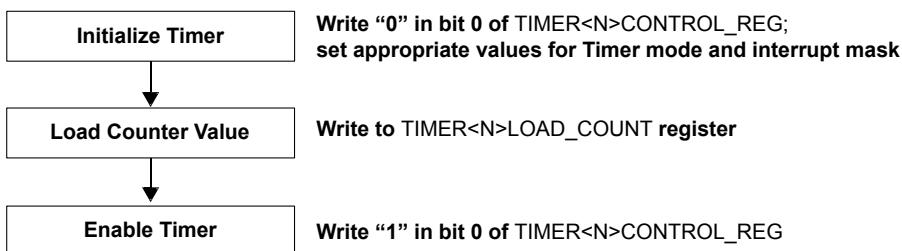


Figure 12-22 Timer Usage Flow



Before writing to the `TIMER<N>LOAD_COUNT` register, you have to 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 a 1 or 0, respectively, to the timer mode bit (bit 1).
  - c. Set the interrupt mask as either masked or not masked by writing a 1 or 0, respectively, to 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 a 1 to bit 0 of `TIMER<N>CONTROL_REG`.

#### 12.3.1.1.2. To Enable a Timer

You use bit 0 of the `TIMER<N>CONTROL_REG`, where N is in the range 1 to 4, to either enable or disable a timer.

If you want to enable a timer, you write 1 to bit 0 of its `TIMER<N>CONTROL_REG` register.

#### 12.3.1.1.3. To Disable a Timer

You use bit 0 of the `TIMER<N>CONTROL_REG`, where N is in the range 1 to 4, to either enable or disable a timer.

To disable a timer, write 0 to bit 0 of its `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. To Load 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** – 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 to bit 1 of `TIMER<N>CONTROL_REG`.
- **Free-running mode** – timer loads the maximum value, which is dependent on the timer 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. To Work 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 (`TIMER<N>EOI` or `TIMERS_EOI` the individual and global end-of-interrupt registers, respectively). 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 global `TIMERS_EOI` 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, you write 1 to `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 register bit `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` =  $(\text{TIMER_NLOADCOUNT2} + 1) * \text{timer\_N\_clk\_clock\_period}$
- Width of `timer_N_toggle LOW period` =  $(\text{TIMER_NLOADCOUNT2} + 1) * \text{timer\_N\_clk\_clock\_period}$

#### 12.3.1.1.6.2. Pulse Width Modulation with 0% and 100% Duty Cycle

The TIM supports the programming for 0% and 100% duty cycle pulse width modulation of 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 is 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` = `TIMERNLOADCOUNT2 * timer_N_clk clock period`
  - Width of `timer_N_toggle LOW period` = `TIMERNLOADCOUNT * 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 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](#), [TIMER<N>LOAD\\_COUNT2](#) Relationship**

Duty Cycle (%)	<a href="#">TIMER&lt;N&gt;LOAD_COUNT</a>	<a href="#">TIMER&lt;N&gt;LOAD_COUNT2</a>
0	X	0
1	99	1
2	98	2
3	97	3
4	96	4
...	...	...
96	4	96
97	3	97
98	2	98

**Table 12-48 Duty Cycle, `TIMER<N>LOAD_COUNT`, `TIMER<N>LOAD_COUNT2` Relationship (continued)**

Duty Cycle (%)	<code>TIMER&lt;N&gt;LOAD_COUNT</code>	<code>TIMER&lt;N&gt;LOAD_COUNT2</code>
99	1	99
100	0	100



When `TIMER<N>LOAD_COUNT`=0 and `TIMER<N>LOAD_COUNT2`=0, timer considers this as 100% by providing higher priority to the `TIMER<N>LOAD_COUNT`.

Following are some points that you must consider while configuring the pulse width modulation 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 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.

#### 12.3.1.1.6.3. Timer Pause Mode

The operation of a timer can be paused by asserting the respective `timer_N_pause` input signal, which is synchronized to the `timer_N_clk` domain.

#### 12.3.2. TIM Registers

Register regions of the TIMs are mapped in the BE-U1000 microcontroller 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	<code>TIM_0</code>	4KB	0x1100_7000	0x1100_7FFF
Periphery 1	<code>TIM_1</code>	4KB	0x1300_7000	0x1300_7FFF



For details, refer to [Memory Map](#) chapter.

In this chapter:

- [Registers Map](#)
- [Register Descriptions](#)

### 12.3.2.1. Registers 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 Registers Map**

Name	Offset	Description	Reset Value
TIMER_1_LOAD_COUNT	0x00	<a href="#">Timer N Load Count Register</a> 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	<a href="#">Timer N Load Count2 Register</a> Used to specify the value to be loaded into timer N when toggle output changes from 0 to 1	0x0
TIMER_2_LOAD_COUNT2	0xB4		
TIMER_3_LOAD_COUNT2	0xB8		
TIMER_4_LOAD_COUNT2	0xBC		
TIMER_1_CURRENT_VALUE	0x04	<a href="#">Timer N Current Value Register</a> Holds the Current Value of timer N	0xFFFFFFFF
TIMER_2_CURRENT_VALUE	0x18		
TIMER_3_CURRENT_VALUE	0x2C		

**Table 12-51 TIM Registers Map (continued)**

Name	Offset	Description	Reset Value
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 of a register description 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]	<code>TIMER&lt;N&gt;LOADCOUNT</code>	R/W	Value to be loaded into Timer N. This is the value from which counting commences. Any value written to this register is loaded into the associated timer. <b>Value After Reset:</b> 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 toggle output changes from 0 to 1.

The width of LOW and HIGH periods of the timer toggle outputs can be separately programmed via 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]	<code>TIMER&lt;N&gt;LOADCOUNT2</code>	R/W	Value to be loaded into timer N when <code>timer_N_toggle</code> output changes from 0 to 1. This value determines the width of the HIGH period of the <code>timer_N_toggle</code> output. <b>Value After Reset:</b> 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]	<code>TIMER&lt;N&gt;CURRENTVAL</code>	R	Current value of Timer. <b>Value After Reset:</b> 0xFFFFFFFF <b>Volatile:</b> 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_ON100PWM_EN	R/W	<p>Allows user to enable or disable the usage of the TIM 0% and 100% mode feature.</p> <p><b>0x0</b> : Timer 0% and 100% PWM duty cycle mode is disabled</p> <p><b>0x1</b>: Timer 0% and 100% PWM duty cycle mode is enabled</p> <p><b>Value After Reset:</b> 0x0</p>
[3]	TIMER_PWM	R/W	<p>Pulse Width Modulation of timer_N_toggle output</p> <p><b>0x0</b> : PWM for timer_N_toggle o/p is disabled</p> <p><b>0x1</b>: PWM for timer_N_toggle o/p is enabled</p> <p><b>Value After Reset:</b> 0x0</p>
[2]	TMR_INT_MASK	R/W	<p>Interrupt mask</p> <p><b>0x0</b>: Timer N interrupt is unmasked</p> <p><b>0x1</b>: Timer N interrupt is masked</p> <p><b>Value After Reset:</b> 0x0</p>
[1]	TMR_MODE	R/W	<p>Operating mode</p> <p><b>0x0</b>: Free Running mode of operation</p> <p><b>0x1</b>: User-Defined mode of operation</p> <p>For more information about these modes, see <a href="#">Timer operation</a></p> <p> You must set the TIMER&lt;N&gt;LOAD_COUNT to all 1s before enabling the Timer in free-running mode.</p> <p><b>Value After Reset:</b> 0x0</p>
[0]	TMR_EN	R/W	<p>Timer enable</p> <p><b>0x0</b>: disabled</p> <p><b>0x1</b>: enabled</p> <p><b>Value After Reset:</b> 0x0</p>

#### 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

**Table 12-56 Fields For Register: TIMER<N>EOI (continued)**

Bits	Name	R/W	Description
[0]	TIMER<N>EOI	R	<p>Reading from this register returns zero (0) and clears the interrupt from Timer N.</p> <p><b>Value After Reset:</b> 0x0</p>

#### 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	<p>Contains the interrupt status for Timer N</p> <p><b>0x0:</b> Timer N Interrupt is inactive</p> <p><b>0x1:</b> Timer N Interrupt is active</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>

#### 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	<p>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 after the interrupt mask has been applied. Reading from this register does not clear any active interrupts:</p> <p><b>0x0:</b> Timer interrupt is not active after masking</p> <p><b>0x1:</b> Timer interrupt is active after masking</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>

### 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]	<code>TIMERSEOI</code>	R	Reading this register returns zero (0) and clears all active interrupts. <b>Value After Reset:</b> 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]	<code>TIMERSRAWINTSTAT</code>	R	The register contains the unmasked interrupt status of all timers in the subsystem <b>0x0:</b> Timer interrupt is not active prior to masking <b>0x1:</b> Timer interrupt is active prior to masking <b>Value After Reset:</b> 0x0 <b>Volatile:</b> true

## 12.4. WDT

This chapter describes the *Watchdog Timer Controller* (below referred to as **WDT**) of the BE-U1000 microcontroller. The WDT is used to prevent system lockup that may be caused by conflicting parts or programs in the microcontroller. The WDT may be reset independently to the other components.

The generated by the WDT\_0 interrupt (`wdt_0_irq`) is passed to the Core 0/1 *Core-Local Interrupt Controller (CLIC)*. The generated by the WDT\_0 reset (`wdt0_sys_RST`) is passed to the Reset Generation 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 generated by the WDT\_1 interrupt (`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.

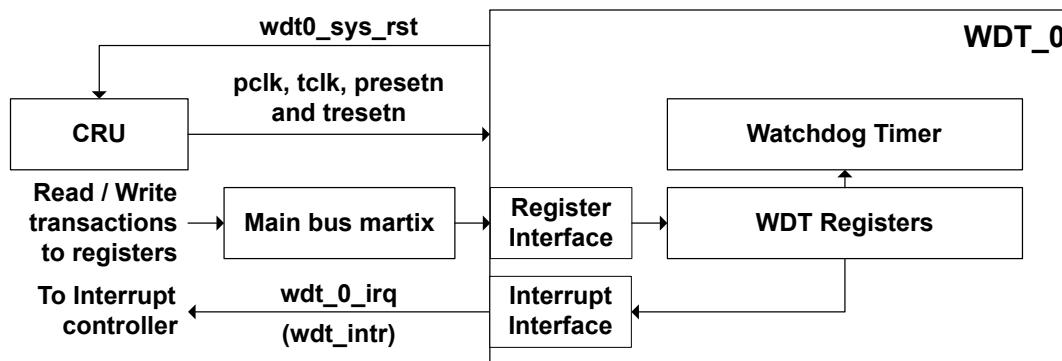


Figure 12-23 WDT\_0 Block Diagram

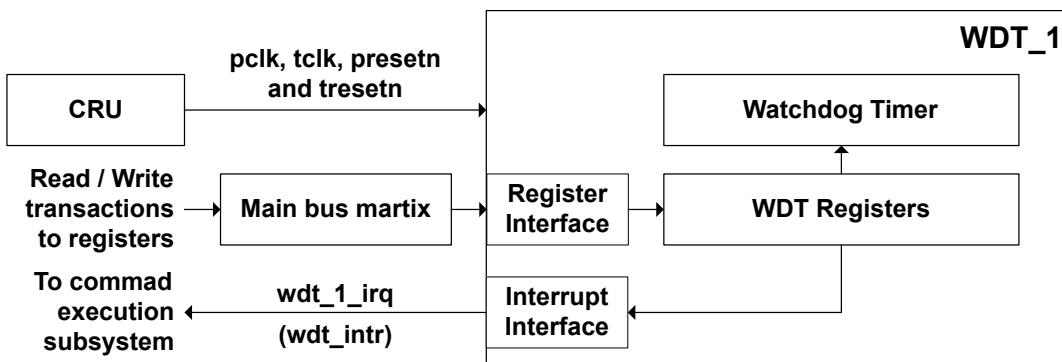


Figure 12-24 WDT\_1 Block Diagram

The WDT supports the following general features:

- Watchdog counter width 32 bits
- Counter counts down from a preset value to 0 to indicate the occurrence of a timeout
- If a timeout occurs, the WDT first generates an interrupt and if this is not cleared by the service routine by the time a second timeout occurs then generates a system reset
- Programmable timeout range (period)
- Programmable reset pulse length
- Prevention of accidental restart of the WDT counter
- Prevention of accidental disabling of the WDT

### 12.4.1. WDT Functional Description

This chapter describes the functional operation of the WDT.

In this chapter:

- [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 Watchdog Timer [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 WDT timers. The following table describes address regions for the WDT timers.

**Table 12-61 Memory Address Regions for WDT Registers**

Region	Block Name	Size	Start Address	End Address
Periphery 0	WDT_0	4KB	0x1100_8000	0x1100_8FFF
Periphery 1	WDT_1	4KB	0x1300_8000	0x1300_8FFF



For details, refer to [Memory Map](#) chapter.

In this chapter:

- [Registers Map](#)
- [Register Descriptions](#)

### 12.4.2.1. Registers Map

The following table provides 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 Registers Map**

Name	Offset	Description	Default Value
WDT_CR	0x0	<a href="#">Control Register</a>	0x40
WDT_TORR	0x4	<a href="#">Timeout Range Register</a>	0x4
WDT_CCVR	0x8	<a href="#">Current Counter Value Register</a>	0xFFFF
WDT_CRR	0xC	<a href="#">Counter Restart Register</a>	0x0
WDT_STAT	0x10	<a href="#">Interrupt Status Register</a>	0x0
WDT_EOI	0x14	<a href="#">Interrupt Clear Register</a>	0x0
WDT_COMP_PARAM_5	0xE4	<a href="#">Component Parameters Register 5</a>	0xFFFFFFFF
WDT_COMP_PARAM_4	0xE8	<a href="#">Component Parameters Register 4</a>	0x0
WDT_COMP_PARAM_3	0xEC	<a href="#">Component Parameters Register 3</a>	0x4
WDT_COMP_PARAM_2	0xF0	<a href="#">Component Parameters Register 2</a>	0xFFFF
WDT_COMP_PARAM_1	0xF4	<a href="#">Component Parameters Register 1</a>	0x10041280

### 12.4.2.2. Register Descriptions

The following sections contain the 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:

- **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	<p>Reset pulse length</p> <p>This is used to select the number of <code>pclk</code> cycles for which the system reset stays asserted. The range of values available is 2 to 256 <code>pclk</code> cycles.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>000:</b> 2 <code>pclk</code> cycles</li> <li><b>001:</b> 4 <code>pclk</code> cycles</li> <li><b>010:</b> 8 <code>pclk</code> cycles</li> <li><b>011:</b> 16 <code>pclk</code> cycles</li> <li><b>100:</b> 32 <code>pclk</code> cycles</li> <li><b>101:</b> 64 <code>pclk</code> cycles</li> <li><b>110:</b> 128 <code>pclk</code> cycles</li> <li><b>111:</b> 256 <code>pclk</code> cycles</li> </ul> <p><b>Value After Reset:</b> 0x4</p>
[1]	RMOD	R/W	<p>Response mode</p> <p>Selects the output response generated to a timeout.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Generate a system reset</li> <li><b>0x1:</b> First generate an interrupt and if it is not cleared by the time a second timeout occurs then generate a system reset</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[0]	WDT_EN	R/W	<p>WDT enable</p> <p>This bit is used to enable and disable the WDT.</p> <p>When disabled, the counter does not decrement. Thus, no interrupts or system resets are generated.</p> <p>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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> WDT disabled</li> <li><b>0x1:</b> WDT enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

#### 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

**Table 12-64 Fields For Register: WDT\_TORR (continued)**

Bits	Name	R/W	Description
			<p>This field is used to select the timeout period from which the watchdog counter restarts. A change of the timeout period takes effect only after the next counter restart (kick).</p> <p>The range of values is limited by the Width of WDT Counter (32 bit). If <b>TOP</b> is programmed to select a range that is greater than the counter width, the timeout period is truncated to fit to the counter width.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0: 0xFF</li> <li>1: 0x1FF</li> <li>2: 0x3FF</li> <li>3: 0x7FF</li> <li>4: 0xFFFF</li> <li>5: 0x1FFFF</li> <li>...</li> <li>14: 0x3FFFFFF</li> <li>15: 0x7FFFFFF</li> </ul> <p><b>Value After Reset:</b> 0x4</p>

#### 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	<p>Current Counter Value Register</p> <p>This register, when read, is the current value of the internal counter. This value is read coherently whenever it is read.</p> <p><b>Value After Reset:</b> 0xFFFF</p>

#### 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	<p>Counter Restart Register</p> <p>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.</p> <p>Reading this register returns zero.</p> <p><b>Value After Reset:</b> 0x0</p>

### 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]	<code>WDT_STAT</code>	R	Interrupt Status Register This register shows the interrupt status of the WDT. <b>Values:</b> <b>0x1:</b> Interrupt is active regardless of polarity <b>0x0:</b> Interrupt is inactive <b>Value After Reset:</b> 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]	<code>WDT_EOI</code>	R	Interrupt Clear Register Clears the watchdog interrupt. This can be used to clear the interrupt without restarting the watchdog counter. <b>Value After Reset:</b> 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]	<code>CP_WDT_USER_TOP_MAX</code>	R	Upper Limit of Timeout Period <b>Values:</b> <b>0xFF:</b> range 0 <b>0x1FF:</b> range 1 <b>0x3FF:</b> range 2 <b>0x7FF:</b> range 3 <b>0xFFFF:</b> range 4 <b>0x1FFF:</b> range 5 ...

**Table 12-69 Fields For Register: WDT\_COMP\_PARAM\_5 (continued)**

Bits	Name	R/W	Description
			<p><b>0x3FFFFFFF:</b> range 14</p> <p><b>0x7FFFFFFF:</b> range 15</p> <p><b>Value After Reset:</b> 0x7FFFFFFF</p>

#### 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	<p>Upper Limit of Initial Timeout Period</p> <p><b>Value After Reset:</b> 0x0</p>

#### 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	<p>WDT Default Counter value</p> <p>The value of this register is derived from the configuration settings that can be read in the <code>WDT_DFLT_TOP</code> field of the <code>WDT_COMP_PARAM_1</code> register.</p> <p><b>Value After Reset:</b> 0x4</p>

#### 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	<p>WDT Counter Value After Reset</p> <p><b>Value After Reset:</b> 0xFFFF</p>

#### 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	Width of WDT Counter <b>0x10:</b> 32 bit <b>Value After Reset:</b> 0x10
[23:20]	WDT_DFLT_TOP_INIT	R	Initial timeout period range selected as default <b>Value After Reset:</b> 0x0
[19:16]	WDT_DFLT_TOP	R	Main timeout period range selected as default <b>Value After Reset:</b> 0x4
[15:13]			Reserved
[12:10]	WDT_DFLT_RPL	R	Default reset pulse length <b>100:</b> 32 <code>pclk</code> cycles <b>Value After Reset:</b> 0x4
[9:8]	REG_IF_DATA_WIDTH	R	Register Interface data bus width <b>0x2:</b> 32 bit <b>Value After Reset:</b> 0x2
[7]	WDT_PAUSE	R	Pause enable input on interface <b>0x1:</b> Included  ⚠ <code>pause</code> signal tied off to 0 in the BE-U1000 <b>Value After Reset:</b> 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. <b>0x0:</b> No <b>Value After Reset:</b> 0x0
[5]	WDT_HC_TOP	R	Hard code timeout period range Since the parameter is set to 0, the timeout period is programmable. <b>0x0:</b> No <b>Value After Reset:</b> 0x0
[4]	WDT_HC_RPL	R	Hard code reset pulse length Since the parameter is set to 0, the reset pulse length is programmable. <b>0x0:</b> No <b>Value After Reset:</b> 0x0
[3]	WDT_HC_RMOD	R	Hard code output response mode

**Table 12-73 Fields For Register: WDT\_COMP\_PARAM\_1 (continued)**

Bits	Name	R/W	Description
			Since the parameter is set to 0, output response mode is programmable. <b>0x0:</b> No <b>Value After Reset:</b> 0x0
[2]	WDT_DUAL_TOP	R	A second timeout period for initialization is included <b>0x0:</b> false <b>Value After Reset:</b> 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. <b>0x0:</b> System reset only <b>Value After Reset:</b> 0x0
[0]	WDT_ALWAYS_EN	R	Enable WDT from reset Since the parameter is set to 0, the Watchdog timer disabled on reset. <b>0x0:</b> No <b>Value After Reset:</b> 0x0

## 13. Low Speed Peripherals

In this chapter:

- [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.

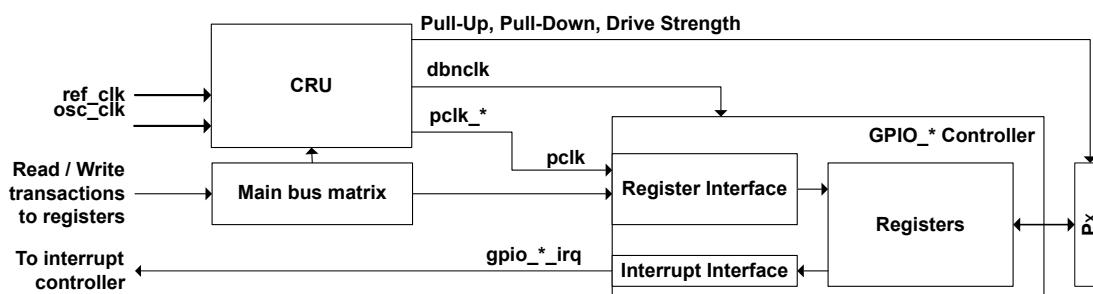


Figure 13-1 GPIO Functional block diagram



In the figure above,  $\text{x}$  shows corresponding I/O port, where  $\text{x} = \text{A}$  for  $\text{GPIO}_0$ ,  $\text{x} = \text{B}$  for  $\text{GPIO}_1$  and  $\text{x} = \text{C}$  for  $\text{GPIO}_2$ . Asterisk symbol (\*) indicates 0 for  $\text{GPIO}_0$ , 1 for  $\text{GPIO}_1$  and 2 for  $\text{GPIO}_2$ .

The GPIO controllers supports the following 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

In this chapter:

- [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.

In this section:

- [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\\_BOTEDGE\[n\]](#) 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	4KB	0x1100_9000	0x1100_9FFF
Periphery 1	GPIO_1	4KB	0x1300_9000	0x1300_9FFF
Periphery 2	GPIO_2	4KB	0x1400_8000	0x1400_8FFF



For details, refer to [Memory Map](#) chapter.

In this chapter:

- [Programming Considerations](#)
- [Registers 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. Registers Map

This section tables contain information about the register map.

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

**Table 13-2 GPIO Registers 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]	<code>GPIO_DR</code>	R/W	Output Data Register Values written to this register are output on the I/O signals if the <code>DDR</code> bits of the <code>GPIO_DDR</code> register are set to Output mode. <b>Value After Reset:</b> 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. <b>0x0:</b> Input data <b>0x1:</b> Output data <b>Value After Reset:</b> 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 <a href="#">GPIO_DR</a> to be configured for interrupts. <b>0x0:</b> Configure bit as normal GPIO signal (default) <b>0x1:</b> Configure bit as interrupt To disable interrupts on the corresponding bits set the value 1 to the corresponding bits of the <a href="#">GPIO_DDR</a> . <b>Value After Reset:</b> 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. <b>0x0:</b> Interrupt bits are unmasked (default) <b>0x1:</b> Mask interrupt <b>Value After Reset:</b> 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: <b>0x0:</b> Level-sensitive (default) <b>0x1:</b> Edge-sensitive <b>Value After Reset:</b> 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 <b>INTTYPE_LEVEL</b> value) that can occur on <a href="#">External Port Register (GPOI_EXT)</a> . 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. <b>0x0:</b> Active-low (default) <b>0x1:</b> Active-high <b>Value After Reset:</b> 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 <b>0x0:</b> Inactive (default) <b>0x1:</b> Active <b>Value After Reset:</b> 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]	<code>GPIO_INTSTATRAW</code>	R	Raw interrupt of status of GPIO Port signals (premasking bits) <b>0x0:</b> Inactive (default) <b>0x1:</b> Active <b>Value After Reset:</b> 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]	<code>GPIO_DEBOUNCE</code>	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. <b>0x0:</b> No debounce (default) <b>0x1:</b> Enables the denouncing circuitry  A signal must be valid for two periods of an external clock before it is internally processed. <b>Value After Reset:</b> 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]	<code>GPIO_EOI</code>	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 <b>0x0:</b> No interrupt clear <b>0x1:</b> Clear Interrupt <b>Value After Reset:</b> 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]	<code>GPIO_EXT</code>	R	This register always reflects the signals value on the External Port. <b>Value After Reset:</b> 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]	<code>GPIO_SYNC</code>	R/W	Synchronization level Level-sensitive interrupts synchronization to <code>pclk</code> : <b>0x0:</b> No synchronization to <code>pclk</code> (default) <b>0x1:</b> Synchronize to <code>pclk</code> <b>Value After Reset:</b> 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]	<code>GPIO_BOTHEDGE</code>	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 <code>GPIO_INTLVL</code> and <code>GPIO_INTPOLARITY</code> 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 <code>GPIO_INTLVL</code> and <code>GPIO_INTPOLARITY</code> registers. <b>0x0:</b> Single edge sensitive <b>0x1:</b> Both edge sensitive

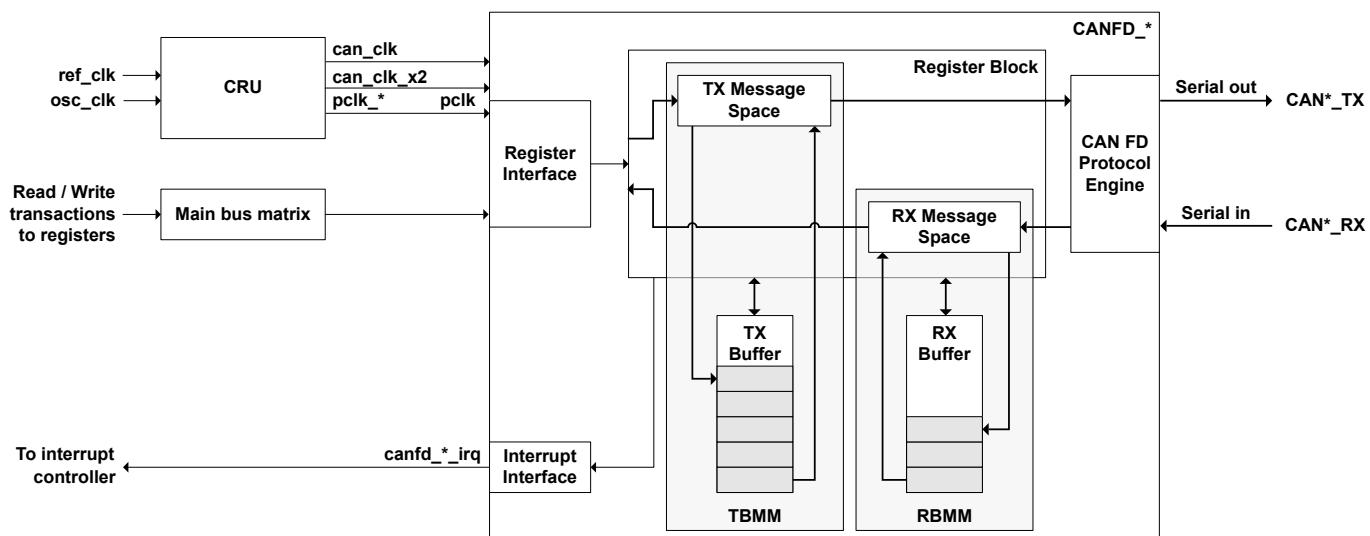
**Table 13-15 Fields For Register: GPIO\_BOTHEDGE (continued)**

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

## 13.2. CAN FD

There are two CAN FD in the BE-U1000.

A simplified block diagram of the CAN FD is shown in the following figure.

**Figure 13-2 CAN FD Functional block diagram**

In the figure above: 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 provides the following main 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 has the following 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

In this chapter:

- [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 State
- Bus-Off Recovery State

The following table defines the modes of operation with corresponding control and status bits.

**Table 13-16 CAN FD Operating Mode Transitions**

System (Hard) Reset	SRR Register Bits		MSR Register Bits			SR Register Bits						Operation Mode	
	SRST	CEN	LB ACK	SL EEP	SN OOP	CON FIG	BSFR _CON FIG	PEE _CON FIG	LB ACK	SL EEP	NOR MAL	SN OOP	
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
		0	0	0	0	0	0	0	0	0	1	0	Normal
		0	0	1	0	0	0	0	0	0	1	1	Snoop
		0	1	0	0	0	0	0	0	1	0	0	Sleep
		1	0	0	0	0	0	0	1	0	0	0	Loopback
		0	X	0	0	1	0	0	0	0	X	0	Bus-Off Recovery
		0	X	X	0	0	1	0	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 are 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 Bit Register](#) and [Arbitration Phase Baud Rate Prescaler Register](#) registers can 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 LBCK, SNOOP, and SLEEP bits in the [MSR](#) register.

## Normal Mode



The CAN FD can enter Normal mode only if SLEEP, LBCK 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	32KB	0x1101_0000	0x1101_7FFF
Periphery 1	CANFD_1	32KB	0x1301_0000	0x1301_7FFF



For details, refer to [Memory Map](#) chapter.

In this chapter:

- [Registers Map](#)
- [Register Descriptions](#)

### 13.2.2.1. Registers Map

The following tables provide 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 Registers Map**

Register	Offset	Description	Default Value
<b>Core Register Space</b>			
SRR	0x00	Software Reset Register	0x0
MSR	0x04	Mode Select Register	0x0
BRPR	0x08	Arbitration Phase Baud Rate Prescaler Register	0x0
BTR	0x0C	Arbitration Phase Bit 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
TxE_WMR	0xA4	TX Event FIFO Watermark Register	0xF

**Table 13-18 CAN FD Registers Map (continued)**

Register	Offset	Description	Default Value
AFR	0xE0	Acceptance Filter (Control) Register	0x0
FSR	0xE8	RX FIFO Status Register	0x0
WMR	0xEC	RX FIFO Watermark Register	0x1F0F0F
<b>TX Message Space</b>			
TBi_ID	0x100 - 0x9B8	TBi Message ID Register (for i = 0; i <=31)	
TBi_DLC	0x104 - 0x9BC	TBi Data Length Code Register (for i = 0; i <=31)	
TBi_DWN	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 - 0xAF0	Acceptance Filter i ID Registers (for i = 0; i <=31)	
<b>TX Event FIFO Status Register Space</b>			
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)	
<b>RX Message Space</b>			
RBi_ID	0x2100 - 0x32B8	RBi Message ID Register (for i = 0; i <=63)	
RBi_DLC	0x2104 - 0x32BC	RBi Data Length Code Register (for i = 0; i <=63)	
RBi_DWN	0x2108 - 0x32FC	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 of a register description 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 during CAN FD operation, recommended resetting the CAN FD so that operation starts over.

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	<p>CAN Enable. This is the Enable bit for the CAN FD.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> The CAN FD is in Configuration mode</li> <li><b>0x1:</b> The CAN FD is in Loopback, Sleep, Snoop, or Normal mode, depending on the <b>LBACK</b>, <b>SLEEP</b>, and <b>SNOOP</b> bits in the <b>MSR</b></li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[0]	SRST	W	<p>Reset. This is the software reset bit for the CAN FD.</p> <p><b>Value:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> CAN FD is reset.</li> </ul> <p>If a 1 is written to this bit, all CAN FD configuration registers (including the <b>SRR</b>) are reset. Reads to this bit always return 0.</p> <p> After performing a soft or hard reset, wait for 16 Register Interface clock cycles before initiating next Register Interface transaction.</p> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p>Internal Timing Optimization. Bit [11] of this register is used for Internal Timing Optimization Enable. Program this field if you intend to</p>

**Table 13-20 Fields For Register: `MSR` (continued)**

Bits	Name	R/W	Description
			<p>configure the nominal and data phase baud rate prescaler as 1 (register value 0). Only change this field when <code>CEN</code> = 0 in <code>SRR</code> register.</p> <p> For CAN FD operation with <code>BRPR.BRP</code>=1 (register value 0), CAN FD implementation makes it mandatory to choose <i>Baud Rate Prescaler (BRP)</i> for nominal and data phase as 1 (register value 0). Additionally, you need to program the <code>MSR</code>.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x00:</b> <code>BRPR.BRP</code> != 1</li> <li><b>0x08:</b> <code>BRPR.BRP</code>=1</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[7]	<code>ABR</code>	R/W	<p>Auto Bus-Off Recovery Request.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> No such request</li> <li><b>0x1:</b> Auto Bus-Off Recovery request</li> </ul> <p>If this bit is set, the node does auto Bus-Off Recovery irrespective of the <code>SBR</code> bit setting in this register. This bit can be written only when the <code>CEN</code> bit in <code>SRR</code> is 0.</p> <p><b>Value After Reset:</b> 0x0</p>
[6]	<code>SBR</code>	R/W	<p>Start Bus-Off Recovery Request.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> No such request</li> <li><b>0x1:</b> Start Bus-Off Recovery request</li> </ul> <p>Node stays in Bus-Off state until the <code>SBR</code> bit is set to 1 (providing that the <code>ABR</code> 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 <code>SRR.CEN</code> deassertion.</p> <p><b>Value After Reset:</b> 0x0</p>
[5]	<code>DPEE</code>	R/W	<p>Disable Protocol Exception Event Detection/Generation.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <i>Protocol Exception Event (PEE)</i> detection/generation is enabled. If the CAN FD receiver detects the <code>res</code> bit in CAN FD frame as 1, it goes to Bus Integration state (<code>SR.PEE_CONFIG</code>) and waits for Bus Idle condition (<code>SR.BIDLE</code>). The error counter remains unchanged.</li> <li><b>0x1:</b> Disable Protocol Exception Event detection/generation by CAN FD receiver if <code>res</code> bit in CAN FD frame is detected as 1. In this case, CAN FD receiver generates Form error.</li> </ul> <p>This bit can be written only when the <code>CEN</code> bit in <code>SRR</code> is 0.</p> <p><b>Value After Reset:</b> 0x0</p>

**Table 13-20 Fields For Register: `MSR` (continued)**

Bits	Name	R/W	Description
[4]	DAR	R/W	<p>Disable Auto-Retransmission.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Disable auto retransmission on the CAN bus to provide single shot transmission</li> <li><b>0x1:</b> Auto retransmission enabled</li> </ul> <p>This bit can be written only when the <code>CEN</code> bit in <code>SRR</code> is 0.</p> <p><b>Value After Reset:</b> 0x0</p>
[3]	BRSD	R/W	<p>CAN FD Bit Rate Switch Disable Override.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Makes the CAN FD transmit frames as per <code>BRS</code> bit in the <a href="#">TB* Data Length Code Register</a></li> <li><b>0x1:</b> Makes the CAN FD transmit frames only in nominal bit rate (by overriding the <a href="#">TB* Data Length Code Register</a> <code>BRS</code> bit setting)</li> </ul> <p>This bit can be written only when the <code>CEN</code> bit in <code>SRR</code> is 0.</p> <p><b>Value After Reset:</b> 0x0</p>
[2]	SNOOP	R/W	<p>SNOOP Mode Select/Request.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> No such request</li> <li><b>0x1:</b> Request CAN FD to be in Snoop mode</li> </ul> <p>This bit can be written only when the <code>CEN</code> bit in <code>SRR</code> is 0. Make sure that Snoop mode is programmed only after system reset or software reset. For the CAN FD to enter Snoop mode, <code>LBACK</code> and <code>SLEEP</code> bits in this register should be set to 0.</p> <p><b>Value After Reset:</b> 0x0</p>
[1]	LBACK	R/W	<p>Loopback Mode Select/Request.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> No such request</li> <li><b>0x1:</b> Request CAN FD to be in Loopback mode</li> </ul> <p>This bit can be written only when the <code>CEN</code> bit in <code>SRR</code> is 0. For the CAN FD to enter Loopback mode, <code>SLEEP</code> and <code>SNOOP</code> bits in this register should be set to 0.</p> <p><b>Value After Reset:</b> 0x0</p>
[0]	SLEEP	R/W	<p>Sleep Mode Select/Request.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> No such request</li> <li><b>0x1:</b> Request CAN FD to be in Sleep mode</li> </ul> <p>This bit is cleared when the CAN FD wakes up from Sleep mode. For the CAN FD to enter Sleep mode, <code>LBACK</code> and <code>SNOOP</code> bits in this register should be set to 0.</p> <p><b>Value After Reset:</b> 0x0</p>

### 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]	<b>BRP</b>	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 <code>CEN</code> bit in <code>SRR</code> is 0. <b>Value After Reset:</b> 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]	<b>SJW</b>	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 <code>CEN</code> bit in <code>SRR</code> is 0. <b>Value After Reset:</b> 0x0
[15]			Reserved
[14:8]	<b>TS2</b>	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 <code>CEN</code> bit in <code>SRR</code> is 0. <b>Value After Reset:</b> 0x0
[7:0]	<b>TS1</b>	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 <code>CEN</code> bit in <code>SRR</code> is 0. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 0x0
[7:0]	TEC	R	Transmit Error Count. Indicates the value of Transmit Error Counter. <b>Value After Reset:</b> 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. <b>Values:</b>  <b>0x0:</b> Indicates a bit error has not occurred in Data Phase (Fast) data rate after the last write to this bit <b>0x1:</b> 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).  <b>Value After Reset:</b> 0x0
[10]	F_STER	R/W	Stuff Error in CAN FD Data Phase. <b>Values:</b>  <b>0x0:</b> Indicates stuff error has not occurred in Data Phase (Fast) data rate. after the last write to this bit. <b>0x1:</b> Indicates stuff error occurred in Data Phase (Fast) data rate.  If this bit is set, writing a 1 clears it. <b>Value After Reset:</b> 0x0

**Table 13-24 Fields For Register: ESR (continued)**

Bits	Name	R/W	Description
[9]	F_FMER	R/W	<p>Form Error in CAN FD Data Phase.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Indicates form error has not occurred in Data Phase (Fast) data rate. after the last write to this bit.</li> <li><b>0x1:</b> Indicates form error occurred in Data Phase (Fast) data rate.</li> </ul> <p>If this bit is set, writing a 1 clears it.</p> <p><b>Value After Reset:</b> 0x0</p>
[8]	F_CRCER	R/W	<p>CRC Error in CAN FD Data Phase.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Indicates CRC error has not occurred in Data Phase (Fast) data rate after the last write to this bit.</li> <li><b>0x1:</b> Indicates CRC error occurred in Data Phase (Fast) data rate.</li> </ul> <p>If this bit is set, writing a 1 clears it.</p> <p><b>Value After Reset:</b> 0x0</p>
[7:5]			Reserved
[4]	ACKER	R/W	<p>ACK Error.</p> <p>Indicates an acknowledgment error.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Indicates an acknowledgment error has not occurred on the bus after the last write to this bit.</li> <li><b>0x1:</b> Indicates an acknowledgment error has occurred.</li> </ul> <p>If this bit is set, writing a 1 clears it.</p> <p><b>Value After Reset:</b> 0x0</p>
[3]	BERR	R/W	<p>Bit Error.</p> <p>Indicates the received bit is not the same as the transmitted bit during bus communication.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Indicates a bit error has not occurred on the bus after the last write to this bit.</li> <li><b>0x1:</b> Indicates a bit error has occurred.</li> </ul> <p>If this bit is set, writing a 1 clears it.</p> <p><b>Value After Reset:</b> 0x0</p>
[2]	STER	R/W	<p>Stuff Error.</p> <p>Indicates an error if there is a stuffing violation.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Indicates a stuff error has not occurred on the bus after the last write to this bit.</li> <li><b>0x1:</b> Indicates a stuff error has occurred.</li> </ul> <p>If this bit is set, writing a 1 clears it.</p> <p><b>Value After Reset:</b> 0x0</p>
[1]	FMER	R/W	Form Error.

**Table 13-24 Fields For Register: ESR (continued)**

Bits	Name	R/W	Description
			<p>Indicates an error in one of the fixed form fields in the message frame.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Indicates a form error has not occurred on the bus after the last write to this bit.</li> <li><b>0x1:</b> Indicates a form error has occurred.</li> </ul> <p>If this bit is set, writing a 1 clears it.</p> <p><b>Value After Reset:</b> 0x0</p>
[0]	CRCER	R/W	<p>CRC Error.</p> <p>Indicates a CRC error has occurred.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Indicates a CRC error has not occurred on the bus after the last write to this bit.</li> <li><b>0x1:</b> Indicates a CRC error has occurred.</li> </ul> <p>If this bit is set, writing a 1 clears it.</p> <p> In case of a CRC Error and a CRC delimiter corruption, only the FMER bit is set.</p> <p><b>Value After Reset:</b> 0x0</p>

#### 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	<p>Transmitter Delay Compensation Value.</p> <p>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 FD clocks. This field is for status purposes.</p> <p><b>Value After Reset:</b> 0x0</p>
[15:13]			Reserved
[12]	SNOOP	R	<p>Snoop Mode.</p> <p><b>Value:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Indicates controller is in Snoop mode provided Normal mode bit is also set in this register.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[11]			Reserved
[10]	BSFR_CONFIG	R	<p>Bus-Off Recovery Mode Indicator.</p> <p><b>Value:</b></p>

**Table 13-25 Fields For Register: SR (continued)**

Bits	Name	R/W	Description
			<p><b>0x1:</b> Indicates the CAN FD is in Bus-Off Recovery mode (Bus Integration State)</p> <p>When this bit is set, the <b>BBSY</b> and <b>NORMAL</b> status bits in this register do not mean anything.</p> <p><b>Value After Reset:</b> 0x0</p>
[9]	PEE_CONFIG	R	<p>PEE Mode Indicator.</p> <p><b>Value:</b></p> <p><b>0x1:</b> Indicates the CAN FD is in PEE mode (Bus Integration State).</p> <p>When this bit is set, the <b>BBSY</b> and <b>NORMAL</b> status bits in this register do not mean anything.</p> <p><b>Value After Reset:</b> 0x0</p>
[8:7]	ESTAT	R	<p>Error Status.</p> <p>Indicates the error status of the CAN FD.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x00:</b> Indicates Configuration mode (<b>CONFIG</b> = 1). Error state is undefined</li> <li><b>0x01:</b> Indicates error active state.</li> <li><b>0x11:</b> Indicates error passive state.</li> <li><b>0x10:</b> Indicates Bus-Off state.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[6]	ERRWRN	R	<p>Error Warning.</p> <p>Indicates that either the Transmit Error counter or the Receive Error counter has exceeded a value of 96.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> one or more error counters have a value of 96</li> <li><b>0x0:</b> neither of the error counters has a value of 96.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[5]	BBSY	R	<p>Indicates the CAN bus status.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Indicates that the CAN FD is either receiving a message or transmitting a message</li> <li><b>0x0:</b> Indicates that the CAN FD is either in Configuration mode or the bus is idle</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[4]	BIDLE	R	<p>Bus Idle.</p> <p>Indicates the CAN bus status.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Indicates no bus communication is taking place</li> <li><b>0x0:</b> Indicates the CAN FD is either in Configuration mode or the bus is busy</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[3]	NORMAL	R	Normal Mode.

**Table 13-25 Fields For Register: SR (continued)**

Bits	Name	R/W	Description
			<p>Indicates that the CAN FD is in Normal mode.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Indicates that the CAN FD is in Normal mode.</li> <li><b>0x0:</b> Indicates that the CAN FD is not in Normal mode.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[2]	SLEEP	R	<p>Sleep Mode.</p> <p>Indicates that the CAN FD is in Sleep mode.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Indicates that the CAN FD is in Sleep mode</li> <li><b>0x0:</b> Indicates that the CAN FD is not in Sleep mode.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[1]	LBACK	R	<p>Loopback Mode.</p> <p>Indicates that the CAN FD is in Loopback mode.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Indicates that the CAN FD is in Loopback mode.</li> <li><b>0x0:</b> Indicates that the CAN FD is not in Loopback mode.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[0]	CONFIG	R	<p>Configuration Mode Indicator.</p> <p>Indicates that the CAN FD is in Configuration mode.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Indicates that the CAN FD is in Configuration mode.</li> <li><b>0x0:</b> Indicates that the CAN FD is not in Configuration mode.</li> </ul> <p><b>Value After Reset:</b> 0x1</p>

#### 13.2.2.1.8. 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	<p>TX Event FIFO Watermark Full Interrupt.</p> <p><b>Value:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Indicates that TX Event FIFO is full based on watermark programming.</li> </ul> <p>The interrupt continues to assert as long as the TX Event FIFO Fill Level is above TX Event FIFO Full watermark.</p>

**Table 13-26 Fields For Register: ISR (continued)**

Bits	Name	R/W	Description
			<p>This bit can be cleared by writing to the respective bit in the <a href="#">ICR</a>.</p> <p><b>Value After Reset:</b> 0x0</p>
[30]	TXEOLW	R	<p>TX Event FIFO Overflow Interrupt.</p> <p><b>Value:</b></p> <p><b>0x1:</b> 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.</p> <p>This bit can be cleared by writing to the respective bit in the <a href="#">ICR</a>. This bit is also cleared when a 0 is written to the <a href="#">CEN</a> bit in <a href="#">SRR</a>.</p> <p><b>Value After Reset:</b> 0x0</p>
[29:18]			Reserved
[17]	RXMNF	R	<p>RX Match Not Finished.</p> <p><b>Value:</b></p> <p><b>0x1:</b> Indicates that Match process did not finish until the start of sixth bit in <a href="#">EOF</a> field and frame was discarded.</p> <p>This bit can be cleared by writing to the respective bit in the <a href="#">ICR</a>.</p> <p><b>Value After Reset:</b> 0x0</p>
[16:15]			Reserved
[14]	TXCRS	R	<p>TX Cancellation Request Served Interrupt.</p> <p><b>Value:</b></p> <p><b>0x1:</b> Indicates that a cancellation request was cleared.</p> <p>This bit can be cleared by writing to the respective bit in the <a href="#">ICR</a>.</p> <p><b>Value After Reset:</b> 0x0</p>
[13]	TXRRS	R	<p>TX Buffer Ready Request Served Interrupt.</p> <p><b>Value:</b></p> <p><b>0x1:</b> Indicates that a Buffer Ready request was cleared.</p> <p>This bit can be cleared by writing to the respective bit in the <a href="#">ICR</a>.</p> <p><b>Value After Reset:</b> 0x0</p>
[12]	RXFWMFL	R	<p>RX FIFO-0 Watermark Full Interrupt.</p> <p><b>Value:</b></p> <p><b>0x1:</b> Indicates that RX FIFO-0 is full based on watermark programming.</p> <p>The interrupt continues to assert as long as the RX FIFO-0 Fill Level is above RX FIFO-0 Full watermark.</p> <p>This bit can be cleared by writing to the respective bit in the <a href="#">ICR</a>.</p>

**Table 13-26 Fields For Register: ISR (continued)**

Bits	Name	R/W	Description
			<b>Value After Reset:</b> 0x0
[11]	WKUP	R	<p>Wake-Up Interrupt.</p> <p><b>Value:</b></p> <p><b>0x1:</b> Indicates that the core entered Normal mode from Sleep mode.</p> <p>This bit can be cleared by writing to the respective bit in the <a href="#">ICR</a>. This bit is also cleared when a 0 is written to the <a href="#">CEN</a> bit in <a href="#">SRR</a>.</p> <p><b>Value After Reset:</b> 0x0</p>
[10]	SLP	R	<p>Sleep Interrupt.</p> <p><b>Value:</b></p> <p><b>0x1:</b> Indicates that the CAN core entered Sleep mode.</p> <p>This bit can be cleared by writing to the respective bit in the <a href="#">ICR</a>. This bit is also cleared when a 0 is written to the <a href="#">CEN</a> bit in <a href="#">SRR</a>.</p> <p><b>Value After Reset:</b> 0x0</p>
[9]	BSOFF	R	<p>Bus-Off Interrupt.</p> <p><b>Value:</b></p> <p><b>0x1:</b> Indicates that the CAN core entered the Bus-Off state.</p> <p>This bit can be cleared by writing to the respective bit in the <a href="#">ICR</a>. This bit is also cleared when a 0 is written to the <a href="#">CEN</a> bit in <a href="#">SRR</a>.</p>
[8]	ERROR	R	<p>Error Interrupt.</p> <p><b>Value:</b></p> <p><b>0x1:</b> Indicates that an error occurred during message transmission or reception.</p> <p>This bit can be cleared by writing to the respective bit in the <a href="#">ICR</a>. This bit is also cleared when a 0 is written to the <a href="#">CEN</a> bit in <a href="#">SRR</a>.</p> <p><b>Value After Reset:</b> 0x0</p>
[7]			Reserved
[6]	RXFOFLW	R	<p>RX FIFO-0 Overflow Interrupt.</p> <p><b>Value:</b></p> <p><b>0x1:</b> 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.</p> <p>This bit can be cleared by writing to the respective bit in the <a href="#">ICR</a>. This bit is also cleared when a 0 is written to the <a href="#">CEN</a> bit in <a href="#">SRR</a>.</p> <p><b>Value After Reset:</b> 0x0</p>
[5]	TSCNT_OFLW	R	<p>Timestamp Counter Overflow Interrupt.</p> <p><b>Values:</b></p>

**Table 13-26 Fields For Register: ISR (continued)**

Bits	Name	R/W	Description
			<p><b>0x1:</b> Indicates that Timestamp counter rolled over (from 0xFFFF to 0x0). This bit can be cleared by writing to the respective bit in the <a href="#">ICR</a>.            This bit is also cleared when a 0 is written to the <a href="#">CEN</a> bit in <a href="#">SRR</a>.</p> <p><b>Value After Reset:</b> 0x0</p>
[4]	RXOK	R	<p>New Message Received Interrupt.  <b>Value:</b></p> <p><b>0x1:</b> 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 <a href="#">ICR</a>. This bit is also cleared when a 0 is written to the <a href="#">CEN</a> bit in <a href="#">SRR</a>.</p> <p> In Loopback mode, both <a href="#">TXOK</a> and <a href="#">RXOK</a> bits are set. The <a href="#">RXOK</a> bit is set before the <a href="#">TXOK</a> bit.</p> <p><b>Value After Reset:</b> 0x0</p>
[3]	BSFRD	R	<p>Bus-Off Recovery Done Interrupt.  <b>Value:</b></p> <p><b>0x1:</b> Indicates that the CAN FD recovered from Bus-Off state.            This bit can be cleared by writing to the respective bit in the <a href="#">ICR</a>. This bit is also cleared when a 0 is written to the <a href="#">CEN</a> bit in <a href="#">SRR</a>.</p> <p><b>Value After Reset:</b> 0x0</p>
[2]	PEE	R	<p>Protocol Exception Event Interrupt.  <b>Value:</b></p> <p><b>0x1:</b> Indicates that the CAN FD receiver has detected PEE event.            This bit can be cleared by writing to the respective bit in the <a href="#">ICR</a>. This bit is also cleared when a 0 is written to the <a href="#">CEN</a> bit in <a href="#">SRR</a>.</p> <p><b>Value After Reset:</b> 0x0</p>
[1]	TXOK	R	<p>Transmission Successful Interrupt.  <b>Value:</b></p> <p><b>0x1:</b> Indicates that a message was transmitted successfully.            This bit can be cleared by writing to the respective bit in the <a href="#">ICR</a>. This bit is also cleared when a 0 is written to the <a href="#">CEN</a> bit in <a href="#">SRR</a>.</p> <p> In Loopback mode, both <a href="#">TXOK</a> and <a href="#">RXOK</a> bits are set. The <a href="#">RXOK</a> bit is set before the <a href="#">TXOK</a> bit.</p> <p><b>Value After Reset:</b> 0x0</p>

**Table 13-26 Fields For Register: ISR (continued)**

Bits	Name	R/W	Description
[0]	ARBLST	R	<p>Arbitration Lost Interrupt.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Indicates that arbitration was lost during message transmission.</li> </ul> <p>This bit can be cleared by writing to the respective bit in the <a href="#">ICR</a>. This bit is also cleared when a 0 is written to the <a href="#">CEN</a> bit in <a href="#">SRR</a>.</p> <p><b>Value After Reset:</b> 0x0</p>

### 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]	ETXEWWMFL	R/W	<p>TX Event FIFO Watermark Full Interrupt Enable.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Disables interrupt generation if <a href="#">TXEWWMFL</a> bit in the <a href="#">ISR</a> is set.</li> <li><b>0x1:</b> Enables interrupt generation if <a href="#">TXEWWMFL</a> bit in the <a href="#">ISR</a> is set.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[30]	ETXEOF	R/W	<p>TX Event FIFO Overflow Interrupt Enable.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Disables interrupt generation if <a href="#">TXEOF</a> bit in the <a href="#">ISR</a> is set.</li> <li><b>0x1:</b> Enables interrupt generation if <a href="#">TXEOF</a> bit in the <a href="#">ISR</a> is set.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[29:18]			Reserved
[17]	ERXMNF	R/W	<p>RX Match Not Finished interrupt Enable.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Disables interrupt generation if <a href="#">RXMN</a> bit in the <a href="#">ISR</a> is set.</li> <li><b>0x1:</b> Enables interrupt generation if <a href="#">RXMN</a> bit in the <a href="#">ISR</a> is set.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[16:15]			Reserved
[14]	ETXCRS	R/W	<p>TX Cancellation Request Served Interrupt Enable.</p> <p><b>Values:</b></p>

**Table 13-27 Fields For Register: IER (continued)**

Bits	Name	R/W	Description
			<p><b>0x0:</b> Disables interrupt generation if <code>TXCRS</code> bit in the <code>ISR</code> is set.</p> <p><b>0x1:</b> Enables interrupt generation if <code>TXCRS</code> bit in the <code>ISR</code> is set.</p> <p><b>Value After Reset:</b> 0x0</p>
[13]	ETXRRS	R/W	<p>TX Buffer Ready Request Served Interrupt Enable.</p> <p><b>Values:</b></p> <p><b>0x0:</b> Disables interrupt generation if <code>TXRRS</code> bit in the <code>ISR</code> is set.</p> <p><b>0x1:</b> Enables interrupt generation if <code>TXRRS</code> bit in the <code>ISR</code> is set.</p> <p><b>Value After Reset:</b> 0x0</p>
[12]	ERXFWMFLL	R/W	<p>RX FIFO-0 Watermark Full Interrupt Enable (Sequential/FIFO Mode).</p> <p><b>Values:</b></p> <p><b>0x0:</b> Disables interrupt generation if <code>RXFWMFLL</code> bit in the <code>ISR</code> is set.</p> <p><b>0x1:</b> Enables interrupt generation if <code>RXFWMFLL</code> bit in the <code>ISR</code> is set.</p> <p><b>Value After Reset:</b> 0x0</p>
[11]	EWKUP	R/W	<p>Wake-Up Interrupt Enable.</p> <p><b>Values:</b></p> <p><b>0x0:</b> Disables interrupt generation if <code>WKUP</code> bit in the <code>ISR</code> is set.</p> <p><b>0x1:</b> Enables interrupt generation if <code>WKUP</code> bit in the <code>ISR</code> is set.</p> <p><b>Value After Reset:</b> 0x0</p>
[10]	ESLP	R/W	<p>Sleep Interrupt Enable.</p> <p><b>Values:</b></p> <p><b>0x0:</b> Disables interrupt generation if <code>SLP</code> bit in the <code>ISR</code> is set.</p> <p><b>0x1:</b> Enables interrupt generation if <code>SLP</code> bit in the <code>ISR</code> is set.</p> <p><b>Value After Reset:</b> 0x0</p>
[9]	EBSOFF	R/W	<p>Bus-Off Interrupt Enable.</p> <p><b>Values:</b></p> <p><b>0x0:</b> Disables interrupt generation if <code>BSOFF</code> bit in the <code>ISR</code> is set.</p> <p><b>0x1:</b> Enables interrupt generation if <code>BSOFF</code> bit in the <code>ISR</code> is set.</p> <p><b>Value After Reset:</b> 0x0</p>
[8]	ERROR	R/W	<p>Error Interrupt Enable.</p> <p><b>Values:</b></p>

**Table 13-27 Fields For Register: IER (continued)**

Bits	Name	R/W	Description
			<p><b>0x0:</b> Disables interrupt generation if <code>ERROR</code> bit in the <code>ISR</code> is set.</p> <p><b>0x1:</b> Enables interrupt generation if <code>ERROR</code> bit in the <code>ISR</code> is set.</p> <p><b>Value After Reset:</b> 0x0</p>
[7]			Reserved
[6]	ERFXOFLW	R/W	<p>RX FIFO-0 Overflow Interrupt Enable (Sequential/FIFO Mode).</p> <p><b>Values:</b></p> <p><b>0x0:</b> Disables interrupt generation if <code>RFXOFLW</code> bit in the <code>ISR</code> is set.</p> <p><b>0x1:</b> Enables interrupt generation if <code>RFXOFLW</code> bit in the <code>ISR</code> is set.</p> <p><b>Value After Reset:</b> 0x0</p>
[5]	ETSCNT_OFLW	R/W	<p>Timestamp Counter Overflow Interrupt Enable.</p> <p><b>Values:</b></p> <p><b>0x0:</b> Disables interrupt generation if <code>TSCNT_OFLW</code> bit in the <code>ISR</code> is set.</p> <p><b>0x1:</b> Enables interrupt generation if <code>TSCNT_OFLW</code> bit in the <code>ISR</code> is set.</p> <p><b>Value After Reset:</b> 0x0</p>
[4]	ERXOK	R/W	<p>New Message Received Interrupt Enable.</p> <p><b>Values:</b></p> <p><b>0x0:</b> Disables interrupt generation if <code>RXOK</code> bit in the <code>ISR</code> is set.</p> <p><b>0x1:</b> Enables interrupt generation if <code>RXOK</code> bit in the <code>ISR</code> is set.</p> <p><b>Value After Reset:</b> 0x0</p>
[3]	EBSFRD	R/W	<p>Bus-Off Recovery Done Interrupt Enable.</p> <p><b>Values:</b></p> <p><b>0x0:</b> Disables interrupt generation if <code>BSFRD</code> bit in the <code>ISR</code> is set.</p> <p><b>0x1:</b> Enables interrupt generation if <code>BSFRD</code> bit in the <code>ISR</code> is set.</p> <p><b>Value After Reset:</b> 0x0</p>
[2]	EPEE	R/W	<p>Protocol Exception Event Interrupt Enable.</p> <p><b>Values:</b></p> <p><b>0x0:</b> Disables interrupt generation if <code>PEE</code> bit in the <code>ISR</code> is set.</p> <p><b>0x1:</b> Enables interrupt generation if <code>PEE</code> bit in the <code>ISR</code> is set.</p> <p><b>Value After Reset:</b> 0x0</p>

**Table 13-27 Fields For Register: IER (continued)**

Bits	Name	R/W	Description
[1]	ETXOK	R/W	<p>Transmission Successful Interrupt Enable.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Disables interrupt generation if TXOK bit in the ISR is set.</li> <li><b>0x1:</b> Enables interrupt generation if TXOK bit in the ISR is set.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[0]	EARBLOST	R/W	<p>Arbitration Lost Interrupt Enable.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Disables interrupt generation if ARBLST bit in the ISR is set.</li> <li><b>0x1:</b> Enables interrupt generation if ARBLST bit in the ISR is set.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

#### 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	<p>Clear TX Event FIFO Watermark Full Interrupt.</p> <p><b>Value:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Clears TX Event FIFO Watermark Full interrupt status bit.</li> </ul> <p>Writing a 1 to this bit clears the respective bit in the ISR. Reads always 0.</p> <p><b>Value After Reset:</b> 0x0</p>
[30]	CTXEOF LW	W	<p>Clear TX Event FIFO Overflow Interrupt.</p> <p><b>Value:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Clears TX Event FIFO Overflow interrupt status bit.</li> </ul> <p>Writing a 1 to this bit clears the respective bit in the ISR. Reads always 0.</p> <p><b>Value After Reset:</b> 0x0</p>
[29:18]			Reserved
[17]	CRXMNF	W	<p>Clear RX Match Not Finished Interrupt.</p> <p><b>Value:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Clears RX Match Not Finished interrupt status bit.</li> </ul>

**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 <a href="#">ISR</a> . Reads always 0. <b>Value After Reset:</b> 0x0
[16:15]			Reserved
[14]	CTXCRS	W	Clear TX Cancellation Request Served Interrupt. <b>Value:</b> <b>0x1:</b> Clears TX Cancellation Request Served Interrupt status bit. Writing a 1 to this bit clears the respective bit in the <a href="#">ISR</a> . Reads always 0. <b>Value After Reset:</b> 0x0
[13]	CTXRRS	W	Clear TX Buffer Ready Request Served Interrupt. <b>Value:</b> <b>0x1:</b> Clears TX Buffer Ready Request Served Interrupt status bit. Writing a 1 to this bit clears the respective bit in the <a href="#">ISR</a> . Reads always 0. <b>Value After Reset:</b> 0x0
[12]	CRXFWMFLL	W	Clear RX FIFO-0 Watermark Full Interrupt. <b>Value:</b> <b>0x1:</b> Clears RX FIFO-0 Watermark Full interrupt status bit . Writing a 1 to this bit clears the respective bit in the <a href="#">ISR</a> . Reads always 0. <b>Value After Reset:</b> 0x0
[11]	CWKUP	W	Clear Wake-Up Interrupt. <b>Value:</b> <b>0x1:</b> Clears Wake-Up interrupt status bit. Writing a 1 to this bit clears the respective bit in the <a href="#">ISR</a> . Reads always 0. <b>Value After Reset:</b> 0x0
[10]	CSLP	W	Clear Sleep Interrupt. <b>Value:</b> <b>0x1:</b> Clears Sleep interrupt status bit. Writing a 1 to this bit clears the respective bit in the <a href="#">ISR</a> . Reads always 0. <b>Value After Reset:</b> 0x0
[9]	CBSOFF	W	Clear Bus-Off Interrupt. <b>Value:</b> <b>0x1:</b> Clears Bus-Off interrupt status bit. Writing a 1 to this bit clears the respective bit in the <a href="#">ISR</a> . Reads always 0. <b>Value After Reset:</b> 0x0

**Table 13-28 Fields For Register: ICR (continued)**

Bits	Name	R/W	Description
[8]	CERROR	W	<p>Clear Error Interrupt.</p> <p><b>Value:</b></p> <p><b>0x1:</b> Clears Error interrupt status bit.</p> <p>Writing a 1 to this bit clears the respective bit in the <a href="#">ISR</a>. Reads always 0.</p> <p><b>Value After Reset:</b> 0x0</p>
[7]			Reserved
[6]	CRFXOFLW	W	<p>Clear RX FIFO-0 Overflow Interrupt.</p> <p><b>Value:</b></p> <p><b>0x1:</b> Clears RX FIFO-0 Overflow interrupt status bit.</p> <p>Writing a 1 to this bit clears the respective bit in the <a href="#">ISR</a>. Reads always 0.</p> <p><b>Value After Reset:</b> 0x0</p>
[5]	ETSCNT_OFLW	W	<p>Clear Timestamp Counter Overflow Interrupt.</p> <p><b>Value:</b></p> <p><b>0x1:</b> Clears Timestamp Counter Overflow Interrupt status bit.</p> <p>Writing a 1 to this bit clears the respective bit in the <a href="#">ISR</a>. Reads always 0.</p> <p><b>Value After Reset:</b> 0x0</p>
[4]	CRXOK	W	<p>Clear Clears New Message Received Interrupt.</p> <p><b>Value:</b></p> <p><b>0x1:</b> Clears New Message Received interrupt status bit.</p> <p>Writing a 1 to this bit clears the respective bit in the <a href="#">ISR</a>. Reads always 0.</p> <p><b>Value After Reset:</b> 0x0</p>
[3]	CBSFRD	W	<p>Clear Bus-Off Recovery Done Interrupt.</p> <p><b>Value:</b></p> <p><b>0x1:</b> Clears Bus-Off Recovery Done interrupt status bit.</p> <p>Writing a 1 to this bit clears the respective bit in the <a href="#">ISR</a>. Reads always 0.</p> <p><b>Value After Reset:</b> 0x0</p>
[2]	CPEE	W	<p>Clear Protocol Exception Event Interrupt.</p> <p><b>Value:</b></p> <p><b>0x1:</b> Clears Protocol Exception Event interrupt status bit.</p> <p>Writing a 1 to this bit clears the respective bit in the <a href="#">ISR</a>. Reads always 0.</p> <p><b>Value After Reset:</b> 0x0</p>
[1]	CTXOK	W	<p>Clear Transmission Successful Interrupt.</p> <p><b>Value:</b></p>

**Table 13-28 Fields For Register: ICR (continued)**

Bits	Name	R/W	Description
			<p><b>0x1:</b> Clears Transmission Successful interrupt status bit.</p> <p><b>Value After Reset:</b> 0x0</p>
[0]	CARBLOST	W	<p>Clear Arbitration Lost Interrupt.</p> <p><b>Value:</b></p> <p><b>0x1:</b> Clears Arbitration Lost interrupt status bit..</p> <p>Writing a 1 to this bit clears the respective bit in the <a href="#">ISR</a>. Reads always 0.</p> <p><b>Value After Reset:</b> 0x0</p>

**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	<p>Timestamp Counter Value.</p> <p>This Status field gives running value of the timestamp counter. This field is cleared when a 0 is written to the <a href="#">CEN</a> bit in the <a href="#">SRR</a>.</p> <p><b>Value After Reset:</b> 0x0</p>
[15:1]			Reserved
[0]	CTS	W	<p>Clear Timestamp Counter.</p> <p>Internal free running counter is cleared to 0 when <a href="#">CTS</a> = 1.</p> <p>This bit only needs to be written once with a 1 to clear the counter.</p> <p>The bit always reads as 0.</p> <p><b>Value After Reset:</b> 0x0</p>

**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	<p><i>Transmitter Delay Compensation (TDC)</i> Enable.</p> <p><b>Values:</b></p> <p><b>0x0:</b> TDC is disabled.</p> <p><b>0x1:</b> Enables TDC function as specified in the CAN FD standard.</p> <p>This bit can be written only when <a href="#">CEN</a> bit in <a href="#">SRR</a> is 0.</p>

**Table 13-30 Fields For Register: DP\_BRPR (continued)**

Bits	Name	R/W	Description
			<b>Value After Reset:</b> 0x0
[15:14]			Reserved
[13:8]	TDCOFF	R/W	<p>Transmitter Delay Compensation Offset.            This offset is specified in CAN clock cycles and is added to the measured transmitter delay to place the <i>Secondary Sample Point (SSP)</i> 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 <i>CEN</i> bit in <i>SRR</i> is 0.</p> <p><b>Value After Reset:</b> 0x0</p>
[7:0]	DP_BRP	R/W	<p>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 <i>CEN</i> bit in <i>SRR</i> is 0.</p> <p><b>Value After Reset:</b> 0x0</p>



The following boundary conditions are imposed on sum of measured loop delay and TDC Offset:  
 $\text{Measured loop delay} + \text{TDCOFF} < 3 \text{ bit times in the data phase.}$

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	<p>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 <i>CEN</i> bit in <i>SRR</i> is 0.</p> <p><b>Value After Reset:</b> 0x0</p>
[15:12]			Reserved
[11:8]	DP_TS2	R/W	<p>Data Phase Time Segment 2.            Indicates the Phase Segment 2 as specified in the CAN FD standard for Data Bit Timing.</p>

**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 <b>CEN</b> bit in <b>SRR</b> is 0. <b>Value After Reset:</b> 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 <b>CEN</b> bit in <b>SRR</b> is 0. <b>Value After Reset:</b> 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]	RR <i>i</i>	R/W	<p>TX Buffer_ <i>i</i> Ready Request (for <i>i</i> = 0; <i>i</i> &lt;=31).</p> <p> RR<i>i</i> corresponds to the bit with number [<i>i</i>], i.e. bit [31] corresponds to the RR31, [30] to the RR30, [29] to the RR29 and etc.</p> <p>This is control bit corresponds to TB<i>i</i> (for <i>i</i> = 0; <i>i</i> &lt;=31) message in TX block RAM.</p> <p>Host writes 1 to indicate buffer is ready for transmission. Core clears this bit when:</p> <ul style="list-style-type: none"> <li>• Buffer transmission is completed on CAN Bus</li> <li>• 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)]</li> <li>• If message is cancelled due to cancellation request</li> <li>• Any combination of the above three.</li> </ul> <p>Host writes to this bit are ignored when this bit is 1.</p> <p> This register remains in reset when SNOOP mode is enabled.</p> <p><b>Value After Reset:</b> 0x0</p>



- 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]	<code>ERRSi</code>	R/W	<p>TX Buffer_i (for i = 0; i &lt;=31) Ready Req Served/Cleared Interrupt Enable.</p> <p> <code>ERRSi</code> corresponds to the bit with number [i], i.e. bit [31] corresponds to the <code>ERRS31</code>, [30] to the <code>ERRS30</code>, [29] to the <code>ERRS29</code> and etc.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <code>TXRRS</code> bit in the <code>ISR</code> does not set if <code>RRi</code> (for i = 0; i &lt;=31) bit in <code>TRR</code> register clears.</li> <li><b>0x1:</b> Enables setting <code>TXRRS</code> bit in the <code>ISR</code> when <code>RRi</code> (for i = 0; i &lt;=31) bit in <code>TRR</code> register clears.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

### 13.2.2.2.1.16. TX Buffer Cancel Request (`TCR`)

The `TCR` register is at offset 0x94.

The following table shows the register bit assignments.

**Table 13-34 Fields For Register: `TCR`**

Bits	Name	R/W	Description
[31:0]	<code>CRI</code>	R/W	<p>TX Buffer_i (for i = 0; i &lt;=31) Cancel Request.</p> <p> <code>CRI</code> corresponds to the bit with number [i], i.e. bit [31] corresponds to the <code>CR31</code>, [30] to the <code>CR30</code>, [29] to the <code>CR29</code> and etc</p> <p>This is cancellation request bit corresponds to <code>RRi</code> (for i = 0; i &lt;=31) bit in <code>TRR</code> register.</p> <p>Host writes 1 to indicate cancellation request of corresponding buffer ready request (that is, <code>RRi</code> bit in <code>TRR</code> register). The core clears this bit when cancellation request is completed.</p> <p>Host writes to this bit are ignored if <code>CRI</code> is 1 or <code>RRi</code> bit of <code>TRR</code> register is 0.</p> <p>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)</p>

**Table 13-34 Fields For Register: TCR (continued)**

Bits	Name	R/W	Description
			<p>and cancellation request is cleared. Along with <code>RR<i>i</i></code> bit this is cleared.</p> <p>If message is transmitted successfully, then <code>RR0</code> bit in <code>TRR</code> clears and cancellation request is cleared anyway.</p> <p> If internal buffer scheduling round is in progress, then cancellation consideration is postponed till it is over.</p> <p><b>Value After Reset:</b> 0x0</p>



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]	<code>ECRS<i>i</i></code>	R/W	<p><code>TX Buffer_</code><i>i</i> (for <i>i</i> = 0; <i>i</i> &lt;=31) Transmission Served/Cleared Interrupt Enable.</p> <p> <code>ECRS<i>i</i></code> corresponds to the bit with number [<i>i</i>], i.e. bit [31] corresponds to the <code>ECRS31</code>, [30] to the <code>ECRS30</code>, [29] to the <code>ECRS29</code> and etc</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <code>TXCRS</code> bit in the <code>ISR</code> does not set if <code>CRI</code> (for <i>i</i> = 0; <i>i</i> &lt;=31) bit in <code>TCR</code> register clears.</li> <li><b>0x1:</b> Enables setting <code>TXCRS</code> bit in the <code>ISR</code> when <code>CRI</code> (for <i>i</i> = 0; <i>i</i> &lt;=31) bit in <code>TCR</code> register clears.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

#### 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]	<code>TxE_FL</code>	R	Fill Level (0-31).

**Table 13-36 Fields For Register: TXE\_FSR (continued)**

Bits	Name	R/W	Description
			<p>Number of stored message in TX Event FIFO starting from Read Index given in this register.</p> <p>For example, if <code>TXE_FL</code> = 0x5 and <code>TXE_RI</code> = 0x3 then TX Event FIFO has five messages starting from Read Index 3 (Start address 0x2018).</p> <p><code>TXE_FL</code> is maintained if <code>CEN</code> bit in <code>SRR</code> is cleared.</p> <p><code>TXE_FL</code> gets reset to 0 if soft or hard reset is asserted.</p> <p><b>Value After Reset:</b> 0x0</p>
[7]	<code>TXE_IRI</code>	W	<p>Increment Read Index by 1.</p> <p>With each Host writes setting this bit as 1, CAN FD increments Read index (<code>TXE_RI</code>) by 1 and update fill level (that is, decrement by 1).</p> <p>If FILL level is 0, setting this bit has no effect. The FILL level might remain unchanged when <code>TXE_IRI</code> is written if CAN FD is just finishing a successful transmission and incrementing internal write index. This bit always read as 0.</p> <p><b>Value After Reset:</b> 0x0</p>
[6:5]			Reserved
[4:0]	<code>TXE_RI</code>	R	<p>Read Index (0 to 63).</p> <p>Each time <code>IRI</code> 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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> next message read starts from location = 0x2000</li> <li><b>0x1:</b> next message read starts from location = 0x2008</li> </ul> <p><code>TXE_RI</code> is maintained if <code>CEN</code> in <code>SRR</code> bit is cleared.</p> <p><code>TXE_RI</code> gets reset to 0 if soft or hard reset is asserted.</p> <p><b>Value After Reset:</b> 0x0</p>

#### 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]	<code>TXE_FWM</code>	R/W	<p>TX Event FIFO Full Watermark.</p> <p>TX Event FIFO generates FULL interrupt based on the value programmed in this field.</p> <p>Set it within (1-31) range.</p> <p>The TX FIFO Full Watermark interrupt in the <code>ISR</code> register continues to assert as long as the TX Event FIFO Fill Level is above TX Event FIFO Full watermark.</p> <p>This field can be written to only when <code>CEN</code> bit in <code>SRR</code> is 0.</p> <p><b>Value After Reset:</b> 0xF</p>

### 13.2.2.1.20. Acceptance Filter (Control) Register (**A<sub>FR</sub>**)

The **A<sub>FR</sub>** register is at offset 0xE0.

There are 32 acceptance filters. Each acceptance filter has the **Acceptance Filter i Mask Register (AFM<sub>Ri</sub>)** and **Acceptance Filter i ID registers Register (AFI<sub>Ri</sub>)** (which is controlled by the Acceptance Filter (Control) Register bits).

Each **AFI<sub>Ri</sub>** and **AFM<sub>Ri</sub>** 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: A<sub>FR</sub>**

Bits	Name	R/W	Description
[31:0]	UAF <sub>i</sub>	R/W	<p>Use Acceptance Filter Mask Pair<sub>i</sub> (for <math>i = 0; i &lt;= 31</math>).</p> <p> UAF<sub>i</sub> corresponds to the bit with number [i], i.e. bit [31] corresponds to the UAF<sub>31</sub>, [30] to the UAF<sub>30</sub>, [29] to the UAF<sub>29</sub> and etc</p> <p>Enables the use of acceptance filter mask pair <b>i</b>.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Indicates AFM<sub>Ri</sub> and AFI<sub>Ri</sub> pair is not used for acceptance filtering.</li> <li><b>0x1:</b> Indicates AFM<sub>Ri</sub> and AFI<sub>Ri</sub> pair is used for acceptance filtering.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

### 13.2.2.1.21. 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	<p>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.</p>

**Table 13-39 Fields For Register: FSR (continued)**

Bits	Name	R/W	Description
			<p>For example, if <b>FL</b> = 0x5 and <b>RI</b> = 0x2, then RX FIFO-0 has five messages starting from Read Index 2 (Start address 0x4190). <b>FL</b> is maintained if <b>CEN</b> bit in <b>SRR</b> is cleared. <b>FL</b> is reset to 0 if a soft or hard reset is asserted.</p> <p><b>Value After Reset:</b> 0x0</p>
[7]	<b>IRI</b>	W	<p>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).</p> <p>If FILL level is 0, setting this bit has no effect. The FILL level might remain unchanged when <b>IRI</b> is written if core is just finishing a successful receive and incrementing internal write index.</p> <p>This bit always read as 0.</p> <p><b>Value After Reset:</b> 0x0</p>
[6]			Reserved
[5:0]	<b>RI</b>	R	<p>RX FIFO-0 Read Index (0 to 63). Each time <b>IRI</b> 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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Next message read starts from location = 0x2100</li> <li><b>0x1:</b> Next message read starts from location = 0x2148</li> </ul> <p><b>RI</b> is maintained if <b>CEN</b> in <b>SRR</b> is cleared.</p> <p><b>RI</b> gets reset to 0 if soft or hard reset is asserted.</p> <p><b>Value After Reset:</b> 0x0</p>

### 13.2.2.1.22. 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]			<p>Reserved</p> <p><b>Value After Reset:</b> 0x1F0F00</p>
[5:0]	<b>RXFWM</b>	R/W	<p>RX FIFO-0 Full Watermark.</p> <p>RX FIFO-0 generates FULL interrupt based on the value programmed in this field.</p> <p>Set it within (1-63) range.</p> <p>The RX FIFO-0 Full Watermark interrupt in the <b>ISR</b> register continues to assert as long as the RX FIFO-0 Fill Level is above RX FIFO-0 Full watermark.</p> <p>This field can be written to only when <b>CEN</b> bit in <b>SRR</b> is 0.</p> <p><b>Value After Reset:</b> 0xF</p>

### 13.2.2.2.2. CAN FD TX Message Space

#### 13.2.2.2.2.1. TBI Message ID Register (**TBiID**)

The **TBiID** (for  $i = 0; i \leq 31$ ) 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	<p>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.</p>
[20]	SRR/RTR/RRS	R/W	<p>Substitute Remote Transmission Request. For Extended CAN frames and Extended CAN FD frame this bit is transmitted at <b>SRR</b> position of the respective frame and must be set as 1. For Standard CAN FD frame this bit is transmitted at <b>RRS</b> 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:</p> <ul style="list-style-type: none"> <li><b>0x0:</b> Indicates that the message frame is a Standard Data CAN Frame</li> <li><b>0x1:</b> Indicates that the message frame is a Standard Remote CAN Frame.</li> </ul> <p> There are no CAN FD remote frames.</p>
[19]	IDE	R/W	<p>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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Indicates the use of a Standard Message Identifier.</li> <li><b>0x1:</b> Indicates the use of an Extended Message Identifier.</li> </ul>
[18:1]	ID[ 17:0 ]	R/W	<p>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.</p>
[0]	RTR/RRS	R/W	<p>Remote Transmission Request. This bit differentiates between CAN extended data frames and CAN extended remote frames.</p> <p><b>Values:</b></p>

**Table 13-41 Fields For Register: TBIID (continued)**

Bits	Name	R/W	Description
			<p><b>0x0:</b> Indicates the message frame is a CAN Data Frame. For Extended CAN FD frame this bit is transmitted at <u>RRS</u> position of the frame and must be set as 0.</p> <p><b>0x1:</b> Indicates the message frame is a CAN Remote Frame.</p> <p>For Standard CAN and CAN FD Frames, writes to this bit should be 0.</p> <p> There are no CAN FD remote frames.</p>

#### 13.2.2.2.2.2. TBi Data Length Code Register (TBiDLC)

The `TBIDLC` (for  $i = 0; i \leq 31$ ) 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	<p>Data Length Code.</p> <p>This is the data length code of the R/W field of the CAN and CAN FD frame.</p>
[27]	EDL/FDF	R/W	<p>Extended Data Length/FD Frame Format.</p> <p>This bit distinguishes between CAN format and CAN FD format frames.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> CAN format frame.</li> <li><b>0x1:</b> CAN FD format frame.</li> </ul>
[26]	BRS	R/W	<p>Bit Rate Switch.</p> <p>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).</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Bit rate is not switched inside a CAN FD frame.</li> <li><b>0x1:</b> 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.</li> </ul> <p> BRS does not exist in CAN format frames and should be set to 0.</p>
[25]			Reserved.
[24]	EFC	R/W	<p>Event FIFO R/W.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Don't store TX events.</li> <li><b>0x1:</b> Store TX Events</li> </ul>

**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.

**13.2.2.2.3. 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 <b>TBiDLC.DLC</b> 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 <b>TBiDLC.DLC</b> 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 <b>TBiDLC.DLC</b> 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 <b>TBiDLC.DLC</b> control field.

**13.2.2.2.4. 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 **AIR** register bits).

Acceptance filtering is performed in this sequence:

- The incoming Identifier is masked with the bits in the Acceptance Filter Mask register.
- The **Acceptance Filter i ID Register** is also masked with the bits in the Acceptance Filter Mask register.
- Both resulting values are compared.
- If both these values are equal, the message is stored in RX FIFO-0.
- 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	<p>Standard Message ID Mask . These bits are used for masking the <a href="#">TBIID.ID[ 28:18 ]</a> in a Standard Frame.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Indicates the corresponding bit in <a href="#">AFIRi</a> is not used when comparing the incoming message identifier.</li> <li><b>0x1:</b> Indicates the corresponding bit in <a href="#">AFIRi</a> is used when comparing the incoming message identifier.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[20]	AMSRR	R/W	<p>Substitute Remote Transmission Request Mask. This bit is used for masking the <a href="#">TBIID.RTR</a> bit in a Standard Frame.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Indicates the corresponding bit in <a href="#">AFIRi</a> is not used when comparing the incoming message identifier.</li> <li><b>0x1:</b> Indicates the corresponding bit in <a href="#">AFIRi</a> is used when comparing the incoming meassage identifier.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[19]	AMIDE	R/W	<p>Identifier Extension Mask. Used for masking the <a href="#">TBIID.IDE</a> bit.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Indicates the corresponding bit in <a href="#">AFIRi</a> is not used when comparing the incoming message identifier.</li> <li><b>0x1:</b> Indicates the corresponding bit in <a href="#">AFIRi</a> is used when comparing the incoming meassage identifier.</li> </ul> <p>If <a href="#">AMIDE</a> = 0x1 and the <a href="#">AIIDE</a> bit in the corresponding <a href="#">AFIRi</a> is 0x0, this mask is applicable to only Standard frames. If <a href="#">AMIDE</a> = 0x1 and the <a href="#">AIIDE</a> bit in the corresponding <a href="#">AFIRi</a> is 0x1, this mask is applicable to only extended frames. If <a href="#">AMIDE</a> = 0x0, this mask is applicable to both Standard and Extended frames.</p> <p><b>Value After Reset:</b> 0x0</p>
[18:1]	AMID[ 17:0 ]	R/W	<p>Extended Message ID Mask . These bits are used for masking the <a href="#">TBIID.ID[ 17:0 ]</a> in an Extended Frame.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Indicates the corresponding bit in <a href="#">AFIRi</a> is not used when comparing the incoming message identifier.</li> <li><b>0x1:</b> Indicates the corresponding bit in <a href="#">AFIRi</a> is used when comparing the incoming meassage identifier.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

**Table 13-44 Fields For Register: AFMRI (continued)**

Bits	Name	R/W	Description
[0]	AMRTR	R/W	<p>Remote Transmission Request Mask . This bit is used for masking the <b>TBiID.RTR</b> bit in an Extended Frame.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Indicates the corresponding bit in <b>AFIRi</b> is not used when comparing the incoming message identifier.</li> <li><b>0x1:</b> Indicates the corresponding bit in <b>AFIRi</b> is used when comparing the incoming meessage identifier.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

### 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. <b>TBiID.RTR</b> 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.

**Table 13-46 Fields For Register: TXE\_FIFO\_TBi\_ID (continued)**

Bits	Name	R/W	Description
			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	<p>Substitute Remote Transmission Request. For Extended CAN frames and Extended CAN FD frame this bit is transmitted at <u>SRR</u> position of the respective frame and must be set as 1. For Standard CAN FD frames, this bit is transmitted at the <u>RRS</u> 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:</p> <ul style="list-style-type: none"> <li><b>0x0:</b> Indicates that the message frame is a Standard Data CAN Frame</li> <li><b>0x1:</b> Indicates that the message frame is a Standard Remote CAN Frame.</li> </ul> <p> There are no CAN FD remote frames.</p>
[19]	IDE	R	<p>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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Indicates the use of a Standard Message Identifier.</li> <li><b>0x1:</b> Indicates the use of an Extended Message Identifier.</li> </ul>
[18:1]	ID[17:0]	R	<p>Extended Message ID. This field indicates the Extended Identifier. Valid only for Extended CAN and CAN FD frames.</p>
[0]	RTR/RRS	R	<p>Remote Transmission Request. This bit differentiates between CAN extended data frames and CAN extended remote frames.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Indicates the message frame is a CAN Data Frame. For Extended CAN FD frame this bit is transmitted at <u>RRS</u> position of the frame and is always equal to 0.</li> <li><b>0x1:</b> Indicates the message frame is a CAN Remote Frame.</li> </ul> <p> There are no CAN FD remote frames.</p>

### 13.2.2.3.2. 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. <b>Values:</b> <b>0x0:</b> CAN format frame. <b>0x1:</b> 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 <a href="#">MSR</a> register). <b>Values:</b> <b>0x0:</b> Bit rate is not switched inside a CAN FD frame. <b>0x1:</b> 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.  <b>BRS</b> does not exist in CAN format frames and is always equal to 0
[25:24]	ET	R	Event Type. <b>Values:</b> <b>0x11:</b> Transmitted. <b>0x01:</b> Transmitted in spite of cancellation request or DAR mode transmissions <b>0x00:</b> Reserved. <b>0x10:</b> Reserved.
[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 <a href="#">SOF</a> 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	<p>Standard Message ID.</p> <p>The Identifier portion for a Standard Frame is 11 bits. These bits indicate the Standard Frame ID.</p> <p>This field is valid for both CAN and CAN FD Standard and Extended Frames.</p>
[20]	SRR/RTR/ RRS	R	<p>Substitute Remote Transmission Request.</p> <p>For Extended CAN frames and Extended CAN FD frame this bit was received at <u>SRR</u> position of the respective frame.</p> <p>For Standard CAN FD frame this bit was received at <u>RRS</u> position of the frame.</p> <p>This bit differentiates between received standard CAN data frames and standard CAN remote frames as following</p> <ul style="list-style-type: none"> <li><b>0x0:</b> Indicates that the message frame is a Standard Data CAN Frame</li> <li><b>0x1:</b> Indicates that the message frame is a Standard Remote CAN Frame.</li> </ul>
[19]	IDE	R	<p>Identifier Extension.</p> <p>This bit differentiates between frames using the Standard Identifier and those using the Extended Identifier.</p> <p>Valid for both CAN and CAN FD Standard and Extended Frames.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Indicates the use of a Standard Message Identifier.</li> <li><b>0x1:</b> Indicates the use of an Extended Message Identifier.</li> </ul>
[18:1]	ID[ 17:0 ]	R	<p>Extended Message ID.</p> <p>This field indicates the Extended Identifier.</p> <p>Valid only for Extended CAN and CAN FD Frames.</p> <p>For Standard CAN and CAN FD Frames, this field is reserved and has no meaning.</p>
[0]	RTR/RRS	R	<p>Remote Transmission Request.</p> <p>This bit differentiates between CAN extended data frames and CAN extended remote frames.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Indicates the message frame is a CAN Data Frame. For Extended CAN FD frame this bit is transmitted at <u>RRS</u> position of the frame and is always equal to 0</li> <li><b>0x1:</b> Indicates the message frame is a CAN Remote Frame.</li> </ul> <p>For Standard CAN and CAN FD frames, this field is reserved and has no meaning.</p>

### 13.2.2.4.2. RBi Data Length Code Register (RBiDLC)

The `RBiDLC` (for  $i = 0; i \leq 63$ ) 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. <b>Values:</b> <b>0x0:</b> CAN format frame. <b>0x1:</b> CAN FD format frame.
[26]	BRS	R	Bit Rate Switch. The <code>BRS</code> bit provides R whether the bit rate was switched inside the received CAN FD format frame or not. <b>Values:</b> <b>0x0:</b> Bit rate is not switched inside a CAN FD frame. <b>0x1:</b> 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 <code>ESI</code> bit provides the error R of the sender/transmitter of the message. <b>Values:</b> <b>0x0:</b> Received frame was sent by error active transmitter This bit has no meaning if the received frame is a CAN frame. <b>0x1:</b> 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 <code>SOF</code> bit.

### 13.2.2.4.3. RBi Data Byte n Register (RBiDWN)

The `RBiDWN` (for  $i = 0; i \leq 63$  and for  $n = 0; n \leq 15$ ) register is offset as  $0x2108 + (i * 0x48) + (n * 0x4)$ .

The following table shows the register bit assignments.

**Table 13-50 Fields For Register: RBiDwn**

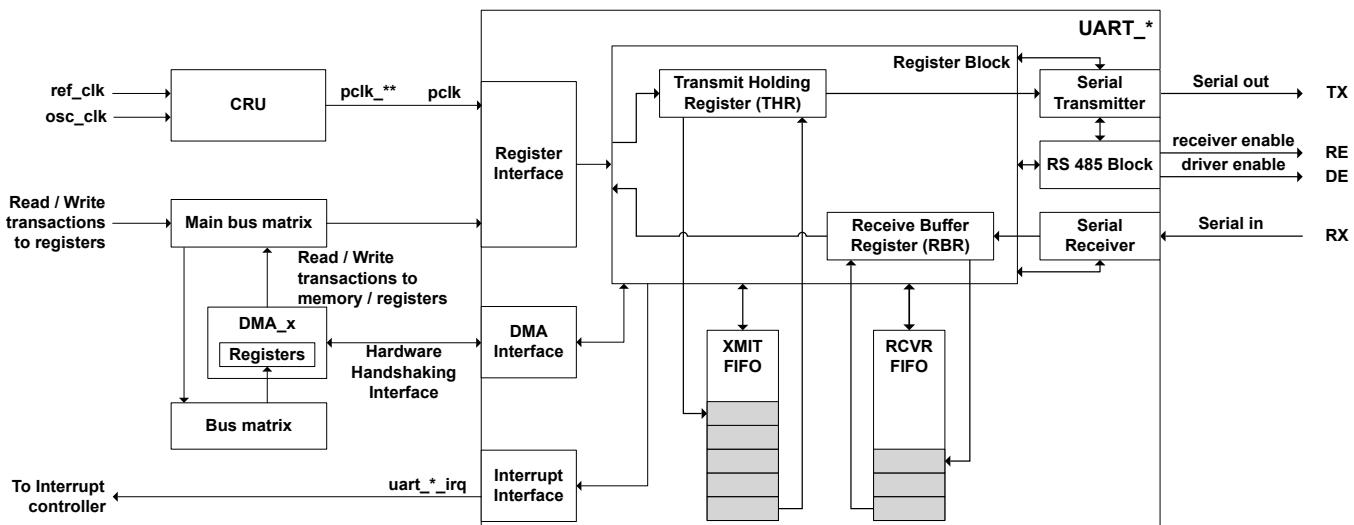
Bits	Name	R/W	Description
[31:24]	Data bytes0 [7:0]	R	Data Byte 0. Data byte that was received with CAN or CAN FD frame based on <a href="#">RBiDLC.DLC</a> control field.
[23:16]	Data bytes1 [7:0]	R	Data Byte 1. Data byte that was received with CAN or CAN FD frame based on <a href="#">RBiDLC.DLC</a> 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 <a href="#">RBiDLC.DLC</a> 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 <a href="#">RBiDLC.DLC</a> control field.

### 13.3. UART

The *Universal Asynchronous Receiver/Transmitter (UART)* is programmable component used for serial communication with peripherals and data sets.

There are eight UARTs in BE-U1000 microcontroller.

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

**Figure 13-3 UART Functional block diagram**

In the figure above:

- $\times$  indicates 0 for DMA\_0 and 1 for DMA\_1. For more information, refer to the [DMA Controller](#).
- 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.
- 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.

The UART has the following features:

- Transmit 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

In this chapter:

- 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.

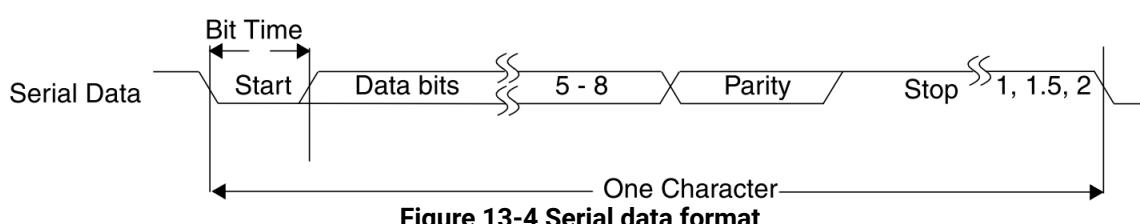


Figure 13-4 Serial data format

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 *Last 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 TxFIFO/THR empty conditions.

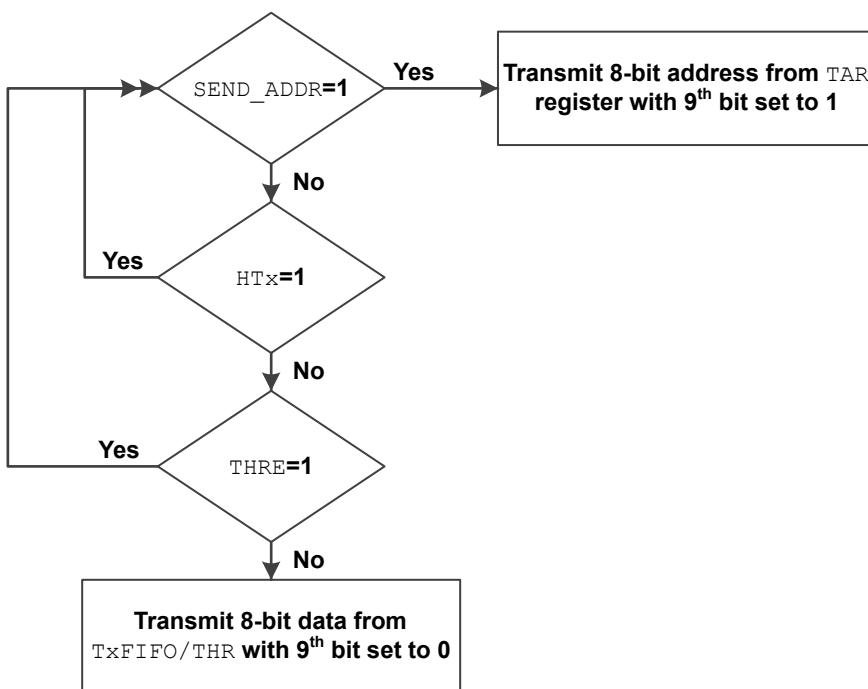


Figure 13-5 Auto address transmit flow chart

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 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 TxFIFO 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 TxFIFO as data byte. Once the TxFIFO 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 TxFIFO. 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:

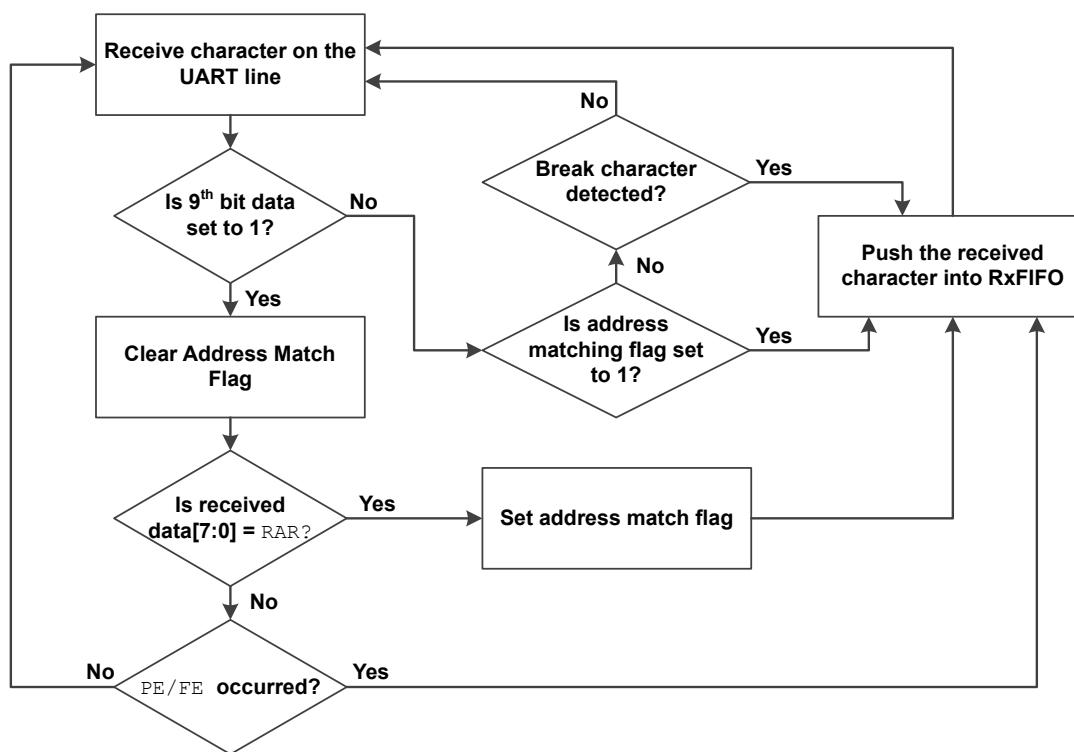
- [Hardware Address Match Receive Mode](#) (when [ADDR\\_MATCH](#) bit of [LCR\\_EXT](#) register is set to 1)
- [Software Address Match Receive Mode](#) (when [ADDR\\_MATCH](#) bit of [LCR\\_EXT](#) register is set to 0)

#### 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 RxFIFO. 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 RxFIFO in FIFO-mode or to `RBR` register in non-FIFO mode and the `ADDR_RCV` 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 RxFIFO or `RBR` register with `ADDR_RCV` 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 RxFIFO 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 RxFIFO 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 RxFIFO 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 RxFIFO 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 TxFIFO.

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 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 Fraction value)/(2<sup>4</sup>) formula and shows in the following table.

**Table 13-51 Divisor Latch Fractional Values**

<b>DLF Value</b>	<b>Fraction</b>	<b>Fractional Value</b>
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

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	4KB	0x1100_1000	0x1000_1FFF
	UART_1	4KB	0x1100_2000	0x1100_2FFF
	UART_2	4KB	0x1100_3000	0x1100_3FFF
Periphery 1	UART_3	4KB	0x1300_1000	0x1300_1FFF
	UART_4	4KB	0x1300_2000	0x1300_2FFF
	UART_5	4KB	0x1300_3000	0x1300_3FFF
Periphery 2	UART_6	4KB	0x1400_9000	0x1400_9FFF
	UART_7	4KB	0x1400_A000	0x1400_AFFF



For details, refer to [Memory Map](#) chapter.

In this chapter:

- [Registers Map](#)
- [Register Descriptions](#)

#### 13.3.2.1. Registers Map

This section contains information about the register memory map.

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 13-53 UART Registers Map**

Name	Offset	Description	Reset Value
RBR	0x0	<a href="#">Receive Buffer Register</a> <b>Dependencies:</b> <a href="#">LCR</a> [7] bit = 0	0x0
THR	0x0	<a href="#">Transmit Holding Register</a> <b>Dependencies:</b> <a href="#">LCR</a> [7] bit = 0	0x0
DLL	0x0	<a href="#">Divisor Latch Low</a> <b>Dependencies:</b> <a href="#">LCR</a> [7] bit = 1	0x0
DLH	0x04	<a href="#">Divisor Latch High</a> <b>Dependencies:</b> <a href="#">LCR</a> [7] bit = 1	0x0
IER	0x04	<a href="#">Interrupt Enable Register</a> <b>Dependencies:</b> <a href="#">LCR</a> [7] bit = 0	0x0
IIR	0x08	<a href="#">Interrupt Identity Register</a>	0x1
FCR	0x08	<a href="#">FIFO Control Register</a>	0x0

**Table 13-53 UART Registers 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 <b>Dependencies:</b> LCR[7] bit = 0	0x0
STHRI (for i=0; i<=15)	0x30 +(i *0x4)	Shadow Transmit Holding Register <b>Dependencies:</b> 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 <b>Exists:</b> UART6 and UART7	0x6
DE_EN	0xB0	Driver Output Enable Register <b>Exists:</b> UART6 and UART7	0x0
RE_EN	0xB4	Receiver Output Enable Register <b>Exists:</b> UART6 and UART7	0x0
DET	0xB8	Driver Output Enable Timing Register <b>Exists:</b> UART6 and UART7	0x0
TAT	0xBC	TurnAround Timing Register <b>Exists:</b> UART6 and UART7	0x0

**Table 13-53 UART Registers 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 of a register description 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	<p>Data byte received on the serial in port (<b>RX</b>)  The data in this register is valid only if the Data Ready (<b>DR</b>) bit in the Line Status Register (<b>LSR</b>) is set.  If FIFOs are disabled (<b>FCR[0]</b> set to 0), the data in the <b>RBR</b> must be read before the next data arrives, otherwise it is overwritten, resulting in an over-run error.  If FIFOs are enabled (<b>FCR[0]</b> 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.</p> <p><b>Value After Reset:</b> 0x0</p>

#### 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 ( <b>TX</b> )

**Table 13-55 Fields For Register: `THR` (continued)**

Bits	Name	R/W	Description
			<p>Data should only be written to the <code>THR</code> when the <code>THR</code> Empty (<code>THRE</code>) bit (<code>LSR[5]</code>) is set.</p> <p>If FIFOs are disabled (<code>FCR[0]=0</code>) and <code>THRE</code> is set, writing a single character to the <code>THR</code> clears the <code>THRE</code>. Any additional writes to the <code>THR</code> before the <code>THRE</code> is set again causes the <code>THR</code> data to be overwritten.</p> <p>If FIFOs are enabled (<code>FCR[0]=1</code>) and <code>THRE</code> is set, 16 number of characters of data may be written to the <code>THR</code> before the FIFO is full. Any attempt to write data when the FIFO is full results in the write data being lost.</p> <p><b>Value After Reset:</b> 0x0</p>

### 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]	<code>DLL</code>	R/W	<p>Lower 8 bits of a 16-bit, read/write, Divisor Latch register that contains the baud rate divisor for the UART.</p> <p>The output baud rate is equal to the serial clock (<code>pclk</code>) frequency divided by sixteen times the value of the baud rate divisor, as follows:</p> $\text{baud rate} = (\text{serial clock freq}) / (16 * \text{divisor})$ <p> With the Divisor Latch Registers (<code>DLL</code> and <code>DLH</code>) set to 0, the baud clock is disabled and no serial communications occur. Also, once the <code>DLL</code> is set, at least 8 clock cycles of the slowest UART clock should be allowed to pass before transmitting or receiving data.</p> <p><b>Value After Reset:</b> 0x0</p>

### 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]	<code>DLH</code>	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
			<p>The output baud rate is equal to the serial clock (<code>pclk</code>) frequency divided by sixteen times the value of the baud rate divisor, as follows:</p> $\text{baud rate} = (\text{serial clock freq}) / (16 * \text{divisor})$ <p> With the Divisor Latch Registers (<code>DLL</code> and <code>DLH</code>) set to 0, the baud clock is disabled and no serial communications occur. Also, once the <code>DLH</code> is set, at least 8 clock cycles of the slowest UART clock should be allowed to pass before transmitting or receiving data.</p> <p><b>Value After Reset:</b> 0x0</p>

### 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]	<code>PTIME</code>	R/W	<p>Programmable <code>THRE</code> Interrupt Mode Enable This bit is used to enable/disable the generation of <code>THRE</code> Interrupt.</p> <p><b>0x0:</b> disabled <b>0x1:</b> enabled</p> <p><b>Value After Reset:</b> 0x0</p>
[6:5]			Reserved and read as 0
[4]	<code>ELCOLR</code>	R/W	<p>Method for clearing the status in the <code>LSR</code> register This is applicable only for Overrun Error, Parity Error, Framing Error, and Break Interrupt status bits.</p> <p><b>0x0:</b> disabled     <code>LSR</code> status bits are cleared either on reading Rx FIFO (<code>RBR</code> Read) or On reading <code>LSR</code> register. <b>0x1:</b> enabled     <code>LSR</code> status bits are cleared only on reading <code>LSR</code> register.</p> <p><b>Value After Reset:</b> 0x0</p>
[3]			Reserved
[2]	<code>ELSI</code>	R/W	<p>Enable Receiver Line Status Interrupt This bit used to enable/disable the generation of Receiver Line Status Interrupt. This is the highest priority interrupt.</p> <p><b>0x0:</b> disabled <b>0x1:</b> enabled</p> <p><b>Value After Reset:</b> 0x0</p>
[1]	<code>ETBEI</code>	R/W	Enable Transmit Holding Register Empty Interrupt

**Table 13-58 Fields For Register: IER (continued)**

Bits	Name	R/W	Description
			<p>This bit used to enable/disable the generation of Transmitter Holding Register Empty Interrupt. This is the third highest priority interrupt.</p> <p><b>0x0:</b> disabled  <b>0x1:</b> enabled</p> <p><b>Value After Reset:</b> 0x0</p>
[0]	ERBFI	R/W	<p>Enable Received Data Available Interrupt</p> <p>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.</p> <p><b>0x0:</b> disabled  <b>0x1:</b> enabled</p> <p><b>Value After Reset:</b> 0x0</p>

### 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]			<p>Reserved and read as 0.</p> <p><b>Value After Reset:</b> 0x0</p>
[7:6]	FIFOE	R	<p>FIFOs Enabled</p> <p>This bit used to indicate whether the FIFOs are enabled or disabled.</p> <p><b>0x0:</b> disabled  <b>0x3:</b> enabled</p> <p><b>Value After Reset:</b> 0x0</p>
[5:4]			<p>Reserved and read as 0</p> <p><b>Value After Reset:</b> 0x0</p>
[3:0]	IID	R	<p>Interrupt ID</p> <p>This bit indicates the highest priority pending interrupt, which can be one of the following types:</p> <p><b>0x1:</b> no interrupt pending  <b>0x2:</b> <a href="#">THR</a> empty  <b>0x4:</b> received data available  <b>0x6:</b> receiver line status  <b>0x7:</b> busy detect  <b>1xC:</b> character timeout</p> <p>Bit 3 indicates an interrupt that can occur only when the FIFOs are enabled and used to distinguish a Character Timeout condition interrupt.</p> <p><b>Value After Reset:</b> 0x1</p>

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 <a href="#">LSR</a>
0	1	0	0	Second	Received data available	Receiver data available (FIFOs disabled, <a href="#">FCR[0]=0</a> ) or RCVR FIFO trigger level reached (FIFOs enabled, <a href="#">FCR[0]=1</a> )	Reading the <a href="#">RBR</a> (FIFOs disabled, <a href="#">FCR[0]=0</a> ) or the FIFO drops below the trigger level (FIFOs enabled, <a href="#">FCR[0]=1</a> )
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 <a href="#">RBR</a>
0	0	1	0	Third	Transmit holding register empty	<a href="#">THR</a> empty (Programmable THRE Interrupt Mode disabled, <a href="#">IER[7]=0</a> ) or xMIT FIFO at or below threshold (Programmable THRE Interrupt Mode enabled, <a href="#">IER[7]=1</a> )	Reading the <a href="#">IIR</a> register or writing into <a href="#">THR</a> (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 <a href="#">LCR</a> while the UART is busy ( <a href="#">USR[0]</a> is set to 1).	Reading the <a href="#">USR</a>

### 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]	<a href="#">RT</a>	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: <b>0x0:</b> 1 character in the FIFO <b>0x1:</b> FIFO ¼ full <b>0x2:</b> FIFO ½ full <b>0x3:</b> FIFO 2 less than full <b>Value After Reset:</b> 0x0
[5:4]	<a href="#">TET</a>	W	Tx Empty Trigger This bit used to select the empty threshold level at which the <a href="#">THRE</a> 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
			<p><b>0x0:</b> FIFO empty  <b>0x1:</b> 2 characters in the FIFO  <b>0x2:</b> FIFO ¼ full  <b>0x3:</b> FIFO ½ full</p> <p><b>Value After Reset:</b> 0x0</p>
[3]			Reserved and read as 0
[2]	FIFOR	W	<p>xMIT FIFO Reset  This resets the control portion of the transmit FIFO and treats the FIFO as empty.</p> <p> This bit is 'self-clearing'. It is not necessary to clear this bit.</p> <p><b>Value After Reset:</b> 0x0</p>
[1]	RFIFOR	W	<p>RCVR FIFO Reset  This resets the control portion of the receive FIFO and treats the FIFO as empty.</p> <p> This bit is 'self-clearing'. It is not necessary to clear this bit.</p> <p><b>Value After Reset:</b> 0x0</p>
[0]	FIFOE	W	<p>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.</p> <p><b>0x0:</b> FIFO disabled  <b>0x1:</b> FIFO enabled</p> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p>Divisor Latch Access Bit  Writeable only when UART is not busy (<a href="#">USR[0]</a> is 0). This bit is used to enable reading and writing of the Divisor Latch register (<a href="#">DLL</a> and <a href="#">DLH</a>) to set the baud rate of the UART. This bit must be cleared after initial baud rate setup in order to access other registers.</p> <p><b>Value After Reset:</b> 0x0</p>
[6]	BC	R/W	<p>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 <a href="#">MCR[4]</a>, the sout line is forced low until the</p>

**Table 13-62 Fields For Register: LCR (continued)**

Bits	Name	R/W	Description
			<p>Break bit is cleared. When in Loopback Mode, the break condition is internally looped back to the receiver.</p> <p><b>Value After Reset:</b> 0x0</p>
[5]	SP	R/W	<p>Stick Parity</p> <p>Writeable only when UART is not busy (<code>USR[0]</code> is 0). This bit is used to force parity value. When <code>PEN</code>, <code>EPS</code>, and <code>SP</code> are set to 1, the parity bit is transmitted and checked as logic 0. If <code>PEN</code> and <code>SP</code> are set to 1 and <code>EPS</code> is logic 0, then parity bit is transmitted and checked as logic 1. If this bit is set to 0, Stick Parity is disabled.</p> <p><b>Value After Reset:</b> 0x0</p>
[4]	EPS	R/W	<p>Even Parity Select</p> <p>Writeable only when UART is not busy (<code>USR[0]</code> is 0). This is used to select between even and odd parity, when parity is enabled (<code>PEN</code> 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.</p> <p><b>Value After Reset:</b> 0x0</p>
[3]	PEN	R/W	<p>Parity Enable</p> <p>Writeable only when UART is not busy (<code>USR[0]</code> is 0). This bit is used to enable and disable parity generation and detection in transmitted and received serial character respectively.</p> <p><b>0x0:</b> parity disabled  <b>0x1:</b> parity enabled</p> <p><b>Value After Reset:</b> 0x0</p>
[2]	STOP	R/W	<p>Number of stop bits</p> <p>Writeable only when UART is not busy (<code>USR[0]</code> 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 (<code>LCR[1:0]</code> set to 0) one and a half stop bits are transmitted. Otherwise, two stop bits are transmitted.</p> <p> Regardless of the number of stop bits selected the receiver checks only the first stop bit.</p> <p><b>0x0:</b> 1 stop bit  <b>0x1:</b> 1.5 stop bits when <code>DLS</code> (<code>LCR[1:0]</code>) is 0, else 2 stop bit</p> <p><b>Value After Reset:</b> 0x0</p>
[1:0]	DLS	R/W	<p>Data Length Select</p> <p>Writeable only when UART is not busy (<code>USR[0]</code> 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 bit that may be selected areas follows:</p> <p><b>0x0:</b> 5 bits  <b>0x1:</b> 6 bits  <b>0x2:</b> 7 bits  <b>0x3:</b> 8 bits</p> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p><b>SIR Mode Enable</b></p> <p> To enable SIR mode, write the appropriate value to the <b>MCR</b> register before writing to the <b>LCR</b> register.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> IrDA SIR Mode disabled</li> <li><b>0x1:</b> IrDA SIR Mode enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[5]	AFCE	R	<p><b>Auto Flow Control Enable</b></p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Auto Flow Control Mode disabled</li> <li><b>0x1:</b> Auto Flow Control Mode enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[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	<p><b>Address Received Bit</b></p> <p>If 9-bit data mode (<b>LCR_EXT[0]=1</b>) 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.</p> <p><b>0x1:</b> Indicates the character is address  <b>0x0:</b> Indicates the character is data</p> <p>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.</p> <p>Reading the <b>LSR</b> clears the 9bit.</p> <p> User needs to ensure that interrupt gets cleared (reading <b>LSR</b> 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.</p>

**Table 13-64 Fields For Register: LSR (continued)**

Bits	Name	R/W	Description
			<b>Value After Reset:</b> 0x0
[7]	RFE	R	<p>Receiver FIFO Error bit This bit is only relevant when FIFOs are enabled (<a href="#">FCR[0]</a> set to 1). This is used to indicate if there is at least one parity error, framing error, or break indication in the FIFO.</p> <p><b>0x0:</b> no error in Rx FIFO <b>0x1:</b> error in Rx FIFO</p> <p>This bit is cleared when the <a href="#">LSR</a> 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.</p> <p><b>Value After Reset:</b> 0x0</p>
[6]	TEMPT	R	<p>Transmitter Empty bit If FIFOs enabled (<a href="#">FCR[0]</a> 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 <a href="#">THR</a> and the Transmitter Shift Register are both empty.</p> <p><b>Value After Reset:</b> 0x1</p>
[5]	THRE	R	<p>Transmit Holding Register Empty bit If THRE mode is disabled (<a href="#">IER[7]</a> set to 0) and regardless of FIFO's being enabled or not, this bit indicates that the <a href="#">THR</a> or Tx FIFO is empty. This bit is set whenever data is transferred from the <a href="#">THR</a> or Tx FIFO to the Transmitter Shift Register and no new data has been written to the <a href="#">THR</a> or Tx FIFO. This also causes a <a href="#">THRE</a> Interrupt to occur, if the <a href="#">THRE</a> Interrupt is enabled. If both modes are active (<a href="#">IER[7]</a> set to 1 and <a href="#">FCR[0]</a> set to 1 respectively), the functionality is switched to indicate the transmitter FIFO is full, and no longer controls <a href="#">THRE</a> interrupts, which are then controlled by the <a href="#">FCR[5:4]</a> threshold setting. For more details, see <a href="#">Programmable THRE Interrupt</a>.</p> <p><b>Value After Reset:</b> 0x1</p>
[4]	BI	R	<p>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 <a href="#">LSR</a> clears the <a href="#">BI</a> bit. In non-FIFO mode, the <a href="#">BI</a> indication occurs immediately and persists until the <a href="#">LSR</a> is read.</p> <p> 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.</p> <p><b>Value After Reset:</b> 0x0</p>
[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
			<p>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.</p> <p>It should be noted that the <i>Framing Error (FE)</i> bit (<code>LSR[3]</code>) is set if a break interrupt has occurred, as indicated by <i>Break Interrupt (BI)</i> bit (<code>LSR[4]</code>). 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.</p> <p><b>0x0:</b> no framing error  <b>0x1:</b> framing error</p> <p>Reading the <code>LSR</code> clears the <code>FE</code> bit.</p> <p><b>Value After Reset:</b> 0x0</p>
[2]	PE	R	<p>Parity Error bit</p> <p>This bit used to indicate the occurrence of a parity error in the receiver if the Parity Enable (<code>PEN</code>) bit (<code>LCR[3]</code>) is set.</p> <p>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.</p> <p>It should be noted that the <i>Parity Error (PE)</i> bit (<code>LSR[2]</code>) can be set if a break interrupt has occurred, as indicated by <i>Break Interrupt (BI)</i> bit (<code>LSR[4]</code>). In this situation, the Parity Error bit is set if parity generation and detection is enabled (<code>LCR[3]=1</code>) and the parity is set to odd (<code>LCR[4]=0</code>).</p> <p><b>0x0:</b> no parity error  <b>0x1:</b> parity error</p> <p>Reading the <code>LSR</code> clears the <code>PE</code> bit.</p> <p><b>Value After Reset:</b> 0x0</p>
[1]	OE	R	<p>Overrun Error bit</p> <p>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.</p> <p>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.</p> <p><b>0x0:</b> no overrun error  <b>0x1:</b> overrun error</p> <p>Reading the <code>LSR</code> clears the <code>OE</code> bit.</p> <p><b>Value After Reset:</b> 0x0</p>
[0]	DR	R	<p>Data Ready bit</p> <p>This bit used to indicate that the receiver contains at least one character in the <code>RBR</code> or the receiver FIFO.</p> <p><b>0x0:</b> no data ready  <b>0x1:</b> data ready</p> <p>This bit is cleared when the <code>RBR</code> is read in non-FIFO mode, or when the receiver FIFO is empty, in FIFO mode.</p>

**Table 13-64 Fields For Register: LSR (continued)**

Bits	Name	R/W	Description
			<b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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: STHR<sub>i</sub>**

Bits	Name	R/W	Description
[31:9]			Reserved and read as 0
[8:0]	STHR	W	<p>This is a shadow register for the <a href="#">THR</a> 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 <a href="#">THR</a> when the <a href="#">THR Empty</a> (<a href="#">THRE</a>) bit (<a href="#">LSR[5]</a>) is set.</p> <p>If FIFOs are disabled (<a href="#">FCR[0]</a> set to 0) and <a href="#">THRE</a> is set, writing a single character to the <a href="#">THR</a> clears the <a href="#">THRE</a>. Any additional writes to the <a href="#">THR</a> before the <a href="#">THRE</a> is set again causes the <a href="#">THR</a> data to be overwritten. If FIFOs are enabled (<a href="#">FCR[0]</a> set to 1) and <a href="#">THRE</a> is set, 16 characters of data may be written to the <a href="#">THR</a> before the FIFO is full.</p> <p><b>Value After Reset:</b> 0x0</p>

**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	<p>This register is used 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 <a href="#">RBR</a> to be written by the master and the <a href="#">THR</a> to be read by the master.</p> <p><b>0x0:</b> FIFO access mode disabled  <b>0x1:</b> FIFO access mode enabled</p> <p> 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.</p> <p><b>Value After Reset:</b> 0x0</p>

**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	<p>Transmit FIFO Read</p> <p>These bits are only valid when FIFO access mode is enabled (<a href="#">FAR[0]</a> 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.</p> <p>When FIFOs are not enabled, reading this register gives the data in the <a href="#">THR</a>.</p>

**Table 13-69 Fields For Register: `TFR` (continued)**

Bits	Name	R/W	Description
			<b>Value After Reset:</b> 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	<p>Receive FIFO Framing Error This bit is only valid when FIFO access mode is enabled (<code>FAR[0]</code> 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 <code>RBR</code>.</p> <p><b>Value After Reset:</b> 0x0</p>
[8]	RFPE	W	<p>Receive FIFO Parity Error This bit is only valid when FIFO access mode is enabled (<code>FAR[0]</code> 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 <code>RBR</code>.</p> <p><b>Value After Reset:</b> 0x0</p>
[7:0]	RFWD	W	<p>Receive FIFO Write Data These bits are only valid when FIFO access mode is enabled (<code>FAR[0]</code> is set to 1). When FIFOs are enabled, the data that is written to the <code>RFWD</code> 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 <code>RFWD</code> is pushed into the <code>RBR</code>.</p> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p>Receive FIFO Full This bit used to indicate that the receive FIFO is completely full.</p> <p><b>0x0:</b> Receive FIFO not full <b>0x1:</b> Receive FIFO Full</p> <p>This bit is cleared when the Rx FIFO is no longer full.</p> <p><b>Value After Reset:</b> 0x0</p>

**Table 13-71 Fields For Register: `USR` (continued)**

Bits	Name	R/W	Description
[3]	RFNE	R	<p>Receive FIFO Not Empty This bit used to indicate that the receive FIFO contains one or more entries.</p> <p><b>0x0:</b> Receive FIFO is empty <b>0x1:</b> Receive FIFO is not empty</p> <p>This bit is cleared when the Rx FIFO is empty. <b>Value After Reset:</b> 0x0</p>
[2]	TFE	R	<p>Transmit FIFO Empty This is used to indicate that the transmit FIFO is completely empty.</p> <p><b>0x0:</b> Transmit FIFO is not empty <b>0x1:</b> Transmit FIFO is empty</p> <p>This bit is cleared when the Tx FIFO is no longer empty. <b>Value After Reset:</b> 0x1</p>
[1]	TFNF	R	<p>Transmit FIFO Not Full This bit used to indicate that the transmit FIFO is not full.</p> <p><b>0x0:</b> Transmit FIFO is full <b>0x1:</b> Transmit FIFO is not full</p> <p>This bit is cleared when the Tx FIFO is full. <b>Value After Reset:</b> 0x1</p>
[0]	BUSY	R	<p>UART Busy This bit indicates that a serial transfer is in progress; when cleared, indicates that the UART is idle or inactive.</p> <p><b>0x0:</b> UART is idle or inactive <b>0x1:</b> UART is busy (actively transferring data)</p> <p>This bit will be set to 1 (busy) under any of the following conditions: Transmission in progress on serial interface Transmit data present in <code>THR</code>, when FIFO access mode is not being used (<code>FAR = 0</code>) and the baud divisor is non-zero (<code>{DLH, DLL}</code> does not equal 0) when the divisor latch access bit is 0 (<code>LCR.DLAB= 0</code>) Reception in progress on the interface Receive data present in <code>RBR</code>, when FIFO access mode is not being used (<code>FAR = 0</code>)</p> <p> 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 <code>THR</code> and <code>RBR</code> 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.</p> <p><b>Value After Reset:</b> 0x0</p>

### 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 indicates the number of data entries in the transmit FIFO. <b>Value After Reset:</b> 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 indicates the number of data entries in the receive FIFO. <b>Value After Reset:</b> 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.  <b>Value After Reset:</b> 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.  <b>Value After Reset:</b> 0x0
[0]	UR	W	UART Reset

**Table 13-74 Fields For Register: [SRR](#) (continued)**

Bits	Name	R/W	Description
			This asynchronously resets the UART and synchronously removes the reset assertion. <b>Value After Reset:</b> 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]	<a href="#">SBCR</a>	R/W	<b>Shadow Break Control Bit</b> This is a shadow register for the Break Control Bit bit ( <a href="#">LCR</a> [6]), this can be used to remove the burden of having to performing a read modify write on the <a href="#">LCR</a> . 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 <a href="#">MCR</a> [4], the sout line is forced low until the Break bit is cleared. If <a href="#">MCR</a> [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. <b>Value After Reset:</b> 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]	<a href="#">SFE</a>	R/W	<b>Shadow FIFO Enable</b> This is a shadow register for the FIFO enable bit ( <a href="#">FCR</a> [0]). This can be used to remove the burden of having to store the previously written value to the <a href="#">FCR</a> 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. <b>Value After Reset:</b> 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	<p>Shadow RCVR Trigger</p> <p>This is a shadow register for the RCVR Trigger bits (<a href="#">FCR[7:6]</a>). This can be used to remove the burden of having to store the previously written value to the <a href="#">FCR</a> in memory and having to mask this value so that only the RCVR Trigger bits get updated.</p> <p>This is used to select the trigger level in the receiver FIFO at which the Received Data Available Interrupt is generated.</p> <p>The following trigger levels are supported:</p> <ul style="list-style-type: none"> <li><b>0x0</b> – 1 character in the FIFO</li> <li><b>0x1</b> – FIFO ¼ full</li> <li><b>0x2</b> – FIFO ½ full</li> <li><b>0x3</b> – FIFO 2 less than full</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

**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	<p>Shadow Tx Empty Trigger</p> <p>This is a shadow register for the Tx Empty Trigger bits (<a href="#">FCR[5:4]</a>). This can be used to remove the burden of having to store the previously written value to the <a href="#">FCR</a> in memory and having to mask this value so that only the Tx Empty Trigger bit gets updated.</p> <p>This is used to select the empty threshold level at which the <a href="#">THRE</a> Interrupts are generated when THRE mode is active. The following trigger levels are supported:</p> <ul style="list-style-type: none"> <li><b>0x0</b> – FIFO empty</li> <li><b>0x1</b> – 2 characters in the FIFO</li> <li><b>0x2</b> – FIFO ¼ full</li> <li><b>0x3</b> – FIFO ½ full</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

**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

**Table 13-79 Fields For Register: HTx (continued)**

Bits	Name	R/W	Description
[0]	HTx	R/W	<p>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.</p> <p><b>0x0:</b> Halt Tx disabled  <b>0x1:</b> Halt Tx enabled</p> <p> If FIFOs are not enabled, the setting of the Halt Tx register has no effect on operation.</p> <p><b>Value After Reset:</b> 0x0</p>

**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	<p>DMA Software Acknowledge</p> <p>This register is use to perform DMA software acknowledge if a transfer needs to be terminated due to an error condition.</p> <p> This bit is 'self-clearing' and it is not necessary to clear this bit.</p> <p><b>Values:</b></p> <p><b>0x1:</b> DMA software acknowledge</p> <p><b>Value After Reset:</b> 0x0</p>

**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 to 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.
[4:3]	XFER_MODE	R/W	<p>Transfer Mode.</p> <p><b>Values:</b></p>

**Table 13-81 Fields For Register: TCR (continued)**

Bits	Name	R/W	Description
			<p><b>0x0:</b> In this mode, transmit and receive can happen simultaneously. The user can enable <a href="#">DE_EN</a>, <a href="#">RE_EN</a> at any point of time.</p> <p> Turn around timing as programmed in the <a href="#">TAT</a> register is not applicable in this mode.</p> <p><b>0x1:</b> In this mode, <a href="#">de</a> and <a href="#">re</a> are mutually exclusive. Either <a href="#">de</a> or <a href="#">re</a> only one of them is expected to be enabled through programming. Hardware will consider the Turn Around timings which are programmed in the <a href="#">TAT</a> 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.</p> <p><b>0x2:</b> In this mode, <a href="#">de</a> and <a href="#">re</a> are mutually exclusive. Once <a href="#">DE_EN</a> / <a href="#">RE_EN</a> is programmed - by default <a href="#">re</a> 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 <a href="#">re</a> and enable <a href="#">de</a> signal. Once the TX FIFO becomes empty, <a href="#">re</a> signal gets enabled and <a href="#">de</a> signal will be disabled. In this mode of operation hardware will consider the Turn Around timings which are programmed in the <a href="#">TAT</a> register while switching from Receiver Enable to Driver Enable or Driver Enable to Receiver Enable. In this mode, <a href="#">de</a> and <a href="#">re</a> signals are strictly complementary to each other.</p> <p><b>Value After Reset:</b> 0x0</p>
[2]	<a href="#">DE_POL</a>	R/W	<p>Driver Enable Polarity.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <a href="#">de</a> signal is active low</li> <li><b>0x1:</b> <a href="#">de</a> signal is active high</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[1]	<a href="#">RE_POL</a>	R/W	<p>Receiver Enable Polarity.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <a href="#">re</a> signal is active low</li> <li><b>0x1:</b> <a href="#">re</a> signal is active high</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[0]	<a href="#">RS485_EN</a>	R/W	<p>RS485 Transfer Enable.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> UART Mode. All other fields in this register are reserved and registers <a href="#">DE_EN</a> / <a href="#">RE_EN</a> / <a href="#">TAT</a> are also reserved.</li> <li><b>0x1:</b> RS485 Mode. In this mode, the transfers will happen in RS485 mode. All other fields of this register are applicable.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p>DE Enable control. The register bit is used to control assertion and de-assertion of Driven enable (<code>de</code>) signal.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> De-assert <code>de</code> signal</li> <li><b>0x1:</b> Assert <code>de</code> signal</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p>RE Enable control. The register bit is used to control assertion and de-assertion of <code>re</code> signal.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0::</b> De-assert <code>re</code> signal</li> <li><b>0x1::</b> Assert <code>re</code> signal</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p>Driver Enable de-assertion time. 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 (<code>de</code>) signal.</p> <p><b>Value After Reset:</b> 0x0</p>

**Table 13-84 Fields For Register: DET (continued)**

Bits	Name	R/W	Description
[15:8]			Reserved and read as 0.
[7:0]	DE_Assertion_Time	R/W	<p>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 (<code>de</code>) signal to serial transmit enable. Any data in transmit buffer, will start on serial output after the transmit enable.</p> <p><b>Value After Reset:</b> 0x0</p>

**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	<p>Receiver Enable to Driver Enable TurnAround time. Turnaround time (in terms of serial clock) for <code>re</code> De-assertion to <code>de</code> assertion.</p> <p> • If the <code>de</code> assertion time in the <code>DET</code> register is 0, then the actual value is the programmed value + 3 • If the <code>de</code> assertion time in the <code>DET</code> register is 1, then the actual value is the programmed value + 2. • If the <code>de</code> assertion time in the <code>DET</code> register is greater than 1, then the actual value is the programmed value + 1.</p> <p><b>Value After Reset:</b> 0x0</p>
[15:0]	DE_to_RE	R/W	<p>Driver Enable to Receiver Enable TurnAround time. Turnaround time (in terms of serial clock) for <code>de</code> De-assertion to <code>re</code> assertion.</p> <p> The actual time is the programmed value + 1.</p> <p><b>Value After Reset:</b> 0x0</p>

**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.

**Table 13-86 Fields For Register: `DLF` (continued)**

Bits	Name	R/W	Description
			<p>The fractional value is added to integer value set by <code>DLH</code>, <code>DLL</code>. For information on <code>DLF</code> values, see the <a href="#">Fractional Baud Rate Support</a>.</p> <p><b>Value After Reset:</b> 0x0</p>

**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]	<code>RAR</code>	R/W	<p>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.</p> <p> • This register is applicable only when <code>LCR_EXT[1]</code> and <code>LCR_EXT[0]</code> bits are set to 1.</p> <p>• <code>RAR</code> 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.</p> <p><b>Value After Reset:</b> 0x0</p>

**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]	<code>TAR</code>	R/W	<p>Address matching register during transmit mode</p> <p>If <code>LCR_EXT[0]</code> 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 <code>SEND_ADDR</code> bit of <code>LCR_EXT</code> register is set to 1.</p> <p> • This register is used only to send the address. The normal data should be sent by programming <code>THR</code> register.</p> <p>• Once the address is started to send on the UART serial lane, then <code>SEND_ADDR</code> bit will be auto-cleared by the hardware.</p> <p><b>Value After Reset:</b> 0x0</p>

### 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]	<code>TRANSMIT_MODE</code>	R/W	<p>Transmit mode control bit This bit is used to control the type of transmit mode during 9-bit data transfers.</p> <p><b>0x1:</b> In this mode of operation, <a href="#">Transmit Holding Register (<code>THR</code>)</a> and <a href="#">Shadow Transmit Holding Register (<code>STHR</code>)</a> are 9-bit wide. The user needs to ensure that the <code>THR/STHR</code> register is written correctly for address/data. Address: 9th bit is set to 1 Data : 9th bit is set to 0.</p> <p> <a href="#">Transmit Address Register (<code>TAR</code>)</a> is not applicable in this mode of operation.</p> <p><b>0x0:</b> In this mode of operation, <a href="#">Transmit Holding Register (<code>THR</code>)</a> and <a href="#">Shadow Transmit Holding Register (<code>STHR</code>)</a> are 8-bit wide. The user needs to program the address into <a href="#">Transmit Address Register (<code>TAR</code>)</a> and data into the <code>THR/STHR</code> register. <code>SEND_ADDR</code> bit is used as a control knob to indicate the UART on when to send the address.</p> <p><b>Value After Reset:</b> 0x0</p>
[2]	<code>SEND_ADDR</code>	R/W	<p>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.</p> <p><b>0x1:</b> 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 <a href="#">Transmit Address Register (<code>TAR</code>)</a>.</p> <p><b>0x0:</b> 9-bit character will be transmitted with 9-th bit set to 0 and the remaining 8-bits will be taken from the TxFIFO which is programmed through 8-bit wide <code>THR/STHR</code> register.</p> <p> 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 <code>DLS_E</code> bit is set to 1 and <code>TRANSMIT_MODE</code> is set to 0.</p> <p><b>Value After Reset:</b> 0x0</p>
[1]	<code>ADDR_MATCH</code>	R/W	<p>Address Match Mode This bit is used to enable the address match feature during receive.</p> <p><b>0x1:</b> 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 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.</p>

**Table 13-89 Fields For Register: LCR\_EXT (continued)**

Bits	Name	R/W	Description
			<p><b>0x0:</b> Normal mode; UART will start to receive the data and 9-bit character will be formed and written into the receive RxFIFO. User is responsible to read the data and differentiate b/n address and data.</p> <p> This field is applicable only when <code>DLS_E</code> is set to 1.</p> <p><b>Value After Reset:</b> 0x0</p>
[0]	<code>DLS_E</code>	R/W	<p>Extension for <code>DLS</code>. This bit is used to enable 9-bit data for transmit and receive transfers.</p> <p><b>Value After Reset:</b> 0x0</p>

### 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]	<code>TIMEOUT_RST</code>	R/W	<p>This field holds reset value of the Register timeout counter.</p> <p><b>Value After Reset:</b> 0x80</p>

## 13.4. SPI & QSPI

This section describes the *Serial Peripheral Interface (SPI)* and *Quad Serial Peripheral Interface (QSPI)* controllers of the BE-U1000.

 Below both SPI and QSPI controllers are referred to as \*SPI.

\*SPI controller is a programmable component used for the serial communication with peripherals.

There are six \*SPI controllers in BE-U1000:

- two SPI controllers that acts as a serial master (below referred to as SPI Master)
- two SPI controllers that acts as a serial slave (below referred to as SPI Slave)
- two *Quad SPI (QSPI)* controllers

A simplified functional block diagram of the \*SPI controller is illustrated in the following figure.

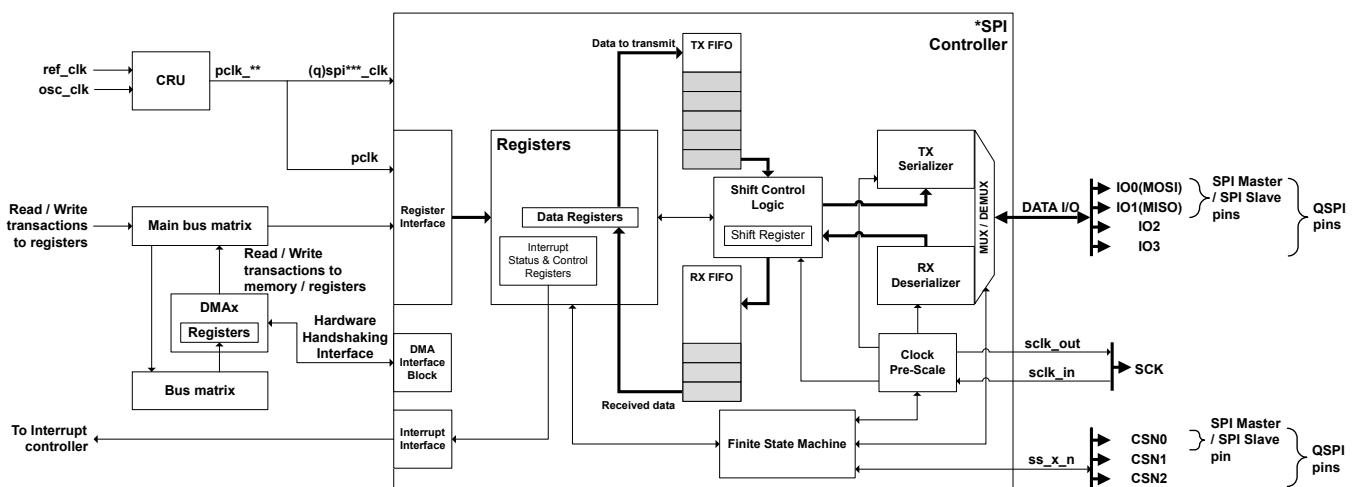


Figure 13-7 \*SPI controller functional block diagram



In the figure above:

- Double asterisk(\*\*) indicates 0 for SPI\_0, SPI\_1, QSPI\_0 and 1 for SPI\_2, SPI\_3, QSPI\_1
- 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 mentioned 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

In this chapter:

- **\*SPI Clock**
- **Endian Conversion Support**

- Transmit and Receive FIFO Buffers
- Transfer Modes
- Dual Data-Rate Support
- 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:

$$F_{sclk\_out} = F_{spi*\_clk} / SCKDV$$

`SCKDV` is a bit field in the programmable register `BAUDR`, holding any even value in the range 0 to 65,534. 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.

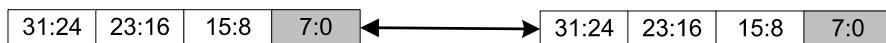
### 13.4.1.2. Endian Conversion Support



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).

data frame size = 8



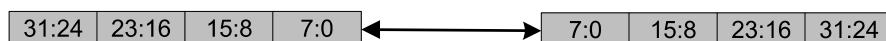
data frame size = 16



data frame size = 24



data frame size = 32



**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*_rxr_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	<code>spi*_txe_intr</code> is asserted when 0 data entries are present in transmit FIFO
001	<code>spi*_txe_intr</code> is asserted when 1 data entries are present in transmit FIFO
010	<code>spi*_txe_intr</code> is asserted when 2 data entries are present in transmit FIFO
011	<code>spi*_txe_intr</code> is asserted when 3 data entries are present in transmit FIFO
100	<code>spi*_txe_intr</code> is asserted when 4 data entries are present in transmit FIFO
101	<code>spi*_txe_intr</code> is asserted when 5 data entries are present in transmit FIFO

**Table 13-92 Transmit FIFO Threshold (TFT) Decode Values (continued)**

TFT Value	Description
110	<code>spi*_txe_intr</code> is asserted when 6 data entries are present in transmit FIFO
111	<code>spi*_txe_intr</code> is asserted when 7 data entries are present in transmit FIFO

**Table 13-93 Receive FIFO Threshold (RFT) Decode Values**

RFT Value	Description
000	<code>spi*_rxf_intr</code> is asserted when 1 data entries are present in receive FIFO
001	<code>spi*_rxf_intr</code> is asserted when 2 data entries are present in receive FIFO
010	<code>spi*_rxf_intr</code> is asserted when 3 data entries are present in receive FIFO
011	<code>spi*_rxf_intr</code> is asserted when 4 data entries are present in receive FIFO
100	<code>spi*_rxf_intr</code> is asserted when 5 data entries are present in receive FIFO
101	<code>spi*_rxf_intr</code> is asserted when 6 data entries are present in receive FIFO
110	<code>spi*_rxf_intr</code> is asserted when 7 data entries are present in receive FIFO
111	<code>spi*_rxf_intr</code> 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 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. 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*_rxr_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

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	4KB	0x1100_0000	0x1100_0FFF
	SPI_0	4KB	0x1100_B000	0x1100_BFFF
	SPI_1	4KB	0x1100_C000	0x1100_CFFF
Periphery 1	QSPI_1	4KB	0x1300_0000	0x1300_0FFF
	SPI_2	4KB	0x1300_B000	0x1300_BFFF
	SPI_3	4KB	0x1300_C000	0x1300_CFFF



For details, refer to [Memory Map](#) chapter.

In this chapter:

- [Registers Map](#)
- [Register Descriptions](#)

#### 13.4.2.1. Registers Map



In the tables below, the offset of a register is relative to the \*SPI controller memory region start address. The following table provides 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 Registers Map**

Register	Offset	Description	Default Value
CTRLR0	0x00	<a href="#">Control Register 0</a>	0x1070000
CTRLR1	0x04	<a href="#">Control Register 1</a> <b>Exists:QSPI/SPI Master</b>	0x0
SPIENR	0x08	<a href="#">SPI Enable Register</a>	0x0
MWCR	0xC	<a href="#">Microwire Control Register</a>	0x0
SER	0x10	<a href="#">Slave Enable Register</a> <b>Exists:QSPI/SPI Master</b>	0x0
BAUDR	0x14	<a href="#">Baud Rate Select Register</a> <b>Exists:QSPI/SPI Master</b>	0x0
TXFTLR	0x18	<a href="#">Transmit FIFO Threshold Level Register</a>	0x0
RXFTLR	0x1C	<a href="#">Receive FIFO Threshold Level Register</a>	0x0
TXFLR	0x20	<a href="#">Transmit FIFO Level Register</a>	0x0
RXFLR	0x24	<a href="#">Receive FIFO Level Register</a>	0x0
SR	0x28	<a href="#">Status Register</a>	0x6

**Table 13-95 \*SPI Registers Map (continued)**

Register	Offset	Description	Default Value
IMR	0x2C	Interrupt Mask Register	QSPI, SPI Master: 0x3FSPI 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 <b>Exists: QSPI</b>	0x200
TXD_DRIVE_EDGE	0xF8	Transmit Drive Edge Register <b>Exists: QSPI</b>	0x0
XIP_CMD	0xFC	XIP Command Register <b>Exists: QSPI</b>	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	<p> This bit field is only available for SPI Master and QSPI controllers.</p> <p>Set the Endianness for XIP and data register reads.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Endian Conversion disabled.</li> <li><b>0x1:</b> Endian Conversion enabled.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[24]	SSTE	R/W	<p><b>Slave Select Toggle Enable</b></p> <p>When operating in SPI mode with clock phase (<i>SCPH</i>) set to 0, this register controls the behavior of the slave select line (<i>ss_*_n</i>) between data frames. If this register field is set to 1 the <i>ss_*_n</i> line will toggle between consecutive data frames, with the serial clock being held to its default value while <i>ss_*_n</i> is high; if this register field is set to 0 the <i>ss_*_n</i> will stay low and serial clock will run continuously for the duration of the transfer.</p> <p><b>Value After Reset:</b> 0x1</p>
[23]			Reserved
[22:21]	SPI_FRF	*Varies	<p><b>SPI Frame Format</b></p> <p>Selects data frame format for Transmitting/Receiving the data Bits.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> (<i>STD_SPI_FRF</i>): Standard SPI Frame Format</li> <li><b>0x1:</b> (<i>DUAL_SPI_FRF</i>): Dual SPI Frame Format</li> <li><b>0x2:</b> (<i>QUAD_SPI_FRF</i>): Quad SPI Frame Format</li> <li><b>0x3:</b> Reserved</li> </ul> <p> The access attributes of this field are the following:</p> <p><b>QSPI:</b> R/W  <b>SPI Slave/SPI Master:</b> R</p> <p><b>Value After Reset:</b> 0x0</p>
[20:16]	DFS_32	R/W	<p><b>Data Frame Size</b> in 32-bit transfer size mode.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x7:</b> 8-bit serial data transfer</li> </ul> <p><b>Value After Reset:</b> 0x7</p>
[15:12]	CFS	R/W	<p><b>Control Frame Size</b></p> <p>Selects the length of the control word for the Microwire frame format.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> 1-bit Control Word</li> <li><b>0x1:</b> 2-bit Control Word</li> <li><b>0x2:</b> 3-bit Control Word</li> <li><b>0x3:</b> 4-bit Control Word</li> <li><b>0x4:</b> 5-bit Control Word</li> <li><b>0x5:</b> 6-bit Control Word</li> </ul>

**Table 13-96 Fields For Register: CTRLR0 (continued)**

Bits	Name	R/W	Description
			<p><b>0x6:</b> 7-bit Control Word  <b>0x7:</b> 8-bit Control Word  <b>0x8:</b> 9-bit Control Word  <b>0x9:</b> 10-bit Control Word  <b>0xA:</b> 11-bit Control Word  <b>0xB:</b> 12-bit Control Word  <b>0xC:</b> 13-bit Control Word  <b>0xD:</b> 14-bit Control Word  <b>0xE:</b> 15-bit Control Word  <b>0xF:</b> 16-bit Control Word</p> <p><b>Value After Reset:</b> 0x0</p>
[11]	SRL	R/W	<p><b>Shift Register Loop</b>            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 <code>ss_in_n</code> and <code>spi*_clk</code> 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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Test mode: Tx &amp; Rx shift reg connected</li> <li><b>0x0:</b> Normal mode operation</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[10]	SLV_OE	R	<p><b>Slave Output Enable</b>            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 after reset.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Slave Output is disabled</li> <li><b>0x0:</b> Slave Output is enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[9:8]	TMOD	R/W	<p><b>Transfer Mode</b>            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.</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>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</p>

**Table 13-96 Fields For Register: CTRLR0 (continued)**

Bits	Name	R/W	Description
			<p>FIFO is considered control data in this mode. This transfer mode is only valid when the QSPI controller is configured as a master device.</p> <p><b>Values for SPI_FRF=0:</b></p> <ul style="list-style-type: none"> <li><b>0x0 (TX_AND_RX):</b> Transmit &amp; receive</li> <li><b>0x1 (TX_ONLY):</b> Transmit only mode or Write (<code>SPI_FRF != 0</code>)</li> <li><b>0x2 (RX_ONLY):</b> Receive only mode or Read (<code>SPI_FRF != 0</code>)</li> <li><b>0x3 (EEPROM_READ):</b> EEPROM Read mode</li> </ul> <p><b>Values for SPI_FRF != 0:</b></p> <ul style="list-style-type: none"> <li><b>0x1 (TX_ONLY):</b> Write</li> <li><b>0x2 (RX_ONLY):</b> Read</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[7]	SCPOL	R/W	<p><b>Serial Clock Polarity</b></p> <p>Valid when the frame format (<code>FRF</code>) is set to Motorola SPI. 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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0 (SCLK_LOW):</b> Inactive state of serial clock is low</li> <li><b>0x1 (SCLK_HIGH):</b> Inactive state of serial clock is high</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[6]	SCPH	R/W	<p><b>Serial Clock Phase</b></p> <p>Valid when the frame format (<code>FRF</code>) is set to Motorola SPI. The serial clock phase selects the relationship of the serial clock with the slave select signal. When <code>SCPH = 0</code>, data are captured on the first edge of the serial clock. When <code>SCPH = 1</code>, 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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0: (SCPH_MIDDLE):</b> Serial clock toggles in middle of first data bit</li> <li><b>0x1: (SCPH_START):</b> Serial clock toggles at start of first data bit</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[5:4]	FRF	R/W	<p><b>Frame Format</b></p> <p>Selects which serial protocol transfers the data. This field should be set to 0.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Motorola SPI Frame Format</li> <li><b>0x1:</b> Texas Instruments SSP Frame Format</li> <li><b>0x2:</b> National Microwire Frame Format</li> <li><b>0x3:</b> Reserved</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[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	<p><b>Number of Data Frames</b>  When <b>TMOD</b> bit of <b>CTRLR0</b> register is 0x2 or <b>TMOD</b> bit of <b>CTRLR0</b> 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.  <b>Value After Reset:</b> 0x0</p>

#### 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	<p><b>SPI* Enable</b>  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.  <b>Values:</b>  <b>0x0:</b> Disables Serial Transfer  <b>0x1:</b> Enables Serial Transfer  <b>Value After Reset:</b> 0x0</p>

#### 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	<p><b>Microwire Handshaking.</b></p> <p>This bit field is only available for QSPI and SPI Master controllers.</p> <p>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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Handshaking interface is disabled</li> <li><b>0x1:</b> Handshaking interface is enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[1]	MDD	R/W	<p><b>Microwire Control.</b></p> <p>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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> SPI* receives data</li> <li><b>0x1:</b> SPI* transmits data</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[0]	MWMOD	R/W	<p><b>Microwire Transfer Mode.</b></p> <p>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 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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Non-Sequential Microwire Transfer</li> <li><b>0x1:</b> Sequential Microwire Transfer</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

### 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. Two slave-select output pins are available on the \*SPI controller master. 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	<p><b>Slave Select Enable Flag</b>  Each bit in this field corresponds to a slave select line (<code>ss_x_n</code>) from the *SPI controller master, where <code>x</code> 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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> No slave selected</li> <li><b>0x1:</b> Slave is selected</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

#### 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	<p><b>SPI* Clock Divider</b>  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 (<code>sclk_out</code>) is disabled. The frequency of the <code>sclk_out</code> is derived from the following equation:  <math>F_{sclk\_out} = F_{spi*\_clk} / SCKDV</math>  where <code>SCKDV</code> is any even value between 2 and 65534. For example:  for <code>F_{spi*\_clk}</code> = 3.6864MHz and <code>SCKDV</code> = 2 <code>F_{sclk\_out}</code> = 3.6864/2 = 1.8432MHz  <b>Value After Reset:</b> 0x0</p>

#### 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	<b>Transmit FIFO Threshold</b> 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. <code>spi*_txe_intr</code> is asserted when TFT or less data entries are present in transmit FIFO <b>Value After Reset:</b> 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	<b>Receive FIFO Threshold</b> 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. <code>spi*_rxf_intr</code> is asserted when RFT or more data entries are present in receive FIFO <b>Value After Reset:</b> 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

**Table 13-104 Fields For Register: TXFLR (continued)**

Bits	Name	R/W	Description
[3:0]	TXTFL	R	<b>Transmit FIFO Level</b> Contains the number of valid data entries in the transmit FIFO. <b>Value After Reset:</b> 0x0 <b>Volatile:</b> 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	<b>Receive FIFO Level</b> Contains the number of valid data entries in the receive FIFO. <b>Value After Reset:</b> 0x0 <b>Volatile:</b> 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	<b>Data Collision Error</b>  This bit field is available only for QSPI and SPI Master controllers. This bit will be set if <code>ss_in_n</code> 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. <b>Values:</b> <b>0x0:</b> No Error <b>0x1:</b> Transmit Data Collision Error <b>Value After Reset:</b> 0x0 <b>Volatile:</b> true

**Table 13-106 Fields For Register: SR (continued)**

Bits	Name	R/W	Description
[5]	TXE	R	<p><b>Transmission Error</b></p>  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. <b>Values:</b> <b>0x0:</b> No Error <b>0x1:</b> Transmission Error <b>Value After Reset:</b> 0x0 <b>Volatile:</b> true
[4]	RFF	R	<p><b>Receive FIFO Full</b></p> 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. <b>Values:</b> <b>0x0:</b> Receive FIFO is not full <b>0x1:</b> Receive FIFO is full <b>Value After Reset:</b> 0x0 <b>Volatile:</b> true
[3]	RFNE	R	<p><b>Receive FIFO Not Empty</b></p> 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. <b>Values:</b> <b>0x0:</b> Receive FIFO is empty <b>0x1:</b> Receive FIFO is not empty <b>Value After Reset:</b> 0x0 <b>Volatile:</b> true
[2]	TFE	R	<p><b>Transmit FIFO Empty</b></p> 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. <b>Values:</b> <b>0x0:</b> Transmit FIFO is not empty <b>0x1:</b> Transmit FIFO is empty <b>Value After Reset:</b> 0x1 <b>Volatile:</b> true
[1]	TFNF	R	<p><b>Transmit FIFO Not Full</b></p> Set when the transmit FIFO contains one or more empty locations, and is cleared when the FIFO is full. <b>Values:</b> <b>0x0:</b> Transmit FIFO is full <b>0x1:</b> Transmit FIFO is not Full <b>Value After Reset:</b> 0x1

**Table 13-106 Fields For Register: SR (continued)**

Bits	Name	R/W	Description
			<b>Volatile:</b> true
[0]	BUSY	R	<b>SPI* Busy Flag</b> When set, indicates that a serial transfer is in progress; when cleared indicates that the *SPI controller is idle or disabled. <b>Values:</b> 0x0: *SPI controller is idle or disabled 0x1: *SPI controller is actively transferring data <b>Value After Reset:</b> 0x0 <b>Volatile:</b> true

#### 13.4.2.2.12. Interrupt Mask Register (IMR)

The IMR register is at offset 0x2C.

This read/write register 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. <b>Multi-Master Contention Interrupt Mask</b> <b>Values:</b> 0x0: <code>spi*_mst_intr</code> interrupt is masked 0x1: <code>spi*_mst_intr</code> interrupt is not masked <b>Value After Reset:</b> 0x1
[4]	RXFIM	R/W	<b>Receive FIFO Full Interrupt Mask</b> <b>Values:</b> 0x0: <code>spi*_rfx_intr</code> interrupt is masked 0x1: <code>spi*_rfx_intr</code> interrupt is not masked <b>Value After Reset:</b> 0x1
[3]	RXOIM	R/W	<b>Receive FIFO Overflow Interrupt Mask</b> <b>Values:</b> 0x0: <code>spi*_rxo_intr</code> interrupt is masked 0x1: <code>spi*_rxo_intr</code> interrupt is not masked <b>Value After Reset:</b> 0x1
[2]	RXUIM	R/W	<b>Receive FIFO Underflow Interrupt Mask</b> <b>Values:</b> 0x0: <code>spi*_rxu_intr</code> interrupt is masked 0x1: <code>spi*_rxu_intr</code> interrupt is not masked <b>Value After Reset:</b> 0x1

**Table 13-107 Fields For Register: IMR (continued)**

Bits	Name	R/W	Description
[1]	TXOIM	R/W	<p>Transmit FIFO Overflow Interrupt Mask</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <code>spi_*_txo_intr</code> interrupt is masked</li> <li><b>0x1:</b> <code>spi_*_txo_intr</code> interrupt is not masked</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[0]	TXEIM	R/W	<p>Transmit FIFO Empty Interrupt Mask</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <code>spi_*_txe_intr</code> interrupt is masked</li> <li><b>0x1:</b> <code>spi_*_txe_intr</code> interrupt is not masked</li> </ul> <p><b>Value After Reset:</b> 0x1</p>

### 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	<p> This bit is available only for QSPI and SPI Master controllers.</p> <p>Multi-Master Contention Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <code>spi_*_mst_intr</code> interrupt not active after masking</li> <li><b>0x1:</b> <code>spi_*_mst_intr</code> interrupt is active after masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[4]	RXFIS	R	<p>Receive FIFO Full Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <code>spi_*_rfx_intr</code> interrupt is not active after masking</li> <li><b>0x1:</b> <code>spi_*_rfx_intr</code> interrupt is full after masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[3]	RXOIS	R	<p>Receive FIFO Overflow Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <code>spi_*_rxo_intr</code> interrupt is not active after masking</li> <li><b>0x1:</b> <code>spi_*_rxo_intr</code> interrupt is active after masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[2]	RXUIS	R	Receive FIFO Underflow Interrupt Status

**Table 13-108 Fields For Register: ISR (continued)**

Bits	Name	R/W	Description
			<b>Values:</b> <b>0x0:</b> <code>spi*_rxu_intr</code> interrupt is not active after masking <b>0x1:</b> <code>spi*_rxu_intr</code> interrupt is active after masking <b>Value After Reset:</b> 0x0 <b>Volatile:</b> true
[1]	TXOIS	R	Transmit FIFO Overflow Interrupt Status <b>Values:</b> <b>0x0:</b> <code>spi*_txo_intr</code> interrupt is not active after masking <b>0x1:</b> <code>spi*_txo_intr</code> interrupt is active after masking <b>Value After Reset:</b> 0x0 <b>Volatile:</b> true
[0]	TXEIS	R	Transmit FIFO Empty Interrupt Status <b>Values:</b> <b>0x0:</b> <code>spi*_txe_intr</code> interrupt is not active after masking <b>0x1:</b> <code>spi*_txe_intr</code> interrupt is active after masking <b>Value After Reset:</b> 0x0 <b>Volatile:</b> 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 <b>Values:</b> <b>0x0:</b> <code>spi*_mst_intr</code> interrupt is not active prior to masking <b>0x1:</b> <code>spi*_mst_intr</code> interrupt is active prior to masking <b>Value After Reset:</b> 0x0 <b>Volatile:</b> true
[4]	RXFIR	R	Receive FIFO Full Raw Interrupt Status <b>Values:</b> <b>0x0:</b> <code>spi*_rfx_intr</code> interrupt is not active prior to masking <b>0x1:</b> <code>spi*_rfx_intr</code> interrupt is active prior to masking <b>Value After Reset:</b> 0x0 <b>Volatile:</b> true

**Table 13-109 Fields For Register: RISR (continued)**

Bits	Name	R/W	Description
[3]	RXOIR	R	<p>Receive FIFO Overflow Raw Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> <code>spi*_rxo_intr</code> interrupt is not active prior to masking</li> <li><b>0x0:</b> <code>spi*_rxo_intr</code> interrupt is active prior masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[2]	RXUIR	R	<p>Receive FIFO Underflow Raw Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <code>spi*_rxu_intr</code> interrupt is not active prior to masking</li> <li><b>0x1:</b> <code>spi*_rxu_intr</code> interrupt is active prior to masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[1]	TXOIR	R	<p>Transmit FIFO Overflow Raw Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <code>spi*_txo_intr</code> interrupt is not active prior to masking</li> <li><b>0x1:</b> <code>spi*_txo_intr</code> interrupt is active prior masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[0]	TXEIR	R	<p>Transmit FIFO Empty Raw Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> <code>spi*_txe_intr</code> interrupt is not active prior to masking</li> <li><b>0x1:</b> <code>spi*_txe_intr</code> interrupt is active prior masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>

**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	<p><b>Clear Transmit FIFO Overflow Interrupt</b></p> <p>This register reflects the status of the interrupt. A read from this register clears the <code>spi*_txo_intr</code> interrupt; writing has no effect.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>

**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	<b>Clear Receive FIFO Overflow Interrupt</b> This register reflects the status of the interrupt. A read from this register clears the <code>spi*_rxo_intr</code> interrupt; writing has no effect. <b>Value After Reset:</b> 0x0 <b>Volatile:</b> 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	<b>Clear Receive FIFO Underflow Interrupt</b> This register reflects the status of the interrupt. A read from this register clears the <code>spi*_rxu_intr</code> interrupt; writing has no effect. <b>Value After Reset:</b> 0x0 <b>Volatile:</b> 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	<b>Clear Multi-Master Contention Interrupt</b> This register reflects the status of the interrupt. A read from this register clears the <code>spi*_mst_intr</code> interrupt; writing has no effect. <b>Value After Reset:</b> 0x0 <b>Volatile:</b> 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

**Table 13-114 Fields For Register: ICR (continued)**

Bits	Name	R/W	Description
[0]	ICR	R	<p><b>Clear Interrupts</b>            This register is set if any of the interrupts below is active. A read clears the <code>spi*_txo_intr</code>, <code>spi*_rxu_intr</code>, <code>spi*_rxo_intr</code>, and the <code>spi*_mst_intr</code> interrupts. Writing to this register has no effect.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>

#### 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	<p><b>Transmit DMA Enable</b>            This bit enables/disables the transmit FIFO DMA channel.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Transmit DMA disabled</li> <li><b>0x1:</b> Transmit DMA enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[0]	RDMAE	R/W	<p><b>Receive DMA Enable</b>            This bit enables/disables the receive FIFO DMA channel</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Receive DMA disabled</li> <li><b>0x1:</b> Receive DMA enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

#### 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	<p><b>Transmit Data Level</b>            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 <code>DMACR.TDMAE = 1</code>.</p> <p><b>Value After Reset:</b> 0x0</p>

### 13.4.2.22. 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	<p><b>Receive Data Level.</b> This bit field controls the level at which a DMA request is made by the receive logic. The watermark level = <b>DMARDL+1</b>; 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 <b>DMACR.RDMAE=1</b>.</p> <p><b>Value After Reset:</b> 0x0</p>

### 13.4.2.23. Data Registers (**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 pldata 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	<p><b>Data Register</b> When writing to this register, you must right-justify the data. Read data are automatically right-justified. All 32 bits are valid.</p> <ul style="list-style-type: none"> <li>• Read = Receive FIFO buffer.</li> <li>• Write = Transmit FIFO buffer.</li> </ul> <p><b>Value After Reset:</b> 0x0 <b>Volatile:</b> true</p>

### 13.4.2.24. 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 rxd 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	<p><b>Data Input Sample Delay</b>            This register is used to delay the sample of the data input port. Each value represents a single <code>spi*_clk</code> delay on the sample of data input.</p> <p> 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.</p> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p><b>Read data strobe enable bit</b>            Once this bit is set to 1 QSPI controller will use Read data strobe to capture read data in DDR mode.  <b>Value After Reset:</b> 0x0</p>
[17]	INST_DDR_EN	R/W	<p><b>Instruction DDR Enable bit</b>            This will enable dual-data rate transfer for instruction phase.  <b>Value After Reset:</b> 0x0</p>
[16]	SPI_DDR_EN	R/W	<p><b>SPI DDR Enable bit</b>            This will enable Dual-data rate transfers in Dual/Quad frame formats of SPI.  <b>Value After Reset:</b> 0x0</p>
[15:11]	WAIT_CYCLES	R/W	<p><b>Wait cycles</b>            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.</p>

**Table 13-120 Fields For Register: SPI\_CTRLR0 (continued)**

Bits	Name	R/W	Description
			<b>Value After Reset:</b> 0x0
[10]			Reserved
[9:8]	INST_L	R/W	<p><b>Instruction Length</b>  Dual/Quad mode instruction length in bits.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0 (INST_L_0):</b> 0-bit (No Instruction)</li> <li><b>0x1 (INST_L_1):</b> 4-bit Instruction</li> <li><b>0x2 (INST_L_2):</b> 8-bit Instruction</li> <li><b>0x3 (INST_L_3):</b> 16-bit Instruction</li> </ul> <p><b>Value After Reset:</b> 0x2</p>
[7:6]			Reserved
[5:2]	ADDR_L	R/W	<p><b>Address Length</b>  This bit defines Length of Address to be transmitted. Only after this much bits are programmed in to the FIFO the transfer can begin.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0 (ADDR_L_0):</b> 0-bit Address Width</li> <li><b>0x1 (ADDR_L_1):</b> 4-bit Address Width</li> <li><b>0x2 (ADDR_L_2):</b> 8-bit Address Width</li> <li><b>0x3 (ADDR_L_3):</b> 12-bit Address Width</li> <li><b>0x4 (ADDR_L_4):</b> 16-bit Address Width</li> <li><b>0x5 (ADDR_L_5):</b> 20-bit Address Width</li> <li><b>0x6 (ADDR_L_6):</b> 24-bit Address Width</li> <li><b>0x7 (ADDR_L_7):</b> 28-bit Address Width</li> <li><b>0x8 (ADDR_L_8):</b> 32-bit Address Width</li> <li><b>0x9 (ADDR_L_9):</b> 36-bit Address Width</li> <li><b>0xA (ADDR_L_10):</b> 40-bit Address Width</li> <li><b>0xB (ADDR_L_11):</b> 44-bit Address Width</li> <li><b>0xC (ADDR_L_12):</b> 48-bit Address Width</li> <li><b>0xD (ADDR_L_13):</b> 52-bit Address Width</li> <li><b>0xE (ADDR_L_14):</b> 56-bit Address Width</li> <li><b>0xF (ADDR_L_15):</b> 60-bit Address Width</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[1:0]	TRANS_TYPE	R/W	<p><b>Address and instruction transfer format</b>  Selects whether QSPI controller will transmit instruction/address either in Standard SPI mode or the SPI mode selected in <a href="#">CTRLR0.SPI_FRF</a> field.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Instruction and Address will be sent in Standard SPI Mode</li> <li><b>0x1:</b> Instruction will be sent in Standard SPI Mode and Address will be sent in the mode specified by <a href="#">CTRLR0.SPI_FRF</a></li> </ul>

**Table 13-120 Fields For Register: SPI\_CTRLR0 (continued)**

Bits	Name	R/W	Description
			<b>0x2:</b> Both Instruction and Address will be sent in the mode specified by <a href="#">CTRLR0.SPI_FRF</a> <b>0x3:</b> Reserved <b>Value After Reset:</b> 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 [TXD](#) register 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	TXD Drive edge - value of which decides the driving edge of transmit data. The maximum value of this register is = <a href="#">(BAUDR/2) -1</a> . <b>Value After Reset:</b> 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 <b>Value After Reset:</b> 0x0
[23:16]	CMD_BYTE_1	R/W	Byte 1 <b>Value After Reset:</b> 0x0

**Table 13-122 Fields For Register: XIP\_CMD (continued)**

Bits	Name	R/W	Description
[15:8]	CMD_BYTE_0	R/W	Byte 0 <b>Value After Reset:</b> 0x0
[7:4]			Reserved
[3]	S2R	R/W	Place to insert extra byte <b>Values:</b> 0x0: address phase then extra byte 0x1: extra byte then address phase <b>Value After Reset:</b> 0x0
[2:1]	CMD_SEL	R/W	Selects the contents of cmd_byte_w[7:0] <b>Values:</b> 0x0: all zeros 0x1: from the CMD_BYTE_0 field 0x2: from the CMD_BYTE_1 field 0x3: from the CMD_BYTE_2 field <b>Value After Reset:</b> 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 <b>Values:</b> 0x0: additions are ignored 0x1: additions are used <b>Value After Reset:</b> 0x0

## 13.5. I2C

This chapter describes the *Inter-Integrated Circuit (I2C)* controllers of the BE-U1000 microcontroller.

The BE-U1000 microcontroller contains four I2C controllers.

The I2C controllers are made up of a Register Interface, an I2C interface, and FIFO logic to maintain coherency between the two interfaces. A simplified block diagram of the I2C is illustrated in the following figure.

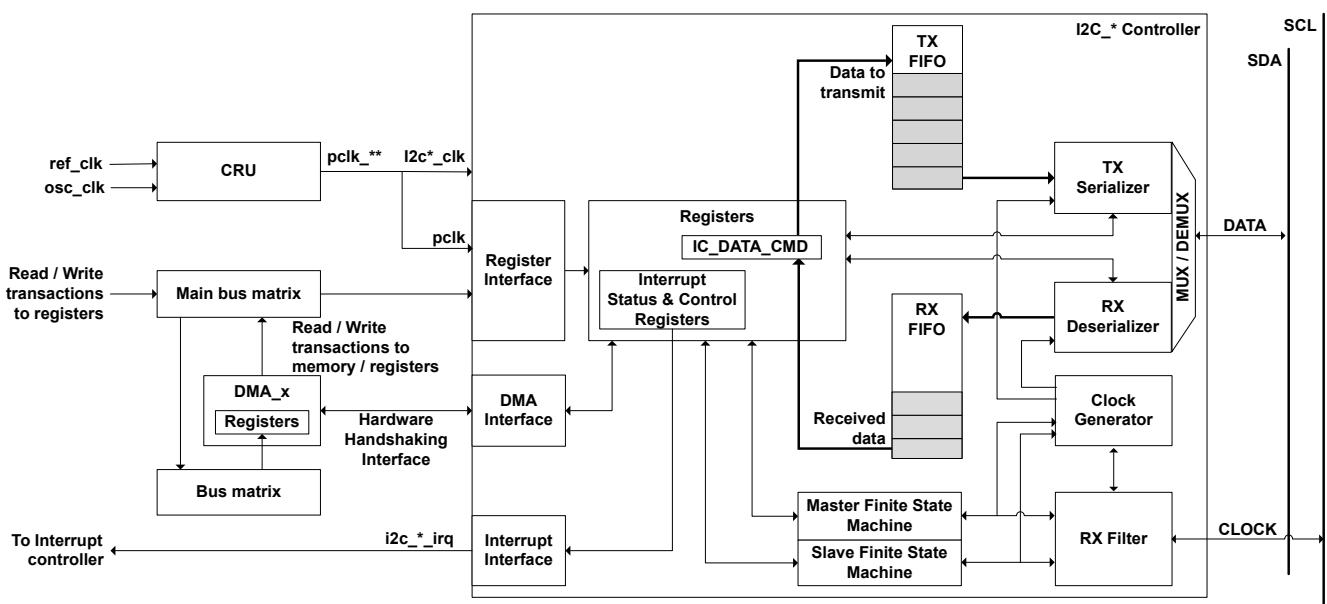


Figure 13-9 I2C Functional Block Diagram



In the figure above:

- $\times$  indicates 0 for DMA\_0 and 1 for DMA\_1. For more information, refer to the [DMA Controller](#).
- Asterisk symbol (\*) indicates 0 for I2C\_0, 1 for I2C\_1, 2 for I2C\_2 and 3 for I2C\_3.
- Double asterisk (\*\*\*) indicates 0 for I2C\_0 an I2C\_1, 1 for I2C\_2 and I2C\_3.

### 13.5.1. I2C Functional Description

In this chapter:

- [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 ( $\overline{R/W}$ ) is set to 0, the Master writes to the Slave. When bit 0 ( $\overline{R/W}$ ) is set to 1, the Master reads from the Slave.

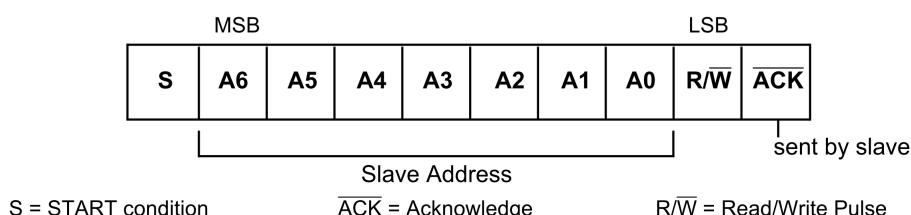
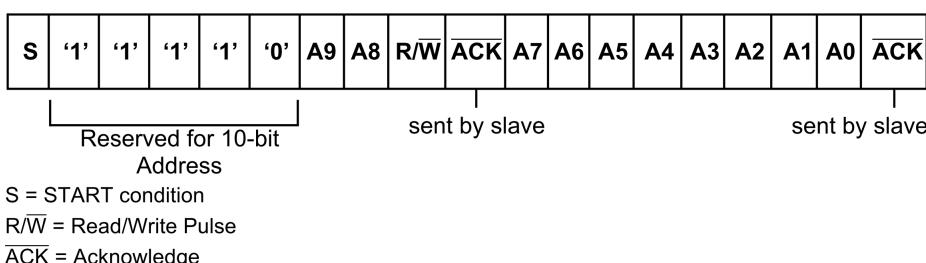


Figure 13-10 7-bit Address Format

## 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 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 50ns.
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 300ns 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).

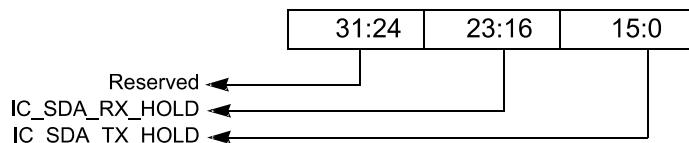


Figure 13-12 IC\_SDA\_HOLD Register

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 <code>IC_SDA_RX_HOLD</code> Value
Standard Mode	<code>IC_SS_SCL_HCNT - IC_FS_SPKLEN - 3</code>
Fast Mode or Fast Mode Plus	<code>IC_FS_SCL_HCNT - IC_FS_SPKLEN - 3</code>
High Speed	<code>IC_FS_SCL_HCNT - IC_FS_SPKLEN - 3,</code> <code>IC_HS_SCL_LCNT - IC_HS_SPKLEN - 3</code>

### 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 `tHD: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 `tHD: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 contains 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	4KB	0x1100_4000	0x1100_4FFF
	I2C1	4KB	0x1100_5000	0x1100_5FFF
Periphery 1	I2C2	4KB	0x1300_4000	0x1300_4FFF
	I2C3	4KB	0x1300_5000	0x1300_5FFF



For details, refer to [Memory Map](#) chapter.

In this chapter:

- [Registers Map](#)
- [Register Descriptions](#)

#### 13.5.2.1. Registers Map

The following table provides top-level summary of each registers for the I2C controllers. To view the detailed description of a register, click the register name in the **Description** column.

**Table 13-127 I2C Registers Map**

Name	Offset	Description	Default Value
IC_CON	0x00	<a href="#">I2C Control Register</a>	0x67
IC_TAR	0x04	<a href="#">I2C Target Address Register</a>	0x55
IC_SAR	0x08	<a href="#">I2C Slave Address Register</a>	0x55
IC_HS_MADDR	0xC	<a href="#">I2C High Speed Master Mode Code Address Register</a>	0x1
IC_DATA_CMD	0x10	<a href="#">I2C Rx/Tx Data Buffer and Command Register</a>	0x0
IC_SS_SCL_HCNT	0x14	<a href="#">Standard Speed I2C Clock SCL High Count Register</a>	0x190
IC_SS_SCL_LCNT	0x18	<a href="#">Standard Speed I2C Clock SCL Low Count Register</a>	0x1D6
IC_FS_SCL_HCNT	0x1C	<a href="#">Fast Mode or Fast Mode Plus I2C Clock SCL High Count Register</a>	0x3C

**Table 13-127 I2C Registers 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 Registers 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 of a register description 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	<p>In Master mode:</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x1: Bus Clear Feature is enabled.</li> <li>0x0: Bus Clear Feature is disabled.</li> </ul> <p>In Slave mode, this register bit is not applicable.</p> <p><b>Value After Reset:</b> 0x0</p>

**Table 13-128 Fields for Register: IC\_CON (continued)**

Bits	Name	R/W	Description
[10]	STOP_DET_IF_MASTER_ACTIVE	R/W	<p>In Master mode:</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Master issues the <code>STOP_DET</code> interrupt only when Master is active</li> <li><b>0x0:</b> Master issues the <code>STOP_DET</code> interrupt irrespective of whether Master is active or not</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[9]	RX_FIFO_FULL_HLD_CTRL	R	<p>Controls whether I2C Controller should hold the bus when the Rx FIFO is physically full to its <code>RX_BUFFER_DEPTH=8</code>.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Hold bus when <code>RX_FIFO</code> is full</li> <li><b>0x0:</b> Overflow when <code>RX_FIFO</code> is full</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[8]	TX_EMPTY_CTRL	R/W	<p>Controls the generation of the <code>TX_EMPTY</code> interrupt, as described in the <code>IC_RAW_INTR_STAT</code> register.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Controlled generation of <code>TX_EMPTY</code> interrupt</li> <li><b>0x0:</b> Default behavior of <code>TX_EMPTY</code> interrupt</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[7]	STOP_DET_IFADDRESSED	R/W	<p>In Slave mode:</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> issues the <code>STOP_DET</code> interrupt only when it is addressed.</li> <li><b>0x0:</b> issues the <code>STOP_DET</code> irrespective of whether it's addressed or not.</li> </ul> <p> During a general call address, this Slave does not issue the <code>STOP_DET</code> interrupt if <code>STOP_DET_IFADDRESSED</code> = 1, even if the Slave responds to the general call address by generating ACK. The <code>STOP_DET</code> interrupt is generated only when the transmitted address matches the Slave address.</p> <p><b>Value After Reset:</b> 0x0</p>
[6]	IC_SLAVE_DISABLE	R/W	This bit controls whether I2C has its Slave disabled. That means once the <code>presetn</code> 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
			<p>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.</p> <p> Software should ensure that if this bit is written with 0, then <code>MASTER_MODE</code> should also be written with a 0.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Slave mode is disabled</li> <li><b>0x0:</b> Slave mode is enabled</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[5]	IC_RESTART_EN	R/W	<p>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:</p> <ul style="list-style-type: none"> <li>• Sending a START BYTE</li> <li>• Performing any high-speed mode operation</li> <li>• High-speed mode operation</li> <li>• Performing direction changes in combined format mode</li> <li>• Performing a read operation with a 10-bit address</li> </ul> <p>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 <code>IC_RAW_INTR_STAT.TX_ABRT</code>.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Master restart enabled</li> <li><b>0x0:</b> Master restart disabled</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[4]	IC_10BITADDR_MASTER_RO	R	<p>The function of this bit is handled by bit 12 of <code>IC_TAR</code> register, and becomes a read-only copy called <code>IC_10BITADDR_MASTER_RO</code>.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Master 10Bit addressing mode</li> <li><b>0x0:</b> Master 7Bit addressing mode</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[3]	IC_10BITADDR_SLAVE	R/W	<p>When acting as a Slave, this bit controls whether the I2C Controller responds to 7- or 10-bit addresses.</p> <p><b>Values:</b></p>

**Table 13-128 Fields for Register: IC\_CON (continued)**

Bits	Name	R/W	Description
			<p><b>0x1:</b> Slave 10Bit addressing. The I2C Controller responds to only 10-bit addressing transfers that match the full 10 bits of the <a href="#">IC_SAR</a> register.</p> <p><b>0x0:</b> Slave 7Bit addressing. The I2C Controller ignores transactions that involve 10-bit addressing; for 7-bit addressing, only the lower 7 bits of the <a href="#">IC_SAR</a> register are compared.</p> <p><b>Value After Reset:</b> 0x0</p>
[2:1]	SPEED	R/W	<p>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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Standard Speed mode of operation (100 kbit/s)</li> <li><b>0x2:</b> Fast (&lt;=400 kbit/s) or Fast Plus (&lt;=1000Kbit/s) mode of operation</li> <li><b>0x3:</b> High speed mode (3.4 Mbit/s)</li> </ul> <p><b>Value After Reset:</b> 0x3</p>
[0]	MASTER_MODE	R/W	<p>This bit controls whether the I2C Master is enabled.</p> <p> Software should ensure that if this bit is written with '1' then <a href="#">IC_SLAVE_DISABLE</a> should also be written with a '1'.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Master mode is enabled</li> <li><b>0x0:</b> Master mode is disabled</li> </ul> <p><b>Value After Reset:</b> 0x1</p>

### 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]	<a href="#">IC_10BITADDR_MASTER</a>	R/W	<p>This bit controls whether the I2C controller starts its transfers in 7- or 10-bit addressing mode when acting as a Master.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Address 10Bit transmission format</li> <li><b>0x0:</b> Address 7Bit transmission format</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[11]	<a href="#">SPECIAL</a>	R/W	<p>This bit indicates whether software performs a Device-ID or General Call or START BYTE command.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Enables programming of GENERAL_CALL or START_BYT transmission to perform special I2C command as specified in <a href="#">GC_OR_START</a> bit</li> <li><b>0x0:</b> Disables programming of GENERAL_CALL or START_BYT transmission to ignore bit 10 <a href="#">GC_OR_START</a> and use <a href="#">IC_TAR</a> normally</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[10]	<a href="#">GC_OR_START</a>	R/W	<p>If bit 11 (<a href="#">SPECIAL</a>) is set to 1, then this bit indicates whether a General Call or START byte command is to be performed by the I2C Controller.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> START byte transmission</li> <li><b>0x0:</b> GENERAL_CALL byte transmission - after issuing a General Call, only writes may be performed. Attempting</li> </ul>

**Table 13-129 Fields for Register: IC\_TAR (continued)**

Bits	Name	R/W	Description
			<p>to issue a read command results in setting the <a href="#">IC_RAW_INTR_STAT.TX_ABRT</a>. The I2C controller remains in General Call mode until the <a href="#">SPECIAL</a> bit value is cleared.</p> <p><b>Value After Reset:</b> 0x0</p>
[9:0]	IC_TAR	R/W	<p>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 <a href="#">IC_TAR</a> and <a href="#">IC_SAR</a> 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.</p> <p><b>Value After Reset:</b> 0x55</p>

**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	<p>The <a href="#">IC_SAR</a> holds the Slave address when the I2C is operating as a Slave. For 7-bit addressing, only <a href="#">IC_SAR[6:0]</a> is used.</p> <p>This register can be written only when the I2C interface is disabled, which corresponds to the <a href="#">IC_ENABLE[0]</a> register being set to 0. Writes at other times have no effect.</p> <p> 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 <a href="#">IC_SAR</a> or <a href="#">IC_TAR</a> to a reserved value. Refer to Table I2C Definition of Bits in First Byte in section <a href="#">Addressing Slave Protocol</a> for a complete list of these reserved values.</p> <p><b>Value After Reset:</b> 0x55</p>

**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\_MADDR**

Bits	Name	R/W	Description
[31:3]			Reserved

**Table 13-131 Fields for Register: IC\_HS\_MADDR (continued)**

Bits	Name	R/W	Description
[2:0]	IC_HS_MAR	R/W	<p>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.</p> <p>This register can be written only when the I2C controller is disabled, which corresponds to the <a href="#">IC_ENABLE[0]</a> register being set to 0. Writes at other times have no effect.</p> <p><b>Value After Reset:</b> 0x1</p>

### 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	<p>Indicates the first data byte received after the address phase for receive transfer in Master receiver or Slave receiver mode.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x1: Non sequential data byte received</li> <li>0x0: Sequential data byte received</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[10:9]			Reserved
[8]	CMD	W	<p>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.</p> <p>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 <a href="#">IC_DATA_CMD</a> is to be transmitted.</p> <p>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 <a href="#">TX_ABRT</a> interrupt (TX_ABRT bit of the <a href="#">IC_RAW_INTR_STAT</a> register), unless the <a href="#">IC_TAR.SPECIAL</a> has been</p>

**Table 13-132 Fields for Register: IC\_DATA\_CMD (continued)**

Bits	Name	R/W	Description
			<p>cleared. If a 1 is written to this bit after receiving a RD_REQ interrupt, then a TX_ABRT interrupt occurs.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x1: Master Read Command</li> <li>0x0: Master Write Command</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[7:0]	DAT	R/W	<p>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.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>

#### 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	<p>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.</p> <p>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.</p> <p>The minimum valid value is 6; hardware prevents values less than this being written, and if attempted results in value 6 being written.</p> <p> 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.</p> <p><b>Value After Reset:</b> 0x190</p>

#### 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	<p>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.</p> <p>This register can be written only when the I2C interface is disabled which corresponds to the <a href="#">IC_ENABLE[0]</a> register being set to 0. Writes at other times have no effect.</p> <p>The minimum valid value is 8; hardware prevents values less than this being written, and if attempted results in value 8 being written.</p> <p><b>Value After Reset:</b> 0x1D6</p>

**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	<p>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.</p> <p>This register can be written only when the I2C interface is disabled, which corresponds to the <a href="#">IC_ENABLE[0]</a> register being set to 0. Writes at other times have no effect.</p> <p>The minimum valid value is 6; hardware prevents values less than this being written, and if attempted results in value 6 being written.</p> <p><b>Value After Reset:</b> 0x3C</p>

**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
			<p>count for fast speed. It is used in high-speed mode to send the Master Code and START BYTE or General CALL.</p> <p>This register can be written only when the I2C interface is disabled, which corresponds to the <code>IC_ENABLE[0]</code> register being set to 0. Writes at other times have no effect.</p> <p>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.</p> <p><b>Value After Reset:</b> 0x82</p>

### 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]	<code>IC_HS_SCL_HCNT</code>	R/W	<p>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.</p> <p>This register can be written only when the I2C interface is disabled, which corresponds to the <code>IC_ENABLE[0]</code> register being set to 0. Writes at other times have no effect.</p> <p>The minimum valid value is 6; hardware prevents values less than this being written, and if attempted results in value 6 being written.</p> <p><b>Value After Reset:</b> 0x6</p>

### 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]	<code>IC_HS_SCL_LCNT</code>	R/W	<p>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.</p> <p>This register can be written only when the I2C interface is disabled, which corresponds to the <code>IC_ENABLE[0]</code> register being set to 0. Writes at other times have no effect.</p> <p>The minimum valid value is 8; hardware prevents values less than this being written, and if attempted results in value 8 being written.</p>

**Table 13-138 Fields for Register: IC\_HS\_SCL\_LCNT (continued)**

Bits	Name	R/W	Description
			<b>Value After Reset:</b> 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	<p>See <code>IC_RAW_INTR_STAT</code> for a detailed description of <code>R_SCL_STUCK_AT_LOW</code> bit.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> <code>R_SCL_STUCK_AT_LOW</code> interrupt is active</li> <li><b>0x0:</b> <code>R_SCL_STUCK_AT_LOW</code> interrupt is inactive</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[13:12]			Reserved
[11]	R_GEN_CALL	R	<p>See <code>IC_RAW_INTR_STAT</code> for a detailed description of <code>R_GEN_CALL</code> bit.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> <code>R_GEN_CALL</code> interrupt is active</li> <li><b>0x0:</b> <code>R_GEN_CALL</code> interrupt is inactive</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[10]	R_START_DET	R	<p>See <code>IC_RAW_INTR_STAT</code> for a detailed description of <code>R_START_DET</code> bit.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> <code>R_START_DET</code> interrupt is active</li> <li><b>0x0:</b> <code>R_START_DET</code> interrupt is inactive</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[9]	R_STOP_DET	R	<p>See <code>IC_RAW_INTR_STAT</code> for a detailed description of <code>R_STOP_DET</code> bit.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> <code>R_STOP_DET</code> interrupt is active</li> <li><b>0x0:</b> <code>R_STOP_DET</code> interrupt is inactive</li> </ul>

**Table 13-139 Fields for Register: IC\_INTR\_STAT (continued)**

Bits	Name	R/W	Description
			<b>Value After Reset:</b> 0x0 <b>Volatile:</b> true
[8]	R_ACTIVITY	R	See <a href="#">IC_RAW_INTR_STAT</a> for a detailed description of R_ACTIVITY bit. <b>Values:</b> <b>0x1:</b> R_ACTIVITY interrupt is active <b>0x0:</b> R_ACTIVITY interrupt is inactive <b>Value After Reset:</b> 0x0 <b>Volatile:</b> true
[7]	R_RX_DONE	R	See <a href="#">IC_RAW_INTR_STAT</a> for a detailed description of R_RX_DONE bit. <b>Values:</b> <b>0x1:</b> R_RX_DONE interrupt is active <b>0x0:</b> R_RX_DONE interrupt is inactive <b>Value After Reset:</b> 0x0 <b>Volatile:</b> true
[6]	R_TX_ABRT	R	See <a href="#">IC_RAW_INTR_STAT</a> for a detailed description of R_TX_ABRT bit. <b>Values:</b> <b>0x1:</b> R_TX_ABRT interrupt is active <b>0x0:</b> R_TX_ABRT interrupt is inactive <b>Value After Reset:</b> 0x0 <b>Volatile:</b> true
[5]	R_RD_REQ	R	See <a href="#">IC_RAW_INTR_STAT</a> for a detailed description of R_RD_REQ bit. <b>Values:</b> <b>0x1:</b> R_RD_REQ interrupt is active <b>0x0:</b> R_RD_REQ interrupt is inactive <b>Value After Reset:</b> 0x0 <b>Volatile:</b> true
[4]	R_TX_EMPTY	R	See <a href="#">IC_RAW_INTR_STAT</a> for a detailed description of R_TX_EMPTY bit. <b>Values:</b> <b>0x1:</b> R_TX_EMPTY interrupt is active <b>0x0:</b> R_TX_EMPTY interrupt is inactive <b>Value After Reset:</b> 0x0 <b>Volatile:</b> true
[3]	R_TX_OVER	R	See <a href="#">IC_RAW_INTR_STAT</a> for a detailed description of R_TX_OVER bit. <b>Values:</b> <b>0x1:</b> R_TX_OVER interrupt is active <b>0x0:</b> R_TX_OVER interrupt is inactive <b>Value After Reset:</b> 0x0

**Table 13-139 Fields for Register: IC\_INTR\_STAT (continued)**

Bits	Name	R/W	Description
			<b>Volatile:</b> true
[2]	R_RX_FULL	R	See <a href="#">IC_RAW_INTR_STAT</a> for a detailed description of R_RX_FULL bit. <b>Values:</b> 0x1: R_RX_FULL interrupt is active 0x0: R_RX_FULL interrupt is inactive <b>Value After Reset:</b> 0x0 <b>Volatile:</b> true
[1]	R_RX_OVER	R	See <a href="#">IC_RAW_INTR_STAT</a> for a detailed description of R_RX_OVER bit. <b>Values:</b> 0x1: R_RX_OVER interrupt is active 0x0: R_RX_OVER interrupt is inactive <b>Value After Reset:</b> 0x0 <b>Volatile:</b> true
[0]	R_RX_UNDER	R	See <a href="#">IC_RAW_INTR_STAT</a> for a detailed description of R_RX_UNDER bit. <b>Values:</b> 0x1: RX_UNDER interrupt is active 0x0: RX_UNDER interrupt is inactive <b>Value After Reset:</b> 0x0 <b>Volatile:</b> 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 <a href="#">IC_INTR_STAT</a> register. <b>Values:</b> 0x1: SCL_STUCK_AT_LOW interrupt is unmasked 0x0: SCL_STUCK_AT_LOW interrupt is masked <b>Value After Reset:</b> 0x1
[13:12]			Reserved
[11]	M_GEN_CALL	R/W	This bit masks the R_GEN_CALL interrupt in <a href="#">IC_INTR_STAT</a> register.

**Table 13-140 Fields for Register: IC\_INTR\_MASK (continued)**

Bits	Name	R/W	Description
			<p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> GEN_CALL interrupt is unmasked</li> <li><b>0x0:</b> GEN_CALL interrupt is masked</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[10]	M_START_DET	R/W	<p>This bit masks the R_START_DET interrupt in <a href="#">IC_INTR_STAT</a> register.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> START_DET interrupt is unmasked</li> <li><b>0x0:</b> START_DET interrupt is masked</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[9]	M_STOP_DET	R/W	<p>This bit masks the R_STOP_DET interrupt in <a href="#">IC_INTR_STAT</a> register.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> STOP_DET interrupt is unmasked</li> <li><b>0x0:</b> STOP_DET interrupt is masked</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[8]	M_ACTIVITY	R/W	<p>This bit masks the R_ACTIVITY interrupt in <a href="#">IC_INTR_STAT</a> register.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> ACTIVITY interrupt is unmasked</li> <li><b>0x0:</b> ACTIVITY interrupt is masked</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[7]	M_RX_DONE	R/W	<p>This bit masks the R_RX_DONE interrupt in <a href="#">IC_INTR_STAT</a> register.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> RX_DONE interrupt is unmasked</li> <li><b>0x0:</b> RX_DONE interrupt is masked</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[6]	M_TX_ABRT	R/W	<p>This bit masks the R_TX_ABRT interrupt in <a href="#">IC_INTR_STAT</a> register.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> TX_ABORT interrupt is unmasked</li> <li><b>0x0:</b> TX_ABORT interrupt is masked</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[5]	M_RD_REQ	R/W	<p>This bit masks the R_RD_REQ interrupt in <a href="#">IC_INTR_STAT</a> register.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> RD_REQ interrupt is unmasked</li> <li><b>0x0:</b> RD_REQ interrupt is masked</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[4]	M_TX_EMPTY	R/W	<p>This bit masks the R_TX_EMPTY interrupt in <a href="#">IC_INTR_STAT</a> register.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> TX_EMPTY interrupt is unmasked</li> <li><b>0x0:</b> TX_EMPTY interrupt is masked</li> </ul>

**Table 13-140 Fields for Register: IC\_INTR\_MASK (continued)**

Bits	Name	R/W	Description
			<b>Value After Reset:</b> 0x1
[3]	M_TX_OVER	R/W	<p>This bit masks the R_TX_OVER interrupt in <a href="#">IC_INTR_STAT</a> register.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> TX_OVER interrupt is unmasked</li> <li><b>0x0:</b> TX_OVER interrupt is masked</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[2]	M_RX_FULL	R/W	<p>This bit masks the R_RX_FULL interrupt in <a href="#">IC_INTR_STAT</a> register.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> RX_FULL interrupt is unmasked</li> <li><b>0x0:</b> RX_FULL interrupt is masked</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[1]	M_RX_OVER	R/W	<p>This bit masks the R_RX_OVER interrupt in <a href="#">IC_INTR_STAT</a> register.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> RX_OVER interrupt is unmasked</li> <li><b>0x0:</b> RX_OVER interrupt is masked</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[0]	M_RX_UNDER	R/W	<p>This bit masks the R_RX_UNDER interrupt in <a href="#">IC_INTR_STAT</a> register.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> RX_UNDER interrupt is unmasked</li> <li><b>0x0:</b> RX_UNDER interrupt is masked</li> </ul> <p><b>Value After Reset:</b> 0x1</p>

#### 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	<p>Indicates whether the SCL Line is stuck at low for the <a href="#">IC_SCL_STUCK_LOW_TIMEOUT</a> number of <a href="#">i2c*_clk</a> periods.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> SCL_STUCK_AT_LOW interrupt is active</li> <li><b>0x0:</b> SCL_STUCK_AT_LOW interrupt is inactive.</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>

**Table 13-141 Fields for Register: IC\_RAW\_INTR\_STAT (continued)**

Bits	Name	R/W	Description
[13:12]			Reserved
[11]	GEN_CALL	R	<p>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 <a href="#">IC_CLR_GEN_CALL</a> register. The I2C stores the received data in the Rx buffer.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> GEN_CALL interrupt is active</li> <li><b>0x0:</b> GEN_CALL interrupt is inactive</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[10]	START_DET	R	<p>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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> START_DET interrupt is active</li> <li><b>0x0:</b> START_DET interrupt is inactive</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[9]	STOP_DET	R	<p>Indicates whether a STOP condition has occurred on the I2C interface regardless of whether the I2C is operating in Slave or Master mode.</p> <p><b>In Slave Mode:</b></p> <p>If <a href="#">IC_CON[7]=1</a>, the STOP_DET interrupt will be issued only if Slave is addressed.</p> <p> During a general call address, this Slave does not issue a STOP_DET interrupt if <a href="#">IC_CON.STOP_DET_IFADDRESSED=1</a>, 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).</p> <p>If <a href="#">IC_CON[7]=0</a>, the STOP_DET interrupt is issued irrespective of whether it is being addressed.</p> <p><b>In Master Mode:</b></p> <p>If <a href="#">IC_CON[10]=1</a>, the STOP_DET interrupt will be issued only if Master is active.</p> <p>If <a href="#">IC_CON[10]=0</a>, the STOP_DET interrupt will be issued irrespective of whether Master is active or not.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> STOP_DET interrupt is active</li> <li><b>0x0:</b> STOP_DET interrupt is inactive</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>

**Table 13-141 Fields for Register: IC\_RAW\_INTR\_STAT (continued)**

Bits	Name	R/W	Description
[8]	ACTIVITY	R	<p>This bit captures the I2C activity and stays set until it is cleared. There are four ways to clear it:</p> <ul style="list-style-type: none"> <li>• Disabling the I2C</li> <li>• Reading the <a href="#">IC_CLR_ACTIVITY</a> register</li> <li>• Reading the <a href="#">IC_CLR_INTR</a> register</li> <li>• System reset</li> </ul> <p>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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> <a href="#">RAW_INTR_ACTIVITY</a> interrupt is active</li> <li><b>0x0:</b> <a href="#">RAW_INTR_ACTIVITY</a> interrupt is inactive</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[7]	RX_DONE	R	<p>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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> <a href="#">RX_DONE</a> interrupt is active</li> <li><b>0x0:</b> <a href="#">RX_DONE</a> interrupt is inactive</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[6]	TX_ABRT	R	<p>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 <a href="#">IC_TX_ABRT_SOURCE</a> register indicates the reason why the transmit abort takes places.</p> <p> The I2C flushes/resets/empties only the Tx FIFO whenever there is a transmit abort caused by any of the events tracked by the <a href="#">IC_TX_ABRT_SOURCE</a> register. The Tx FIFO remains in this flushed state until the register <a href="#">IC_CLR_TX_ABRT</a> 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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> <a href="#">TX_ABRT</a> interrupt is active</li> <li><b>0x0:</b> <a href="#">TX_ABRT</a> interrupt is inactive</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>

**Table 13-141 Fields for Register: IC\_RAW\_INTR\_STAT (continued)**

Bits	Name	R/W	Description
[5]	RD_REQ	R	<p>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 <a href="#">IC_DATA_CMD</a> register. This bit is set to 0 just after the processor reads the <a href="#">IC_CLR_RD_REQ</a> register.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> RD_REQ interrupt is active</li> <li><b>0x0:</b> RD_REQ interrupt is inactive</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[4]	TX_EMPTY	R	<p>The behavior of the TX_EMPTY interrupt status differs based on the TX_EMPTY_CTRL setting in the <a href="#">IC_CON</a> register.</p> <p>If TX_EMPTY_CTRL = 0 this bit is set to 1 when the transmit buffer is at the threshold value set in the <a href="#">IC_TX_TL</a> register or below it.</p> <p>If TX_EMPTY_CTRL = 1 this bit is set to 1 when the transmit buffer is at the threshold value set in the <a href="#">IC_TX_TL</a> register or below it and the transmission of the address/data from the internal shift register for the most recently popped command is completed.</p> <p>It is automatically cleared by hardware when the buffer level goes above the threshold. When <a href="#">IC_ENABLE[0]</a> 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 <a href="#">IC_EN=0</a> in <a href="#">IC_ENABLE_STATUS</a> register, this bit is set to 0.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> TX_EMPTY interrupt is active</li> <li><b>0x0:</b> TX_EMPTY interrupt is inactive</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[3]	TX_OVER	R	<p>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 <a href="#">IC_DATA_CMD</a> register. When the module is disabled, this bit keeps its level until the Master or Slave state machines go into idle, and when <a href="#">IC_EN</a> in <a href="#">IC_ENABLE_STATUS</a> register goes to 0, this interrupt is cleared.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> TX_OVER interrupt is active</li> <li><b>0x0:</b> TX_OVER interrupt is inactive</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

**Table 13-141 Fields for Register: IC\_RAW\_INTR\_STAT (continued)**

Bits	Name	R/W	Description
			<b>Volatile:</b> true
[2]	RX_FULL	R	<p>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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x1: RX_FULL interrupt is active</li> <li>0x0: RX_FULL interrupt is inactive</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[1]	RX_OVER	R	<p>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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x1: RX_OVER interrupt is active</li> <li>0x0: RX_OVER interrupt is inactive</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[0]	RX_UNDER	R	<p>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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x1: RX_UNDER interrupt is active</li> <li>0x0: RX_UNDER interrupt is inactive</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>

### 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. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 0x0 <b>Volatile:</b> 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). <b>Value After Reset:</b> 0x0 <b>Volatile:</b> 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). <b>Value After Reset:</b> 0x0 <b>Volatile:</b> 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). <b>Value After Reset:</b> 0x0 <b>Volatile:</b> 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). <b>Value After Reset:</b> 0x0 <b>Volatile:</b> 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. <b>Value After Reset:</b> 0x0 <b>Volatile:</b> 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). <b>Value After Reset:</b> 0x0 <b>Volatile:</b> 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	<p>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 <a href="#">IC_RAW_INTR_STAT</a> register).</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>

**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	<p>Read this register to clear the STOP_DET interrupt (STOP_DET bit of the <a href="#">IC_RAW_INTR_STAT</a> register).</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>

**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	<p>Read this register to clear the START_DET interrupt (START_DET bit of the <a href="#">IC_RAW_INTR_STAT</a> register).</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>

**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). <b>Value After Reset:</b> 0x0 <b>Volatile:</b> 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 <b>Values:</b> 0x1: Master initiates the SDA stuck at low recovery mechanism. 0x0: Master disabled the SDA stuck at low recovery mechanism. <b>Value After Reset:</b> 0x0
[2]	TX_CMD_BLOCK	R/W	In Master mode: <b>Values:</b> 0x1: Blocks the transmission of data on the I2C bus even if Tx FIFO has data to transmit. 0x0: The transmission of data starts on the I2C bus automatically, as soon as the first data is available in the Tx FIFO.  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. <b>Value After Reset:</b> 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
			<p>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.</p> <p>For a detailed description on how to abort the I2C transfers, refer to Aborting I2C Transfers in the <a href="#">Operation Modes</a></p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x1: ABORT operation in progress</li> <li>0x0: ABORT operation not in progress or done</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[0]	ENABLE	R/W	<p>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 the <a href="#">Operation Modes</a>.</p> <p>When I2C Controller is disabled, the following occurs:</p> <ul style="list-style-type: none"> <li>• The TX FIFO and RX FIFO get flushed.</li> <li>• Status bits in the <a href="#">IC_INTR_STAT</a> register are still active until I2C goes into IDLE state.</li> </ul> <p>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.</p> <p>pclk and i2c*_clk signals are asynchronous. There is a two i2c*_clk delay when enabling or disabling the I2C Controller. For a detailed description on how to disable I2C Controller, refer to Disabling I2C Controller in the <a href="#">Operation Modes</a>.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>0x1: Enables the I2C Controller</li> <li>0x0: Disables the I2C Controller (TX and RX FIFOs are held in an erased state)</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

### 13.5.2.29. 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
- Bits 3 and 10 are 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	<p>This bit indicates that SDA stuck at low is not recovered after the recovery mechanism. In Slave mode, this register bit is not applicable.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> SDA Stuck at low is recovered after recovery mechanism.</li> <li><b>0x0:</b> SDA Stuck at low is not recovered after recovery mechanism.</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[10]			Reserved
[9]	SLV_HOLD_TX_FIFO_EMPTY	R	<p>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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Slave holds the bus due to Tx FIFO is empty</li> <li><b>0x0:</b> Slave is not holding the bus or Bus hold is not due to Tx FIFO is empty</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[8:7]			Reserved
[6]	SLV_ACTIVITY	R	<p>Slave FSM Activity Status</p> <p>When the Slave <i>Finite State Machine (FSM)</i> is not in the IDLE state, this bit is set.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Slave FSM is not in IDLE state so the Slave part of the I2C Controller is Active</li> <li><b>0x0:</b> Slave FSM is in IDLE state so the Slave part of the I2C Controller is not Active</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[5]	MST_ACTIVITY	R	Master FSM Activity Status

**Table 13-156 Fields for Register: IC\_STATUS (continued)**

Bits	Name	R/W	Description
			<p>When the Master <i>Finite State Machine (FSM)</i> is not in the IDLE state, this bit is set.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Master FSM is not in IDLE state so the Master part of I2C Controller is Active</li> <li><b>0x0:</b> Master FSM is in IDLE state so the Master part of the I2C Controller is not Active</li> </ul> <p> The ACTIVITY bit 0 in <a href="#">IC_STATUS</a> register is the OR of <a href="#">SLV_ACTIVITY</a> and <a href="#">MST_ACTIVITY</a> bits.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[4]	RFF	R	<p>Receive FIFO Completely Full</p> <p>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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Rx FIFO is full</li> <li><b>0x0:</b> Rx FIFO is not full</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[3]	RFNE	R	<p>Receive FIFO Not Empty</p> <p>This bit is set when the receive FIFO contains one or more entries; it is cleared when the receive FIFO is empty.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Rx FIFO is not empty</li> <li><b>0x0:</b> Rx FIFO is empty</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[2]	TFE	R	<p>Transmit FIFO Completely Empty</p> <p>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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Tx FIFO is empty</li> <li><b>0x0:</b> Tx FIFO is not empty</li> </ul> <p><b>Value After Reset:</b> 0x1</p> <p><b>Volatile:</b> true</p>
[1]	TFNF	R	<p>Transmit FIFO Not Full</p> <p>Set when the transmit FIFO contains one or more empty locations, and is cleared when the FIFO is full.</p> <p><b>Values:</b></p>

**Table 13-156 Fields for Register: IC\_STATUS (continued)**

Bits	Name	R/W	Description
			<p><b>0x1:</b> Tx FIFO not full  <b>0x0:</b> Tx FIFO is full</p> <p><b>Value After Reset:</b> 0x1  <b>Volatile:</b> true</p>
[0]	ACTIVITY	R	<p>I2C Activity Status  <b>Values:</b></p> <p><b>0x1:</b> I2C is active  <b>0x0:</b> I2C is idle</p> <p><b>Value After Reset:</b> 0x0  <b>Volatile:</b> true</p>

### 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	<p>Transmit FIFO Level. Contains the number of valid data entries in the transmit FIFO.</p> <p><b>Value After Reset:</b> 0x0  <b>Volatile:</b> true</p>

### 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. <b>Value After Reset:</b> 0x0 <b>Volatile:</b> 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 <code>i2c*_clk</code> period, when the I2C Controller acts as a receiver. <b>Value After Reset:</b> 0x0
[15:0]	IC_SDA_TX_HOLD	R/W	Sets the required SDA hold time in units of <code>i2c*_clk</code> period, when the I2C Controller acts as a transmitter. <b>Value After Reset:</b> 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	<p>This field indicates the number of Tx FIFO Data Commands which are flushed due to <code>TX_ABRT</code> interrupt. It is cleared whenever I2C is disabled.</p> <p><b>Role of the I2C Controller:</b> Master-Transmitter or Slave-Transmitter</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[22:18]			Reserved
[17]	ABRT_SDA_STUCK_AT_LOW	R	<p>This is a Master-mode-only bit</p> <p>Master detects the SDA Stuck at low for the <code>IC_SDA_STUCK_AT_LOW_TIMEOUT</code> value of <code>i2c*_clk</code>.</p> <p><b>Role of the I2C Controller:</b> Master</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> This abort is generated because of SDA stuck at low for <code>IC_SDA_STUCK_AT_LOW_TIMEOUT</code> value of <code>i2c*_clk</code></li> <li><b>0x0:</b> This abort is not generated</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[16]	ABRT_USER_ABRT	R	<p>This is a Master-mode-only bit</p> <p>Master has detected the transfer abort (<code>IC_ENABLE[1]</code>)</p> <p><b>Role of the I2C Controller:</b> Master-Transmitter</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Transfer abort detected by Master</li> <li><b>0x0:</b> Transfer abort detected by Master- scenario not present</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[15]	ABRT_SLVRD_INTX	R	<p>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 <code>CMD</code> of <code>IC_DATA_CMD</code> register.</p> <p><b>Role of the I2C Controller:</b> Slave-Transmitter</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Slave trying to transmit to remote Master in read mode</li> <li><b>0x0:</b> Slave trying to transmit to remote Master in read mode- scenario not present</li> </ul>

**Table 13-160 Fields for Register: IC\_TX\_ABRT\_SOURCE (continued)**

Bits	Name	R/W	Description
			<b>Value After Reset:</b> 0x0 <b>Volatile:</b> true
[14]	ABRT_SLV_ARBLOST	R	<p>This field indicates that a Slave has lost the bus while transmitting data to a remote Master. <a href="#">IC_TX_ABRT_SOURCE[12]</a> is set at the same time.</p> <p> 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.</p> <p><b>Role of the I2C Controller:</b> Slave-Transmitter  <b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Slave lost arbitration to remote Master</li> <li><b>0x0:</b> Slave lost arbitration to remote Master- scenario not present</li> </ul> <p><b>Value After Reset:</b> 0x0  <b>Volatile:</b> true</p>
[13]	ABRT_SLVFLUSH_TXFIFO	R	<p>This field specifies that the Slave has received a read command and some data exists in the TX FIFO, so the Slave issues a <a href="#">TX_ABRT</a> interrupt to flush old data in TX FIFO.</p> <p><b>Role of the I2C Controller:</b> Slave-Transmitter  <b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Slave flushes existing data in TX-FIFO upon getting read command</li> <li><b>0x0:</b> Slave flushes existing data in TX-FIFO upon getting read command- scenario not present</li> </ul> <p><b>Value After Reset:</b> 0x0  <b>Volatile:</b> true</p>
[12]	ARB_LOST	R	<p>This field specifies that the Master has lost arbitration, or if <a href="#">IC_TX_ABRT_SOURCE[14]</a> is also set, then the Slave transmitter has lost arbitration.</p> <p><b>Role of the I2C Controller:</b> Master-Transmitter or Slave-Transmitter  <b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Master or Slave-Transmitter lost arbitration</li> <li><b>0x0:</b> Master or Slave-Transmitter lost arbitration- scenario not present</li> </ul> <p><b>Value After Reset:</b> 0x0  <b>Volatile:</b> true</p>
[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
			<p><b>Role of the I2C Controller:</b> Master-Transmitter or Master-Receiver</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> User initiating Master operation when MASTER disabled</li> <li><b>0x0:</b> User initiating Master operation when MASTER disabled- scenario not present</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[10]	ABRT_10B_RD_NORSTRT	R	<p>This field indicates that the restart is disabled (<a href="#">IC_CON[5]=0</a>) and the Master sends a read command in 10-bit addressing mode.</p> <p><b>Role of the I2C Controller:</b> Master-Receiver</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Master trying to read in 10Bit addressing mode when RESTART disabled</li> <li><b>0x0:</b> Master not trying to read in 10Bit addressing mode when RESTART disabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[9]	ABRT_SBYTE_NORSTRT	R	<p>To clear this bit, the source of the <a href="#">ABRT_SBYTE_NORSTRT</a> must be fixed first; restart must be enabled (<a href="#">IC_CON[5]=1</a>), the <a href="#">SPECIAL</a> bit must be cleared (<a href="#">IC_TAR[11]</a>), or the <a href="#">GC_OR_START</a> bit must be cleared (<a href="#">IC_TAR[10]</a>). Once the source of the <a href="#">ABRT_SBYTE_NORSTRT</a> is fixed, then this bit can be cleared in the same manner as other bits in this register. If the source of the <a href="#">ABRT_SBYTE_NORSTRT</a> is not fixed before attempting to clear this bit, <a href="#">ABRT_SBYTE_NORSTRT</a> clears for one cycle and then gets reasserted. When this field is set to 1, the restart is disabled (<a href="#">IC_CON[5] =0</a>) and the user is trying to send a START Byte.</p> <p><b>Role of the I2C Controller:</b> Master</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> User trying to send START byte when RESTART disabled</li> <li><b>0x0:</b> User trying to send START byte when RESTART disabled- scenario not present</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[8]	ABRT_HS_NORSTRT	R	<p>This field indicates that the restart is disabled (<a href="#">IC_CON[5]=0</a>) and the user is trying to use the Master to transfer data in High Speed mode.</p>

**Table 13-160 Fields for Register: IC\_TX\_ABRT\_SOURCE (continued)**

Bits	Name	R/W	Description
			<p><b>Role of the I2C Controller:</b> Master-Transmitter or Master-Receiver</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> User trying to switch Master to HS mode when RESTART disabled</li> <li><b>0x0:</b> User trying to switch Master to HS mode when RESTART disabled- scenario not present</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[7]	ABRT_SBYTE_ACKDET	R	<p>This field indicates that the Master has sent a START Byte and the START Byte was acknowledged (wrong behavior).</p> <p><b>Role of the I2C Controller:</b> Master</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> ACK detected for START byte</li> <li><b>0x0:</b> ACK detected for START byte- scenario not present</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[6]	ABRT_HS_ACKDET	R	<p>This field indicates that the Master is in High Speed mode and the High Speed Master code was acknowledged (wrong behavior).</p> <p><b>Role of the I2C Controller:</b> Master</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> HS Master code ACKed in HS Mode</li> <li><b>0x0:</b> HS Master code ACKed in HS Mode- scenario not present</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[5]	ABRT_GCALL_READ	R	<p>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 (<a href="#">IC_DATA_CMD[9]</a> is set to 1).</p> <p><b>Role of the I2C Controller:</b> Master-Transmitter</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> <code>GCALL</code> is followed by read from bus</li> <li><b>0x0:</b> <code>GCALL</code> is followed by read from bus-scenario not present</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[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
			<p><b>Role of the I2C Controller:</b> Master-Transmitter</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> GCALL is not ACKed by any Slave</li> <li><b>0x0:</b> GCALL is not ACKed by any Slave-scenario not present</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[3]	ABRT_TXDATA_NOACK	R	<p>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).</p> <p><b>Role of the I2C Controller:</b> Master-Transmitter</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Transmitted data not ACKed by addressed Slave</li> <li><b>0x0:</b> Transmitted data non-ACKed by addressed Slave-scenario not present</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[2]	ABRT_10ADDR2_NOACK	R	<p>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.</p> <p><b>Role of the I2C Controller:</b> Master-Transmitter or Master-Receiver</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Byte 2 of 10Bit Address not ACKed by any Slave</li> <li><b>0x0:</b> This abort is not generated</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[1]	ABRT_10ADDR1_NOACK	R	<p>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.</p> <p><b>Role of the I2C Controller:</b> Master-Transmitter or Master-Receiver</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Byte 1 of 10Bit Address not ACKed by any Slave</li> <li><b>0x0:</b> This abort is not generated</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[0]	ABRT_7B_ADDR_NOACK	R	<p>This field indicates that the Master is in 7-bit addressing mode and the address sent was not acknowledged by any Slave.</p> <p><b>Role of the I2C Controller:</b> Master-Transmitter or Master-Receiver</p>

**Table 13-160 Fields for Register: IC\_TX\_ABRT\_SOURCE (continued)**

Bits	Name	R/W	Description
			<p><b>Values:</b></p> <p><b>0x1:</b> This abort is generated because of NOACK for 7-bit address</p> <p><b>0x0:</b> This abort is not generated</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>

### 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	<p>Transmit DMA Enable This bit enables/disables the transmit FIFO DMA channel.</p> <p><b>Values:</b></p> <p><b>0x1:</b> Transmit FIFO DMA channel enabled</p> <p><b>0x0:</b> transmit FIFO DMA channel disabled</p> <p><b>Value After Reset:</b> 0x0</p>
[0]	RDMAE	R/W	<p>Receive DMA Enable This bit enables/disables the receive FIFO DMA channel.</p> <p><b>Values:</b></p> <p><b>0x1:</b> Receive FIFO DMA channel enabled</p> <p><b>0x0:</b> Receive FIFO DMA channel disabled</p> <p><b>Value After Reset:</b> 0x0</p>

### 13.5.2.2.35. DMA Transmit Data Level Register (IC\_DMA\_TDRL)

The `IC_DMA_TDRL` 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	<p>Transmit Data Level</p> <p>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 <a href="#">IC_DMA_CR.TDMAE = 1</a>.</p> <p><b>Value After Reset:</b> 0x0</p>

**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	<p>Receive Data Level</p> <p>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 <a href="#">IC_DMA_CR.RDMAE=1</a>. For instance, when DMARDL is 0, then Receive request is made when 1 or more data entries are present in the receive FIFO.</p> <p><b>Value After Reset:</b> 0x0</p>

**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

**Table 13-164 Fields for Register: IC\_SDA\_SETUP (continued)**

Bits	Name	R/W	Description
[7:0]	SDA_SETUP	R/W	<p>It is recommended that if the required delay is 1000ns, then for an <code>i2c*_clk</code> frequency of 10 MHz, <code>IC_SDA_SETUP</code> should be programmed to a value of 11. <code>IC_SDA_SETUP</code> must be programmed with a minimum value of 2.</p> <p><b>Value After Reset:</b> 0x64</p>



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	<p>ACK General Call</p> <p>When set to 1, I2C Controller responds with a ACK when it receives a General Call. Otherwise, I2C Controller responds with a NACK.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Generate ACK for a General Call</li> <li><b>0x0:</b> Generate NACK for General Call</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

### 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
[2]	SLV_RX_DATA_LOST	R	Slave Received Data Lost

**Table 13-166 Fields for Register: IC\_ENABLE\_STATUS (continued)**

Bits	Name	R/W	Description
			<p>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 <code>IC_ENABLE[0]</code> 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.</p> <p> If the remote I2C Master terminates the transfer with a STOP condition before the I2C Controller has a chance to NACK a transfer, and <code>IC_ENABLE[0]</code> has been set to 0, then this bit is also set to 1.</p> <p>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.</p> <p> The CPU can safely read this bit when <code>IC_EN</code> is read as 0.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Slave RX Data is lost</li> <li><b>0x0:</b> Slave RX Data is not lost</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[1]	SLV_DISABLED_WHILE_BUSY	R	<p>Slave Disabled While Busy (Transmit, Receive)</p> <p>This bit indicates if a potential or active Slave operation has been aborted due to the setting bit 0 of the <code>IC_ENABLE</code> register from 1 to 0. This bit is set when the CPU writes a 0 to the <code>IC_ENABLE</code> register while:</p> <ul style="list-style-type: none"> <li>• I2C Controller is receiving the address byte of the Slave-Transmitter operation from a remote Master;</li> <li>• OR, I2C Controller is receiving address and data bytes of the Slave-Receiver operation from a remote Master.</li> </ul> <p>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 (<code>IC_SAR</code> register) OR if the transfer is completed before <code>IC_ENABLE</code> is set to 0 but has not taken effect.</p> <p> If the remote I2C Master terminates the transfer with a STOP condition before the I2C Controller has a chance to NACK a transfer, and <code>IC_ENABLE[0]</code> has been set to 0, then this bit will also be set to 1.</p>

**Table 13-166 Fields for Register: IC\_ENABLE\_STATUS (continued)**

Bits	Name	R/W	Description
			<p>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.</p> <p> The CPU can safely read this bit when IC_EN is read as 0.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Slave is disabled when it is active</li> <li><b>0x0:</b> Slave is disabled when it is idle</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>
[0]	IC_EN	R	<p>I2C interface enable. This bit always reflects the value driven on the output port.</p> <p>When read as 1, the I2C Controller is deemed to be in an enabled state.</p> <p>When read as 0, the I2C Controller is deemed completely inactive.</p> <p> 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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> I2C enabled</li> <li><b>0x0:</b> I2C disabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>



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	<p>This register must be set before any I2C bus transaction can take place to ensure stable operation. This register sets the duration, measured in <code>i2c*_clk</code> 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 <code>IC_ENABLE[0]</code> 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.</p> <p><b>Value After Reset:</b> 0x5</p>

#### 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	<p>This register must be set before any I2C bus transaction can take place to ensure stable operation. This register sets the duration, measured in <code>i2c*_clk</code> cycles, of the longest spike in the SCL or SDA lines that will be filtered out by the spike suppression logic.</p> <p>This register can be written only when the I2C interface is disabled which corresponds to the <code>IC_ENABLE[0]</code> register being set to 0. Writes at other times have no effect.</p> <p>The minimum valid value is 1; hardware prevents values less than this being written, and if attempted results in 1 being set.</p> <p><b>Value After Reset:</b> 0x1</p>

#### 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 ( <code>IC_RAW_INTR_STAT.SCL_STUCK_AT_LOW</code> )

**Table 13-169 Fields for Register: IC\_SCL\_STUCK\_AT\_LOW\_TIMEOUT (continued)**

Bits	Name	R/W	Description
			<p>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.</p> <p><b>Value After Reset:</b> 0xFFFFFFFF</p>

#### 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	<p>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.</p> <p><b>Value After Reset:</b> 0xFFFFFFFF</p>

#### 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	<p>Read this register to clear the SCL_STUCK_AT_LOW interrupt (SCL_STUCK_AT_LOW bit of the IC_RAW_INTR_STAT register).</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Volatile:</b> true</p>

#### 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**

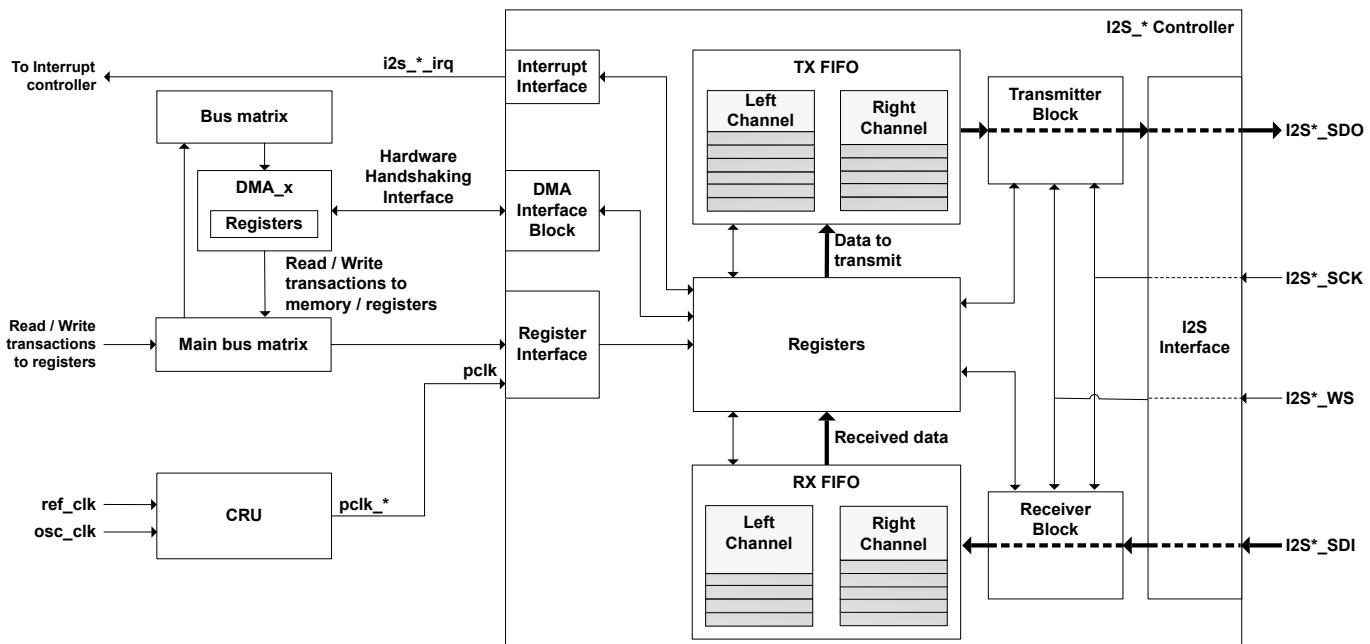
Bits	Name	R/W	Description
[31:8]			Reserved
[7:0]	REG_TIMEOUT_RST	R/W	This field holds reset value of REG_TIMEOUT_RST counter register. <b>Value After Reset:</b> 0x8 <b>Volatile:</b> true

## 13.6. I2S

This chapter describes the *Inter-IC Sound (I2S)* controller of the BE-U1000 microcontroller.

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 the [DMA Controller](#).

**Figure 13-13 I2S Block Diagram**

I2S controller has the following 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 of 12, 16, 20, 24, and 32 bits
- FIFO depth of 8 words, with the word size of 32 bit

### 13.6.1. I2S Functional Description

This chapter describes the functional operation of the I2S Controller.

In this chapter:

- I<sub>S</sub>S Controller Enable
- I<sub>S</sub>S Controller as Transmitter
- I<sub>S</sub>S Controller as Receiver

### 13.6.1.1. I<sub>S</sub>S Controller Enable

You must enable the I<sub>S</sub>S Controller before any data is received or transmitted into the FIFOs. To enable the component, set the I<sub>S</sub>S Enable (`IEN`) bit of the I<sub>S</sub>S Enable Register (`IER`) to 1. When you disable the device it acts as a global disable. To disable the I<sub>S</sub>S 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. I<sub>S</sub>S Controller as Transmitter

The I<sub>S</sub>S Controller component supports 1 stereo I<sub>S</sub>S 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.

The diagram below illustrates the basic usage flow for the I<sub>S</sub>S Controller when it acts as a transmitter.

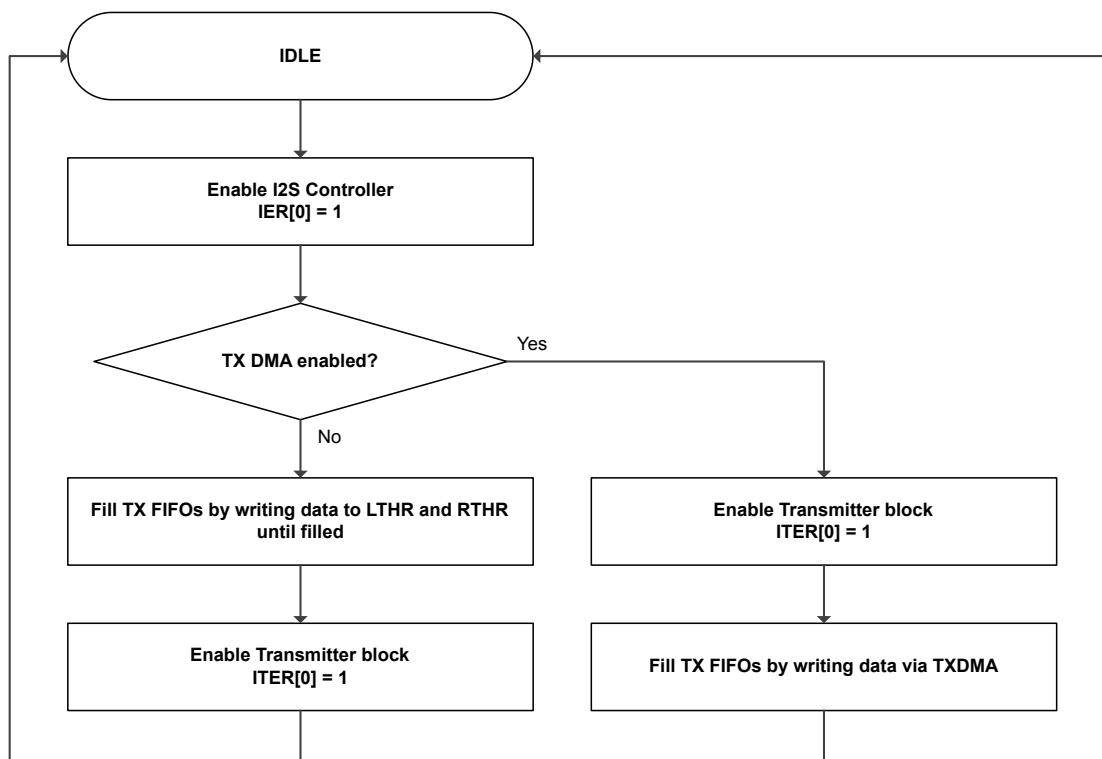


Figure 13-14 Basic Usage Flow - I<sub>S</sub>S Controller as Transmitter

### 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 Registers (`TCCR0[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 (`ISRO`). The `ISRO.TXFE` bit indicates the status of the 'TX FIFO Empty' interrupt, while the `ISRO.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 `ISRO` 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.

The following diagram illustrates the basic usage flow for I2S Controller when it acts as a receiver.

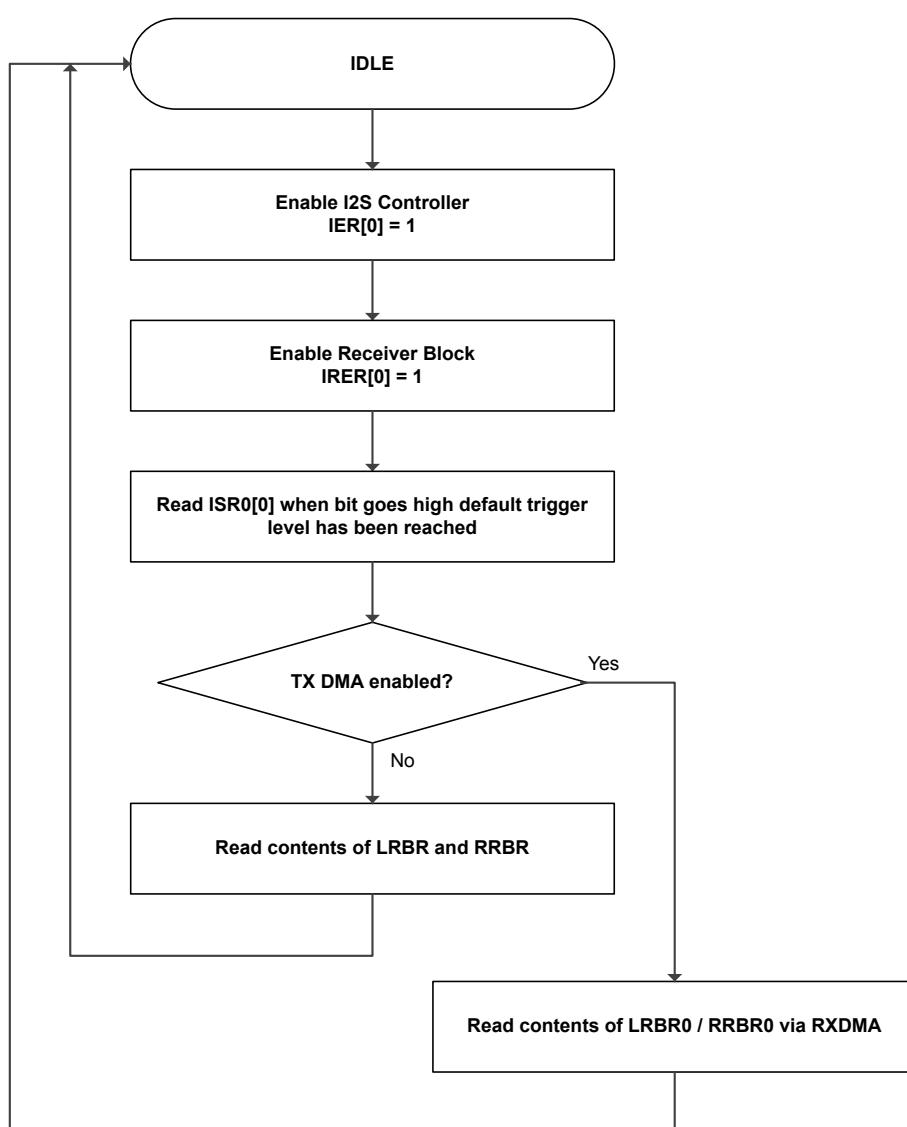


Figure 13-15 Basic Usage Flow - I2S Controller as Receiver

#### 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 ( $\text{RXFFR}$ ) bit of the Receiver FIFO Flush Register ( $\text{RXFFR}[0] = 1$ )
- Flush the channel's RX FIFO by programming the Receive Channel FIFO Reset ( $\text{RXCHFR}$ ) bit of the Receive FIFO Flush Register ( $\text{RFFO}[0] = 1$ )

On reset,  $\text{IER}$  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 ( $\text{RERO}[0]$ ). To enable the RX Channel, write a 1 to  $\text{RERO}[0]$ . To disable this channel, write a 0 to  $\text{RERO}[0]$ .

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 ( $\text{RFFO}$ ). When the channel is enabled it resumes receiving on the next left stereo data cycle.

On reset, the  $\text{RFFO}[0]$  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 ( $\text{WLEN}$ ) bits of the Receive Configuration registers ( $\text{RCR0}[3:0]$ ). 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 ways, as described as follows:

- on reset
- by disabling I2S Controller ( $\text{IER}[0] = 0$ )

- 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 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) 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 `RXCHO` bit of the Receive Overrun Register (`ROR0`).

The interrupt status of the RX channel can be determined by polling the Interrupt Status Register (`ISRO`). The `ISRO.RXDA` bit indicates the status of the RX FIFO Data Available interrupt; the `ISRO.RXFO` bit indicates the status of the ‘RX FIFO Data Overrun’ interrupt. Both the Receive Empty Threshold and Data Overrun interrupts can be masked by writing a 1 in the Receive Empty Threshold Mask (`RDM`) and Receive Overrun Mask (`ROM`) bits of the Interrupt Mask Register (`IMR0`), respectively. This prevents the interrupts from driving their output lines; however, the `ISRO` 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 (`LRBRO`) and the Right Receive Buffer Register (`RRBRO`). All stereo data pairs must be read using the following two-stage process:

1. Read the left stereo data from `LRBRO`
2. Read the right stereo data from `RRBRO`



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 `LRBRO` and `RRBRO`.

## 13.6.2. I2S Registers

The BE-U1000 microcontroller contains 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	4KB	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	4KB	0x1300_6000	0x1300_6FFF



For details, refer to [Memory Map](#) chapter.

In this chapter:

- [Registers Map](#)
- [Register Descriptions](#)

### 13.6.2.1. Registers Map

The following table provides 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 Registers Map**

Name	Offset	Description	Default Value
IER	0x0	<a href="#">I2S Controller Enable Register</a>	0x0
IRER	0x4	<a href="#">I2S Receiver Block Enable Register</a>	0x0
ITER	0x8	<a href="#">I2S Transmitter Block Enable Register</a>	0x0
RXFFR	0x14	<a href="#">Receiver Block FIFO Reset Register</a>	0x0
TXFFR	0x18	<a href="#">Transmitter Block FIFO Reset Register</a>	0x0
LRBR0	0x20	<a href="#">Left Receive Buffer Register</a>	0x0
LTHR0	0x20	<a href="#">Left Transmit Holding Register</a>	0x0
RRBR0	0x24	<a href="#">Right Receive Buffer Register</a>	0x0
RTHR0	0x24	<a href="#">Right Transmit Holding Register</a>	0x0
RERO	0x28	<a href="#">Receive Enable Register</a>	0x1
TER0	0x2C	<a href="#">Transmit Enable Register</a>	0x1
RCR0	0x30	<a href="#">Receive Configuration Register</a>	0x5
TCR0	0x34	<a href="#">Transmit Configuration Register</a>	0x2
ISR0	0x38	<a href="#">Interrupt Status Register</a>	0x10
IMR0	0x3C	<a href="#">Interrupt Mask Register</a>	0x33
ROR0	0x40	<a href="#">Receive Overrun Register</a>	0x0

**Table 13-174 I2S Registers 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 of a register description 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]	<b>IEN</b>	R/W	<p>I2S Controller enable A disable on this bit overrides any other block or channel enables and flushes all FIFOs.</p> <p><b>Values:</b></p> <p><b>0x1:</b> Enable I2S Controller <b>0x0:</b> Disable I2S Controller</p> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p>Receiver block enable A disable on this bit overrides receive channel enable.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Enable receiver</li> <li><b>0x0:</b> Disable receiver</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p>Transmitter block enable A disable on this bit overrides transmit channel enable.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Enable transmitter</li> <li><b>0x0:</b> Disable transmitter</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p>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.</p> <p><b>Values:</b></p>

**Table 13-178 Fields For Register: RXFFR (continued)**

Bits	Name	R/W	Description
			<p><b>0x1:</b> Flushes the RX FIFO  <b>0x0:</b> Does not flush the RX FIFO</p> <p><b>Value After Reset:</b> 0x0</p>

**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	<p>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.</p> <p><b>Values:</b></p> <p><b>0x1:</b> Flushes the TX FIFO  <b>0x0:</b> Does not flush the TX FIFO</p> <p><b>Value After Reset:</b> 0x0</p>

**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	<p>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.)</p> <p> Before reading this register again, the right stereo data MUST be read from RRBR0, or the status/interrupts will not be valid.</p> <p><b>Value After Reset:</b> 0x0</p>

**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	<p>The left stereo data to be transmitted serially through the transmit channel output is written through this register.</p> <p>Writing is a two-stage process:</p> <ol style="list-style-type: none"> <li>1. A write to this register passes the left stereo sample to the transmitter.</li> <li>2. This MUST be followed by writing the right stereo sample to the <a href="#">RTHR0</a> register.</li> </ol> <p>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.</p> <p><b>Value After Reset:</b> 0x0</p>

**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	<p>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 <a href="#">LRBR0</a> followed by a read from <a href="#">RRBR0</a>) 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.)</p> <p> Prior to reading this register, the left stereo data MUST be read from <a href="#">LRBR0</a>, or the status/interrupts will not be valid.</p> <p><b>Value After Reset:</b> 0x0</p>

**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	<p>The right stereo data to be transmitted serially through the transmit channel output is written through this register. Writing is a two-stage process:</p> <ol style="list-style-type: none"> <li>1. A left stereo sample MUST first be written to the <a href="#">LTHR0</a> register.</li> <li>2. A write to this register passes the right stereo sample to the transmitter.</li> </ol> <p>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.</p> <p><b>Value After Reset:</b> 0x0</p>

### 13.6.2.2.10. Receive Enable Register (**RERO**)

The **RERO** register is at offset 0x28

The following table shows the register bit assignments.

**Table 13-184 Fields For Register: RERO**

Bits	Name	R/W	Description
[31:1]			Reserved
[0]	RXCHEN0	R/W	<p>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 (<b>IER[0]</b> = 0) or the Receiver block (<b>IER[0]</b> = 0) overrides this value.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Enable</li> <li><b>0x0:</b> Disable</li> </ul> <p><b>Value After Reset:</b> 0x1</p>

### 13.6.2.2.11. Transmit Enable Register (**TERO**)

The **TERO** 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	<p>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 (<b>IER[0]</b> = 0) or Transmitter block (<b>IER[0]</b> = 0) overrides this value.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Enable</li> <li><b>0x0:</b> Disable</li> </ul> <p><b>Value After Reset:</b> 0x1</p>

### 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	<p>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 <a href="#">LRBRO</a> (or <a href="#">RRBRO</a>) register.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Ignore word length</li> <li><b>0x1:</b> 12 bit resolution</li> <li><b>0x2:</b> 16 bit resolution</li> <li><b>0x3:</b> 20 bit resolution</li> <li><b>0x4:</b> 24 bit resolution</li> <li><b>0x5:</b> 32 bit resolution</li> </ul> <p>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).</p> <p>The channel must be disabled prior to any changes in this value (<a href="#">RERO[0]</a> = 0).</p> <p><b>Value After Reset:</b> 0x5</p>

**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	<p>These bits are used to program the data resolution of the transmitter and ensure the MSB of the data is transmitted first.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Ignore word length</li> <li><b>0x1:</b> 12 bit resolution</li> <li><b>0x2:</b> 16 bit resolution</li> <li><b>0x3:</b> 20 bit resolution</li> <li><b>0x4:</b> 24 bit resolution</li> <li><b>0x5:</b> 32 bit resolution</li> </ul> <p>Programmed resolution must be less than or equal to 32. If the selected resolution is greater than 32, the transmit channel defaults back 32 bit resolution (0x5).</p> <p>The channel must be disabled prior to any changes in this value (<a href="#">TERO[0]</a> = 0).</p> <p><b>Value After Reset:</b> 0x2</p>

**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. <b>Values:</b> 0x0: TX FIFO write valid 0x1: TX FIFO write overrun <b>Value After Reset:</b> 0x0
[4]	TXFE	R	Status of Transmit Empty Trigger interrupt TX FIFO is empty. <b>Values:</b> 0x0: Trigger level reached 0x1: Trigger level not reached <b>Value After Reset:</b> 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. <b>Values:</b> 0x0: RX FIFO write valid 0x1: RX FIFO write overrun <b>Value After Reset:</b> 0x0
[0]	RXDA	R	Status of Receive Data Available interrupt RX FIFO data available. <b>Values:</b> 0x1: Trigger level not reached 0x0: Trigger level reached <b>Value After Reset:</b> 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 <b>Values:</b> 0x1: Masks interrupt 0x0: Unmasks interrupt <b>Value After Reset:</b> 0x1
[4]	TXFEM	R/W	Masks TX FIFO Empty interrupt

**Table 13-189 Fields For Register: IMR0 (continued)**

Bits	Name	R/W	Description
			<b>Values:</b> <b>0x1:</b> Masks interrupt <b>0x0:</b> Unmasks interrupt <b>Value After Reset:</b> 0x1
[3:2]			Reserved
[1]	RXFOM	R/W	Masks RX FIFO Overrun interrupt <b>Values:</b> <b>0x1:</b> Masks interrupt <b>0x0:</b> Unmasks interrupt <b>Value After Reset:</b> 0x1
[0]	RXDAM	R/W	Masks RX FIFO Data Available interrupt <b>Values:</b> <b>0x1:</b> Masks interrupt <b>0x0:</b> Unmasks interrupt <b>Value After Reset:</b> 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]	RXCHO	R	Read this bit to clear the RX FIFO Data Overrun interrupt. <b>Values:</b> <b>0x0:</b> RX FIFO write valid <b>0x1:</b> RX FIFO write overrun <b>Value After Reset:</b> 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]	TXCHO	R	Read this bit to clear the TX FIFO Data Overrun interrupt <b>Values:</b>

**Table 13-191 Fields For Register: TOR0 (continued)**

Bits	Name	R/W	Description
			<p><b>0x0:</b> TX FIFO write valid  <b>0x1:</b> TX FIFO write overrun</p> <p><b>Value After Reset:</b> 0x0</p>

**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	<p>These bits program the trigger level in the RX FIFO at which the Received Data Available interrupt is generated.</p> <p>Trigger Level = Programmed Value + 1            (for example, 1 to 16)</p> <p>Valid <b>RXCHDT</b> values: 0 to 15</p> <p>If an illegal value is programmed, these bits saturate to 15.</p> <p>The channel must be disabled prior to any changes in this value (that is, <b>RERO[0] = 0</b>).</p> <p><b>Value After Reset:</b> 0x3</p>

**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	<p>Transmit Channel Empty Trigger</p> <p>These bits program the trigger level in the TX FIFO at which the Empty Threshold Reached Interrupt is generated.</p> <p>Trigger Level = TXCHET</p> <p>TXCHET values: 0 to 15.</p> <p>If an illegal value is programmed, these bits saturate to 15.</p> <p>The channel must be disabled prior to any changes in this value (that is, <b>TERO[0] = 0</b>).</p> <p><b>Value After Reset:</b> 0x3</p>

**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	<p>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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Does not flush an individual RX FIFO</li> <li><b>0x1:</b> Flushes an individual RX FIFO</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p>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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Do not flush TX FIFO of the channel</li> <li><b>0x1:</b> Flush TX FIFO of the channel</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

### 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 LRBR0 and RRBRO 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. <b>Value After Reset:</b> 0x0

### 13.6.2.2.23. Reset Receiver Block DMA Register (RRXDMA)

The RXDMA register is at offset 0x1C4

The RXDMA can be reset to the lowest enabled Channel via the RXDMA register. The RXDMA 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. RXDMA Reset
4. Ch0 - Left Data
5. Ch0 - Right Data
6. Ch1 - Left Data
7. RXDMA Reset - No effect (read not complete)
8. Ch1 - Right Data, etc.
9. Ch2 - Left Data
10. Ch2 - Right Data
11. RXDMA Reset
12. Ch0 - Left Data
13. Ch0 - Right Data

The following table shows the register bit assignments.

**Table 13-197 Fields For Register: RXDMA**

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). <b>Value After Reset:</b> 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	<p>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.</p> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p>Reset Transmitter Block DMA Register Writing a 1 to this self-clearing register resets the <b>TXDMA</b> register mid-cycle to point to the lowest enabled Transmit channel.</p> <p> 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).</p> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p>DMA Enable for transmit block The corresponding bits of this field enables/disables the DMA handshake logic for transmitter block.</p>

**Table 13-200 Fields For Register: DMACR (continued)**

Bits	Name	R/W	Description
			<p><b>Values:</b></p> <p><b>0x1:</b> Enable  <b>0x0:</b> Disable</p> <p><b>Value After Reset:</b> 0x0</p>
[16]	DMAEN_RXBLOCK	R/W	<p>DMA Enable for receive block            The corresponding bits of this field enables/disables the DMA handshake logic for receiver block.</p> <p><b>Values:</b></p> <p><b>0x1:</b> Enable  <b>0x0:</b> Disable</p> <p><b>Value After Reset:</b> 0x0</p>
[15:0]			Reserved

## 14. USB 2.0

In this chapter:

- [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.

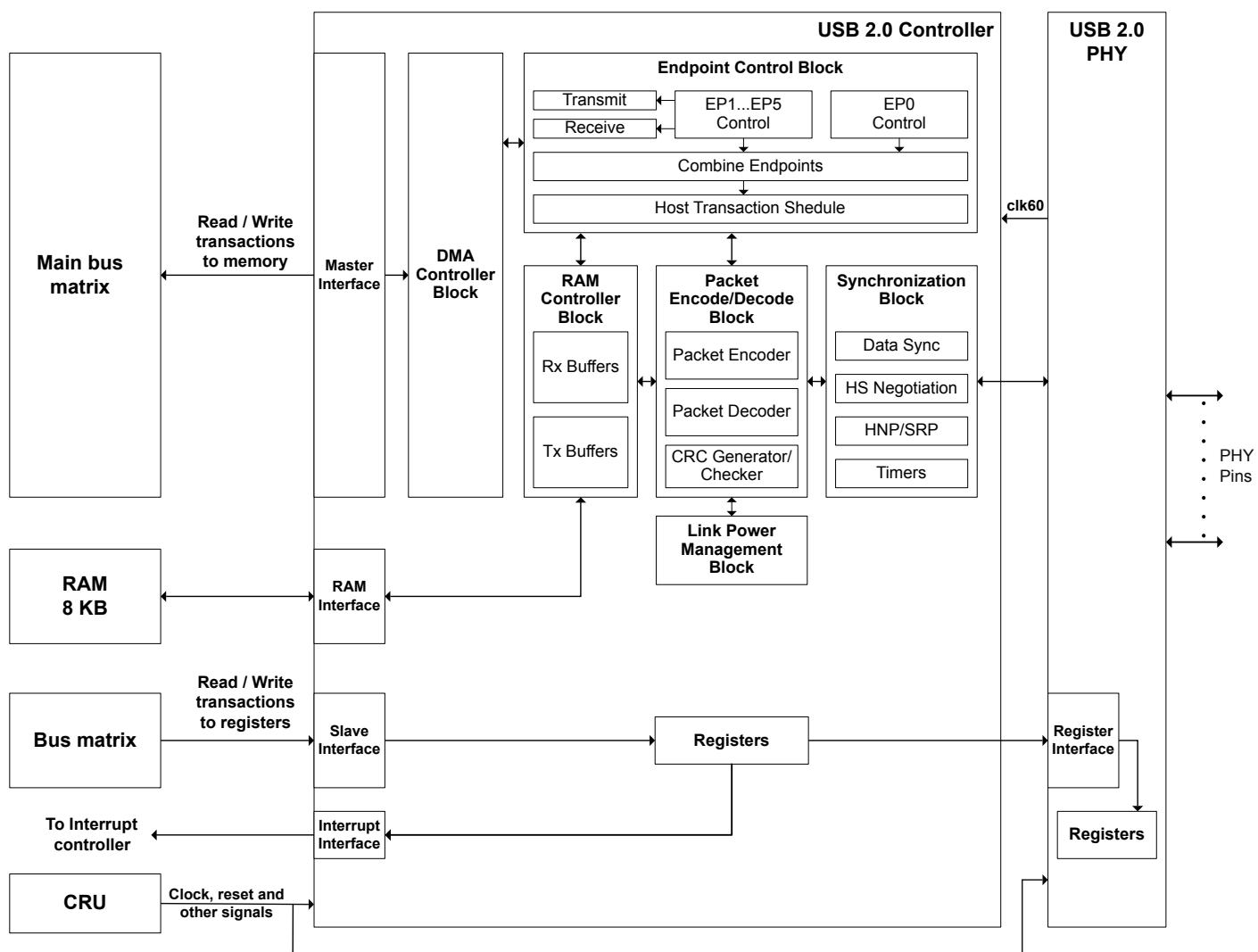


Figure 14-1 USB 2.0 Controller Block Diagram

The USB 2.0 Controller has the following 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.
- Single total RAM with size of 8K.
- 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

In this section:

- [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

In this section:

- 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 (D5) 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 (D4) 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 (D5) 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 (D4) 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 5V 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 2V. 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 in 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.

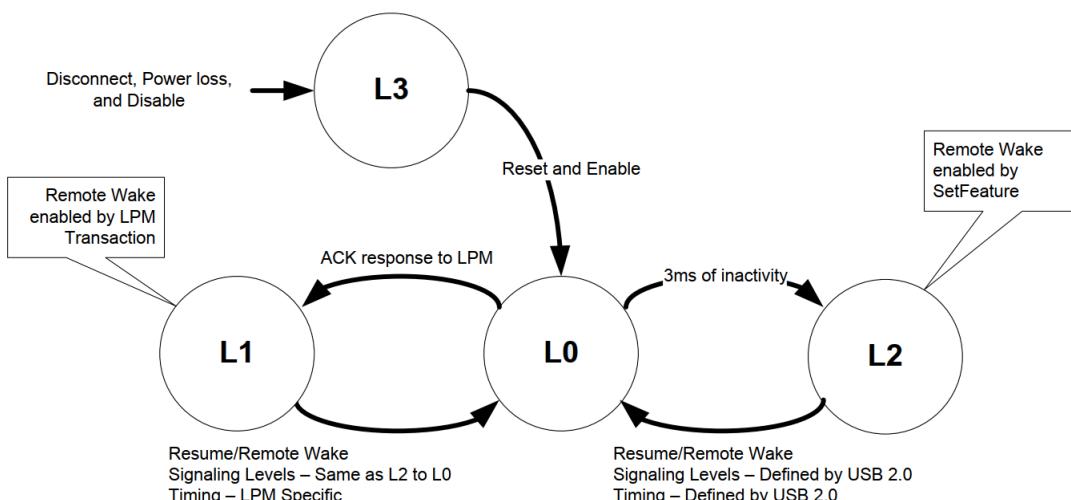
The procedures for session request and for transferring host/peripheral roles between the devices at either end of the connection are described in [OTG Session Request](#) and [Host Negotiation](#) sections, respectively. The transfers that are made are all subject to the standard USB data transfer protocols.

#### 14.1.1.3. Suspend/Resume

In this section:

- [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)

In this section:

- [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 (D2) to '1'.

The software should leave 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` (D0) 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 20ms. 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` (D2) 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)

In this section:

- [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_CTRL` appropriately. The register field `LPMEN` (in the `LPM_CTRL`) 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**

<code>LPMXMT</code> Field Value	<code>LPM_CTRL</code> Register Value	Data Pending (Data resides in the TX FIFOs)	Response to the next LPM transaction
1'b0	2'b00	Don't Care	Timeout
1'b0	2'b10		
1'b1	2'b00		
1'b1	2'b10		
1'b0	2'b01	Don't Care	STALL
1'b1			
1'b0	2'b11	Don't Care	NYET
1'b1	2'b11	Yes	NYET
1'b1	2'b11	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 50us to 1200us. 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 9us after transmitting the ACK. Resume signalling can be driven by the Host or the USB 2.0 Controller 50us after this event. During this 9us 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_CTRL` 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` + 10us.

### 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` + 10us. 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_CTRL`. If `LPMNAK` is set to 0x1, all endpoints will respond to any transaction (other than 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_CTRL` 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 in the `LPM_CTRL`.

### 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_CTR`. This bit is self clearing. Writing a 0x1 will cause resume signaling to be driven on the bus for 50us. The host will respond by driving resume for 60us to 990us. 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_CTR`.
- 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 8us. 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 8us 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_CTR`. 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 + 10us, 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 50us. The USB 2.0 Controller will immediately begin driving resume signaling on the BUS and will do so for 60us. 10us 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

In this section:

- 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 D3 – D0, the transaction type is selected through bits D5 – D4, and the operating speed is selected through bits D7 – D6.

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 D7 – D6 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 Guide.

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 & 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

In this section:

- [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`(D4) goes high). The speed of the device that has been connected can be determined by reading the `DEVCTL` where the FSDEV bit (D6) will be high for a high-speed/full-speed device and the LSDEV bit (D5) 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 (D4).

The CPU should keep the Reset bit set for 20ms 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

In this section:

- 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.

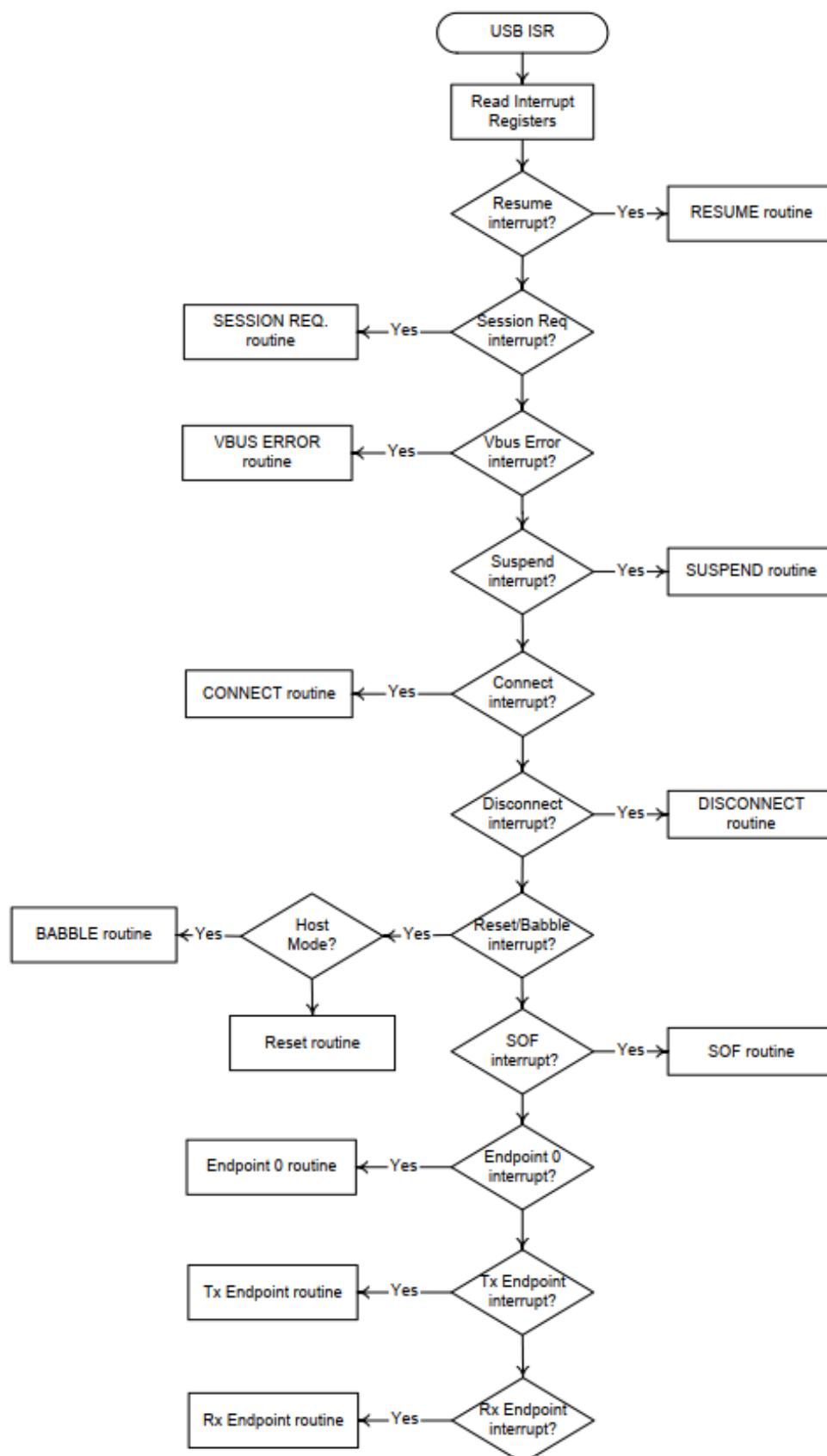
After a hardware reset (`nrst = 0`), `POWER.SOFT_CONN` is cleared to 0. The USB 2.0 Controller will therefore appear disconnected until the software has set `SOFT_CONN` to 1. The application software can then choose when to set the PHY into its normal mode. Systems with a lengthy initialization procedure 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 for the USB Interrupt Service Routine is given in the following flowchart.


**Figure 14-3 USB Interrupt Service Routine Diagram**

#### 14.1.1.7. OTG Session Request

In this section:

- 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 (D7) 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[1:0]` bits in the `DEVCTL` (D4 – D3) to go to 11b).

The USB 2.0 Controller will then wait for a peripheral to be connected. When a peripheral is detected, a Connect interrupt (`INTRUSB.CONN(D4)`) will be generated (if enabled) and either the `FSDEV` or `LSDEV` bit in the `DEVCTL` (D6/D5 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 (D5) 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 (D0) 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[1:0]` bits (D4 – D3) in the `DEVCTL` going to 00b) – and the line state has been SEO 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[1:0]` bits (D4 – D3) in the `DEVCTL` going to 10b) 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 (D0) in the DEVCTL. When Reset signaling is detected on the bus, a Reset interrupt ([INTRUSB.RESET](#) or [BABBLE](#) (D2)) 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\[1:0\]](#) bits (D4 – D3) in the DEVCTL going to 01b), 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](#) (D5)) 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](#) (D6) – if enabled) to indicate that the ‘B’ device is requesting a session. The software should then start a session by setting the SESSION bit (D0) 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](#) (D7) = 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](#) (D7) = 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](#) (D1). This bit can be set either at the same time as requesting a Session Start by setting the SESSION bit (D0) 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](#) (D4)) if this is enabled. It will also set the [RESET](#) bit (D3) 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 (D1) 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](#) (D5)) 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](#).

Underlying the support for an external DMA controller is a separate DMA request line for each Tx endpoint and each Rx endpoint.

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` (D12) or `RXCSRH` (D13) 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.

The required Request Mode needs to be selected even if we are using an external DMA controller.

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 (D0) 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 (D0) of the `CSR0L`.



When operating in Host mode, if either the `RX_STALL` bit (D5) of the `TXCSRL` or `ERROR` bit (D2) of the `TXCSRH` 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**

<code>DMAREQENAB</code>	<code>DMAREQMODE</code>	Interrupt generated?
0	X	Yes
1	0	No
1	1	Only if short packet

**Table 14-3 EP Interrupt Associated with `TXPKTRDY` Being Cleared**

<code>DMAREQENAB</code>	<code>DMAREQENAB</code>	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

In this section:

- [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 1K byte block.



The requirement to start a new burst at 1K 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 1K boundary. Bits D10–9 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

In this section:

- [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 (D13) 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 `RXCSR1.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 `TXCSR1.DMAREQENAB` bit (D12) 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 `TXCSR1.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 `INTR_RXE` 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_RQPCTCOUNT` 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 `RXCSR.L.RXPKTRDY` bit.

In Peripheral mode or where `EPn_RQPCTCOUNT` 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_RQPCTCOUNT` non-zero, the core will decrement the value in the `EPn_RQPCTCOUNT` 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 `TXCSR1.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

In this section:

- 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 and 4.75V)
- Session Valid for ‘A’ device (required to be between 0.8V and 2.1V)
- Session End (required to be between 0.2V and 0.8V)

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[1:0]` (`DEVCTL`).  
`[D4:D3]`) = 11b, `SESSION` bit (D0) 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.D4). The software should reset and enumerate the connected ‘B’ device.

- **VBus > Session Valid with session initiated by ‘B’ device** (i.e. `VBUS[1:0]` (D4:D3) of the `DEVCTL`) = 10b, `SESSION` bit (D0) 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 (D6)). 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 (D2) of the `DEVCTL`.

- **VBus below VBus Valid while the Session bit remains set** (i.e. `VBUS[1:0]` (D4:D3) of the `DEVCTL`) ≠ 11b, `SESSION` bit (D0) 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 (D7)).

#### 14.1.1.11.2. Actions as ‘B’ Device

- **VBus > Session Valid** (i.e. `VBUS[1:0]` (`DEVCTL`.[D4:D3]) = 10b, `SESSION` bit (D0) 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[1:0]` (`DEVCTL`.[D4:D3]) = 01b, `SESSION` bit (D0) 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 (D0) of the `DEVCTL` and generate a Disconnect interrupt (`INTRUSB.DISCON` bit (D5)). The software should end the session.

- **VBus < Session End** (i.e. `VBUS[1:0]` (`DEVCTL`.[D4:D3]) = 00b).

This is the condition under which a ‘B’ device can initiate a session request. If the `SESSION` bit (D0) of the `DEVCTL` is set, then after 2ms 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 8K bytes, 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	<a href="#">TX Endpoint #n FIFO Size (<code>TXFIFOSZ</code>)</a>

**Table 14-4 Endpoint #n FIFO Parameters Registers (continued)**

Address	ID
0x63	RX Endpoint #n FIFO Size ( <a href="#">RXFIFOSZ</a> )
0x64	TX Endpoint #n FIFO Address ( <a href="#">TXFIFOAD</a> )
0x65	
0x66	RX Endpoint #n FIFO Address ( <a href="#">RXFIFOAD</a> )
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)

In this section:

- [In Peripheral Mode](#)
- [In Host Mode](#)

##### 14.1.1.13.1. In Peripheral Mode

In this section:

- [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 (D0) 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 (D6) (indicating that the command has been read from the FIFO) and to set the `DATA_END` bit (D3) (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 (D6) and to set the `SEND_STALL` bit (D5). 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 (D2) 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 (D2) 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 (D0) 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 (D6) (indicating that the command has been read from the FIFO) but in this case the `DATA_END` bit (D3) 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 (D0) 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 (D6) and to set the `SEND_STALL` bit (D5). 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 (D2) of the `CSR0L` will be set.

If the host sends more data after the `DATAEND` 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 (D2) of the `CSR0L` will be set.

#### 14.1.13.1.3. 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 (D0) 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 (D6) (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 (D1) (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 (D3) (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 (D6) and to set the `SEND_STALL` bit (D5) 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 (D2) of the `CSR0L` will be set.

If the host requests more data after `DATA_END` (D3) 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 (D2) of the `CSR0L` will be set.

#### 14.1.13.1.4. 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.

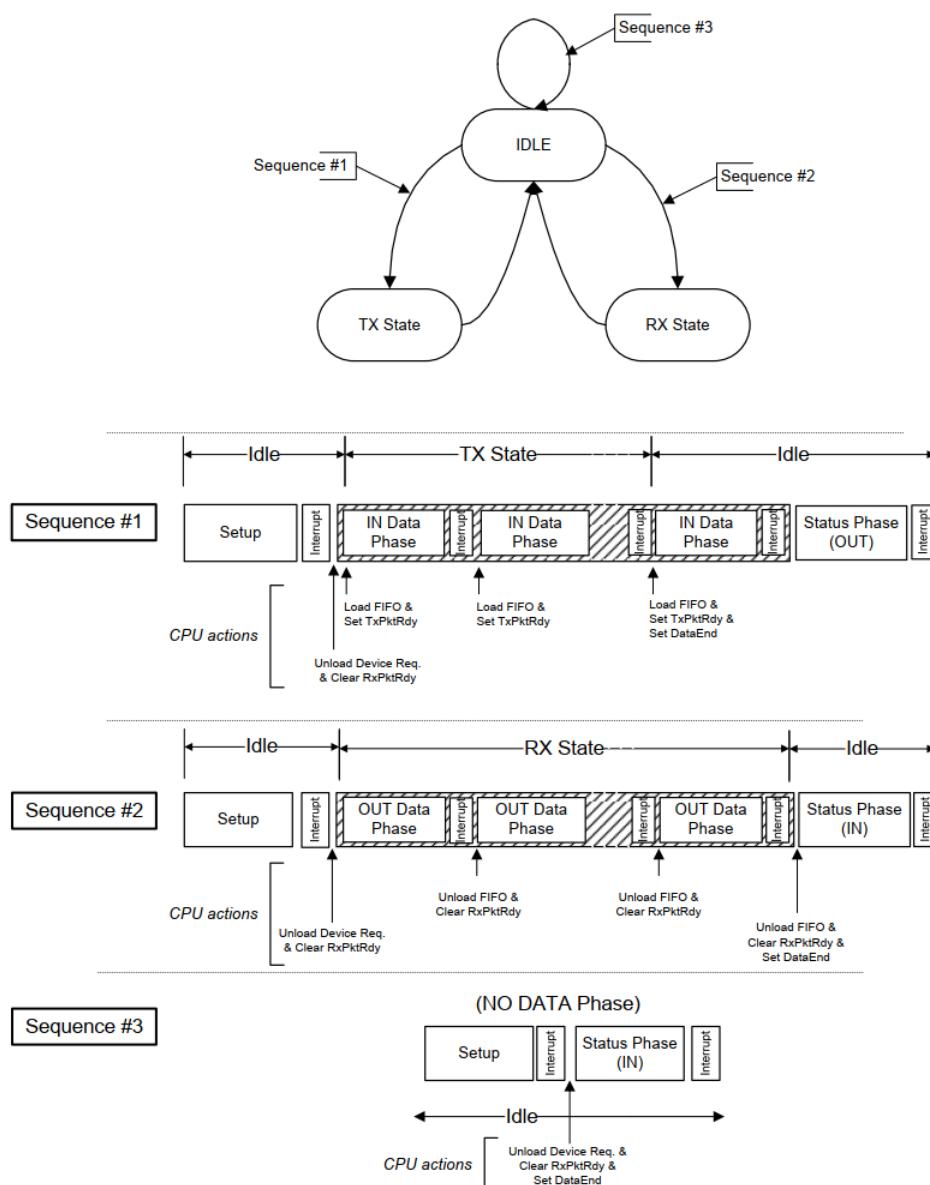
The default mode on power-up or reset should be IDLE.

RXPKTRDY bit (D0) 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 (D0) 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 (D1) 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 (D2) of the `CSR0L` after a control transaction is ended due to a protocol violation.
- When the core sets the `SETUP_END` bit (D4) of the `CSR0L` because a control transfer has ended before `DATA_END` bit (D3) 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.

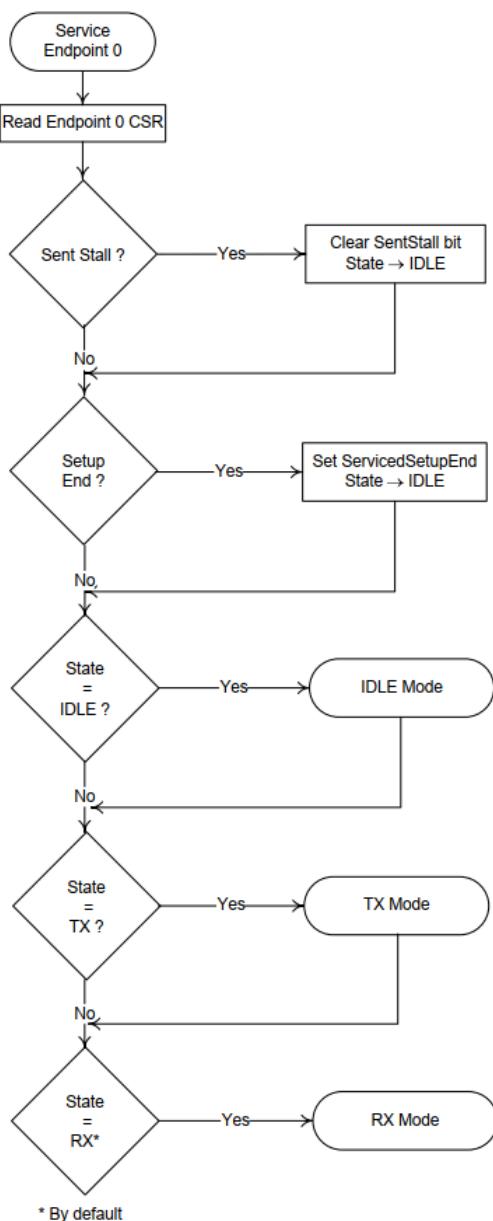
**If Endpoint 0 is in IDLE state**, the only valid reason an interrupt can be generated is as a result of the core receiving data from the USB bus. The service routine must check for this by testing the `RXPKTRDY` bit (D0) of the `CSR0L`. If this bit is set, then the core has received a SETUP packet. 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 (D6) of the `CSR0L` to indicate that it has read the data in the FIFO and leave the endpoint in RX state.

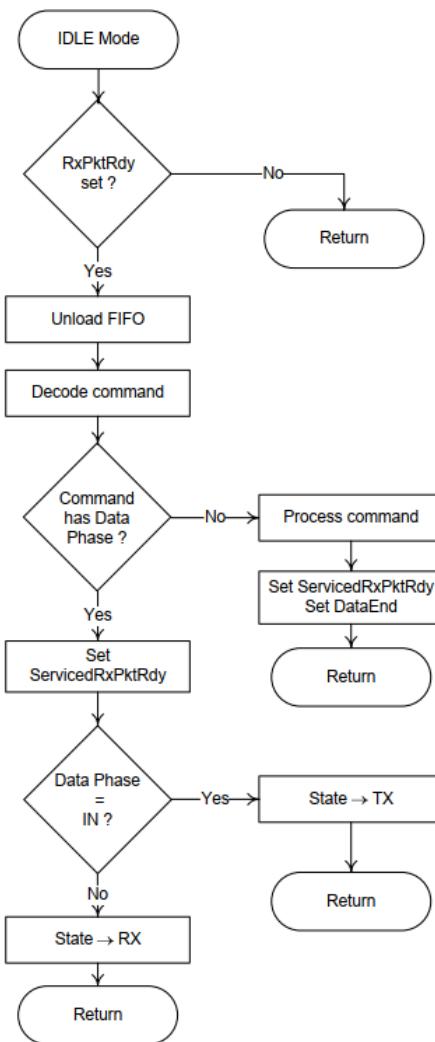


**Figure 14-5 Flow for Endpoint 0 Service Routine**

#### 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

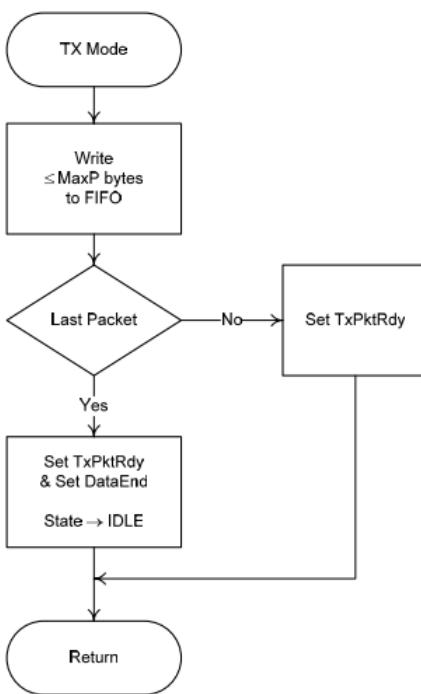
When the endpoint is in TX state, all arriving IN tokens need to be treated as part of a Data phase until the required amount of data has been sent to the host. If either a SETUP or an OUT token is received whilst the endpoint is in the TX state, this will cause a SetupEnd condition to occur as the core 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 (D4) 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 (D3) 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

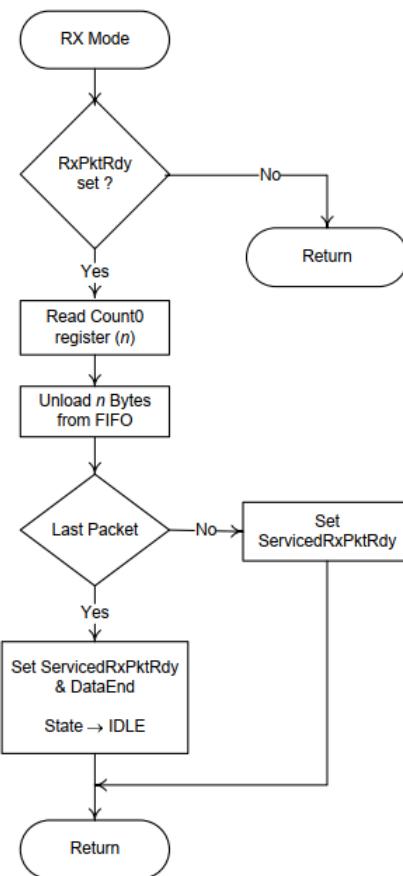
In RX mode, all arriving data should be treated as part of a Data phase until the expected amount of data has been received. If either a SETUP or an IN token is received while the endpoint is in RX state, this will cause a SetupEnd condition to occur as the core expects only OUT tokens.

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 (D4) 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 (`RXPTRDY` bit (D0) of the `CSR0L` set) and to clear `RXPTRDY` by setting the `SERVICED_RXPTRDY` bit (D6) 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 (D3) 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 (D3) 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 (D3) 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 (D2) 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 (D4) 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 (D7) of the `CSR0L`, and return to the IDLE state. If the

RXPKTRDY bit (D0) 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 (D5) of the [CSR0L](#). The USB 2.0 Controller will then send a STALL packet to the host, set the SENT\_STALL bit (D2) 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 (D4) of the [CSR0L](#).

#### 14.1.1.13.2. In Host Mode

In this section:

- [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` (D3) and `TXPKTRDY` (D1) 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` (D0)). The software should then read `CSR0L` to establish whether the `RXSTALL` bit (D2), the `ERROR` bit (D4) or the `NAK_TIMEOUT` bit (D7) 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 (D5) 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 `REQPKT` bit (D0) of the `INTRTX`), read `CSR0L` to establish whether the `RXSTALL` bit (D2), the `ERROR` bit (D4) or the `NAK_TIMEOUT` bit (D7) or `RXPKTRDY` (D0) 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 (D1) 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 (D0) of the `INTRTX`). The software should then read `CSR0L` to establish whether the `RXSTALL` bit (D2), the `ERROR` bit (D4) or the `NAK_TIMEOUT` bit (D7) 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` (D5) and `STATUSPKT` (D6) 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 (D0) of the `INTRTX`), read `CSR0L` to establish whether the `RXSTALL` bit (D2), the `ERROR` bit (D4) or the `NAK_TIMEOUT` bit (D7) or `RXPKTRDY` (D0) 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` (D6) and `TXPKTRDY` (D1) 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 (D0) of the `INTRTX`). The software should then read `CSR0L` to establish whether the `RXSTALL` bit (D2), the `ERROR` bit (D4) or the `NAK_TIMEOUT` bit (D7) 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

In this section:

- [In Peripheral Mode](#)
- [In Host Mode](#)
- [Employing DMA](#)

##### 14.1.1.14.1. In Peripheral Mode

In this section:

- [Bulk IN Transactions](#)
- [Bulk OUT Transactions](#)

###### 14.1.1.14.1.1. Bulk IN Transactions

In this section:

- [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 (D0) 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 (D6) 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
D15	AUTOSET	0/1	Set to 1 if the AutoSet feature is required.
D14	ISO	0	Set to 0 to enable Bulk protocol.
D13	MODE	1	Set to 1 to ensure the FIFO is enabled (only necessary if the FIFO is shared with an Rx endpoint).
D12	DMAREQENAB	0/1	Set to 1 if DMA Requests are required.   If set to 1, will also need to select the chosen DMAREQMODE ( <b>TXCSRH</b> .D2).
D11	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 (D0). 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 (D6) of the **CONFIGDATA**).

Since 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  $m \times \text{USB-payload}$  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 (D4) 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 (D5) of the `TXCSRL` and generate an interrupt.

When the software receives an interrupt with the `SENT_STALL` bit (D5) of the `TXCSRL` set, it should clear the `SENT_STALL` bit. It should however leave the `SEND_STALL` bit (D4) 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 (D6) in the `TXCSRL`.

#### 14.1.1.14.1.2. Bulk OUT Transactions

In this section:

- [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 (D0) of the `RXCSR` 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 (D7) 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 `RXCSR` should be set as shown below:

**Table 14-6 Values of RXCSR Register**

Bits	Name	Value	Description
D15	AUTOCLEAR	0/1	Set to 1 if the AutoSet feature is required.
D14	ISO	0	Set to 0 to enable Bulk protocol.
D13	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 <code>DMAREQMODE</code> ( <code>RXCSR</code> .D3).
D12	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 `RXCSR` should be written to set the `CLRDATATOG` bit (D7). 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 (D0) of the `RXCSR` being set), they should be flushed by setting `FLUSHFIFO` bit (D4) of the `RXCSR`.



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 (D0) 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 (D7) 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  $m \times wMaxPacketSize$  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 (D5) 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 (D6) of the `RXCSRL` and generate an interrupt.

When the software receives an interrupt with the `SENTSTALL` bit (D6) of the `RXCSRL` set, it should clear this bit. It should however leave the `SENDSTALL` bit (D5) 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 (D7) in the `RXCSRL`.

#### 14.1.1.14.2. In Host Mode

In this section:

- Bulk IN Transactions
- Bulk OUT Transactions

#### 14.1.1.14.2.1. Bulk IN Transactions

In this section:

- [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 (D5) 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 (D0) 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 (D7) 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 D7, D6 set to select the operating speed, bits D5, D4 = 10 (to select a Bulk transfer) and bits D3 – D0 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 – 215 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
D15	AUTOCLEAR	0/1	Set to 1 if the AutoClear feature is required.
D14	AUTOREQ	0/1	Set to 1 if the AutoRequest feature is required.
D13	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 ( <code>RXCSRH.D3</code> ).
D12	DISNYET	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` (D2 and D9) to toggle the current setting or by writing the `RXCSRL` to set the `CLRDATATOG` bit (D7). 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 (D0) of the `RXCSRL` being set), they should be flushed by setting the `FLUSHFIFO` bit (D4) 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 (D5) 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 (D0) of the `RXCSRL` is set. If the USB peripheral responds with a STALL, `RXSTALL` bit (D6) 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 (D7) 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  $m \times wMaxPacketSize$  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  $m \times wMaxPacketSize$ , 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 (D6) of the `RXCSR` being set.

#### 14.1.1.14.2.2. Bulk OUT Transactions

In this section:

- [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 (D0) 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 (D6) 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 D7, D6 set to select the operating speed, bits D5, D4 = 10 (to select a Bulk transfer) and bits D3 – D0 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 – 215 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
D15	AUTOSET	0/1	Set to 1 if the AutoSet feature is required.

**Table 14-8 Values of TXCSRH Register (continued)**

Bits	Name	Value	Description
D13	MODE	1	Set to 1 to ensure the FIFO is enabled (only necessary if the FIFO is shared with an Rx endpoint).
D12	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 (TXCSR.DMAREQMODE).
D11	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 to toggle the current setting or by writing the TXCSR to set the CLR\_DATA\_TOG bit (D6). 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 (D1) of the TXCSR being set), they should be flushed by setting the FLUSH\_FIFO bit (D3) of the TXCSR.



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 (D0) in the corresponding TXCSR. 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 (D0) of the TXCSR. If the USB peripheral responds with a STALL, RX\_STALL bit (D5) of the TXCSR 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 (TXCSR.ERROR set).

The USB 2.0 Controller then generates the appropriate endpoint interrupt, whereupon the software should read the corresponding TXCSR register to determine whether the RX\_STALL (D5), ERROR (D2) or NAK\_TIMEOUT (D7) 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[D10:D0] 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 (D6) of the CONFIGDATA). Where this option has been selected, packets up to 32 times the size specified

by `TXMAXP[D10:D0]` 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  $m \times \text{USB-payload}$  where `TXMAXP[D15:D11] = 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 be avoided 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 (D5) of the `TXCSRL` being set.

#### 14.1.1.14.3. Employing DMA

In this section:

- [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` (bits D15, D12 and D10 respectively).

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 (D2) in the `TXCSR` 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 `TXCSR` 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 `RXCSR`.`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 `RXCSR` 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

In this section:

- [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 11 (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

In this section:

- [In Peripheral Mode](#)
- [In Host Mode](#)

#### 14.1.1.16.1. In Peripheral Mode

In this section:

- [IN Transactions](#)
- [OUT Transactions](#)

#### 14.1.1.16.1.1. IN Transactions

In this section:

- [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 (D0) 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
D15	AUTOSET	0/1	Set to 1 if the AutoSet feature is required.
D14	ISO	1	Set to 0 to enable Isochronous protocol.
D13	MODE	1	Set to 1 to ensure the FIFO is enabled (only necessary if the FIFO is shared with an Rx endpoint).
D12	DMAREQENAB	0/1	Set to 1 if a DMA request is required for this endpoint.

**Table 14-9 Values of TXCSRH Register (continued)**

Bits	Name	Value	Description
			 If set to 1, will also need to select the chosen DMAREQMODE ( <a href="#">TXCSRH.D2</a> ).
D11	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 under run 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 (D0) 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 (D0) 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 (D7) 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 (D2) 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 (D0) in the `TXCSR` 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 (D3) in the `TXCSR`, or it may choose to skip the current packet.

#### 14.1.1.16.1.2. OUT Transactions

In this section:

- [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 (D0) of the `RXCSR` 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
D15	AUTOSET	0/1	Set to 1 if the AutoSet feature is required.
D14	ISO	1	Set to 0 to enable Isochronous protocol.
D13	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 ( <a href="#">RXCSRH</a> .D3).
D12	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 (D0) 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 (D2) 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 (D0) and the [DATAERROR](#) bit (D3) of the [RXCSRL](#)). It is left up to the application how this error condition is handled.

### 14.1.1.16.2. In Host Mode

In this section:

- [IN Transactions](#)
- [OUT Transactions](#)

#### 14.1.1.16.2.1. IN Transactions

In this section:

- [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 (D0) of the `RXCSR` 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 (D5) of the `RXCSR` 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 D7, D6 set to select the operating speed, bits D5, D4 = 01 (to select an Isochronous transfer) and bits D3 – D0 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
D15	AUTOCLEAR	0/1	Set to 1 if the AutoClear feature is required.
D14	AUTOREQ	0/1	Set to 1 if the AutoRequest feature is required.
D13	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 ( <code>RXCSRH</code> .D3).
D12	DISNYET	0	Ignored in Isochronous mode.

#### 14.1.1.16.2.1.2. Operation

The operation starts with the CPU setting `REQPKT` bit (D5) of the `RXCSR`. 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 (D0) in the `RXCSR` 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 `RXCSR`.

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 (D3) of the `RXCSR` is set to indicate that the data may be corrupt.

### 14.1.1.16.2.2. OUT Transactions

In this section:

- [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 (D0) 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 D7, D6 set to select the operating speed, bits D5, D4 = 01 (to select an Isochronous transfer) and bits D3 – D0 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
D15	AUTOSET	0/1	Set to 1 if the AutoSet feature is required.

**Table 14-12 Values of TXCSR<sub>H</sub> Register (continued)**

Bits	Name	Value	Description
D13	MODE	1	Set to 1 to ensure FIFO is enabled (only necessary if the FIFO is shared with an Rx endpoint).
D12	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 ( <a href="#">TXCSR<sub>H</sub>.D2</a> ).
D11	DISNYET	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 (D0) of the [TXCSR<sub>L</sub>](#). 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 (D0) in the [TXCSR<sub>L</sub>](#) 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 (D0) in the [TXCSR<sub>L</sub>](#).

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.

- 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 D11 and D12 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 1Kbyte.

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 (D7) in the `TXCSR1`.

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 (D8) in the `RXCSR1`. 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

In this section:

- [Data Field Format](#)
- [Registers Map](#)
- [Register Descriptions](#)

The following table describes BE-U1000 microcontroller USB 2.0 subsystem register region, mapped in the static IO memory of the Application Processor.

**Table 14-13 Memory Address Region for USB Registers**

Block Name	Size	Start Address	End Address
USB*	1MB	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 registers region coincides with USB 2.0 Subsystem registers 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 [Registers Map](#) and [Register Descriptions](#) sections below.

### 14.1.2.1. Data Field Format

Based on the requirements of ***Universal Serial Bus Specification Revision 2.0, Section 8.3.4 Data Field*** and for the convenience of the reader, in the text below and above you can find abbreviations like D0, D1, D2, D3, D4, D5, D6, D7, which can be used both individually (D0, D1-D7 and so on) and in form like, for example: `CSR0H.D0` - this abbreviations means bit number in a byte. In the above example (`CSR0H.D0`) it means reference to [0] bit of the `CSR0H`.

Some registers contains of two bytes of data, divided by one byte and in this case menitioned in text numbering starts from the lower byte to higher. For example this type of registers is:

- TX Endpoint #n Control and Status Register 0 ([TXCSR<sub>L</sub>](#)) (D0-D7)
- TX Endpoint #n Control and Status Register 1 ([TXCSR<sub>H</sub>](#)) (D8-15)
- RX Endpoint #n Control and Status Register 0 ([RXCSR<sub>L</sub>](#)) (D0-D7)
- RX Endpoint #n Control and Status Register 1 ([RXCSR<sub>H</sub>](#)) (D8-15)

#### 14.1.2.2. Registers 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 high-level summary of each register. To view the detailed description of the register, click the register name in the **Description** column.

**Table 14-14 USB 2.0 Controller Registers Map**

Name	Offset	Description	Default Value
<b>Common USB Registers</b>			
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
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
<b>Indexed Registers</b>			
TXMAXP	0x10	Maximum Packet Size for TX Endpoint #n <b>Exists:</b> INDEX.SELECTED_EP != 0x0	0x0
CSR0L	0x12	Endpoint 0 Control and Status Register 0 <b>Exists:</b> INDEX.SELECTED_EP = 0x0	0x0
TXCSR <sub>L</sub>	0x12	TX Endpoint #n Control and Status Register 0	0x0

**Table 14-14 USB 2.0 Controller Registers Map (continued)**

Name	Offset	Description	Default Value
		<b>Exists:</b> INDEX.SELECTED_EP != 0x0	
CSR0H	0x13	Endpoint 0 Control and Status Register 1 <b>Exists:</b> INDEX.SELECTED_EP = 0x0	0x0
TXCSRH	0x13	TX Endpoint #n Control and Status Register 1 <b>Exists:</b> INDEX.SELECTED_EP != 0x0	0x0
RXMAXP	0x14	Maximum Packet Size for RX Endpoint #n <b>Exists:</b> INDEX.SELECTED_EP != 0x0	0x0
RXCSRL	0x16	RX Endpoint #n Control and Status Register 0 <b>Exists:</b> INDEX.SELECTED_EP != 0x0	0x0
RXCSRH	0x17	RX Endpoint #n Control and Status Register 1 <b>Exists:</b> INDEX.SELECTED_EP != 0x0	0x0
COUNT0	0x18*	Endpoint 0 Data Bytes Counter Register <b>Exists:</b> INDEX.SELECTED_EP = 0x0	0x0
RXCOUNT	0x18*	RX Endpoint #n Data Bytes Counter Register <b>Exists:</b> INDEX.SELECTED_EP != 0x0	0x0
TYPE0	0x1A*	Endpoint 0 Speed Register <b>Exists:</b> DEVCTL.HOST_MODE = 0x1 && INDEX.SELECTED_EP = 0x0	0x0
TXTYPE	0x1A*	TX Endpoint #n Settings Register <b>Exists:</b> DEVCTL.HOST_MODE = 0x1 && INDEX.SELECTED_EP != 0x0	0x0
NAKLIMIT0	0x1B*	Endpoint 0 NAK Limit Register <b>Exists:</b> DEVCTL.HOST_MODE = 0x1 && INDEX.SELECTED_EP = 0x0	0x0
TXINTERVAL	0x1B*	TX Endpoint #n NAK Limit/Polling Interval Register <b>Exists:</b> DEVCTL.HOST_MODE = 0x1 && INDEX.SELECTED_EP != 0x0	0x0
RXTYPE	0x1C	RX Endpoint #n Settings Register <b>Exists:</b> DEVCTL.HOST_MODE = 0x1 && INDEX.SELECTED_EP != 0x0	0x0
RXINTERVAL	0x1D	RX Endpoint #n NAK Limit/Polling Interval Register <b>Exists:</b> DEVCTL.HOST_MODE = 0x1 && INDEX.SELECTED_EP != 0x0	0x0
CONFIGDATA	0x1F	Configuration Data Register <b>Exists:</b> INDEX.SELECTED_EP = 0x0	0xDE

**Table 14-14 USB 2.0 Controller Registers Map (continued)**

Name	Offset	Description	Default Value
	0x1F	Reserved	
<b>FIFOs</b>			
<b>FIFON</b> (where <b>n</b> = 0...5)	0x20 + ( <b>n</b> *0x4)	<a href="#">Endpoint #n FIFO</a>	0x0
<b>Additional Control &amp; Configuration Registers</b>			
DEVCTL	0x60	<a href="#">Device Control Register</a>	0x80
	0x61	Reserved	
TXFIFOSZ	0x62	<a href="#">TX Endpoint #n FIFO Size</a> <b>Exists:</b> <code>INDEX.SELECTED_EP != 0x0</code>	0x0
RXFIFOSZ	0x63	<a href="#">RX Endpoint #n FIFO Size</a> <b>Exists:</b> <code>INDEX.SELECTED_EP != 0x0</code>	0x0
TXFIFOADD	0x64	<a href="#">TX Endpoint #n FIFO Address</a> <b>Exists:</b> <code>INDEX.SELECTED_EP != 0x0</code>	0x0
RXFIFOADD	0x66	<a href="#">RX Endpoint #n FIFO Address</a> <b>Exists:</b> <code>INDEX.SELECTED_EP != 0x0</code>	0x0
VCONTROL	0x68**	<a href="#">PHY Signals Control Register</a> <b>Exists:</b> This register is presented when the write at the address 0x68 is performed	0x0
VSTATUS	0x68**	<a href="#">PHY Signals Status Register</a> <b>Exists:</b> This register is presented when the read at the address 0x68 is performed	0x0
EPINFO	0x78	<a href="#">Information About Numbers of TX and Rx Endpoints</a>	0x55
RAMINFO	0x79	<a href="#">Information About RAM</a>	0xB*
LINKINFO	0x7A	<a href="#">Information About Delays</a>	0x5C
VPLEN	0x7B	<a href="#">Duration of the VBus Pulsing Charge</a>	0x3C
HS_EOF1	0x7C	<a href="#">Time Buffer Available on High-Speed Transactions</a>	0x80
FS_EOF1	0x7D	<a href="#">Time Buffer Available on Full-Speed Transactions</a>	0x77
LS_EOF1	0x7E	<a href="#">Time Buffer Available on Low-Speed Transactions</a>	0x72
	0x7F	Reserved	
<b>Target Address Registers</b>			
TXFUNCADDR (where <b>n</b> = 0...5)	0x80 + ( <b>n</b> *0x8)	<a href="#">TX Endpoint #n Function Address</a> <b>Exists:</b> <code>DEVCTL.HOST_MODE = 0x1</code>	0x0

**Table 14-14 USB 2.0 Controller Registers Map (continued)**

Name	Offset	Description	Default Value
TXHUBADDR (where n = 0...5)	0x82 + (n*0x8)	TX Endpoint #n Hub Address <b>Exists:</b> DEVCTL.HOST_MODE = 0x1	0x0
TXHUBPORT (where n = 0...5)	0x83 + (n*0x8)	TX Endpoint #n Hub Port <b>Exists:</b> DEVCTL.HOST_MODE = 0x1	0x0
RXFUNCADDR (where n = 0...5)	0x84 + (n*0x8)	RX Endpoint #n Function Address <b>Exists:</b> DEVCTL.HOST_MODE = 0x1	0x0
RXHUBADDR (where n = 0...5)	0x86 + (n*0x8)	RX Endpoint #n Hub Address <b>Exists:</b> DEVCTL.HOST_MODE = 0x1	0x0
RXHUBPORT (where n = 0...5)	0x87 + (n*0x8)	RX Endpoint #n Hub Port <b>Exists:</b> DEVCTL.HOST_MODE = 0x1	0x0
<b>DMA Registers</b>			
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
<b>Extended Registers</b>			
EPn_RQPKTCOUNT (where n = 1...5)	0x300 + (n*0x4)	Number of Requested Packets for RX Endpoint #n <b>Exists:</b> 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
C_T_HSBT	0x348	High Speed Timeout Increase Register	0x0
<b>LPM Registers</b>			
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 <b>Exists:</b> 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.

### 14.1.2.3. 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

#### 14.1.2.3.1. 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.D2=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. <b>Value After Reset:</b> 0x0

#### 14.1.2.3.2. 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	<p>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.</p> <p> Only valid in Peripheral Mode. Also, this bit only affects endpoints performing Isochronous transfers.</p> <p> This bit is not valid in Host Mode.</p> <p><b>Value After Reset:</b> 0x0</p>
[6]	SOFT_CONN	R/W	R	<p>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.</p> <p> Only valid in Peripheral Mode.</p> <p> This bit is not valid in Host Mode.</p> <p><b>Value After Reset:</b> 0x0</p>
[5]	HS_ENAB	R/W	R	<p>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.</p> <p><b>Value After Reset:</b> 0x1</p>
[4]	HS_MODE	R	R/W	<p>When set, this bit indicates High-speed mode successfully negotiated during USB reset. 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.</p> <p> Allowance is made for Tiny-J signaling in determining the transfer speed to select.</p> <p><b>Value After Reset:</b> 0x0</p>
[3]	RESET	*Varies	R/W	<p>This bit is set when Reset signaling is present on the bus.</p> <p> This bit is Read/Write from the software in Host Mode but Read-Only in Peripheral Mode.</p> <p>The access attributes of this field are as follows:</p>

**Table 14-16 Fields For Register: POWER (continued)**

Bits	Name	Software Access	USB Access	Description
				<ul style="list-style-type: none"> <li>• Peripheral Mode           <ul style="list-style-type: none"> <li>◦ Software Access: R</li> </ul> </li> <li>• Host Mode           <ul style="list-style-type: none"> <li>◦ Software Access: R/W</li> </ul> </li> </ul> <b>Value After Reset:</b> 0x0
[2]	RESUME	R/W	*Varies	<p>Set by the software to generate Resume signaling when the device is in Suspend mode.</p> <p>In Peripheral mode, the software should clear this bit after 10 ms (a maximum of 15 ms), to end Resume signaling.</p> <p>In Host mode, the software should clear this bit after 20 ms.</p> <p>The access attributes of this field are as follows:</p> <ul style="list-style-type: none"> <li>• Peripheral Mode           <ul style="list-style-type: none"> <li>◦ USB Access: R</li> </ul> </li> <li>• Host Mode           <ul style="list-style-type: none"> <li>◦ USB Access: R/WS</li> </ul> </li> </ul> <b>Value After Reset:</b> 0x0
[1]	SUSPEND_MODE	*Varies	*Varies	<p>In Host mode, this bit is set by the software to enter Suspend mode.</p> <p>In Peripheral mode, this bit is set on entry into Suspend mode.</p> <p>It is cleared when the software reads the interrupt register, or sets the Resume bit above.</p> <p>The access attributes of this field are as follows:</p> <ul style="list-style-type: none"> <li>• Peripheral Mode           <ul style="list-style-type: none"> <li>◦ Software Access: R</li> <li>◦ USB Access: R/W</li> </ul> </li> <li>• Host Mode           <ul style="list-style-type: none"> <li>◦ Software Access: WS</li> <li>◦ USB Access: WC</li> </ul> </li> </ul> <b>Value After Reset:</b> 0x0
[0]	ENABLE_SUSPENDM	R/W	R	<p>Set by the software to enable the suspendm output.</p> <b>Value After Reset:</b> 0x0

#### 14.1.2.3.3. 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 <b>Value After Reset:</b> 0x0
[4]	EP4_TX	R	WS	TX Endpoint 4 interrupt <b>Value After Reset:</b> 0x0
[3]	EP3_TX	R	WS	TX Endpoint 3 interrupt <b>Value After Reset:</b> 0x0
[2]	EP2_TX	R	WS	TX Endpoint 2 interrupt <b>Value After Reset:</b> 0x0
[1]	EP1_TX	R	WS	TX Endpoint 1 interrupt <b>Value After Reset:</b> 0x0
[0]	EP0	R	WS	Endpoint 0 interrupt <b>Value After Reset:</b> 0x0

#### 14.1.2.3.4. 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 <b>Value After Reset:</b> 0x0
[4]	EP4_RX	R	WS	RX Endpoint 4 interrupt <b>Value After Reset:</b> 0x0
[3]	EP3_RX	R	WS	RX Endpoint 3 interrupt <b>Value After Reset:</b> 0x0
[2]	EP2_RX	R	WS	RX Endpoint 2 interrupt <b>Value After Reset:</b> 0x0

**Table 14-18 Fields For Register: INTRRX (continued)**

Bits	Name	Software Access	USB Access	Description
[1]	EP1_RX	R	WS	RX Endpoint 1 interrupt <b>Value After Reset:</b> 0x0
[0]				Reserved

#### 14.1.2.3.5. 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 included in the design 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 <b>Value After Reset:</b> 0x1
[4]	EP4_TX_E	RW	R	TX Endpoint 4 interrupt enable <b>Value After Reset:</b> 0x1
[3]	EP3_TX_E	RW	R	TX Endpoint 3 interrupt enable <b>Value After Reset:</b> 0x1
[2]	EP2_TX_E	RW	R	TX Endpoint 2 interrupt enable <b>Value After Reset:</b> 0x1
[1]	EP1_TX_E	RW	R	TX Endpoint 1 interrupt enable <b>Value After Reset:</b> 0x1
[0]	EP0_E	RW	R	Endpoint 0 interrupt enable <b>Value After Reset:</b> 0x1

#### 14.1.2.3.6. 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 included in the design 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 <b>Value After Reset:</b> 0x1
[4]	EP4_RX_E	RW	R	RX Endpoint 4 interrupt enable <b>Value After Reset:</b> 0x1
[3]	EP3_RX_E	RW	R	RX Endpoint 3 interrupt enable <b>Value After Reset:</b> 0x1
[2]	EP2_RX_E	RW	R	RX Endpoint 2 interrupt enable <b>Value After Reset:</b> 0x1
[1]	EP1_RX_E	RW	R	RX Endpoint 1 interrupt enable <b>Value After Reset:</b> 0x1
[0]				Reserved

#### 14.1.2.3.7. 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 <code>vbusvalid</code> threshold during a session. Only valid when USB 2.0 Controller is A' device. <b>Value After Reset:</b> 0x0
[6]	SESS_REQ	R	WS	Set when Session Request signaling has been detected. Only valid when USB 2.0 Controller is A' device. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 0x0

**Table 14-21 Fields For Register: INTRUSB (continued)**

Bits	Name	Software Access	USB Access	Description
[4]	CONN	R	WS	Set when a device connection is detected. Only valid in Host mode. Valid at all transaction speeds. <b>Value After Reset:</b> 0x0
[3]	SOF	R	WS	Set when a new frame starts. <b>Value After Reset:</b> 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.  <b>Value After Reset:</b> 0x0
[1]	RESUME	R	WS	Set when Resume signaling is detected on the bus while the USB 2.0 Controller is in Suspend mode. <b>Value After Reset:</b> 0x0
[0]	SUSPEND	R	WS	Set when Suspend signaling is detected on the bus. Only valid in Peripheral mode. <b>Value After Reset:</b> 0x0

#### 14.1.2.3.8. 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 <b>Value After Reset:</b> 0x0
[6]	SESS_REQ_E	R/W	R	SESS_REQ Enable <b>Value After Reset:</b> 0x0
[5]	DISCON_E	R/W	R	DISCON Enable <b>Value After Reset:</b> 0x0
[4]	CONN_E	R/W	R	CONN Enable <b>Value After Reset:</b> 0x0
[3]	SOF_E	R/W	R	SOF Enable <b>Value After Reset:</b> 0x0

**Table 14-22 Fields For Register: INTRUSBE (continued)**

Bits	Name	Software Access	USB Access	Description
[2]	RESET_E_OR_BABBLE_E	R/W	R	RESET_E: RESET Enable BABBLE: BABBLE Enable <b>Value After Reset:</b> 0x1
[1]	RESUME_E	R/W	R	RESUME Enable <b>Value After Reset:</b> 0x1
[0]	SUSPEND_E	R/W	R	SUSPEND Enable <b>Value After Reset:</b> 0x0

**14.1.2.3.9. 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 <b>FRAME_NUMBER[10]</b> is the <b>Most Significant Bit (MSB)</b> and <b>FRAME_NUMBER[0]</b> is the <b>Last Significant Bit (LSB)</b> <b>Value After Reset:</b> 0x0

**14.1.2.3.10. 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

**Table 14-24 Fields For Register: INDEX (continued)**

Bits	Name	Software Access	USB Access	Description
				Here <code>SELECTED_EP[4]</code> is the <i>Most Significant Bit (MSB)</i> and <code>SELECTED_EP[0]</code> is the <i>Last Significant Bit (LSB)</i> <b>Value After Reset:</b> 0x0

#### 14.1.2.3.11. 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]	<code>FORCE_HOST</code>	R/W	R	<p>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, <code>hostdiscon</code> and <code>linestate[1:0]</code> signals are ignored. The core will then remain in Host mode until the <code>SESSION</code> bit is cleared, even if a device is disconnected, and if the <code>FORCE_HOST</code> bit remains set, will re-enter Host mode the next time the <code>SESSION</code> bit is set.</p> <p>While in this mode, the status of the <code>HOSTDISCON</code> signal from the PHY may be read from bit [7] of the <code>DEVCTL</code>.</p> <p>The operating speed is determined from the <code>FORCE_HS</code> and <code>FORCE_FS</code> bits as follows:</p> <ul style="list-style-type: none"> <li>• <code>FORCE_HS=0x0</code> and <code>FORCE_FS=0x0</code>: Low Speed;</li> <li>• <code>FORCE_HS=0x0</code> and <code>FORCE_FS=0x1</code>: Full Speed;</li> <li>• <code>FORCE_HS=0x1</code> and <code>FORCE_FS=0x0</code>: High Speed;</li> <li>• <code>FORCE_HS=0x1</code> and <code>FORCE_FS=0x1</code>: Undefined.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[6]	<code>FIFO_ACCESS</code>	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.

**Table 14-25 Fields For Register: TESTMODE (continued)**

Bits	Name	Software Access	USB Access	Description
				<b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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 <b><i>Universal Serial Bus Specification Revision 2.0, Section 7.1.20 (and in section 28.4).</i></b>  The test packet has a fixed format and must be loaded into the Endpoint 0 FIFO before the test mode is entered. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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 <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 0x0

**14.1.2.3.12. 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, D15 – D13 is not implemented and D12 – D11(if included) is 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 **TXCSR**) 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]	MULTIPLIER_VAL	RW	R	<p>This bit field defines the value of the multiplier m which is equal to one more than the value recorded.</p> <p><b>Value After Reset:</b> 0x0</p>

**Table 14-26 Fields For Register: TXMAXP (continued)**

Bits	Name	Software Access	USB Access	Description
[10:0]	MAX_PAYLOAD	RW	R	<p>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.</p> <p><b>Value After Reset:</b> 0x0</p>

#### 14.1.2.3.13. 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	<p>The software writes a 1 to this bit to clear the SETUP_END bit. It is cleared automatically.</p> <p><b>Value After Reset:</b> 0x0</p>
[6]	SERVICED_RXPKTRDY	WS	R	<p>The software writes a 1 to this bit to clear the RXPKTRDY bit. It is cleared automatically.</p> <p><b>Value After Reset:</b> 0x0</p>
[5]	SEND_STALL	WS	R	<p>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.</p> <p><b>Value After Reset:</b> 0x0</p>
[4]	SETUP_END	R	WS	<p>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 software writing a 1 to the SERVICED_SETUP_END bit.</p> <p><b>Value After Reset:</b> 0x0</p>
[3]	DATA_END	WS	R	The software sets this bit:

**Table 14-27 Fields For Register: CSR0L (DEVCTL.HOST\_MODE = 0x0) (continued)**

Bits	Name	Software Access	USB Access	Description
				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. <b>Value After Reset:</b> 0x0
[2]	SENT_STALL	R/WC	WS	This bit is set when a STALL handshake is transmitted. The software should clear this bit. <b>Value After Reset:</b> 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). <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 0x0
[5]	REQPKT	RW	RW	The software sets this bit to request an IN transaction. It is cleared when RXPKTRDY is set. <b>Value After Reset:</b> 0x0

**Table 14-28 Fields For Register: CSROL (DEVCTL.HOST\_MODE = 0x1) (continued)**

Bits	Name	Software Access	USB Access	Description
[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. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 0x0
[2]	RXSTALL	R/WC	WS	This bit is set when a STALL handshake is received. The software should clear this bit. <b>Value After Reset:</b> 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). <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 0x0

#### 14.1.2.3.14. 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	<p>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.</p> <p> In anything other than isochronous transfers, this bit will always return 0.</p> <p><b>Value After Reset:</b> 0x0</p>
[6]	CLR_DATA_TOG	WS	R/WC	<p>The software writes a 1 to this bit to reset the endpoint data toggle to 0.</p> <p><b>Value After Reset:</b> 0x0</p>
[5]	SENT_STALL	R/WC	WS	<p>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.</p> <p><b>Value After Reset:</b> 0x0</p>
[4]	SEND_STALL	RW	R	<p>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.</p> <p> This bit has no effect where the endpoint is being used for Isochronous transfers.</p> <p><b>Value After Reset:</b> 0x0</p>
[3]	FLUSH_FIFO	WS	R	<p>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.</p> <p>May be set simultaneously with TXPKTRDY to abort the packet that is currently being loaded into the FIFO.</p> <p> 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.</p> <p><b>Value After Reset:</b> 0x0</p>
[2]	UNDER_RUN	R/WC	WS	<p>The USB sets this bit if an IN token is received when TXPKTRDY is not set. The software should clear this bit.</p> <p><b>Value After Reset:</b> 0x0</p>
[1]	FIFO_NOT_EMPTY	R/WC	WS	<p>The USB sets this bit when there is at least 1 packet in the TX FIFO.</p> <p><b>Value After Reset:</b> 0x0</p>

**Table 14-29 Fields For Register: TXCSRL (DEVCTL.HOST\_MODE = 0x0) (continued)**

Bits	Name	Software Access	USB Access	Description
[0]	TXPKTRDY	R/WS	WC	<p>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.</p> <p><b>Value After Reset:</b> 0x0</p>

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	<p>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 <a href="#">TXINTERVAL</a>. The CPU should clear this bit to allow the endpoint to continue.</p> <p>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.</p> <p><b>Value After Reset:</b> 0x0</p>
[6]	CLR_DATA_TOG	WS	R/WC	<p>The software writes a 1 to this bit to reset the endpoint data toggle to 0.</p> <p><b>Value After Reset:</b> 0x0</p>
[5]	RX_STALL	R/WC	WS	<p>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.</p> <p><b>Value After Reset:</b> 0x0</p>
[4]	SETUP_PKT	RW	R	<p>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.</p> <p><b>Value After Reset:</b> 0x0</p>
[3]	FLUSH_FIFO	WS	R	<p>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</p>

**Table 14-30 Fields For Register: TXCSRL (`DEVCTL.HOST_MODE = 0x1`) (continued)**

Bits	Name	Software Access	USB Access	Description
				<p>with TXPKTRDY to abort the packet that is currently being loaded into the FIFO.</p> <p> <code>FLUSH_FIFO</code> should only be used when <code>TXPKTRDY</code> is set. At other times, it may cause data to be corrupted. Also note that, if the FIFO is double-buffered, <code>FLUSH_FIFO</code> may need to be set twice to completely clear the FIFO.</p> <p><b>Value After Reset:</b> 0x0</p>
[2]	ERROR	R/WC	WS	<p>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, <code>TXPKTRDY</code> 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.</p> <p><b>Value After Reset:</b> 0x0</p>
[1]	FIFO_NOT_EMPTY	R/WC	WS	<p>The USB sets this bit when there is at least 1 packet in the TX FIFO.</p> <p><b>Value After Reset:</b> 0x0</p>
[0]	TXPKTRDY	R/WS	WC	<p>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). <code>TXPKTRDY</code> is also automatically cleared prior to loading a second packet into a double-buffered FIFO.</p> <p><b>Value After Reset:</b> 0x0</p>

#### 14.1.2.3.15. 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	<p>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 <code>CSR0L.TXPKTRDY/CSR0L.RXPKTRDY</code> bit (below) is cleared.</p> <p> <b>FLUSH_FIFO</b> should only be used when <code>TXPKTRDY/RXPKTRDY</code> is set. At other times, it may cause data to be corrupted.</p> <p><b>Value After Reset:</b> 0x0</p>

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	<p>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).</p> <p><b>Value After Reset:</b> 0x0</p>
[2]	DATA_TOGGLE_WRITE_ENABLE	WS	R	<p>The software writes a 1 to this bit to enable the current state of the Endpoint 0 data toggle to be written (see <code>DATA_TOGGLE</code> bit, below). This bit is automatically cleared once the new value is written.</p> <p><b>Value After Reset:</b> 0x0</p>
[1]	DATA_TOGGLE	R/W	R/W	<p>When read, this bit indicates the current state of the Endpoint 0 data toggle. If D10 is high, this bit may be written with the required setting of the data toggle. If D10 is low, any value written to this bit is ignored.</p> <p><b>Value After Reset:</b> 0x0</p>
[0]	FLUSH_FIFO	WS	R	<p>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 <code>CSR0L.TXPKTRDY/CSR0L.RXPKTRDY</code> bit (below) is cleared.</p> <p> <b>FLUSH_FIFO</b> should only be used when <code>TXPKTRDY/RXPKTRDY</code> is set. At</p>

**Table 14-32 Fields For Register: CSR0H (DEVCTL.HOST\_MODE=0x1) (continued)**

Bits	Name	Software Access	USB Access	Description
				 other times, it may cause data to be corrupted. <b>Value After Reset:</b> 0x0

#### 14.1.2.3.16. 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	<p>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.</p> <p> Should not be set for either high-bandwidth Isochronous endpoints or high-bandwidth Interrupt endpoints.</p> <p><b>Value After Reset:</b> 0x0</p>
[6]	ISO	R/W	R	<p>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.</p> <p> This bit only has any effect in Peripheral mode. In Host mode, it always returns zero.</p> <p><b>Value After Reset:</b> 0x0</p>
[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. <b>Value After Reset:</b> 0x0
[4]	DMAREQENAB	R/W	R	The software sets this bit to enable the DMA request for the TX endpoint. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 0x0
[2]	DMAREQMODE	R/W	R	Valid values: <b>1:</b> DMA Request Mode 1. <b>0:</b> DMA Request Mode 0.  This bit must not be cleared either before or in the same cycle as the above DMAREQENAB bit is cleared. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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	<p>The software sets this bit to enable the endpoint direction as TX, and clears it to enable the endpoint direction as Rx.</p> <p> This bit only has any effect where the same endpoint FIFO is used for both TX and Rx transactions.</p> <p><b>Value After Reset:</b> 0x0</p>
[4]	DMAREQENAB	R/W	R	<p>The software sets this bit to enable the DMA request for the TX endpoint.</p> <p><b>Value After Reset:</b> 0x0</p>
[3]	FRCDATATOG	R/W	R	<p>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.</p> <p><b>Value After Reset:</b> 0x0</p>
[2]	DMAREQMODE	R/W	R	<p>Valid values:</p> <ul style="list-style-type: none"> <li><b>1:</b> DMA Request Mode 1.</li> <li><b>0:</b> DMA Request Mode 0.</li> </ul> <p> This bit must not be cleared either before or in the same cycle as the above DMAREQENAB bit is cleared.</p> <p><b>Value After Reset:</b> 0x0</p>
[1]	DATA_TOGGLE_WRITE_ENABLE	WS	R	<p>Valid values:</p> <ul style="list-style-type: none"> <li><b>1:</b> 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).</li> <li><b>0:</b> This bit is automatically cleared once the new value is written.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[0]	DATA_TOGGLE	R/W	R/W	<p>When read, this bit indicates the current state of the TX Endpoint data toggle.</p> <p>If D1 is high, this bit may be written with the required setting of the data toggle.</p> <p>If D1 is low, any value written to this bit is ignored.</p> <p><b>Value After Reset:</b> 0x0</p>

#### 14.1.2.3.17. 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, D15 – D13 is not implemented and D12 – D11 (if included) is 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]	MULTIPLIER_VAL	RW	R	<p>This bit field defines the value of the multiplier m which is equal to one more than the value recorded..</p> <p><b>Value After Reset:</b> 0x0</p>
[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
				<p>The value set can be up to 1024 bytes but is subject to the constraints placed by the <b>Universal Serial Bus Specification, Revision 2.0</b> on packet sizes for Bulk, Interrupt and Isochronous transfers in Full-speed and High-speed operations.</p> <p><b>Value After Reset:</b> 0x0</p>

**14.1.2.3.18. 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	<p>The software writes a 1 to this bit to reset the endpoint data toggle to 0.</p> <p><b>Value After Reset:</b> 0x0</p>
[6]	SENTSTALL	R/WC	WS	<p>This bit is set when a STALL handshake is transmitted. The software should clear this bit.</p> <p><b>Value After Reset:</b> 0x0</p>
[5]	SENDSTALL	R/W	R	<p>The software writes a 1 to this bit to issue a STALL handshake. The software clears this bit to terminate the stall condition.</p> <p> This bit has no effect where the endpoint is being used for Isochronous transfers.</p> <p><b>Value After Reset:</b> 0x0</p>
[4]	FLUSHFIFO	WS	R	<p>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.</p> <p> 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,</p>

**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. <b>Value After Reset:</b> 0x0
[3]	DATAERROR	R	WS	<p>This bit is set when <code>RXPKTRDY</code> is set if the data packet has a CRC or bit-stuff error. It is cleared when <code>RXPKTRDY</code> is cleared.</p>  This bit is only valid when the endpoint is operating in ISO mode. In Bulk mode, it always returns zero. <b>Value After Reset:</b> 0x0
[2]	OVERRUN	R/WC	WS	<p>This bit is set if an OUT packet cannot be loaded into the Rx FIFO. The software should clear this bit.</p>  This bit is only valid when the endpoint is operating in ISO mode. In Bulk mode, it always returns zero. <b>Value After Reset:</b> 0x0
[1]	FIFOFULL	R	WS	<p>This bit is set when no more packets can be loaded into the Rx FIFO.</p> <b>Value After Reset:</b> 0x0
[0]	RXPKTRDY	R/WC	WS	<p>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.</p> <b>Value After Reset:</b> 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	<p>The software writes a 1 to this bit to reset the endpoint data toggle to 0.</p> <b>Value After Reset:</b> 0x0
[6]	RXSTALL	R/WC	WS	<p>When a STALL handshake is received, this bit is set and an interrupt is generated. The software should clear this bit.</p> <b>Value After Reset:</b> 0x0
[5]	REQPKT	R/W	R/W	<p>The software writes a 1 to this bit to request an IN transaction. It is cleared when <code>RXPKTRDY</code> is set.</p> <b>Value After Reset:</b> 0x0

**Table 14-37 Fields For Register: RXCSRL ([DEVCTL](#).HOST\_MODE= 0x1) (continued)**

Bits	Name	Software Access	USB Access	Description
[4]	FLUSHFIFO	WS	R	<p>The software writes a 1 to this bit to flush the next packet to be read from the endpoint Rx FIFO.</p> <p>The FIFO pointer is reset and the <a href="#">RXPKTRDY</a> bit (below) is cleared.</p> <p> <a href="#">FLUSHFIFO</a> should only be used when <a href="#">RXPKTRDY</a> is set. At other times, it may cause data to be corrupted. Also note that, if the FIFO is double-buffered, <a href="#">FLUSHFIFO</a> may need to be set twice to completely clear the FIFO.</p> <p><b>Value After Reset:</b> 0x0</p>
[3]	DATAERROR_OR_NAK_TIMEOUT	R/WC	WS	<p>When operating in ISO mode, this bit is set when <a href="#">RXPKTRDY</a> is set if the data packet has a CRC or bit-stuff error and cleared when <a href="#">RXPKTRDY</a> 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 <a href="#">RXINTERVAL</a>. The software should clear this bit to allow the endpoint to continue.</p> <p><b>Value After Reset:</b> 0x0</p>
[2]	ERROR	R/WC	WS	<p>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.</p> <p> This bit is only valid when the Rx endpoint is operating in Bulk or Interrupt mode. In ISO mode, it always returns zero.</p> <p><b>Value After Reset:</b> 0x0</p>
[1]	FIFOFULL	R	WS	<p>This bit is set when no more packets can be loaded into the Rx FIFO.</p> <p><b>Value After Reset:</b> 0x0</p>
[0]	RXPKTRDY	R/WC	WS	<p>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.</p> <p><b>Value After Reset:</b> 0x0</p>

#### 14.1.2.3.19. RX Endpoint #n Control and Status Register 1 (RXCSR1)

The [RXCSR1](#) 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	<p>If the software sets this bit then the <code>RXPKTRDY</code> bit will be automatically cleared when a packet of <code>RXMAXP</code> bytes has been unloaded from the Rx FIFO. When packets of less than the maximum packet size are unloaded, <code>RXPKTRDY</code> will have to be cleared manually.</p> <p>When using a DMA to unload the RxFIFO, data is read from the RxFIFO in 4 byte chunks regardless of the <code>RXMAXP</code>. Therefore, the <code>RXPKTRDY</code> bit will be cleared as follows:</p> <p style="text-align: center;"><b>Table 14-39</b></p> <table border="1"> <thead> <tr> <th>Remainder (<code>RXMAXP/4</code>)</th> <th>Actual Bytes Read</th> <th>Packet Sizes that will clear <code>RXPKTRDY</code></th> </tr> </thead> <tbody> <tr> <td>0 (i.e. <code>RXMAXP = 64 bytes</code>)</td> <td><code>RXMAXP</code></td> <td><code>RXMAXP, RXMAXP-1, RXMAXP-2, RXMAXP-3</code></td> </tr> <tr> <td>3 (i.e. <code>RXMAXP = 63 bytes</code>)</td> <td><code>RXMAXP+1</code></td> <td><code>RXMAXP, RXMAXP-1, RXMAXP-2</code></td> </tr> <tr> <td>2 (i.e. <code>RXMAXP = 62 bytes</code>)</td> <td><code>RXMAXP+2</code></td> <td><code>RXMAXP, RXMAXP-1</code></td> </tr> <tr> <td>1 (i.e. <code>RXMAXP = 61 bytes</code>)</td> <td><code>RXMAXP+3</code></td> <td><code>RXMAXP</code></td> </tr> </tbody> </table> <p> Should not be set for high-bandwidth Isochronous endpoints.</p> <p><b>Value After Reset: 0x0</b></p>	Remainder ( <code>RXMAXP/4</code> )	Actual Bytes Read	Packet Sizes that will clear <code>RXPKTRDY</code>	0 (i.e. <code>RXMAXP = 64 bytes</code> )	<code>RXMAXP</code>	<code>RXMAXP, RXMAXP-1, RXMAXP-2, RXMAXP-3</code>	3 (i.e. <code>RXMAXP = 63 bytes</code> )	<code>RXMAXP+1</code>	<code>RXMAXP, RXMAXP-1, RXMAXP-2</code>	2 (i.e. <code>RXMAXP = 62 bytes</code> )	<code>RXMAXP+2</code>	<code>RXMAXP, RXMAXP-1</code>	1 (i.e. <code>RXMAXP = 61 bytes</code> )	<code>RXMAXP+3</code>	<code>RXMAXP</code>
Remainder ( <code>RXMAXP/4</code> )	Actual Bytes Read	Packet Sizes that will clear <code>RXPKTRDY</code>																	
0 (i.e. <code>RXMAXP = 64 bytes</code> )	<code>RXMAXP</code>	<code>RXMAXP, RXMAXP-1, RXMAXP-2, RXMAXP-3</code>																	
3 (i.e. <code>RXMAXP = 63 bytes</code> )	<code>RXMAXP+1</code>	<code>RXMAXP, RXMAXP-1, RXMAXP-2</code>																	
2 (i.e. <code>RXMAXP = 62 bytes</code> )	<code>RXMAXP+2</code>	<code>RXMAXP, RXMAXP-1</code>																	
1 (i.e. <code>RXMAXP = 61 bytes</code> )	<code>RXMAXP+3</code>	<code>RXMAXP</code>																	

**Table 14-38 Fields For Register: RXCSRH (DEVCTL.HOST\_MODE= 0x0) (continued)**

Bits	Name	Software Access	USB Access	Description
[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. <b>Value After Reset:</b> 0x0
[5]	DMAREQENAB	R/W	R	The software sets this bit to enable the DMA request for the Rx endpoint. <b>Value After Reset:</b> 0x0
[4]	DISNYET_OR_PID_ERROR	*Varies	*Varies	Bulk/Interrupt Transactions ( <u>DISNYET</u> ): 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.  <span style="color: yellow;">[Note]</span> 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: <ul style="list-style-type: none"><li>• Software Access: R/W</li><li>• USB Access: R</li></ul> ISO Transactions ( <u>PID_ERROR</u> ): The core sets this bit to indicate a PID error in the received packet. The access attributes of this field are as follows: <ul style="list-style-type: none"><li>• Software Access: R</li><li>• USB Access: R/W</li></ul> <b>Value After Reset:</b> 0x0
[3]	DMAREQMODE	R/W	R	Valid values: <b>1:</b> DMA Request Mode 1. <b>0:</b> DMA Request Mode 0. <b>Value After Reset:</b> 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 <u>RXPKTRDY</u> is cleared.  <span style="color: yellow;">[Note]</span> In anything other than Isochronous transfer, this bit will always return 0.  <b>Value After Reset:</b> 0x0

The following table shows the register bit assignments for USB Controller in **Host Mode**.

**Table 14-40 Fields For Register: RXCSRH (DEVCTL.HOST\_MODE= 0x1)**

Bits	Name	Software Access	USB Access	Description															
[7]	AUTOCLEAR	R/W	R	<p>If the software sets this bit then the <code>RXPKTRDY</code> bit will be automatically cleared when a packet of <code>RXMAXP</code> bytes has been unloaded from the Rx FIFO. When packets of less than the maximum packet size are unloaded, <code>RXPKTRDY</code> will have to be cleared manually.</p> <p>When using a DMA to unload the Rx FIFO, data is read from the Rx FIFO in 4 byte chunks regardless of the <code>RXMAXP</code>. Therefore, the <code>RXPKTRDY</code> bit will be cleared as follows:</p> <p style="text-align: center;"><b>Table 14-41</b></p> <table border="1"> <thead> <tr> <th>Remainder (<code>RXMAXP/4</code>)</th><th>Actual Bytes Read</th><th>Packet Sizes that will clear <code>RXPKTRDY</code></th></tr> </thead> <tbody> <tr> <td>0 (i.e. <code>RXMAXP</code> = 64 bytes)</td><td><code>RXMAXP</code></td><td><code>RXMAXP</code>, <code>RXMAXP-1</code>, <code>RXMAXP-2</code>, <code>RXMAXP-3</code></td></tr> <tr> <td>3 (i.e. <code>RXMAXP</code> = 63 bytes)</td><td><code>RXMAXP+1</code></td><td><code>RXMAXP</code>, <code>RXMAXP-1</code>, <code>RXMAXP-2</code></td></tr> <tr> <td>2 (i.e. <code>RXMAXP</code> = 62 bytes)</td><td><code>RXMAXP+2</code></td><td><code>RXMAXP</code>, <code>RXMAXP-1</code></td></tr> <tr> <td>1 (i.e. <code>RXMAXP</code> = 61 bytes)</td><td><code>RXMAXP+3</code></td><td><code>RXMAXP</code></td></tr> </tbody> </table> <p> Should not be set for high-bandwidth Isochronous endpoints.</p> <p><b>Value After Reset:</b> 0x0</p>	Remainder ( <code>RXMAXP/4</code> )	Actual Bytes Read	Packet Sizes that will clear <code>RXPKTRDY</code>	0 (i.e. <code>RXMAXP</code> = 64 bytes)	<code>RXMAXP</code>	<code>RXMAXP</code> , <code>RXMAXP-1</code> , <code>RXMAXP-2</code> , <code>RXMAXP-3</code>	3 (i.e. <code>RXMAXP</code> = 63 bytes)	<code>RXMAXP+1</code>	<code>RXMAXP</code> , <code>RXMAXP-1</code> , <code>RXMAXP-2</code>	2 (i.e. <code>RXMAXP</code> = 62 bytes)	<code>RXMAXP+2</code>	<code>RXMAXP</code> , <code>RXMAXP-1</code>	1 (i.e. <code>RXMAXP</code> = 61 bytes)	<code>RXMAXP+3</code>	<code>RXMAXP</code>
Remainder ( <code>RXMAXP/4</code> )	Actual Bytes Read	Packet Sizes that will clear <code>RXPKTRDY</code>																	
0 (i.e. <code>RXMAXP</code> = 64 bytes)	<code>RXMAXP</code>	<code>RXMAXP</code> , <code>RXMAXP-1</code> , <code>RXMAXP-2</code> , <code>RXMAXP-3</code>																	
3 (i.e. <code>RXMAXP</code> = 63 bytes)	<code>RXMAXP+1</code>	<code>RXMAXP</code> , <code>RXMAXP-1</code> , <code>RXMAXP-2</code>																	
2 (i.e. <code>RXMAXP</code> = 62 bytes)	<code>RXMAXP+2</code>	<code>RXMAXP</code> , <code>RXMAXP-1</code>																	
1 (i.e. <code>RXMAXP</code> = 61 bytes)	<code>RXMAXP+3</code>	<code>RXMAXP</code>																	
[6]	AUTOREQ	R/W	R	<p>If the software sets this bit, the <code>REQPKT</code> bit will be automatically set when the <code>RXPKTRDY</code> bit is cleared.</p> <p> This bit is automatically cleared when a short packet is received.</p> <p><b>Value After Reset:</b> 0x0</p>															
[5]	DMAREQENAB	R/W	R	The software sets this bit to enable the DMA request for the Rx endpoint.															

**Table 14-40 Fields For Register: RXCSRH (DEVCTL.HOST\_MODE= 0x1) (continued)**

Bits	Name	Software Access	USB Access	Description
				<b>Value After Reset:</b> 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.  <b>Value After Reset:</b> 0x0
[3]	DMAREQMODE	R/W	R	Valid values:  <b>1:</b> DMA Request Mode 1. <b>0:</b> DMA Request Mode 0.  <b>Value After Reset:</b> 0x0
[2]	DATA_TOGGLE_WRITE_ENABLE	R	R	Valid values:  <b>1:</b> 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). <b>0:</b> This bit is automatically cleared once the new value is written.  <b>Value After Reset:</b> 0x0
[1]	DATA_TOGGLE	R	R	When read, this bit indicates the current state of the Endpoint 0 data toggle. If D10 is high, this bit may be written with the required setting of the data toggle. If D10 is low, any value written to this bit is ignored.  <b>Value After Reset:</b> 0x0
[0]	INCOMPTRX	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.   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).  <b>Value After Reset:</b> 0x0

#### 14.1.2.3.20. 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 14-42 Fields For Register: COUNT0**

Bits	Name	Software Access	USB Access	Description
[6:0]	COUNT	R	W	<p>Number of received data bytes in the Endpoint 0 FIFO.</p> <p>The value returned changes as the contents of the FIFO change and is only valid while <a href="#">CSR0L.RXPKTRDY</a> is set.</p> <p><b>Value After Reset:</b> 0x0</p>

#### 14.1.2.3.21. 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 14-43 Fields For Register: RXCOUNT**

Bits	Name	Software Access	USB Access	Description
[13:0]	COUNT	R	W	<p>Number of data bytes in the packet currently in line to be read from the Rx FIFO.</p> <p>If the packet was transmitted as multiple bulk packets, the number given will be for the combined packet.</p> <p> The value returned changes as the FIFO is unloaded and is only valid while <a href="#">RXCSRL.RXPKTRDY</a> is set.</p> <p><b>Value After Reset:</b> 0x0</p>

#### 14.1.2.3.22. 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 14-44 Fields For Register: TXTYPE**

Bits	Name	Software Access	USB Access	Description
[7:6]	SPEED	R/W	R	<p>Operating speed of the target device when the core is configured with the multipoint option.</p> <p>Valid values:</p> <ul style="list-style-type: none"> <li><b>00:</b> Unused</li> <li> If selected, the target will be assumed to be using the same connection speed as the core.</li> <li><b>01:</b> High</li> <li><b>10:</b> Full</li> <li><b>11:</b> Low</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[5:4]	PROTOCOL	R/W	R	<p>The software should set this to select the required protocol for the TX endpoint.</p> <p>Valid values:</p> <ul style="list-style-type: none"> <li><b>00:</b> Control</li> <li><b>01:</b> Isochronous</li> <li><b>10:</b> Bulk</li> <li><b>11:</b> Interrupt</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[3:0]	TARGET_ENDPOINT_NUMBER	R/W	R	<p>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.</p> <p><b>Value After Reset:</b> 0x0</p>

#### 14.1.2.3.23. 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 14-45 Fields For Register: TYPE0**

Bits	Name	Software Access	USB Access	Description
[7:6]	SPEED	R/W	R	<p>Operating speed of the target device.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>00:</b> Unused</li> <li> If selected, the target will be assumed to be using the same connection speed as the core.</li> <li><b>01:</b> High</li> </ul>

**Table 14-45 Fields For Register: TYPE0 (continued)**

Bits	Name	Software Access	USB Access	Description
				<b>10:</b> Full <b>11:</b> Low <b>Value After Reset:</b> 0x0
[5:0]				Reserved

**14.1.2.3.24. 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 14-46 Fields For Register: NAKLIMIT0**

Bits	Name	Software Access	USB Access	Description
[4:0]	LIMIT_VAL	R/W	R	<b>Value After Reset:</b> 0x0

**14.1.2.3.25. 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 14-47 Fields For Register: TXINTERVAL**

Bits	Name	Software Access	USB Access	Description
[7:0]	POLLING_INTERVAL_OR_NAK_LIMIT	R/W	R	POLLING_INTERVAL:

**Table 14-47 Fields For Register: TXINTERVAL (continued)**

Bits	Name	Software Access	USB Access	Description
				<ul style="list-style-type: none"> <li>For Low Speed or Full Speed Interrupt transfers polling interval is <math>m</math> frames. Here <math>m</math> corresponds to POLLING_INTERVAL field value and can be set from 1 to 255</li> <li>For High Speed Interrupt transfers polling interval is <math>2^{(m-1)}</math> microframes. Here <math>m</math> corresponds to POLLING_INTERVAL field value and can be set from 1 to 16.</li> <li>For Full Speed or High Speed Isochronous transfers polling interval is <math>2^{(m-1)}</math> frames/microframes. Here <math>m</math> corresponds to POLLING_INTERVAL field value and can be set from 1 to 16.</li> </ul> <p><b>NAK_LIMIT:</b></p> <ul style="list-style-type: none"> <li>For Full Speed or High Speed Bulk transfers NAK Limit is <math>2^{(m-1)}</math> frames/microframes. Here <math>m</math> corresponds to POLLING_INTERVAL field value and can be set from 2 to 16.</li> </ul> <p> A value of 0 or 1 disables the NAK timeout function.</p> <p><b>Value After Reset:</b> 0x0</p>

#### 14.1.2.3.26. 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 14-48 Fields For Register: RXTYPE**

Bits	Name	Software Access	USB Access	Description
[7:6]	SPEED	R/W	R	<p>Operating speed of the target device when the core is configured with the multipoint option.</p> <p>Valid values:</p> <ul style="list-style-type: none"> <li><b>00:</b> Unused</li> <li> If selected, the target will be assumed to be using the same connection speed as the core.</li> </ul> <p><b>01:</b> High</p> <p><b>10:</b> Full</p> <p><b>11:</b> Low</p> <p><b>Value After Reset:</b> 0x0</p>
[5:4]	PROTOCOL	R/W	R	<p>The software should set this to select the required protocol for the RX endpoint.</p> <p>Valid values:</p> <ul style="list-style-type: none"> <li><b>00:</b> Control</li> <li><b>01:</b> Isochronous</li> <li><b>10:</b> Bulk</li> <li><b>11:</b> Interrupt</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[3:0]	TARGET_ENDPOINT_NUMBER	R/W	R	<p>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.</p> <p><b>Value After Reset:</b> 0x0</p>

#### 14.1.2.3.27. 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 14-49 Fields For Register: RXINTERVAL**

Bits	Name	Software Access	USB Access	Description
[7:0]	POLLING_INTERVAL_OR_NAK_LIMIT	R/W	R	<code>POLLING_INTERVAL:</code>

**Table 14-49 Fields For Register: RXINTERVAL (continued)**

Bits	Name	Software Access	USB Access	Description
				<ul style="list-style-type: none"> <li>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</li> <li>For High Speed Interrupt transfers polling interval is <math>2^{(m-1)}</math> microframes. Here m corresponds to POLLING_INTERVAL field value and can be set from 1 to 16.</li> <li>For Full Speed or High Speed Isochronous transfers polling interval is <math>2^{(m-1)}</math> frames/microframes. Here m corresponds to POLLING_INTERVAL field value and can be set from 1 to 16.</li> </ul> <p><b>NAK_LIMIT:</b></p> <ul style="list-style-type: none"> <li>For Full Speed or High Speed Bulk transfers NAK Limit is <math>2^{(m-1)}</math> frames/microframes. Here m corresponds to POLLING_INTERVAL field value and can be set from 2 to 16.</li> </ul> <p> A value of 0 or 1 disables the NAK timeout function.</p> <p><b>Value After Reset:</b> 0x0</p>

#### 14.1.2.3.28. 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 14-50 Fields For Register: CONFIGDATA**

Bits	Name	Software Access	USB Access	Description
[7]	MPRXE	R	R	<p>When set to '1', automatic amalgamation of bulk packets is selected (see <a href="#">Bulk Transactions</a> section).</p> <p><b>Value After Reset:</b> 0x1</p>
[6]	MPTXE	R	R	<p>When set to '1', automatic splitting of bulk packets is selected (see <a href="#">Bulk Transactions</a> section).</p> <p><b>Value After Reset:</b> 0x1</p>

**Table 14-50 Fields For Register: CONFIGDATA (continued)**

Bits	Name	Software Access	USB Access	Description
[5]	BIGENDIAN	R	R	Always “0”. Indicates Little Endian ordering. <b>Value After Reset:</b> 0x0
[4]	HBRXE	R	R	When set to ‘1’ indicates High-bandwidth Rx ISO Endpoint Support selected. <b>Value After Reset:</b> 0x1
[3]	HBTXE	R	R	When set to ‘1’ indicates High-bandwidth TX ISO Endpoint Support selected. <b>Value After Reset:</b> 0x1
[2]	DYNFIFO_SIZING	R	R	When set to ‘1’ indicates Dynamic FIFO Sizing option selected. <b>Value After Reset:</b> 0x1
[1]	SOFTCONE	R	R	Always ‘1’ . Indicates Soft Connect/Disconnect. <b>Value After Reset:</b> 0x1
[0]				Reserved

#### 14.1.2.3.29. 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 TxFIFO for the corresponding endpoint. Reading from these addresses unloads data from the RxFIFO 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 14-51 Fields For Register: FIFOn**

Bits	Name	Software Access	USB Access	Description
[31:0]	DATA	R/W	R/W	FIFOn data. <b>Value After Reset:</b> 0x0

#### 14.1.2.3.30. 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 14-52 Fields For Register: DEVCTL**

Bits	Name	Software Access	USB Access	Description
[7]	B_DEVICE	R	R/W	<p>This bit indicates whether the USB 2.0 Controller is operating as the ‘A’ device or the ‘B’ device. Valid values:</p> <ul style="list-style-type: none"> <li><b>0:</b> ‘A’ device</li> <li><b>1:</b> ‘B’ device</li> </ul> <p>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. <b>Value After Reset:</b> 0x1</p>
[6]	FSDEV	R	R/W	<p>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 full-speed by checking for high-speed chirps when the device is reset). Only valid in Host mode. <b>Value After Reset:</b> 0x0</p>
[5]	LSDEV	R	R/W	<p>This bit is set when a low-speed device has been detected being connected to the port. Only valid in Host mode. <b>Value After Reset:</b> 0x0</p>
[4:3]	VBUS	R	R/W	<p>These bits encode the current VBus level as follows:</p> <ul style="list-style-type: none"> <li><b>00:</b> Below <code>sessend</code></li> <li><b>01:</b> Above <code>sessend</code>, below <code>avalid</code></li> <li><b>10:</b> Above <code>avalid</code>, below <code>vbusvalid</code></li> <li><b>11:</b> Above <code>vbusvalid</code></li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[2]	HOST_MODE	R	R/W	<p>This bit is set when the USB 2.0 Controller is acting as a Host. <b>Value After Reset:</b> 0x0</p>
[1]	HOST_REQ	R/W	R/WC	<p>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 <a href="#">Host Negotiation</a> section (‘B’ device only). <b>Value After Reset:</b> 0x0</p>

**Table 14-52 Fields For Register: DEVCTL (continued)**

Bits	Name	Software Access	USB Access	Description
[0]	SESSION	R/W	R/W	<p>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.</p> <p> Clearing this bit when the core is not suspended will result in undefined behavior.</p> <p><b>Value After Reset:</b> 0x0</p>

#### 14.1.2.3.31. 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 14-53 Fields For Register: TXFIFOSZ**

Bits	Name	Software Access	USB Access	Description
[4]	DPB	R/W	R	<p>Defines whether double-packet buffering supported. Valid values:  <b>1:</b> Double-packet buffering.  <b>0:</b> Only single-packet buffering.</p> <p><b>Value After Reset:</b> 0x0</p>
[3:0]	SZ	R/W	R	<p>Maximum packet size to be allowed for (before any splitting within the FIFO of Bulk/High-Bandwidth packets prior to transmission). Valid values:</p> <ul style="list-style-type: none"> <li><b>0000:</b> 8 bytes</li> <li><b>0001:</b> 16 bytes</li> <li><b>0010:</b> 32 bytes</li> <li><b>0011:</b> 64 bytes</li> <li><b>0100:</b> 128 bytes</li> <li><b>0101:</b> 256 bytes</li> <li><b>0110:</b> 512 bytes</li> <li><b>0111:</b> 1024 bytes</li> <li><b>1000:</b> 2048 bytes</li> <li><b>1001:</b> 4096 bytes</li> </ul>

**Table 14-53 Fields For Register: TXFIFOSZ (continued)**

Bits	Name	Software Access	USB Access	Description
				 <ul style="list-style-type: none"> <li>• If <b>DPB</b> = 0, the FIFO will also be this size;</li> <li>• If <b>DPB</b> = 1, the FIFO will be twice this size.</li> </ul> <b>Value After Reset:</b> 0x0

#### 14.1.2.3.32. 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 14-54 Fields For Register: RXFIFOSZ**

Bits	Name	Software Access	USB Access	Description
[4]	DPB	R/W	R	Defines whether double-packet buffering supported. Valid values: <b>1</b> : Double-packet buffering. <b>0</b> : Only single-packet buffering. <b>Value After Reset:</b> 0x0
[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: <b>0000</b> : 8 bytes <b>0001</b> : 16 bytes <b>0010</b> : 32 bytes <b>0011</b> : 64 bytes <b>0100</b> : 128 bytes <b>0101</b> : 256 bytes <b>0110</b> : 512 bytes <b>0111</b> : 1024 bytes <b>1000</b> : 2048 bytes <b>1001</b> : 4096 bytes  <ul style="list-style-type: none"> <li>• If <b>DPB</b> = 0, the FIFO will also be this size;</li> <li>• If <b>DPB</b> = 1, the FIFO will be twice this size.</li> </ul> <b>Value After Reset:</b> 0x0

#### 14.1.2.3.33. 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 14-55 Fields For Register: TXFIFOAD**

Bits	Name	Software Access	USB Access	Description
[13]				Reserved
[12:0]	AD	R/W	R	<p>Start address of the endpoint FIFO in units of 8 bytes.            Valid values:</p> <ul style="list-style-type: none"> <li><b>0x0000:</b> 0x0000</li> <li><b>0x0001:</b> 0x0008</li> <li><b>0x0002:</b> 0x0010</li> <li>...</li> <li><b>0x1FFF:</b> 0xFFFF</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

#### 14.1.2.3.34. 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 14-56 Fields For Register: RXFIFOAD**

Bits	Name	Software Access	USB Access	Description
[13]				Reserved
[12:0]	AD	R/W	R	<p>Start address of the endpoint FIFO in units of 8 bytes.            Valid values:</p> <ul style="list-style-type: none"> <li><b>0x0000:</b> 0x0000</li> <li><b>0x0001:</b> 0x0008</li> <li><b>0x0002:</b> 0x0010</li> <li>...</li> <li><b>0x1FFF:</b> 0xFFFF</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

#### 14.1.2.3.35. 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 14-57 Fields For Register: vCONTROL**

Bits	Name	Software Access	USB Access	Description
[23]	OTG_SUSPENDM	W	N/A	<p>Controls <code>otg_suspendm</code> 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 1'b0. In OTG/HOST application this signal should be connected to 1'b1.</p> <p>Valid values:</p> <ul style="list-style-type: none"> <li><b>0:</b> VBUS detection output <code>vbus_valid</code>, <code>sessvalid</code>, <code>sessend</code>, <code>bvalid</code> will give inaccurate output, but good enough for VBUS negotiation in low power status.</li> <li><b>1:</b> VBUS detection output <code>vbus_valid</code>, <code>sessvalid</code>, <code>sessend</code>, <code>bvalid</code> will give accurate VBUS detection result.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[22]	FSLS_SERIALMODE	W	N/A	<p>Controls <code>fsls_serialmode</code> 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.</p> <p>Valid values:</p> <ul style="list-style-type: none"> <li><b>0:</b> FS packets are sent using the parallel interface</li> <li><b>1:</b> FS and LS packets are sent using the serial interface</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[21]	PLL_EN	W	N/A	<p>Controls <code>pll_en</code> signal.</p> <p><b>Value After Reset:</b> 0x0</p>
[20]	TX_SE0	W	N/A	<p>Controls <code>tx_se0</code> signal.</p> <p><b>Value After Reset:</b> 0x0</p>
[19]	TX_DAT	W	N/A	<p>Controls <code>tx_dat</code> signal.</p> <p><b>Value After Reset:</b> 0x0</p>
[18]	TX_ENABLE_N	W	N/A	<p>Controls <code>tx_enable_n</code> signal.</p> <p><b>Value After Reset:</b> 0x0</p>
[17]	REFCLK_MODE	W	N/A	<p>Controls <code>refclk_mode</code> signal. This signal is used for reference clock input selection.</p> <p>Valid values:</p> <ul style="list-style-type: none"> <li><b>0:</b> input clock is 25MHz</li> <li><b>1:</b> input clock is 12MHz</li> </ul> <p><b>!</b> It is recommended to use the 25MHz reference clock frequency.</p> <p><b>Value After Reset:</b> 0x0</p>

**Table 14-57 Fields For Register: vCONTROL (continued)**

Bits	Name	Software Access	USB Access	Description
[16]	CFG_RSTN	W	N/A	Controls <code>cfg_rstn</code> signal. <b>Active Level:</b> Low <b>Value After Reset:</b> 0x0
[15]	CFG_RDEN	W	N/A	Controls <code>cfg_rden</code> signal. <b>Active Level:</b> High <b>Value After Reset:</b> 0x0
[14]	CFG_WREN	W	N/A	Controls <code>cfg_wren</code> signal. <b>Active Level:</b> High <b>Value After Reset:</b> 0x0
[13:8]	CFG_REGADDR	W	N/A	Controls <code>cfg_regaddr[5:0]</code> signal. <b>Value After Reset:</b> 0x0
[7:0]	CFG_DATAWR	W	N/A	Controls <code>cfg_datawr[7:0]</code> signal. <b>Value After Reset:</b> 0x0

**14.1.2.3.36. 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:  
 $2Hc + Xc$  and  $3Hc + 6Xc$ ,  
where  $Hc$  is a cycle of `hclk` and  $Xc$  is a cycle of `clk60`.

The following table shows the register bit assignments.

**Table 14-58 Fields For Register: vSTATUS**

Bits	Name	Software Access	USB Access	Description
[7:0]	CFG_DATARD	R	N/A	Controls <code>cfg_datard[7:0]</code> signal. <b>Value After Reset:</b> 0x0

**14.1.2.3.37. 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 14-59 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. <b>Value After Reset:</b> 0x5
[3:0]	TXENDPOINTS	R	R	The number of Tx endpoints implemented in the design. <b>Value After Reset:</b> 0x5

**14.1.2.3.38. 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 14-60 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. <b>Value After Reset:</b> 0xB

**14.1.2.3.39. 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 14-61 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 533.3ns (the default setting corresponds to 2.667gs). <b>Value After Reset:</b> 0x5
[3:0]	WTID	R/W	R	Sets the delay to be applied from <code>idpullup</code> being asserted to <code>iddig</code> being considered valid in units of 4.369ms (the default setting corresponds to 52.43ms). <b>Value After Reset:</b> 0xC

**14.1.2.3.40. 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 14-62 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 $\mu$ s (the default setting corresponds to 32.77ms). <b>Value After Reset:</b> 0x3C

#### 14.1.2.3.41. 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 14-63 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 133.3ns (the default setting corresponds to 17.07 $\mu$ s). <b>Value After Reset:</b> 0x80

#### 14.1.2.3.42. 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 14-64 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 533.3ns (the default setting corresponds to 63.46 $\mu$ s). <b>Value After Reset:</b> 0x77

#### 14.1.2.3.43. 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 14-65 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 1.067µs (the default setting corresponds to 121.6µs). <b>Value After Reset:</b> 0x72

#### 14.1.2.3.44. 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 8-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 14-66 Fields For Register: TXFUNCADDR**

Bits	Name	Software Access	USB Access	Description
[6:0]	ADDR	R/W	R	Address of target function. <b>Value After Reset:</b> 0x0

#### 14.1.2.3.45. 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 14-67 Fields For Register: TXHUBADDR**

Bits	Name	Software Access	USB Access	Description
[7]	MULTIPLE_TRANSLATORS	R/W	R	Multiple transaction translators.

**Table 14-67 Fields For Register: TXHUBADDR (continued)**

Bits	Name	Software Access	USB Access	Description
				Valid values: <b>0:</b> Single transaction translator <b>1:</b> Multiple transaction translators <b>Value After Reset:</b> 0x0
[6:0]	HUB_ADDRESS	R/W	R	Address of USB 2.0 hub. <b>Value After Reset:</b> 0x0

**14.1.2.3.46. 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 14-68 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. <b>Value After Reset:</b> 0x0

**14.1.2.3.47. 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 14-69 Fields For Register: RXFUNCADDR**

Bits	Name	Software Access	USB Access	Description
[6:0]	ADDR	R/W	R	Address of target function. <b>Value After Reset:</b> 0x0

#### 14.1.2.3.48. RX Endpoint #n Hub Address (RXHUBADDR)

The **RXHUBADDR** register is at offset  $0x86 + (n * 0x8)$ , where  $n = 0 \dots 5$ .

**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 EP $n$  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 14-70 Fields For Register: RXHUBADDR**

Bits	Name	Software Access	USB Access	Description
[7]	MULTIPLE_TRANSLATORS	R/W	R	Multiple transaction translators. Valid values: <b>0:</b> Single transaction translator <b>1:</b> Multiple transaction translators <b>Value After Reset:</b> 0x0
[6:0]	HUB_ADDRESS	R/W	R	Address of USB 2.0 hub. <b>Value After Reset:</b> 0x0

#### 14.1.2.3.49. RX Endpoint #n Hub Port (RXHUBPORT)

The **RXHUBPORT** register is at offset  $0x87 + (n * 0x8)$ , where  $n = 0 \dots 5$ .

**Exists:** `DEVCTL.HOST_MODE` = 0x1

**RXHUBPORT** only need to be written where a full- or low-speed device is connected to Rx Endpoint EP $n$  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 14-71 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. <b>Value After Reset:</b> 0x0

#### 14.1.2.3.50. 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 14-72 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. <b>Value After Reset:</b> 0x0
[3]	DMA_CHANNEL_4_INTERRUPT	R/W	W	DMA Channel 4 Interrupt. <b>Value After Reset:</b> 0x0
[2]	DMA_CHANNEL_3_INTERRUPT	R/W	W	DMA Channel 3 Interrupt. <b>Value After Reset:</b> 0x0
[1]	DMA_CHANNEL_2_INTERRUPT	R/W	W	DMA Channel 2 Interrupt. <b>Value After Reset:</b> 0x0
[0]	DMA_CHANNEL_1_INTERRUPT	R/W	W	DMA Channel 1 Interrupt. <b>Value After Reset:</b> 0x0

#### 14.1.2.3.51. 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 14-73 Fields For Register: DMA\_CNTL**

Bits	Name	Software Access	USB Access	Description
[10:9]	DMA_BRSTM	R/W	R	Burst Mode.

**Table 14-73 Fields For Register: DMA\_CNTL (continued)**

Bits	Name	Software Access	USB Access	Description
				<p>Valid values:</p> <ul style="list-style-type: none"> <li><b>00:</b> Burst Mode 0: Bursts of unspecified length.</li> <li><b>01:</b> Burst Mode 1: INCR4 or unspecified length.</li> <li><b>10:</b> Burst Mode 2: INCR8, INCR4 or unspecified length.</li> <li><b>11:</b> Burst Mode 3: INCR16, INCR8, INCR4 or unspecified length.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[8]	DMA_ERR	R/W	R/W	<p>Bus Error Bit. Indicates that a bus error has been observed on the input <code>hresp[1:0]</code>. This bit is cleared by software.</p> <p><b>Value After Reset:</b> 0x0</p>
[7:4]	DMAEP	R/W	R	<p>The endpoint number this channel is assigned to. Valid values:</p> <ul style="list-style-type: none"> <li><b>0000:</b> Endpoint 0</li> <li><b>0001:</b> Endpoint 1</li> <li><b>0010:</b> Endpoint 2</li> <li><b>0011:</b> Endpoint 3</li> <li><b>0100:</b> Endpoint 4</li> <li><b>0101:</b> Endpoint 5</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[3]	DMAIE	R/W	R	<p>DMA Interrupt Enable. <b>Value After Reset:</b> 0x0</p>
[2]	DMAMODE	R/W	R	<p>This bit selects the DMA Transfer Mode. Valid values:</p> <ul style="list-style-type: none"> <li><b>0:</b> DMA Mode0 Transfer</li> <li><b>1:</b> DMA Mode1 Transfer</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[1]	DMA_DIR	R/W	R	<p>This bit selects the DMA Transfer Direction. Valid values:</p> <ul style="list-style-type: none"> <li><b>0:</b> DMA Write (RX Endpoint)</li> <li><b>1:</b> DMA Read (TX Endpoint)</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[0]	DMA_ENAB	R/W	R	<p>This bit enables the DMA transfer and will cause the transfer to begin. <b>Value After Reset:</b> 0x0</p>

#### 14.1.2.3.52. 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 transferred.

The following table shows the register bit assignments.

Table 14-74 Fields For Register: `DMA_ADDR`

Bits	Name	Software Access	USB Access	Description
[31:2]	<code>DMA_ADDR</code>	R/W	R/W	The DMA memory address. Note that the initial memory address written to this register must have a value such that its modulo 4 value is equal to 0. That is, <code>DMA_ADDR[1:0]</code> must be equal to 2'b00. The lower two bits of this register are read only and cannot be set by software.
[1:0]		R	R/W	<b>Value After Reset:</b> 0x0

#### 14.1.2.3.53. 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 transferred.

The following table shows the register bit assignments.

Table 14-75 Fields For Register: `DMA_COUNT`

Bits	Name	Software Access	USB Access	Description
[31:0]	<code>DMA_COUNT</code>	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.  <b>Value After Reset:</b> 0x0

#### 14.1.2.3.54. 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 14-76 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 <a href="#">RXCSRH.AUTOREQ</a> is set. Has no effect in Peripheral mode or when <a href="#">AUTOREQ</a> is not set. <b>Value After Reset:</b> 0x0

#### 14.1.2.3.55. 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 67ns time intervals before the timeout occurs.

The following table shows the register bit assignments.

**Table 14-77 Fields For Register: [C\\_T\\_UCH](#)**

Bits	Name	Software Access	USB Access	Description
[15:0]	<a href="#">C_T_UCH</a>	R/W	N/A	Configurable Chirp Timeout timer. <b>Value After Reset:</b> 0x101D0

#### 14.1.2.3.56. 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 67ns time intervals before the timeout occurs.

The following table shows the register bit assignments.

**Table 14-78 Fields For Register: [C\\_T\\_HSRTN](#)**

Bits	Name	Software Access	USB Access	Description
[15:0]	<a href="#">C_T_HSRTN</a>	R/W	N/A	The delay from the end of High Speed resume signaling to enabling normal operating mode. <b>Value After Reset:</b> 0x34BC

#### 14.1.2.3.57. 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 minimum high speed timeout period of 736 bit times. The timeout period can be increased in increments of 64 high speed bit times (133 ns). There are 16 possible values.

By default, the adder is 0 thus setting the high speed timeout to its minimum value. Use of this register will allow the high speed timeout to be set to values that are greater than the maximum specified in **Universal Serial Bus Specification, Revision 2.0** making the USB 2.0 Controller non-compliant.

The following table shows the register bit assignments.

**Table 14-79 Fields For Register: C\_T\_HSBT**

Bits	Name	Software Access	USB Access	Description
[3:0]	C_T_HSBT	R/W	R	<p>The value added to the minimum High Speed Timeout period (736 bit times) in increments of 64 High Speed bit times. This allows the turn around timeout period to be set to 16 possible values as follows:</p> <ul style="list-style-type: none"> <li><b>0:</b> 736 1.534</li> <li><b>1:</b> 800 1.667</li> <li><b>2:</b> 864 1.801</li> <li><b>3:</b> 928 1.934</li> <li><b>4:</b> 992 2.067</li> <li><b>5:</b> 1056 2.201</li> <li><b>6:</b> 1120 2.334</li> <li><b>7:</b> 1184 2.467</li> <li><b>8:</b> 1248 2.601</li> <li><b>9:</b> 1312 2.734</li> <li><b>10:</b> 1376 2.868</li> <li><b>11:</b> 1440 3.001</li> <li><b>12:</b> 1504 3.134</li> <li><b>13:</b> 1568 3.268</li> <li><b>14:</b> 1632 3.401</li> <li><b>15:</b> 1696 3.534</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

#### 14.1.2.3.58. 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. <b>Value After Reset:</b> 0x0
[11:9]				Reserved
[8]	RMTWAK	R	R/W	Remote wakeup enable bit. Valid values: <b>0:</b> Not enabled. <b>1:</b> 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. <b>Value After Reset:</b> 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: <i>Resume Time = 50us + HIRD*75us.</i> This results a range 50us to 1200us. <b>Value After Reset:</b> 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: <b>0000:</b> Reserved <b>0001:</b> Sleep State (L1) <b>0010...0011:</b> Reserved <b>Value After Reset:</b> 0x0

### 14.1.2.3.59. 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	<p>This bit is used to place all end points in a state such that the response to all transactions other than an LPM transaction will be a NAK.</p> <p>This bit will only take effect after the USB 2.0 Controller has been LPM suspended.</p> <p>In this case, the USB 2.0 Controller will continue to NAK until this bit has been cleared by software.</p> <p><b>Value After Reset:</b> 0x0</p>
[3:2]	LPMEN	R/W	R	<p>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.</p> <p>Valid values:</p> <ul style="list-style-type: none"> <li><b>00...10:</b> LPM and Extended transactions are not supported.</li> <li>In this case, the USB 2.0 Controller will not respond to LPM transactions and the transaction will timeout.</li> <li><b>01:</b> LPM is not supported but extended transactions are supported.</li> <li>In this case, the USB 2.0 Controller will respond to an LPM transaction with a STALL.</li> <li><b>11:</b> The USB 2.0 Controller supports LPM extended transactions.</li> <li>In this case, the USB 2.0 Controller will respond with a NYET or an ACK as determined by the value of <code>LPMXMT</code> field below and other conditions.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[1]	LPMRES	R/W	WC	<p>This bit is used by software to initiate resume (remote wakeup).</p> <p>This bit differs from the classic <code>POWER.RESUME</code> bit, where the <code>resume</code> signal timing is controlled by hardware.</p> <p>When software writes this bit, resume signaling will be asserted for 50us.</p>

**Table 14-81 Fields For Register: LPM\_CNTRL ([DEVCTL](#).HOST\_MODE = 0x0) (continued)**

Bits	Name	Software Access	USB Access	Description
				<p>This bit is self clearing.  <b>Value After Reset:</b> 0x0</p>
[0]	LPMXMT	R/W	WC	<p>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.  This bit is only effective if <a href="#">LPMEN</a> field above is set to 2'b11.  This bit can be set in the same cycle as <a href="#">LPMEN</a>. If this bit is set to 1'b1 and <a href="#">LPMEN</a> = 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.</p> <p><b>Value After Reset:</b> 0x0</p>

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	<p>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 <a href="#">LPM_ATTR.HIRD</a> field.  This bit is self clearing.  <b>Value After Reset:</b> 0x0</p>
[0]	LPMXMT	R/W	R/W	<p>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.  <b>Value After Reset:</b> 0x0</p>

#### 14.1.2.3.60. 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: <b>0:</b> Disable the <code>LPMERR</code> interrupt <b>1:</b> Enable the <code>LPMERR</code> interrupt <b>Value After Reset:</b> 0x0
[4]	LPMRESEN	R	R	Valid values: <b>0:</b> Disable the <code>LPMRES</code> interrupt <b>1:</b> Enable the <code>LPMRES</code> interrupt <b>Value After Reset:</b> 0x0
[3]	LPMNCEN	R/W	R	Valid values: <b>0:</b> Disable the <code>LPMNC</code> interrupt <b>1:</b> Enable the <code>LPMNC</code> interrupt <b>Value After Reset:</b> 0x0
[2]	LPMACKEN	R/W	R	Valid values: <b>0:</b> Disable the <code>LPMACK</code> interrupt <b>1:</b> Enable the <code>LPMACK</code> interrupt <b>Value After Reset:</b> 0x0
[1]	LPMNYEN	R/W	R	Valid values: <b>0:</b> Disable the <code>LPMNY</code> interrupt <b>1:</b> Enable the <code>LPMNY</code> interrupt <b>Value After Reset:</b> 0x0
[0]	LPMSTEN	R/W	R	Valid values: <b>0:</b> Disable the <code>LPMST</code> interrupt <b>1:</b> Enable the <code>LPMST</code> interrupt <b>Value After Reset:</b> 0x0

#### 14.1.2.3.61. 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
[5]	LPMERR	R	WS	<p>This bit is set if an LPM transaction is received that has a <b>LINKSTATE</b> field that is not supported. In this case, the response to the transaction will be a <b>STALL</b>. However, the <b>LPM_INTR</b> register will be updated so that software can observe the non compliant LPM packet payload.</p> <p><b>Value After Reset:</b> 0x0</p>
[4]	LPMRES	R	WS	<p>This bit is set if the USB 2.0 Controller has been resumed for any reason.</p> <p>This bit is mutually exclusive from the <b>POWER.RESUME</b> bit.</p> <p><b>Value After Reset:</b> 0x0</p>
[3]	LPMNC	R	WS	<p>This bit is set when an LPM transaction is received and the USB 2.0 Controller responds with a <b>NYET</b> due to data pending in the RX FIFOs.</p> <p><b>Value After Reset:</b> 0x0</p>
[2]	LPMACK	R	WS	<p>This bit is set when an LPM transaction is received and the USB 2.0 Controller responds with an <b>ACK</b>.</p> <p><b>Value After Reset:</b> 0x0</p>
[1]	LPMNY	R	WS	<p>This bit is set when an LPM transaction is received and the USB 2.0 Controller responds with a <b>NYET</b>.</p> <p><b>Value After Reset:</b> 0x0</p>
[0]	LPMST	R	WS	<p>This bit is set when an LPM transaction is received and the USB 2.0 Controller responds with a <b>STALL</b>.</p> <p><b>Value After Reset:</b> 0x0</p>

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	<p>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.</p> <p><b>Value After Reset:</b> 0x0</p>
[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 <code>POWER.RESUME</code> bit. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 0x0
[2]	LPMACK	R	WS	This bit is set when an LPM transaction is transmitted and the device responds with an ACK. <b>Value After Reset:</b> 0x0
[1]	LPMNY	R	WS	This bit is set when an LPM transaction is transmitted and the device responds with a NYET. <b>Value After Reset:</b> 0x0
[0]	LPMST	R	WS	This bit is set when an LPM transaction is transmitted and the device responds with a STALL. <b>Value After Reset:</b> 0x0

#### 14.1.2.3.62. 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.

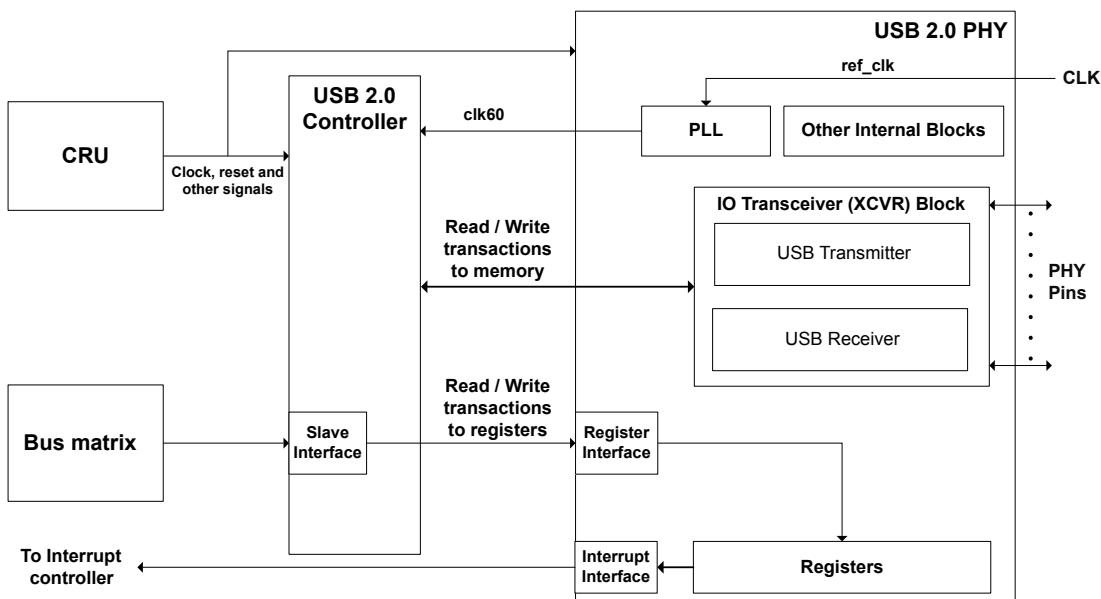
The following table shows the register bit assignments.

**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. <b>Value After Reset:</b> 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**

The USB 2.0 PHY has the following features:

- Supports HS(480Mbps)/FS(12Mbps) /LS(1.5Mbps) 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

In this section:

- [Registers 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 [Registers Map](#) and [Register Descriptions](#) sections below.

### 14.2.1.1. Registers Map

This section tables contain information about the USB 2.0 PHY registers memory map.

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

**Table 14-87 USB 2.0 PHY Registers Mp**

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	Access	Description
[7:5]	HS_RES_ADJ	R/W	<p>Fine tune the 45ohm termination resistor (HS).</p> <p><b>Valid values:</b></p> <ul style="list-style-type: none"> <li><b>000:</b> 40 ohm</li> <li><b>100:</b> 45 ohm (default)</li> <li><b>110:</b> 50 ohm</li> </ul> <p><b>Value After Reset:</b> 0x4</p>
[4:3]			Reserved
[2:0]	CLK_MODE	R/W	<p>Clock Gating Control Signal.</p> <p>CLK_MODE[2]: Clock Gating Control Signal for Tx.</p> <p><b>Valid values:</b></p> <ul style="list-style-type: none"> <li><b>1:</b> Normal power consumption mode</li> <li><b>0:</b> Low power consumption mode (default)</li> </ul> <p>CLK_MODE[1:0]: Clock Gating Control Signal for Rx.</p> <p><b>Valid values:</b></p> <ul style="list-style-type: none"> <li><b>1X:</b> Normal power consumption mode</li> <li><b>00:</b> Lower power consumption mode (default)</li> <li><b>01:</b> Lowest power consumption mode</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

#### 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	Access	Description
[7:4]	ADJ_PW_HS[3:0]	R/W	<p>Super power saving with reduced output swing mode control bits (for HS mode only).</p> <p><b>Valid values:</b></p> <ul style="list-style-type: none"> <li><b>0001:</b> 100</li> <li><b>0011:</b> 200</li> <li><b>0111:</b> 300</li> <li><b>1111:</b> 400 (default)</li> </ul> <p><b>Value After Reset:</b> 0xF</p>
[3:0]	ADJ_VREF_SQ[3:0]	R/W	<p>Squelch detection threshold voltage control bits.</p> <p><b>Valid values:</b></p> <ul style="list-style-type: none"> <li><b>0000:</b> 92mV</li> <li><b>1000:</b> 124mV (default)</li> <li><b>1111:</b> 152mV</li> </ul> <p><b>Value After Reset:</b> 0x8</p>

### 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	Access	Description
[7]			Reserved
[6:4]	ADJ_VSW_HS[2:0]	R/W	Output eye shape adjustment control bits. <b>Valid values:</b> <b>000:</b> 320mV <b>100:</b> 400mV (default) <b>111:</b> 460mV <b>Value After Reset:</b> 0x4
[3:0]	ADJ_IREF_RES[3:0]	R/W	Internal bias current adjustment control bits. <b>Valid values:</b> <b>0000:</b> 125uA <b>0111:</b> 100uA (default) <b>1111:</b> 78uA <b>Value After Reset:</b> 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	Access	Description
[7]			Reserved
[6:4]	ADJ_TXCLK_PHASE[2:0]	R/W	TX Clock phase adjust signal. <b>Value After Reset:</b> 0x0
[3:0]	ADJ_PLL[3:0]	R/W	PLL adjustment signal. <b>Value After Reset:</b> 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	Access	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	Access	Description
			<b>Value After Reset:</b> 0x0
[3]	PHY_MODE_SEL	R/W	PHY mode select. <b>Value After Reset:</b> 0x0
[2]	CLK_SEL	R/W	clk60 source select. <b>Value After Reset:</b> 0x0
[1]	COUNTER_SEL	R/W	SIE input sample enable. <b>Value After Reset:</b> 0x0
[0]	RXGAP_FIX_EN	R/W	RXGAP fix enable. <b>Value After Reset:</b> 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	Access	Description
[7]			Reserved
[6:4]	HS_TX2RX_DLY_CNT_SEL	R/W	PHY High-Speed bus turn-around time select. <b>Valid values:</b> <b>000:</b> 25*T <b>001:</b> 1*T <b>010:</b> 3*T <b>011:</b> 5*T <b>100:</b> 7*T (Default) <b>101:</b> 11*T <b>110:</b> 15*T <b>111:</b> 19*T   T - is the period of <code>clk_480</code> . <b>Value After Reset:</b> 0x4
[3]	LS_KPALV_EN	R/W	LS mode keep alive enable. <b>Value After Reset:</b> 0x0
[2]	PRE_HPHY_LSIE	R/W	dis_preamble enable. <b>Value After Reset:</b> 0x0
[1]	DET_FSEOP_EN	R/W	FS EOP detect enable. <b>Value After Reset:</b> 0x0
[0]	LS_PAR_EN	R/W	LS mode with parallel enable. <b>Value After Reset:</b> 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	Access	Description
[7:5]			Reserved
[4]	VREF_PD18_SEL	R/W	Inner <code>vref_pd18</code> control signal. <b>Valid values:</b> 0: <code>vref_pd18</code> is only controlled by <code>pll_en</code> 1: <code>vref_pd18</code> is controlled by <code>pll_en</code>   <code>suspendm</code> <b>Value After Reset:</b> 0x0
[3]	PLL_PD_SEL	R/W	Inner pd control signal. <b>Valid values:</b> 0: <code>pll_pd</code> is only controlled by <code>pll_en</code> 1: <code>pll_pd</code> is controlled by <code>pll_en</code>   <code>suspendm</code> <b>Value After Reset:</b> 0x0
[2]	CLK480_SEL	R/W	<code>clk_480</code> output time select. <b>Valid values:</b> 0: <code>clk_480</code> is valid after PLL is locked and add about 300us delay 1: <code>clk_480</code> is valid after PLL is locked <b>Value After Reset:</b> 0x0
[1:0]	CNT_NUM	R/W	3ms counter select. <b>Valid values:</b> 00: 392us (Default) 01: 682us 10: 1.36ms 11: 2.72ms <b>Value After Reset:</b> 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	Access	Description
[7:0]	RESERVED_OUT1[7:0]	R/W	<b>Value After Reset:</b> 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	Access	Description
[7:0]	RESERVED_OUT1[15:8]	R/W	<b>Value After Reset:</b> 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	Access	Description
[7:0]	RESERVED_OUT2[7:0]	R/W	<b>Value After Reset:</b> 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	Access	Description
[7:0]	RESERVED_OUT2[15:8]	R/W	<b>Value After Reset:</b> 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	Access	Description
[7:0]	RESERVED_IN1[7:0]	R	<b>Value After Reset:</b> 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	Access	Description
[7:0]	RESERVED_IN1[15:8]	R	<b>Value After Reset:</b> 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	Access	Description
[7:0]	RESERVED_IN2[7:0]	R	<b>Value After Reset:</b> 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	Access	Description
[7:0]	RESERVED_IN2[15:8]	R	<b>Value After Reset:</b> 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 accessible 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 `SOFTDBGGEN` bit of the `SYSCR0` CRU register. `SYSCR0`.`SOFTDBGGEN` 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`.`SOFTDBGGEN` 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 of the `SOFTDBGGEN` bit by software.



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

### 15.1. Software JTAG Register

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	4B	0x4000_A010	0x4000_A013



For details, refer to [Memory Map](#) chapter.

#### 15.1.1. Register Map

The following table provides 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	<a href="#">Software Debug</a>

#### 15.1.2. Register Description

In the tables below, the **R/W** column of a register description 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 <b>Value After Reset:</b> 0x0
[3]	TDI	R/W	Data that is propagated to the PA[3] I/O pin <b>Value After Reset:</b> 0x0
[2]	TDO	R	Data that is present at the PA[2] I/O pin <b>Value After Reset:</b> 0x0
[1]	TMS	R/W	Data that is propagated to the PA[1] I/O pin <b>Value After Reset:</b> 0x0
[0]	TCK	R/W	Data that is propagated to the PA[0] I/O pin <b>Value After Reset:</b> 0x0

## 16. Tracer

The BE-U1000 contains the Tracing System that is a part of Debug System. The Tracing System contains a number of components, and significant component between them is a Tracer Block, that shown in the diagram below.

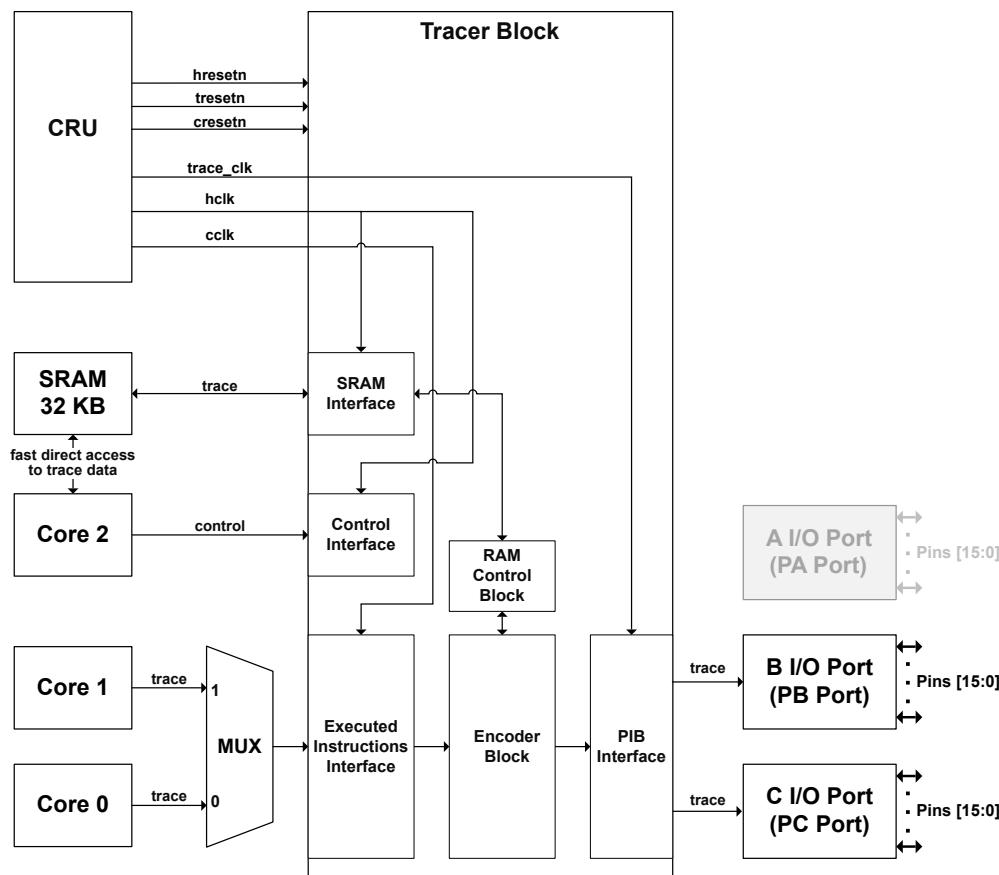


Figure 16-1 Tracer Block Diagram

From the BE-U1000 three cores - Core 0...2, - only Core 2 is responsible for support of Tracing System interfaces and Core 0/1 operates in a trace collecting process as a sources of trace.

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

In this section:

- [Registers Maps](#)
- [Register Descriptions](#)

Registers 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	
<i>Pin Interface Block (PIB)</i>	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 select (from which core - Core 0 or Core 1 collect trace) is done via the multiplexor that is controlled via the logical OR of the `TRACE_SEL.ALTTTRACESEL` bit and `SYSCRO.TRACESEL` bit of the CRU register

You can find list and description of the registers in the [Registers Maps](#) and [Register Descriptions](#) sections below.

### 16.1.1. Registers Maps

In this section:

- [Trace Select Register Map](#)
- [Encoder Block Registers Map](#)
- [RAM Control Block Registers Map](#)
- [Pin Interface Block \(PIB Interface\) Registers Map](#)

#### 16.1.1.1. Trace Select Register Map

This section tables contain information about the trace source registers memory map.

The following table provides high-level summary of each 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	<a href="#">Trace Source Select Register</a>	0x0

#### 16.1.1.2. Encoder Block Registers Map

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

The following table provides 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 Registers 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 Registers Map**

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

The following table provides 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 Registers 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) Registers Map**

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

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

**Table 16-6 PIB Block Registers Map**

Name	Offset	Description	Default Value
TRPIBCONTROL	0x0	PIB Block Control Register	0x118

## 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	<p>Alternative trace source select.</p> <p> Trace source select (from which core - Core 0 or Core 1 collect trace) is done via the multiplexor that is controlled via the logical OR of the <code>ALTTRACESEL</code> bit and <code>SYSCR0 . TRACESEL</code> bit of the CRU register</p> <p>Valid values:</p> <ul style="list-style-type: none"> <li>1: Core 1</li> <li>0: Core 0</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

### 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	<p>Trace recording/protocol format.</p> <p>Valid values:</p> <ul style="list-style-type: none"> <li>1: Nexus format: 6 data bits (MDO) + 2 control bits (MSEO)</li> </ul>

**Table 16-8 Fields For Register: TRTECONTROL (continued)**

Bits	Name	R/W	Description
			<b>Value After Reset:</b> 0x1
[23:20]	TR_TE_INST_SYNC_MAX	R/W	<p>The maximum interval (in units determined by <code>TR_TE_SYNC_MODE</code> bits below) between instruction trace synchronization messages/packets.</p> <p>Generate synchronization when count reaches <math>2^{(\text{TR\_TE\_INST\_SYNC\_MAX}+4)}</math>.</p> <p>If an instruction trace synchronization message/packet is generated for another reason, the internal counter should be reset.</p> <p><b>Value After Reset:</b> 0x0</p>
[17:16]	TR_TE_SYNC_MODE	R/W	<p>Select the periodic instruction trace synchronization message/packet generation mechanism. At least one non-zero mechanism must be implemented.</p> <p>Valid values:</p> <ul style="list-style-type: none"> <li><b>0x0:</b> Periodic synchronization disabled</li> <li><b>0x1:</b> Tracing messages counting</li> <li><b>0x2:</b> Core clock ticks counting</li> <li><b>0x3:</b> Executed instructions counting (16 bit)</li> </ul> <p>Once the max value of periodic counter is reached, an instruction trace synchronization message/packet should be generated.</p> <p><b>Value After Reset:</b> 0x0</p>
[15]	TR_TE_INHIBIT_SRC	R	<p>Adding a core number to the trace messages.</p> <p>Valid values:</p> <ul style="list-style-type: none"> <li><b>1:</b> Disable inclusion of source field in trace messages/packets.</li> <li><b>0:</b> Messages/packets generated by the trace encoder include a message source field if the source width held in <code>TRTEINSTFEATURES.TR_TE_SRC_BITS</code> is not 0.</li> </ul> <p><b>Value After Reset:</b> 0x1</p>
[14]			Reserved
[13]	TR_TE_INST_STALL_EN	R/W	<p>Permission of <code>stall</code> signal generation in a case of internal queues overflow.</p> <p>Valid values:</p> <ul style="list-style-type: none"> <li><b>1:</b> 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.</li> <li><b>0:</b> 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.</li> </ul>

**Table 16-8 Fields For Register: TRTECONTROL (continued)**

Bits	Name	R/W	Description
			<b>Value After Reset:</b> 0x0
[12]	TR_TE_INST_STALL_OR_OVERFLOW	RW1C	<p>Set to 1 by hardware when trace buffer overflow (also known as trace lost) occurs, or when the Encoder Block requests a hart stall.</p> <p>Clears to 0 at Encoder Block reset or when the trace is enabled (<code>TR_TE_ENABLE</code> bit below set to 1). Write 1 to clear.</p> <p><b>Value After Reset:</b> 0x0</p>
[11]	TR_TE_TRIG_ENABLE	R/W	<p>Permission of <code>TR_TE_TRACING</code> control bit managing by the outer triggers.</p> <p>Valid values:</p> <ul style="list-style-type: none"> <li><b>1:</b> Allows <code>TR_TE_TRACING</code> bit below to be set or cleared by Trace-on and Trace-off signals generated by the corresponding trigger module.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[10]			Reserved
[9]	TR_TE_CONTEXT	R	<p>Enable sending trace messages/fields with scontext/mcontext values and/or privilege levels.</p> <p><b>Value After Reset:</b> 0x0</p>
[8:7]			Reserved
[6:4]	TR_TE_INST_MODE	R/W	<p>Instruction trace generation mode.</p> <p>Valid values:</p> <ul style="list-style-type: none"> <li><b>110:</b> Branch History Messaging</li> <li><b>011:</b> Branch Trace messaging</li> <li><b>000:</b> Full Instruction trace is disabled, but other trace (data trace) may be emitted.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[3]	TR_TE_EMPTY	R	<p>Internal queues fullness state.</p> <p>If this bit =1, this means that all generated messages are sent to selected devices and no new messages in present.</p> <p><b>Value After Reset:</b> 0x1</p>
[2]	TR_TE_TRACING	R/W	<p>Trace generation permission.</p> <p>Valid values:</p> <ul style="list-style-type: none"> <li><b>1:</b> Instruction trace is being generated. Written from a trace tool (after a write to <code>TR_TE_ENABLE</code> bit below) or controlled by triggers.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>
[1]	TR_TE_ENABLE	R/W	<p>Encoder Block enable.</p> <p>Valid values:</p>

**Table 16-8 Fields For Register: TRTECONTROL (continued)**

Bits	Name	R/W	Description
			<p><b>1:</b> 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.</p> <p><b>0:</b> 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.</p> <p><b>Value After Reset:</b> 0x0</p>
[0]	TR_TE_ACTIVE	R/W	<p>Trace system enable.</p> <p>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.</p> <p>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.</p> <p><b>Value After Reset:</b> 0x0</p>

#### 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	<p>Trace Protocol Minor Version.</p> <p>As specified by specification governing TRTECONTROL.TR_TE_FORMAT.</p> <p><b>Value After Reset:</b> 0x0</p>
[19:16]	TR_TE_PROTOCOL_MAJOR	R	<p>Trace Protocol Major Version.</p> <p>As specified by specification governing TRTECONTROL.TR_TE_FORMAT.</p> <p><b>Value After Reset:</b> 0x1</p>
[15:12]			Reserved
[11:8]	TR_TE_COMP_TYPE	R	<p>Component type.</p> <p><b>Value After Reset:</b> 0x1</p>
[7:4]	TR_TE_VER_MINOR	R	<p>Encoder Block Minor Version.</p> <p>Value 0 means the component is compliant with <b>RISC-V Trace Control Interface Specification, Version 1.0_rc42, July 22, 2024.</b></p> <p><b>Value After Reset:</b> 0x0</p>
[3:0]	TR_TE_VER_MAJOR	R	Encoder Block Major Version.

**Table 16-9 Fields For Register: TRTEIMPL (continued)**

Bits	Name	R/W	Description
			<p>Value 1 means the component is compliant with <i>RISC-V Trace Control Interface Specification, Version 1.0_rc42, July 22, 2024.</i></p> <p><b>Value After Reset:</b> 0x1</p>

#### 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]	<code>TR_TE_SRC_BITS</code>	R	<p>The number of bits in the trace source field, unless disabled by <code>TRTECONTROL.TR_TE_INHIBIT_SRC</code>.</p> <p><b>Value After Reset:</b> 0x0</p>
[27:16]	<code>TR_TE_SRC_ID</code>	R	<p>Trace source ID assigned to this trace encoder.</p> <p>If <code>TR_TE_SRC_BITS</code> is not 0 and trace source is not disabled by <code>TRTECONTROL.TR_TE_INHIBIT_SRC</code>, then trace messages from this Trace Encoder will all include a trace source field of <code>TR_TE_SRC_BITS</code> bits and all messages from this Trace Encoder will use this value as trace source field.</p> <p><b>Value After Reset:</b> 0x0</p>
[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]	<code>TR_TS_WIDTH</code>	R	<p>Width of timestamp in bits.</p> <p><b>Value After Reset:</b> 0x20</p>
[23:16]			Reserved
[15]	<code>TR_TS_ENABLE</code>	R/W	<p>Enable for timestamp field in trace messages/packets (for Trace Encoder only).</p> <p><b>Value After Reset:</b> 0x0</p>
[14:10]			Reserved
[9:8]	<code>TR_TS_PRESCALE</code>	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 $2^{(2 \cdot \text{TR_TS_PRESCALE})}$ . It will be divided by 1, 4, 16, 64 respectively. <b>Value After Reset:</b> 0x0
[7]			Reserved
[6:4]	TR_TS_TYPE	R/W	Mode used by Timestamp unit (clock source types): <b>000:</b> None (Encoder Block disabled) <b>001:</b> External <b>010:</b> Internal System Bus clock based <b>011:</b> Internal Core clock based <b>100:</b> Shared with other blocks counter <b>Value After Reset:</b> 0x0
[3]	TR_TS_RUN_IN_DEBUG	R/W	Counter run mode (even in debug mode): <b>1:</b> Counter runs when hart is halted (in debug mode) <b>0:</b> Stopped <b>Value After Reset:</b> 0x0
[2]	TR_TS_RESET	R/W	Timestamp counter reset bit - write 1 to reset. <b>Value After Reset:</b> 0x0
[1]	TR_TS_COUNT	R/W	Counter enable: <b>1:</b> Counter enable <b>0:</b> Stopped <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 0x0

**16.1.2.2.5. Timestamp Counter Lower Bits (TRTS COUNTERLOW)**

The TRTS COUNTERLOW register is at offset 0x48.

The following table shows the register bit assignments.

**Table 16-12 Fields For Register: TRTS COUNTERLOW**

Bits	Name	R/W	Description
[31:0]	TR_TS_COUNTER_LOW	R	Lower 32 bits of timestamp counter. <b>Value After Reset:</b> 0x0

**16.1.2.2.6. Timestamp Counter Upper Bits (TRTS COUNTERHIGH)**

The TRTS COUNTERHIGH 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. <b>Value After Reset:</b> 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_ASYNC_FREQ	R	Synchronization of aligning packets size control. Valid values:  <b>000:</b> Alignment synchronization (Async) packets disabled (may be the only choice for some protocols) <b>001-111:</b> Different levels of alignment synchronization (bigger number, bigger distance).  Details should be defined in the specification of each trace protocol. <b>Value After Reset:</b> 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 <code>TR_RAM_ENABLE</code> field below of this register will be 0). <b>Value After Reset:</b> 0x0
[7:5]			Reserved
[4]	TR_RAM_MODE	R/W	RAM Control Block operating mode. Valid values:  <b>1:</b> SRAM mode <b>0:</b> SMEM 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. <b>Value After Reset:</b> 0x0
[3]	TR_RAM_EMPTY	R	Reads 1 when Trace RAM Sink internal buffers are empty, which means that all trace data is flushed. <b>Value After Reset:</b> 0x1
[2]			Reserved
[1]	TR_RAM_ENABLE	R/W	RAM Control Block enable. Valid values: <b>1:</b> Trace RAM Sink enabled. <b>0:</b> 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. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 0x0
[12]	TR_RAM_HAS_SRAM	R	SRAM mode supported or not. <b>Value After Reset:</b> 0x1
[11:8]	TR_RAM_COMP_TYPE	R	Trace RAM Sink Component Type (RAM Sink). <b>Value After Reset:</b> 0x9
[7:4]	TR_RAM_VER_MINOR	R	Trace RAM Sink Component Minor Version. Value 0 means the component is compliant with <b>RISC-V Trace Control Interface Specification, Version 1.0_rc42, July 22, 2024</b> .

**Table 16-15 Fields For Register: TRRAMIMPL (continued)**

Bits	Name	R/W	Description
			<b>Value After Reset:</b> 0x0
[3:0]	TR_RAM_VER_MAJOR	R	Trace RAM Sink Component Major Version. Value 1 means the component is compliant with <b>RISC-V Trace Control Interface Specification, Version 1.0_rc42, July 22, 2024.</b> <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 0x0
[0]	TR_RAM_WRAP	RC1	RAM memory (cyclic buffer) is full flag. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 0x0

**16.1.2.3.9. RAM Memory Read Pointer Address Low Bits Register (TRRAMRPLLOW)**

The **TRRAMRPLLOW** register is at offset 0x28.

The following table shows the register bit assignments.

**Table 16-22 Fields For Register: TRRAMRPLLOW**

Bits	Name	R/W	Description
[31:0]	TR_RAM_RP_LOW	R/W	RAM memory (cyclic buffer) read pointer address low bits. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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 deactivated. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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 <b>TR_PIB_MODE</b> bit below selects one of the parallel protocols. <b>Value After Reset:</b> 0x1
[7:4]	TR_PIB_MODE	R	Select mode for output pins. Valid values: <b>10:</b> Parallel mode*

**Table 16-25 Fields For Register: TRPIBCONTROL (continued)**

Bits	Name	R/W	Description
			 *For description on this mode, refer to <i>RISC-V Control interface Specification, Version 1.0_rc42, July 22, 2024</i> . <b>Value After Reset:</b> 0x10
[3]	TR_PIB_EMPTY	R	Reads 1 when PIB internal buffers are empty. <b>Value After Reset:</b> 0x1
[2]			Reserved
[1]	TR_PIB_ENABLE	R/W	Enable of data transmitting via the PIB interface. Valid values: <b>1:</b> Enable PIB to generate output. <b>0:</b> PIB does not accept input but holds output(s) at idle state defined by pibMode. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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

Registers 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	4KB	0x1510_8000	0x1510_8FFF



For details, refer to [Memory Map](#) chapter.

In this chapter:

- [Register Map](#)
- [Register Descriptions](#)



In this section the offsets of the registers is given regarding the start bit address (0x15108000)

#### 17.1.1. Registers Map

The following table provides 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 Registers Map**

Name	Offset	Description	Default Value
PL1	0x00	<a href="#">Arbitration Priority Master 1 Register</a>	0x1
PL2	0x04	<a href="#">Arbitration Priority Master 2 Register</a>	0x2
EBTCOUNT	0x3C	<a href="#">Early Burst Termination Count Register</a>	0x0
EBT_EN	0x40	<a href="#">Early Burst Termination Enable</a>	0x0
EBT	0x44	<a href="#">Early Burst Termination Register</a>	0x0
DFLT_MASTER	0x48	<a href="#">Default Master ID Number Register</a>	0x0

#### 17.1.2. 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.

### 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_LEVEL EL1	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, with a priority 0 signifying that the master has been disabled. The highest value indicates the highest priority. <b>Value After Reset:</b> 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_LEVEL EL2	R/W	Arbitration priority for master 2 register used for programming the Core 2 priority. The priority level ranges from 0 to 15, with a priority 0 signifying that the master has been disabled. The highest value indicates the highest priority. <b>Value After Reset:</b> 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. <b>Value After Reset:</b> 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	<p>Early burst termination enable is to enable or disable <i>Early Burst Termination (EBT)</i> through the software programming.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x0:</b> Disables the Early Burst Termination</li> <li><b>0x1:</b> Enables the Early Burst Termination</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

#### 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	<p>Early burst termination register. Set when an Early Burst Termination takes place. The register is cleared when read by the processor.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li><b>0x1:</b> Early Burst Termination Occured.</li> <li><b>0x0:</b> No Early Burst Termination Occured.</li> </ul> <p><b>Value After Reset:</b> 0x0</p>

#### 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	<p>Default master ID number register. The default master is the master that is granted by the bus when no master has requested ownership.</p> <p><b>Value After Reset:</b> 0x0</p>