TMS320C6457 DSP
Serial RapidIO (SRIO)

User's Guide

Literature Number: SPRUGK4D
March 2009–Revised February 2011

Preface **15**
1 Overview **16**
1.1 General RapidIO System 16
1.2 RapidIO Feature Support in SRIO 19
1.3 Standards 20
1.4 External Devices Requirements 20
1.5 TMS320C6457 SRIO Interface Characteristics 20
2 SRIO Functional Description **21**
2.1 Overview 21
2.2 SRIO Pins 26
2.3 Functional Operation 26
3 Logical/Transport Error Handling and Logging **88**
4 Interrupt Conditions **90**
4.1 CPU Interrupts 90
4.2 General Description 90
4.3 Interrupt Condition Status and Clear Registers 91
4.4 Interrupt Condition Routing Registers 99
4.5 Interrupt Status Decode Registers 103
4.6 Interrupt Generation 105
4.7 Interrupt Pacing 106
4.8 Interrupt Handling 106
5 SRIO Registers **108**
5.1 Peripheral Identification Register (PID) 118
5.2 Peripheral Control Register (PCR) 118
5.3 Peripheral Settings Control Register (PER_SET_CNTL) 120
5.4 Peripheral Settings Control Register 1 (PER_SET_CNTL1) 123
5.5 Peripheral Global Enable Register (GBL_EN) 124
5.6 Peripheral Global Enable Status Register (GBL_EN_STAT) 125
5.7 Block *n* Enable Register (BLK*n*_EN) 127
5.8 Block *n* Enable Status Register (BLK*n*_EN_STAT) 128
5.9 RapidIO DEVICEID1 Register (DEVICEID_REG1) 129
5.10 RapidIO DEVICEID2 Register (DEVICEID_REG2) 130
5.11 RapidIO DEVICEID3 Register (DEVICEID_REG3) 131
5.12 RapidIO DEVICEID4 Register (DEVICEID_REG4) 132
5.13 Packet Forwarding Register *n* for 16-Bit Device IDs (PF_16B_CNTL*n*) 133
5.14 Packet Forwarding Register *n* for 8-Bit Device IDs (PF_8B_CNTL*n*) 134
5.15 SERDES Receive Channel Configuration Register *n* (SERDES_CFGRX*n*_CNTL) 135
5.16 SERDES Transmit Channel Configuration Register *n* (SERDES_CFGTX*n*_CNTL) 138
5.17 SERDES Macro Configuration Register *n* (SERDES_CFG*n*_CNTL) 140
5.18 DOORBELL*n* Interrupt Condition Status Register (DOORBELL*n*_ICSR) 142
5.19 DOORBELL*n* Interrupt Condition Clear Register (DOORBELL*n*_ICCR) 143
5.20 RX CPPI Interrupt Status Register (RX_CPPI_ICSR) 144

5.21 RX CPPI Interrupt Clear Register (RX_CPPI_ICCR) 145
5.22 TX CPPI Interrupt Status Register (TX_CPPI_ICSR) 146
5.23 TX CPPI Interrupt Clear Register (TX_CPPI_ICCR) 147
5.24 LSU Interrupt Condition Status Register (LSU_ICSR) 148
5.25 LSU Interrupt Condition Clear Register (LSU_ICCR) 151
5.26 Error, Reset, and Special Event Interrupt Condition Status Register (ERR_RST_EVNT_ICSR) 152
5.27 Error, Reset, and Special Event Interrupt Condition Clear Register (ERR_RST_EVNT_ICCR) 153
5.28 DOORBELL*n* Interrupt Condition Routing Registers (DOORBELL*n*_ICRR and DOORBELL*n*_ICRR2) 154
5.29 RX CPPI Interrupt Condition Routing Registers (RX_CPPI_ICRR and RX_CPPI_ICRR2) 155
5.30 TX CPPI Interrupt Condition Routing Registers (TX_CPPI_ICRR and TX_CPPI_ICRR2) 156
5.31 LSU Interrupt Condition Routing Registers (LSU_ICRR0-LSU_ICRR3) 157
5.32 Error, Reset, and Special Event Interrupt Condition Routing Registers (ERR_RST_EVNT_ICRR, ERR_RST_EVNT_ICRR2, and ERR_RST_EVNT_ICRR3) 159
5.33 Interrupt Status Decode Register (INTDST*n*_DECODE) 160
5.34 INTDST*n* Interrupt Rate Control Register (INTDST*n*_RATE_CNTL) 164
5.35 LSU*n* Control Register 0 (LSU*n*_REG0) 165
5.36 LSU*n* Control Register 1 (LSU*n*_REG1) 166
5.37 LSU*n* Control Register 2 (LSU*n*_REG2) 167
5.38 LSU*n* Control Register 3 (LSU*n*_REG3) 168
5.39 LSU*n* Control Register 4 (LSU*n*_REG4) 169
5.40 LSU*n* Control Register 5 (LSU*n*_REG5) 170
5.41 LSU*n* Control Register 6 (LSU*n*_REG6) 171
5.42 LSU*n* Congestion Control Flow Mask Register (LSU***n***_FLOW_MASKS) 172
5.43 Queue *n* Transmit DMA Head Descriptor Pointer Register (QUEUE*n*_TXDMA_HDP) 174
5.44 Queue *n* Transmit DMA Completion Pointer Register (QUEUE*n*_TXDMA_CP) 175
5.45 Queue *n* Receive DMA Head Descriptor Pointer Register (QUEUE*n*_RXDMA_HDP) 176
5.46 Queue *n* Receive DMA Completion Pointer Register (QUEUE*n*_RXDMA_CP) 177
5.47 Transmit Queue Teardown Register (TX_QUEUE_TEAR_DOWN) 178
5.48 Transmit CPPI Supported Flow Mask Registers (TX_CPPI_FLOW_MASKS[0-7]) 179
5.49 Receive Queue Teardown Register (RX_QUEUE_TEAR_DOWN) 182
5.50 Receive CPPI Control Register (RX_CPPI_CNTL) 183
5.51 Transmit CPPI Weighted Round-Robin Control Registers (TX_QUEUE_CNTL[0-3]) 184
5.52 Mailbox to Queue Mapping Registers (RXU_MAP_L*n* and RXU_MAP_H*n*) 188
5.53 Flow Control Table Entry Register *n* (FLOW_CNTL*n*) 192
5.54 Device Identity CAR (DEV_ID) 193
5.55 Device Information CAR (DEV_INFO) 194
5.56 Assembly Identity CAR (ASBLY_ID) 195
5.57 Assembly Information CAR (ASBLY_INFO) 196
5.58 Processing Element Features CAR (PE_FEAT) 197
5.59 Source Operations CAR (SRC_OP) 199
5.60 Destination Operations CAR (DEST_OP) 200
5.61 Processing Element Logical Layer Control CSR (PE_LL_CTL) 201
5.62 Local Configuration Space Base Address 0 CSR (LCL_CFG_HBAR) 202
5.63 Local Configuration Space Base Address 1 CSR (LCL_CFG_BAR) 202
5.64 Base Device ID CSR (BASE_ID) 203
5.65 Host Base Device ID Lock CSR (HOST_BASE_ID_LOCK) 204
5.66 Component Tag CSR (COMP_TAG) 205
5.67 1x/4x LP Serial Port Maintenance Block Header Register (SP_MB_HEAD) 206

5.68 Port Link Time-Out Control CSR (SP_LT_CTL) 207
5.69 Port Response Time-Out Control CSR (SP_RT_CTL) 208
5.70 Port General Control CSR (SP_GEN_CTL) 209
5.71 Port Link Maintenance Request CSR *n* (SP*n*_LM_REQ) 210
5.72 Port Link Maintenance Response CSR *n* (SP*n*_LM_RESP) 211
5.73 Port Local AckID Status CSR *n* (SP*n*_ACKID_STAT) 212
5.74 Port Error and Status CSR *n* (SP*n*_ERR_STAT) 213
5.75 Port Control CSR *n* (SP*n*_CTL) 216
5.76 Error Reporting Block Header Register (ERR_RPT_BH) 219
5.77 Logical/Transport Layer Error Detect CSR (ERR_DET) 220
5.78 Logical/Transport Layer Error Enable CSR (ERR_EN) 222
5.79 Logical/Transport Layer High Address Capture CSR (H_ADDR_CAPT) 224
5.80 Logical/Transport Layer Address Capture CSR (ADDR_CAPT) 225
5.81 Logical/Transport Layer Device ID Capture CSR (ID_CAPT) 226
5.82 Logical/Transport Layer Control Capture CSR (CTRL_CAPT) 227
5.83 Port-Write Target Device ID CSR (PW_TGT_ID) 228
5.84 Port Error Detect CSR *n* (SP*n*_ERR_DET) 229
5.85 Port Error Rate Enable CSR *n* (SP*n*_RATE_EN) 231
5.86 Port *n* Attributes Error Capture CSR 0 (SP*n*_ERR_ATTR_CAPT_DBG0) 233
5.87 Port *n* Error Capture CSR 1 (SP*n*_ERR_CAPT_DBG1) 234
5.88 Port *n* Error Capture CSR 2 (SP*n*_ERR_CAPT_DBG2) 235
5.89 Port *n* Error Capture CSR 3 (SP*n*_ERR_CAPT_DBG3) 236
5.90 Port *n* Error Capture CSR 4 (SP*n*_ERR_CAPT_DBG4) 237
5.91 Port Error Rate CSR *n* (SP*n*_ERR_RATE) 238
5.92 Port Error Rate Threshold CSR *n* (SP*n*_ERR_THRESH) 238
5.93 Port IP Discovery Timer for 4x Mode Register (SP_IP_DISCOVERY_TIMER) 240
5.94 Port IP Mode CSR (SP_IP_MODE) 241
5.95 Port IP Prescaler Register (IP_PRESCAL) 243
5.96 Port-Write-In Capture CSRs (SP_IP_PW_IN_CAPT[0-3]) 244
5.97 Port Reset Option CSR *n* (SP*n*_RST_OPT) 245
5.98 Port Control Independent Register *n* (SP*n*_CTL_INDEP) 246
5.99 Port Silence Timer *n* Register (SP*n*_SILENCE_TIMER) 248
5.100 Port Multicast-Event Control Symbol Request Register *n* (SP*n*_MULT_EVNT_CS) 249
5.101 Port Control Symbol Transmit *n* Register (SP*n*_CS_TX) 250

Appendix A Examples 251

A.1 SRIO Initialization Example 251
A.2 LSU Programming Example 253
A.3 Message Passing Software 254
A.4 Interrupt Handling 255

Appendix B Software-Assisted Error Recovery 257

Appendix C Revision History 259

List of Figures

1 RapidIO Architectural Hierarchy 17
2 RapidIO Interconnect Architecture 18
3 Serial RapidIO Device-to-Device Interface Diagrams 19
4 SRIO Peripheral Block Diagram 22
5 Operation Sequence 23
6 1x/4x RapidIO Packet Data Stream (Streaming-Write Class) 24
7 Serial RapidIO Control Symbol Format 24
8 SRIO Component Block Diagram 27
9 SERDES Macro Configuration Register 0 (SERDES_CFG0_CNTL) 28
10 SERDES Receive Channel Configuration Register *n* (SERDES_CFGRX*n*_CNTL) 32
11 SERDES Transmit Channel Configuration Register *n* (SERDES_CFGTX*n*_CNTL) 34
12 Load/Store Registers for RapidIO (Address Offset: LSU1 400h-418h, LSU2 420h-438h, LSU3 440h-458h, LSU4 460h-478h) 36
13 LSU Registers Timing 39
14 Example Burst NWRITE_R 40
15 Load/Store Module Data Flow Diagram 41
16 CPPI RX Scheme for RapidIO 45
17 Message Request Packet 46
18 Mailbox to Queue Mapping Register Pair 47
19 RX Buffer Descriptor Fields 48
20 RX CPPI Mode Explanation 50
21 CPPI Boundary Diagram 52
22 TX Buffer Descriptor Fields 53
23 Weighted Round-Robin Programming Registers - Address Offset 7E0h-7ECh 56
24 RX Buffer Descriptors 63
25 TX Buffer Descriptors 64
26 Doorbell Operation 65
27 Flow Control Table Entry Registers - Address Offset 0900h-093Ch 68
28 Transmit Source Flow Control Masks 69
29 Fields Within Each Flow Mask 69
30 Configuration Bus Example 70
31 GBL_EN (Address 0030h) 73
32 GBL_EN_STAT (Address 0034h) 73
33 BLK0_EN (Address 0038h) 75
34 BLK0_EN_STAT (Address 003Ch) 75
35 BLK1_EN (Address 0040h) 75
36 BLK1_EN_STAT (Address 0044h) 75
37 BLK8_EN (Address 0078h) 75
38 BLK8_EN_STAT (Address 007Ch) 75
39 Peripheral Control Register (PCR) - Address Offset 0004h 77
40 Bootload Operation 82
41 Logical/Transport Layer Error Detect CSR (ERR_DET) 88
42 RapidIO DOORBELL Packet for Interrupt Use 90
43 Doorbell 0 Interrupt Condition Status and Clear Registers 92
44 Doorbell 1 Interrupt Condition Status and Clear Registers 92
45 Doorbell 2 Interrupt Condition Status and Clear Registers 93
46 Doorbell 3 Interrupt Condition Status and Clear Registers 93

47 RX CPPI Interrupt Condition Status and Clear Registers 94
48 TX CPPI Interrupt Condition Status and Clear Registers 94
49 LSU Interrupt Condition Status and Clear Registers 95
50 Error, Reset, and Special Event Interrupt Condition Status and Clear Registers 97
51 Doorbell 0 Interrupt Condition Routing Registers 100
52 RX CPPI Interrupt Condition Routing Registers 101
53 TX CPPI Interrupt Condition Routing Registers 101
54 LSU Interrupt Condition Routing Registers 102
55 Error, Reset, and Special Event Interrupt Condition Routing Registers 103
56 Interrupt Status Decode Register (INTDST*n*_DECODE) 104
57 Interrupt Sources Assigned to ISDR Bits 104
58 Example Diagram of Interrupt Status Decode Register Mapping 105
59 INTDST*n*_RATE_CNTL Interrupt Rate Control Register 106
60 Peripheral ID Register (PID) - Address Offset 0000h 118
61 Peripheral Control Register (PCR) - Address Offset 0004h 118
62 Peripheral Settings Control Register (PER_SET_CNTL) - Address Offset 0020h 120
63 Peripheral Settings Control Register 1 (PER_SET_CNTL1) - Address Offset 0024h 123
64 Peripheral Global Enable Register (GBL_EN) - Address Offset 0030h 124
65 Peripheral Global Enable Status Register (GBL_EN_STAT) - Address Offset 0034h 125
66 Block *n* Enable Register (BLK*n*_EN) 127
67 Block *n* Enable Status Register (BLK*n*_EN) 128
68 RapidIO DEVICEID1 Register (DEVICEID_REG1) - Address Offset 0080h 129
69 RapidIO DEVICEID2 Register (DEVICEID_REG2) - Address Offset 0084h 130
70 RapidIO DEVICEID3 Register (DEVICEID_REG3) - Address Offset 0088h 131
71 RapidIO DEVICEID4 Register (DEVICEID_REG4) - Address Offset 008Ch 132
72 Packet Forwarding Register *n* for 16-Bit Device IDs (PF_16B_CNTL*n*) 133
73 Packet Forwarding Register *n* for 8-Bit Device IDs (PF_8B_CNTL*n*) 134
74 SERDES Receive Channel Configuration Register *n* (SERDES_CFGRX*n*_CNTL) 135
75 SERDES Transmit Channel Configuration Register *n* (SERDES_CFGTX*n*_CNTL) 138
76 SERDES Macro Configuration Register *n* (SERDES_CFG*n*_CNTL) 140
77 Doorbell *n* Interrupt Condition Status Register (DOORBELL*n*_ICSR) 142
78 Doorbell *n* Interrupt Condition Clear Register (DOORBELL*n*_ICCR) 143
79 RX CPPI Interrupt Condition Status Register (RX_CPPI_ICSR) - Address Offset 0240h 144
80 RX CPPI Interrupt Condition Clear Register (RX_CPPI_ICCR) - Address Offset 0248h 145
81 TX CPPI Interrupt Condition Status Register (TX_CPPI_ICSR) - Address Offset 0250h 146
82 TX CPPI Interrupt Condition Clear Register (TX_CPPI_ICCR) - Address Offset 0258h 147
83 LSU Interrupt Condition Status Register (LSU_ICSR) - Address Offset 0260h 148
84 LSU Interrupt Condition Clear Register (LSU_ICCR) - Address Offset 0268h 151
85 Error, Reset, and Special Event Interrupt Condition Status Register (ERR_RST_EVNT_ICSR) - Address Offset 0270h 152
86 Error, Reset, and Special Event Interrupt Condition Clear Register (ERR_RST_EVNT_ICCR) - Address Offset 0278h 153
87 Doorbell *n* Interrupt Condition Routing Registers 154
88 RX CPPI Interrupt Condition Routing Registers 155
89 TX CPPI Interrupt Condition Routing Registers 156
90 LSU Interrupt Condition Routing Registers 157
91 Error, Reset, and Special Event Interrupt Condition Routing Registers 159
92 Interrupt Status Decode Register (INTDST*n*_DECODE) 160
93 INTDST*n* Interrupt Rate Control Register (INTDST*n*_RATE_CNTL) 164

94 LSU*n* Control Register 0 (LSU*n*_REG0) 165
95 LSU*n* Control Register 1 (LSU*n*_REG1) 166
96 LSU*n* Control Register 2 (LSU*n*_REG2) 167
97 LSU*n* Control Register 3 (LSU*n*_REG3) 168
98 LSU*n* Control Register 4 (LSU*n*_REG4) 169
99 LSU*n* Control Register 5 (LSU*n*_REG5) 170
100 LSU*n* Control Register 6 (LSU*n*_REG6) 171
101 LSU*n* Congestion Control Flow Mask Register (LSU*n*_FLOW_MASKS) 172
102 LSU*n* FLOW_MASK Fields 172
103 Queue *n* Transmit DMA Head Descriptor Pointer Register (QUEUE*n*_TXDMA_HDP) 174
104 Queue *n* Transmit DMA Completion Pointer Register (QUEUE*n*_TXDMA_CP) 175
105 Queue *n* Receive DMA Head Descriptor Pointer Register (QUEUE*n*_RXDMA_HDP) 176
106 Queue *n* Receive DMA Completion Pointer Register (QUEUE*n*_RXDMA_CP) 177
107 Transmit Queue Teardown Register (TX_QUEUE_TEAR_DOWN) - Address Offset 0700h 178
108 Transmit CPPI Supported Flow Mask Registers 180
109 TX Queue *n* FLOW_MASK Fields 180
110 Receive Queue Teardown Register (RX_QUEUE_TEAR_DOWN) - Address Offset 0740h 182
111 Receive CPPI Control Register (RX_CPPI_CNTL) - Address Offset 0744h 183
112 Transmit CPPI Weighted Round-Robin Control Registers 184
113 Mailbox to Queue Mapping Register Pair 190
114 Flow Control Table Entry Register *n* (FLOW_CNTL*n*) 192
115 Device Identity CAR (DEV_ID) - Address Offset 1000h 193
116 Device Information CAR (DEV_INFO) - Address Offset 1004h 194
117 Assembly Identity CAR (ASBLY_ID) - Address Offset 1008h 195
118 Assembly Information CAR (ASBLY_INFO) - Address Offset 100Ch 196
119 Processing Element Features CAR (PE_FEAT) - Address Offset 1010h 197
120 Source Operations CAR (SRC_OP) - Address Offset 1018h 199
121 Destination Operations CAR (DEST_OP) - Address Offset 101Ch 200
122 Processing Element Logical Layer Control CSR (PE_LL_CTL) - Address Offset 104Ch 201
123 Local Configuration Space Base Address 0 CSR (LCL_CFG_HBAR) -Address Offset 1058h 202
124 Local Configuration Space Base Address 1 CSR (LCL_CFG_BAR) -Address Offset 105Ch 202
125 Base Device ID CSR (BASE_ID) - Address Offset 1060h 203
126 Host Base Device ID Lock CSR (HOST_BASE_ID_LOCK) - Address Offset 1068h 204
127 Component Tag CSR (COMP_TAG) - Address Offset 106Ch 205
128 1x/4x LP_Serial Port Maintenance Block Header Register (SP_MB_HEAD) - Address Offset 1100h 206
129 Port Link Time-Out Control CSR (SP_LT_CTL) - Address Offset 1120h 207
130 Port Response Time-Out Control CSR (SP_RT_CTL) - Address Offset 1124h 208
131 Port General Control CSR (SP_GEN_CTL) - Address Offset 113Ch 209
132 Port Link Maintenance Request CSR *n* (SP*n*_LM_REQ) 210
133 Port Link Maintenance Response CSR *n* (SP*n*_LM_RESP) 211
134 Port Local AckID Status CSR *n* (SP*n*_ACKID_STAT) 212
135 Port Error and Status CSR *n* (SP*n*_ERR_STAT) 213
136 Port Control CSR *n* (SP*n*_CTL) 216
137 Error Reporting Block Header Register (ERR_RPT_BH) - Address Offset 2000h 219
138 Logical/Transport Layer Error Detect CSR (ERR_DET) - Address Offset 2008h 220
139 Logical/Transport Layer Error Enable CSR (ERR_EN) - Address Offset 200Ch 222
140 Logical/Transport Layer High Address Capture CSR (H_ADDR_CAPT) - Address Offset 2010h 224
141 Logical/Transport Layer Address Capture CSR (ADDR_CAPT) - Address Offset 2014h 225
142 Logical/Transport Layer Device ID Capture CSR (ID_CAPT) - Address Offset 2018h 226

143 Logical/Transport Layer Control Capture CSR (CTRL_CAPT) - Address Offset 201Ch 227
144 Port-Write Target Device ID CSR (PW_TGT_ID) - Address Offset 2028h 228
145 Port Error Detect CSR *n* (SP*n*_ERR_DET) 229
146 Port Error Rate Enable CSR *n* (SP*n*_RATE_EN) 231
147 Port *n* Attributes Error Capture CSR 0 (SP*n*_ERR_ATTR_CAPT_DBG0) 233
148 Port *n* Error Capture CSR 1 (SP*n*_ERR_CAPT_DBG1) 234
149 Port *n* Error Capture CSR 2 (SP*n*_ERR_CAPT_DBG2) 235
150 Port *n* Error Capture CSR 3 (SP*n*_ERR_CAPT_DBG3) 236
151 Port *n* Error Capture CSR 4 (SP*n*_ERR_CAPT_DBG4) 237
152 Port Error Rate CSR *n* (SP*n*_ERR_RATE) 238
153 Port Error Rate Threshold CSR *n* (SP*n*_ERR_THRESH) 239
154 Port IP Discovery Timer for 4x Mode Register (SP_IP_DISCOVERY_TIMER) - Address Offset 12000h 240
155 Port IP Mode CSR (SP_IP_MODE) - Address Offset 12004h 241
156 Port IP Prescaler Register (IP_PRESCAL) - Address Offset 12008h 243
157 Port-Write-In Capture CSRs 244
158 Port Reset Option CSR *n* (SP*n*_RST_OPT) 245
159 Port Control Independent Register *n* (SP*n*_CTL_INDEP) 246
160 Port Silence Timer *n* Register (SP*n*_SILENCE_TIMER) 248
161 Port Multicast-Event Control Symbol Request Register *n* (SP*n*_MULT_EVNT_CS) 249
162 Port Control Symbol Transmit *n* Register (SP*n*_CS_TX) 250
163 Software Error Recovery Sequence 257

List of Tables

1 TMS320C6457 SRIO Interface Characteristics 20
2 Registers Checked for Multicast DeviceID 21
3 Packet Types 25
4 Pin Description 26
5 SERDES Macro Configuration Register 0 (SERDES_CFG0_CNTL) Field Descriptions 29
6 Line Rate versus PLL Output Clock Frequency 30
7 Effect of the RATE Bits 30
8 Frequency Range versus MPY Value 30
9 SERDES Receive Channel Configuration Register *n* (SERDES_CFGRX*n*_CNTL) Field Descriptions 32
10 EQ Bits 33
11 SERDES Transmit Channel Configuration Register *n* (SERDES_CFGTX*n*_CNTL) Field Descriptions 34
12 DE Bits of SERDES_CFGTX*n*_CNTL 35
13 SWING Bits of SERDES_CFGTX*n*_CNTL 35
14 LSU Control/Command Register Fields 36
15 LSU Status Register Fields 37
16 RX DMA State Head Descriptor Pointer (HDP) - Address Offset 600h-63Ch 47
17 RX DMA State Completion Pointer (CP) - Address Offset 680h-6BCh 47
18 RX Buffer Descriptor Field Descriptions 48
19 TX DMA State Head Descriptor Pointer (HDP) - Address Offset 500h-53Ch 52
20 TX DMA State Completion Pointer (CP) - Address Offset 58h-5BCh 53
21 TX Buffer Descriptor Field Definitions 53
22 Weighted Round-Robin Programming Registers - Address Offset 7E0h-7ECh 57
23 Examples of DOORBELL_INFO Designations (See) 66
24 Flow Control Table Entry Register *n* (FLOW_CNTL*n*) Field Descriptions 68
25 Fields Within Each Flow Mask 69
26 DMA Little-Endian Swapping Modes 71
27 Bits for Little-Endian Swapping 71
28 Reset Hierarchy 72
29 Global Enable and Global Enable Status Field Descriptions 74
30 Block Enable and Block Enable Status Field Descriptions 75
31 Peripheral Control Register (PCR) Field Descriptions 77
32 Port Mode Register Settings 80
33 DESTID Checking Control Modes 83
34 Multicast DeviceID Operation 85
35 Hardware Packet Forwarding and Multicast Operation 86
36 Logical/Transport Layer Error Detect CSR (ERR_DET) Field Descriptions 88
37 Interrupt Condition Status and Clear Bits 92
38 Interrupt Conditions Shown in LSU_ICSR and Cleared With LSU_ICCR 95
39 Interrupt Conditions Shown in ERR_RST_EVNT_ICSR and Cleared With ERR_RST_EVNT_ICCR 97
40 Interrupt Clearing Sequence for Special Event Interrupts 97
41 Interrupt Condition Routing Options 99
42 SRIO Registers 108
43 Peripheral ID Register (PID) Field Descriptions 118
44 Peripheral Control Register (PCR) Field Descriptions 118
45 Peripheral Settings Control Register (PER_SET_CNTL) Field Descriptions 120
46 Peripheral Settings Control Register 1 (PER_SET_CNTL1) Field Descriptions 123
47 Peripheral Global Enable Register (GBL_EN) Field Descriptions 124

48 Peripheral Global Enable Status Register (GBL_EN_STAT) Field Descriptions 125
49 Block *n* Enable Registers and the Associated Blocks 127
50 Block *n* Enable Register (BLK*n*_EN) Field Descriptions 127
51 Block *n* Enable Status Registers and the Associated Blocks 128
52 Block *n* Enable Status Register (BLK*n*_EN_STAT) Field Descriptions 128
53 RapidIO DEVICEID1 Register (DEVICEID_REG1) Field Descriptions 129
54 RapidIO DEVICEID2 Register (DEVICEID_REG2) Field Descriptions 130
55 RapidIO DEVICEID3 Register (DEVICEID_REG3) Field Descriptions 131
56 RapidIO DEVICEID4 Register (DEVICEID_REG4) Field Descriptions 132
57 PF_16B_CNTL Registers 133
58 Packet Forwarding Register *n* for 16-Bit DeviceIDs (PF_16B_CNTL*n*) Field Descriptions 133
59 PF_8B_CNTL Registers 134
60 Packet Forwarding Register *n* for 8-Bit DeviceIDs (PF_8B_CNTL*n*) Field Descriptions 134
61 SERDES_CFGRX*n*_CNTL Registers and the Associated Ports 135
62 SERDES Receive Channel Configuration Register *n* (SERDES_CFGRX*n*_CNTL) Field Descriptions 135
63 EQ Bits 136
64 SERDES_CFGTX*n*_CNTL Registers and the Associated Ports 138
65 SERDES Transmit Channel Configuration Register *n* (SERDES_CFGTX*n*_CNTL) Field Descriptions 138
66 DE Bits of SERDES_CFGTX*n*_CNTL 139
67 SWING Bits of SERDES_CFGTX*n*_CNTL 139
68 SERDES_CFG*n*_CNTL Registers and the Associated Ports 140
69 SERDES Macro Configuration Register *n* (SERDES_CFG*n*_CNTL) Field Descriptions 140
70 DOORBELL*n*_ICSR Registers 142
71 DOORBELL*n* Interrupt Condition Status Register (DOORBELL*n*_ICSR) Field Descriptions 142
72 DOORBELL*n*_ICCR Registers 143
73 DOORBELL*n* Interrupt Condition Clear Register (DOORBELL*n*_ICCR) Field Descriptions 143
74 RX CPPI Interrupt Condition Status Register (RX_CPPI_ICSR) Field Descriptions 144
75 RX CPPI Interrupt Condition Clear Register (RX_CPPI_ICCR) Field Descriptions 145
76 TX CPPI Interrupt Condition Status Register (TX_CPPI_ICSR) Field Descriptions 146
77 TX CPPI Interrupt Condition Clear Register (TX_CPPI_ICCR) Field Descriptions 147
78 LSU Interrupt Condition Status Register (LSU_ICSR) Field Descriptions 148
79 LSU Interrupt Condition Clear Register (LSU_ICCR) Field Descriptions 151
80 Error, Reset, and Special Event Interrupt Condition Status Register (ERR_RST_EVNT_ICSR) Field Descriptions 152
81 Error, Reset, and Special Event Interrupt Condition Clear Register (ERR_RST_EVNT_ICCR) Field Descriptions 153
82 DOORBELL*n*_ICRR Registers 154
83 DOORBELL*n* Interrupt Condition Routing Register Field Descriptions 154
84 RX CPPI Interrupt Condition Routing Register Field Descriptions 155
85 TX CPPI Interrupt Condition Routing Register Field Descriptions 156
86 LSU Interrupt Condition Routing Register Field Descriptions 158
87 Error, Reset, and Special Event Interrupt Condition Routing Register Field Descriptions 159
88 INTDST*n*_DECODE Registers and the Associated Interrupt Destinations 160
89 Interrupt Status Decode Register (INTDST*n*_DECODE) Field Descriptions 160
90 INTDST*n*_RATE_CNTL Registers and the Associated Interrupt Destinations 164
91 INTDST*n* Interrupt Rate Control Register (INTDST*n*_RATE_CNTL) Field Descriptions 164
92 LSU*n*_REG0 Registers and the Associated LSUs 165
93 LSU*n* Control Register 0 (LSU*n*_REG0) Field Descriptions 165
94 LSU*n*_REG1 Registers and the Associated LSUs 166

95 LSU*n* Control Register 1 (LSU*n*_REG1) Field Descriptions 166
96 LSU*n*_REG2 Registers and the Associated LSUs 167
97 LSU*n* Control Register 2 (LSU*n*_REG2) Field Descriptions 167
98 LSU*n*_REG3 Registers and the Associated LSUs 168
99 LSU*n* Control Register 3 (LSU*n*_REG3) Field Descriptions 168
100 LSU*n*_REG4 Registers and the Associated LSUs 169
101 LSU*n* Control Register 4 (LSU*n*_REG4) Field Descriptions 169
102 LSU*n*_REG5 Registers and the Associated LSUs 170
103 LSU*n* Control Register 5 (LSU*n*_REG5) Field Descriptions 170
104 LSU*n*_REG6 Registers and the Associated LSUs 171
105 LSU*n* Control Register 6 (LSU*n*_REG6) Field Descriptions 171
106 LSU*n*_FLOW_MASKS Registers and the Associated LSUs 172
107 LSU*n* Congestion Control Flow Mask Register (LSU*n*_FLOW_MASKS) Field Descriptions 172
108 LSU*n* FLOW_MASK Fields 172
109 QUEUE*n*_TXDMA_HDP Registers 174
110 Queue *n* Transmit DMA Head Descriptor Pointer Register (QUEUE*n*_TXDMA_HDP) Field Descriptions 174
111 QUEUE*n*_TXDMA_CP Registers 175
112 Queue Transmit DMA Completion Pointer Registers (QUEUE*n*_TXDMA_CP) Field Descriptions 175
113 QUEUE*n*_RXDMA_HDP Registers 176
114 Queue *n* Receive DMA Head Descriptor Pointer Register (QUEUE*n*_RXDMA_HDP) Field Descriptions 176
115 QUEUE*n*_RXDMA_CP Registers 177
116 Queue *n* Receive DMA Completion Pointer Register (QUEUE*n*_RXDMA_CP) Field Descriptions 177
117 Transmit Queue Teardown Register (TX_QUEUE_TEAR_DOWN) Field Descriptions 178
118 TX_CPPI_FLOW_MASKS Registers and the Associated TX Queues 179
119 TX Queue *n* FLOW_MASK Field Descriptions 180
120 Receive Queue Teardown Register (RX_QUEUE_TEAR_DOWN) Field Descriptions 182
121 Receive CPPI Control Register (RX_CPPI_CNTL) Field Descriptions 183
122 Transmit CPPI Weighted Round-Robin Control Register Field Descriptions 185
123 Mailbox to Queue Mapping Registers and the Associated RX Mappers 188
124 Mailbox-to-Queue Mapping Register L*n* (RXU_MAP_L*n*) Field Descriptions 190
125 Mailbox-to-Queue Mapping Register H*n* (RXU_MAP_H*n*) Field Descriptions 191
126 FLOW_CNTL*n* Registers 192
127 Flow Control Table Entry Register *n* (FLOW_CNTL*n*) Field Descriptions 192
128 Device Identity CAR (DEV_ID) Field Descriptions 193
129 Device Information CAR (DEV_INFO) Field Descriptions 194
130 Assembly Identity CAR (ASBLY_ID) Field Descriptions 195
131 Assembly Information CAR (ASBLY_INFO) Field Descriptions 196
132 Processing Element Features CAR (PE_FEAT) Field Descriptions 197
133 Source Operations CAR (SRC_OP) Field Descriptions 199
134 Destination Operations CAR (DEST_OP) Field Descriptions 200
135 Processing Element Logical Layer Control CSR (PE_LL_CTL) Field Descriptions 201
136 Local Configuration Space Base Address 0 CSR (LCL_CFG_HBAR) Field Descriptions 202
137 Local Configuration Space Base Address 1 CSR (LCL_CFG_BAR) Field Descriptions 202
138 Base Device ID CSR (BASE_ID) Field Descriptions 203
139 Host Base Device ID Lock CSR (HOST_BASE_ID_LOCK) Field Descriptions 204
140 Component Tag CSR (COMP_TAG) Field Descriptions 205
141 1x/4x LP_Serial Port Maintenance Block Header Register (SP_MB_HEAD) Field Descriptions 206
142 Port Link Timeout Control CSR (SP_LT_CTL) Field Descriptions 207
143 Port Response Time-Out Control CSR (SP_RT_CTL) Field Descriptions 208

144 Port General Control CSR (SP_GEN_CTL) Field Descriptions 209
145 SPn_LM_REQ Registers and the Associated Ports 210
146 Port Link Maintenance Request CSR *n* (SP*n*_LM_REQ) Field Descriptions 210
147 SPn_LM_RESP Registers and the Associated Ports 211
148 Port Link Maintenance Response CSR *n* (SP*n*_LM_RESP) Field Descriptions 211
149 SP*n*_ACKID_STAT Registers and the Associated Ports 212
150 Port Local AckID Status CSR *n* (SP*n*_ACKID_STAT) Field Descriptions 212
151 SP*n*_ERR_STAT Registers and the Associated Ports 213
152 Port Error and Status CSR *n* (SP*n*_ERR_STAT) Field Descriptions 213
153 SP*n*_CTL Registers and the Associated Ports 216
154 Port Control CSR *n* (SP*n*_CTL) Field Descriptions 216
155 Error Reporting Block Header Register (ERR_RPT_BH) Field Descriptions 219
156 Logical/Transport Layer Error Detect CSR (ERR_DET) Field Descriptions 220
157 Logical/Transport Layer Error Enable CSR (ERR_EN) Field Descriptions 222
158 Logical/Transport Layer High Address Capture CSR (H_ADDR_CAPT) Field Descriptions 224
159 Logical/Transport Layer Address Capture CSR (ADDR_CAPT) Field Descriptions 225
160 Logical/Transport Layer Device ID Capture CSR (ID_CAPT) Field Descriptions 226
161 Logical/Transport Layer Control Capture CSR (CTRL_CAPT) Field Descriptions 227
162 Port-Write Target Device ID CSR (PW_TGT_ID) Field Descriptions 228
163 SP*n*_ERR_DET Registers and the Associated Ports 229
164 Port Error Detect CSR *n* (SP*n*_ERR_DET) Field Descriptions 229
165 SP*n*_RATE_EN Registers and the Associated Ports 231
166 Port Error Rate Enable CSR *n* (SP*n*_RATE_EN) Field Descriptions 231
167 SP*n*_ERR_ATTR_CAPT_DBG0 Registers and the Associated Ports 233
168 Port *n* Attributes Error Capture CSR 0 (SP*n*_ERR_ATTR_CAPT_DBG0) Field Descriptions 233
169 SP*n*_ERR_CAPT_DBG1 Registers and the Associated Ports 234
170 Port *n* Error Capture CSR 1 (SP*n*_ERR_CAPT_DBG1) Field Descriptions 234
171 SP*n*_ERR_CAPT_DBG2 Registers and the Associated Ports 235
172 Port *n* Error Capture CSR 2 (SP*n*_ERR_CAPT_DBG2) Field Descriptions 235
173 SP*n*_ERR_CAPT_DBG3 Registers and the Associated Ports 236
174 Port *n* Error Capture CSR 3 (SP*n*_ERR_CAPT_DBG3) Field Descriptions 236
175 SP*n*_ERR_CAPT_DBG4 Registers and the Associated Ports 237
176 Port *n* Error Capture CSR 4 (SP*n*_ERR_CAPT_DBG4) Field Descriptions 237
177 SP*n*_ERR_RATE Registers and the Associated Ports 238
178 Port Error Rate CSR *n* (SP*n*_ERR_RATE) Field Descriptions 238
179 SP*n*_ERR_THRESH Registers and the Associated Ports 239
180 Port Error Rate Threshold CSR *n* (SP*n*_ERR_THRESH) Field Descriptions 239
181 Port IP Discovery Timer for 4x Mode Register (SP_IP_DISCOVERY_TIMER) Field Descriptions 240
182 Port IP Mode CSR (SP_IP_MODE) Field Descriptions 241
183 Port IP Prescaler Register (IP_PRESCAL) Field Descriptions 243
184 Port-Write-In Capture CSR Field Descriptions 244
185 SP*n*_RST_OPT Registers and the Associated Ports 245
186 Port Reset Option CSR *n* (SP*n*_RST_OPT) Field Descriptions 245
187 SP*n*_CTL_INDEP Registers and the Associated Ports 246
188 Port Control Independent Register *n* (SP*n*_CTL_INDEP) Field Descriptions 246
189 SP*n*_SILENCE_TIMER Registers and the Associated Ports 248
190 Port Silence Timer *n* Register (SP*n*_SILENCE_TIMER) Field Descriptions 248
191 SP*n*_MULT_EVNT_CS Registers and the Associated Ports 249
192 Port Multicast-Event Control Symbol Request Register *n* (SP*n*_MULT_EVNT_CS) Field Descriptions 249

193 SP*n*_CS_TX Registers and the Associated Ports 250
194 Port Control Symbol Transmit *n* Register (SP*n*_CS_TX) Field Descriptions..... 250
195 SRIO Revision History..... 259

Read This First

About This Manual

This document describes the Serial RapidIO® (SRIO) peripheral on the TMS320C6457 devices.

Notational Conventions

This document uses the following conventions.

- Hexadecimal numbers are shown with the suffix h. For example, the following number represents 40 hexadecimal (decimal 64): 40h.
- Registers in this document are shown in figures and described in tables.
  - Each register figure shows a rectangle divided into fields that represent the fields of the register. Each field is labeled with its bit name, its beginning and ending bit numbers above, and its read/write properties below. A legend explains the notation used for the properties.
  - Reserved bits in a register figure designate a bit that is used for future device expansion.

Related Documentation From Texas Instruments

The following documents describe the C6000™ devices and related support tools. Copies of these documents are available on the Internet at www.ti.com. *Tip:* Enter the literature number in the search box provided at www.ti.com.

SPRU189 — *TMS320C6000 DSP CPU and Instruction Set Reference Guide.* Describes the CPU architecture, pipeline, instruction set, and interrupts for the TMS320C6000 digital signal processors (DSPs).

SPRU198 — *TMS320C6000 Programmer's Guide.* Describes ways to optimize C and assembly code for the TMS320C6000™ DSPs and includes application program examples.

SPRU301 — *TMS320C6000 Code Composer Studio Tutorial.* Introduces the Code Composer Studio™ integrated development environment and software tools.

SPRU321 — *Code Composer Studio Application Programming Interface Reference Guide.* Describes the Code Composer Studio™ application programming interface (API), which allows you to program custom plug-ins for Code Composer.

SPRU871 — *TMS320C64x+ Megamodule Reference Guide.* Describes the TMS320C64x+ digital signal processor (DSP) megamodule. Included is a discussion on the internal direct memory access (IDMA) controller, the interrupt controller, the power-down controller, memory protection, bandwidth management, and the memory and cache.

SPRUEC6 — *TMS320C645x/C647x Bootloader User's Guide.* This document describes the features of the on-chip Bootloader provided with the TMS320C645x/C647x digital signal processors (DSPs).

C6457 SRIO

1 Overview

The RapidIO peripheral used in the TMS320C6457 device is called a serial RapidIO (SRIO). This section describes the general operation of a RapidIO system, how this module is connected to the outside world, the definitions of terms used within this document, and the features supported and not supported for SRIO.

1.1 General RapidIO System

RapidIO is a non-proprietary high-bandwidth system level interconnect. It is a packet-switched interconnect intended primarily as an intra-system interface for chip-to-chip and board-to-board communications at Gigabyte-per-second performance levels. Uses for the architecture can be found in connected microprocessors, memory, and memory mapped I/O devices that operate in networking equipment, memory subsystems, and general purpose computing. Principle features of RapidIO include:

- Flexible system architecture allowing peer-to-peer communication
- Robust communication with error detection features
- Frequency and port width scalability
- Operation that is not software intensive
- High bandwidth interconnect with low overhead
- Low pin count
- Low power
- Low latency

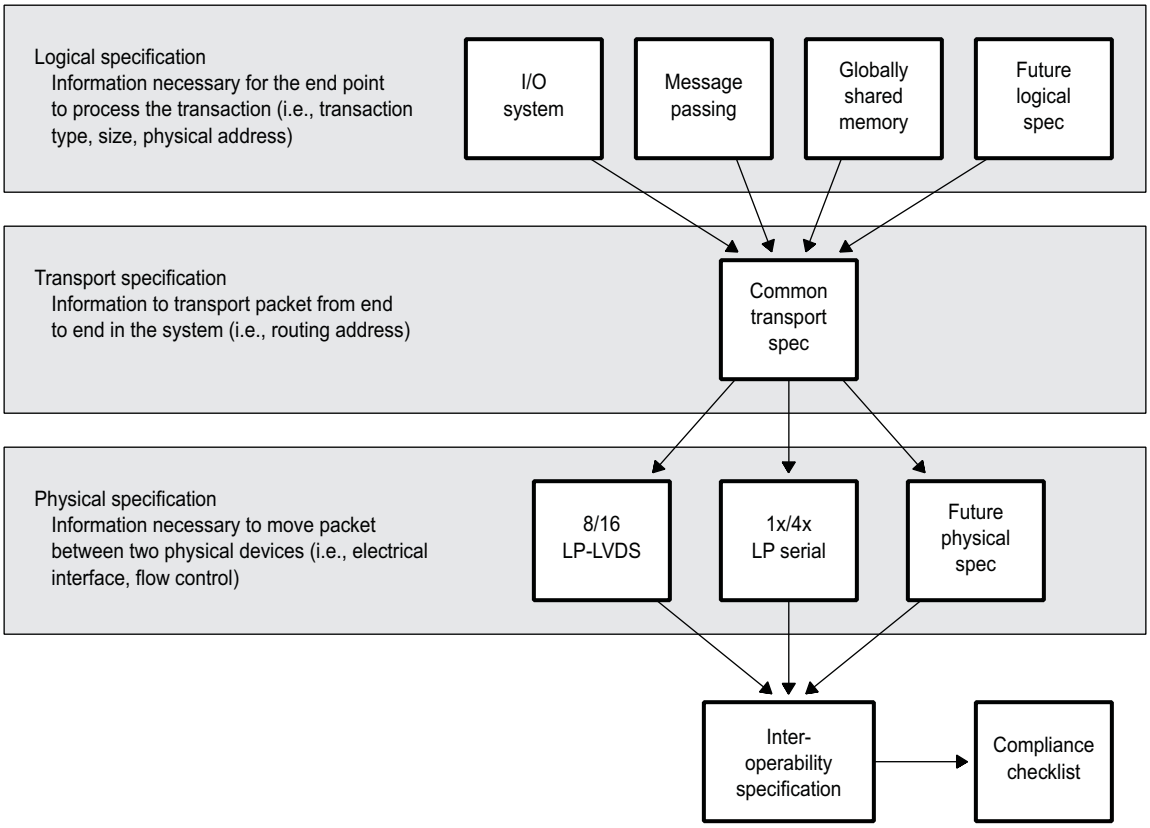
1.1.1 RapidIO Architectural Hierarchy

RapidIO is defined as a 3-layer architectural hierarchy.

- **Logical layer:** Specifies the protocols, including packet formats, which are needed by endpoints to process transactions
- **Transport layer:** Defines addressing schemes to correctly route information packets within a system
- **Physical layer:** Contains the device level interface information such as the electrical characteristics, error management data, and basic flow control data

In the RapidIO architecture, a single specification for the transport layer is compatible with differing specifications for the logical and physical layers (see Figure 1).

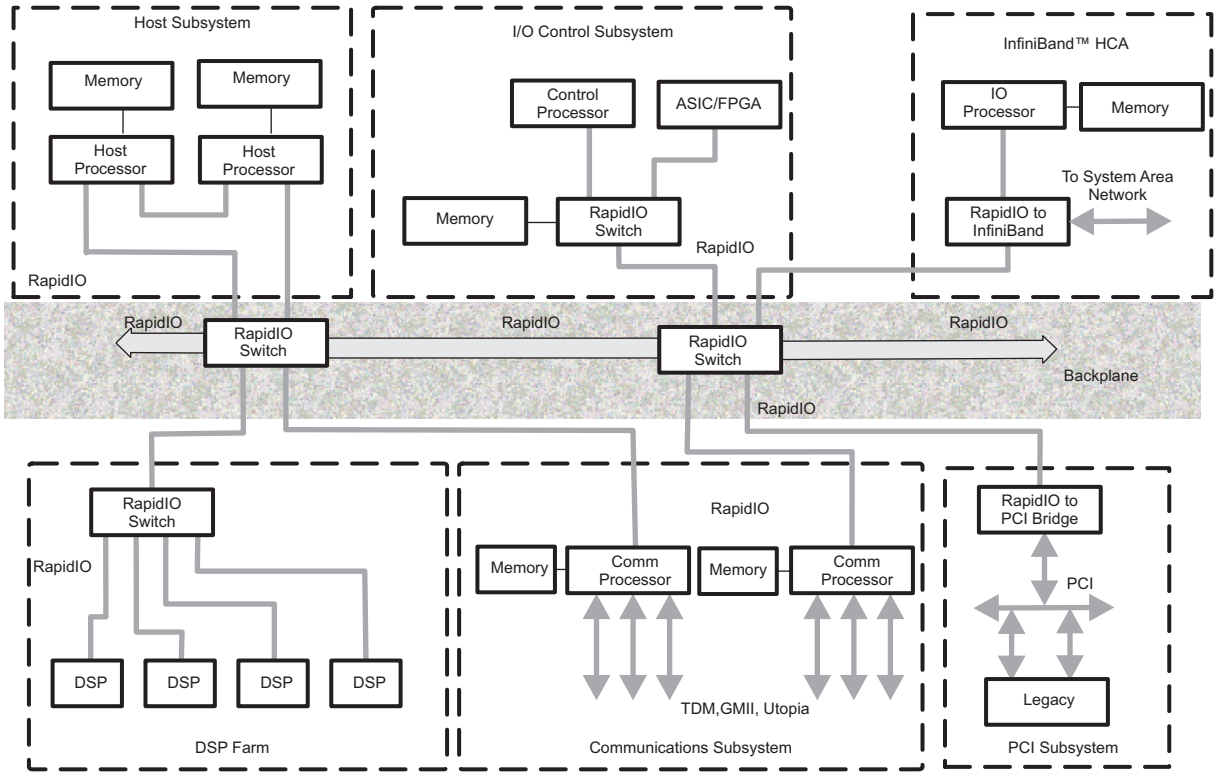
Figure 1. RapidIO Architectural Hierarchy



1.1.2 RapidIO Interconnect Architecture

The interconnect architecture is defined as a packet switched protocol independent of a physical layer implementation. Figure 2 illustrates the interconnection system.

Figure 2. RapidIO Interconnect Architecture



A InfiniBand™ is a trademark of the InfiniBand Trade Association.

1.1.3 Physical Layer 1x/4x LP-Serial Specification

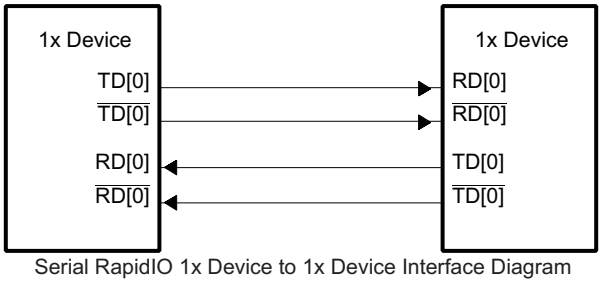
Currently, there are two physical layer specifications recognized by the RapidIO Trade Association: 8/16 LP-LVDS and 1x/4x LP-Serial. The 8/16 LP-LVDS specification is a point-to-point synchronous clock sourcing DDR interface. The 1x/4x LP-Serial specification is a point-to-point, AC coupled, clock recovery interface. The two physical layer specifications are not compatible.

SRIO complies with the 1x/4x LP-Serial specification. The serializer/deserializer (SERDES) technology in SRIO also aligns with that specification.

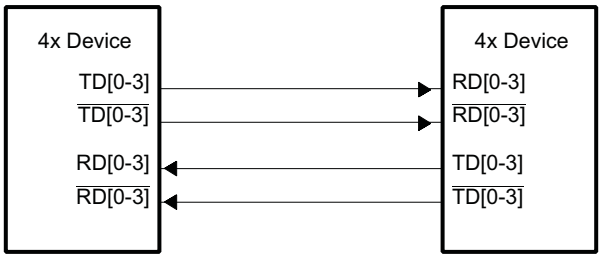
The RapidIO *Physical Layer 1x/4x LP-Serial Specification* currently covers three frequency points: 1.25, 2.5, and 3.125 Gbps. This defines the total bandwidth of each differential pair of I/O signals. An 8-bit/10-bit encoding scheme ensures ample data transitions for the clock recovery circuits. Due to the 8-bit/10-bit encoding overhead, the effective data bandwidth per differential pair is 1.0, 2.0, and 2.5 Gbps respectively. Serial RapidIO only specifies these rates for both the 1x and 4x ports. A 1x port is defined as 1 TX and 1 RX differential pair. A 4x port is a combination of four of these pairs. This document describes a 4x RapidIO port that can also be configured as four 1x ports, thus providing a scalable interface capable of supporting a data bandwidth of 1 to 10 Gbps.

Figure 3 shows how to interface two 1x devices and two 4x devices. Each *positive* transmit data line (TDx) on one device is connected to a *positive* receive data line (RDx) on the other device. Likewise, each *negative* transmit data line ($\overline{\text{TDx}}$) is connected to a *negative* receive data line ($\overline{\text{RDx}}$).

Figure 3. Serial RapidIO Device-to-Device Interface Diagrams



Serial RapidIO 1x Device to 1x Device Interface Diagram



Serial RapidIO 4x Device to 4x Device Interface Diagram

1.2 RapidIO Feature Support in SRIO

Features Supported in the SRIO Peripheral:

- *RapidIO Interconnect Specification* V1.2 compliance, Errata 1.2
- *Physical Layer 1x/4x LP-Serial Specification* V1.2 compliance
- 4x Serial RapidIO with auto-negotiation to 1x port, optional operation for four 1x ports
- Integrated clock recovery with TI SERDES
- Hardware error handling including Cyclic Redundancy Code (CRC)
- Differential CML signaling supporting AC coupling
- Support for 1.25, 2.5, and 3.125 Gbps rates
- Power-down option for unused ports
- Read, write, write with response, streaming write, outgoing Atomic, and maintenance operations
- Generates interrupts to the CPU (Doorbell packets and internal scheduling)
- Support for 8-bit and 16-bit device ID
- Support for receiving 34-bit addresses
- Support for generating 34-bit, 50-bit, and 66-bit addresses
- Support for the following data sizes: byte, half-word, word, double-word
- Big endian data transfers
- Little endian swapping at 1 byte, 2 byte, 4 byte, and 8 byte boundaries
- Direct I/O transfers
- Message passing transfers
- Data payloads of up to 256 bytes
- Single messages consisting of up to 16 packets
- Elastic storage FIFOs for clock domain handoff
- Short run and long run compliance
- Support for Error Management Extensions
- Support for Congestion Control Extensions
- Support for three multicast IDs

Features Not Supported:

- Compliance with the Global Shared Memory specification (GSM)
- 8/16 LP-LVDS compatible

- Destination support of RapidIO Atomic Operations
- Simultaneous mixing of frequencies between 1x ports (all ports must be the same frequency)
- Target atomic operations (including increment, decrement, test-and-swap, set, and clear) for internal L2 memory and registers

1.3 Standards

The SRIO peripheral is compliant to V1.2 of the *RapidIO Interconnect Specification* and V1.2 of the RapidIO *Physical Layer 1x/4x LP-Serial Specification*. These and the various associated documents listed herein can be found at the official RapidIO website: www.RapidIO.org.

1.4 External Devices Requirements

SRIO provides a seamless interface to all devices which are compliant to V1.2 of the RapidIO *Physical Layer 1x/4x LP-Serial Specification*. This includes ASIC, microprocessor, DSP, and switch fabric devices from multiple vendors. Compliance to the specification can be verified with bus-functional models available through the RapidIO Trade Association, as well as test suites currently available for licensing.

1.5 TMS320C6457 SRIO Interface Characteristics

Table 1. TMS320C6457 SRIO Interface Characteristics

Number of DSP Cores (CPUs)	Number of Ports	Number of Lanes	Configurations	SRIO Module Frequency
1	4	4	1x/4x, 1x/1x	DSP frequency ÷ 3

2 SRIO Functional Description

2.1 Overview

2.1.1 Peripheral Data Flow

This peripheral is designed to be an externally driven slave module that is capable of acting as a master in the DSP system. This means that an external device can push (burst write) data to the DSP as needed, without having to generate an interrupt to the CPU or without relying on the DSP EDMA. This has several benefits. It cuts down on the total number of interrupts, it reduces handshaking (latency) associated with read-only peripherals, and it frees up the EDMA for other tasks.

SRIO specifies data packets with payloads up to 256 bytes. Many times, transactions will span across multiple packets. RapidIO specifies a maximum of 16 transactions per message. Although a request is generated for each packet transaction so that the DMA can transfer the data to L2 memory, an interrupt is only generated after the final packet of the message. This interrupt notifies the CPU that data is available in L2 Memory for processing.

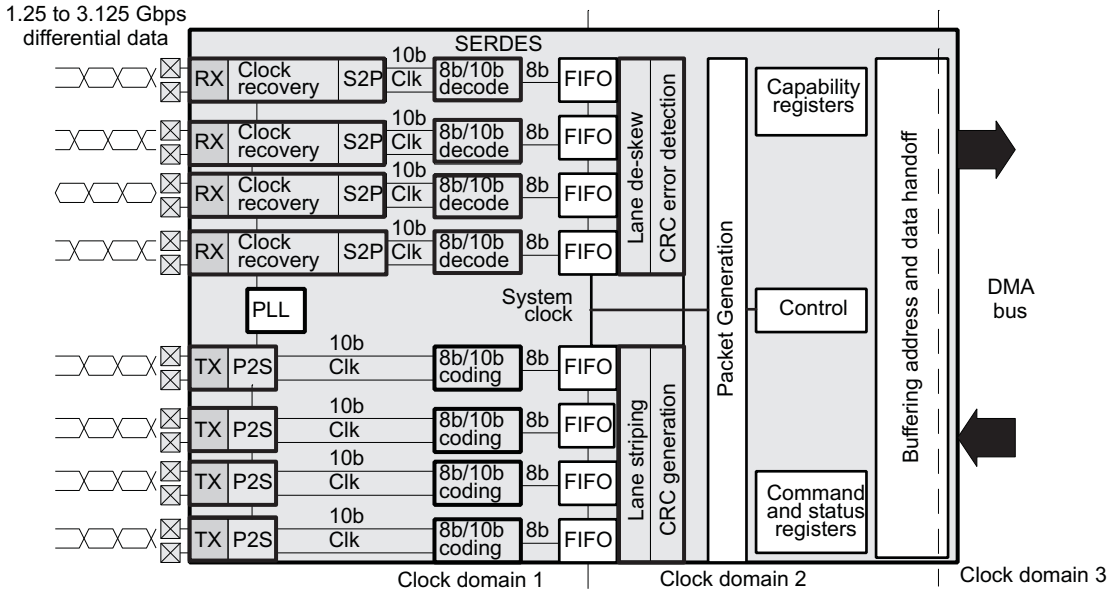
As an endpoint device, the peripheral accepts packets based on the destination ID. Two options exist for packet acceptance and are mode selectable. The first option is to only accept packets whose DESTIDs match the local deviceID in 0x0080. This provides a level of security. The second option is system multicast operation. When multicast is enabled in SP_IP_MODE (offset 12004h) bit 5, incoming packets matching the deviceID in the registers shown in Table 2 are accepted.

Table 2. Registers Checked for Multicast DeviceID

Name	Address Offset
Local DeviceID Register	0080h
Multicast DeviceID Register	0084h
Multicast DeviceID Register	0088h
Multicast DeviceID Register	008Ch

Data flow through the peripheral can be explained using the high-level block diagram shown in Figure 4. High-speed data enters from the device pins into the RX block of the SERDES macro. The RX block is a differential receiver expecting a minimum of 175 mV peak-to-peak differential input voltage (Vid). Level shifting is performed in the RX block, such that the output is single ended CMOS. The serial data is then fed to the SERDES clock recovery block. The sole purpose of this block is to extract a clock signal from the data stream. To do this, a low-frequency reference clock is required. Typically, this clock comes from an off-chip stable crystal oscillator and is a LVDS device input separate to the SERDES. This clock is distributed to the SERDES PLL block which multiplies that frequency up to that of the data rate. Multiple high-speed clock phases are created and routed to the clock recovery blocks. The clock recovery blocks further interpolate between these clocks to provide maximum Unit Interval (UI) resolution on the recovered clock. The clock recovery block samples the incoming data and monitors the relative positions of the data edges. With this information, it can provide the data and a center-aligned clock to the S2P block. The S2P block uses the newly recovered clock to de-multiplex the data into 10-bit words. At this point, the data leaves the SERDES macro at 1/10th the pin data rate, accompanied by an aligned byte clock.

Figure 4. SRIO Peripheral Block Diagram



For the number of ports, see Table 1.

Within the physical layer, the data next goes to the 8-bit/10-bit (8b/10b) decode block. 8b/10b encoding is used by RapidIO to ensure adequate data transitions for the clock recovery circuits. Here the 20% encoding overhead is removed as the 10-bit data is decoded to the raw 8-bit data. At this point, the recovered byte clock is still being used.

The next step is clock synchronization and data alignment. These functions are handled by the FIFO and lane de-skewing blocks. In the *RapidIO Interconnect Specification*, a lane is one serial differential pair. The FIFO provides an elastic store mechanism used to hand off between the recovered clock domains and a common system clock. After the FIFO, the four lanes are synchronized in frequency and phase, whether 1X or 4X mode is being used. The FIFO is 8 words deep. The lane de-skew is only meaningful in the 4X mode, where it aligns each channel's word boundaries, such that the resulting 32-bit word is correctly aligned.

The CRC error detection block keeps a running tally of the incoming data and computes the expected CRC value for the 1X or 4X mode. The expected value is compared against the CRC value at the end of the received packet.

After the packet reaches the logical layer, the packet fields are decoded and the payload is buffered. Depending on the type of received packet, the packet routing is handled by functional blocks which control the DMA access.

2.1.2 SRIO Packets

The SRIO data stream consists of data fields pertaining to the logical layer, the transport layer, and the physical layer.

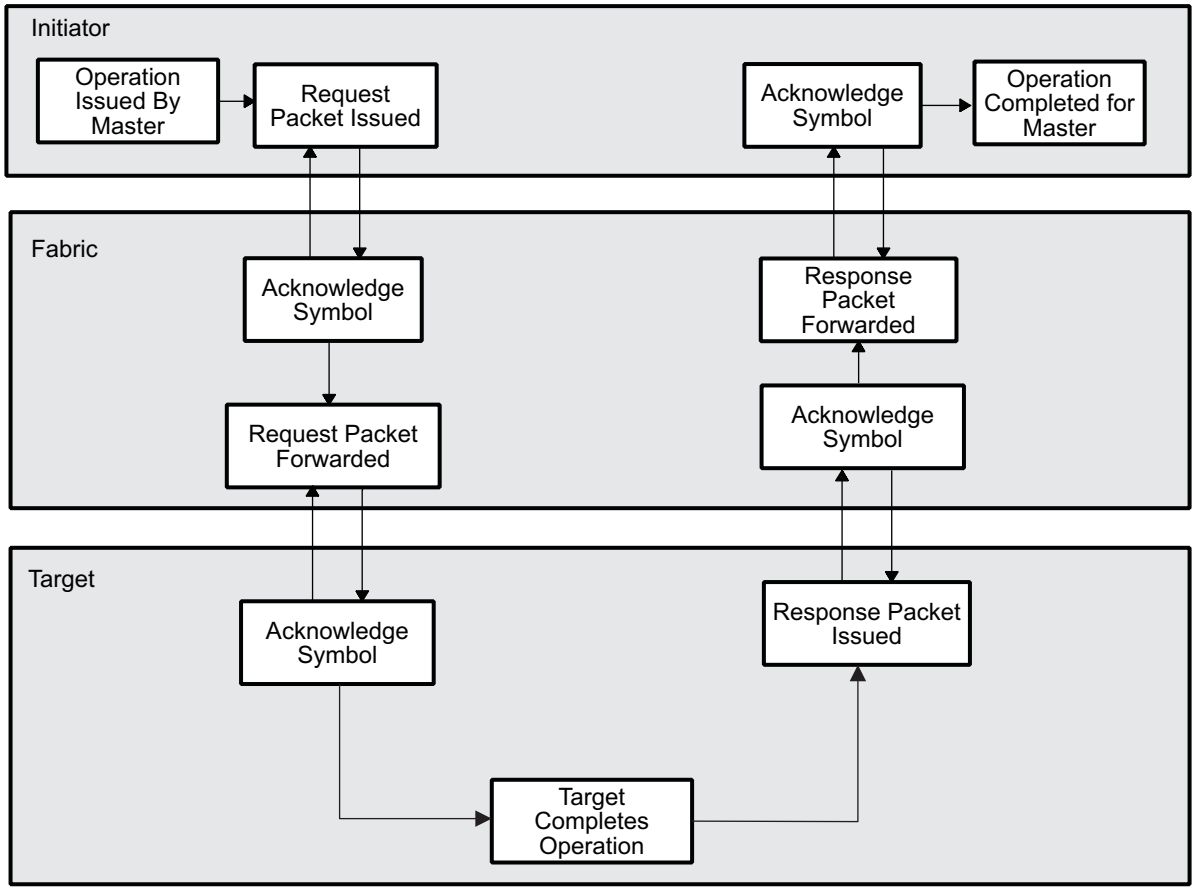
- The logical layer consists of the header (defining the type of access) and the payload (if present).
- The transport layer is partially dependent on the physical topology in the system, and consists of source and destination IDs for the sending and receiving devices.
- The physical layer is dependent on the physical interface (i.e., serial versus parallel RapidIO) and includes priority, acknowledgment, and error checking fields.

2.1.2.1 Operation Sequence

SRIO transactions are based on request and response packets. Packets are the communication element between endpoint devices in the system. A master or initiator generates a request packet which is transmitted to a target. The target then generates a response packet back to the initiator to complete the transaction.

SRIO endpoints are typically not connected directly to each other but instead have intervening connection fabric devices. Control symbols are used to manage the flow of transactions in the SRIO physical interconnect. Control symbols are used for packet acknowledgment, flow control information, and maintenance functions. Figure 5 shows how a packet progresses through the system.

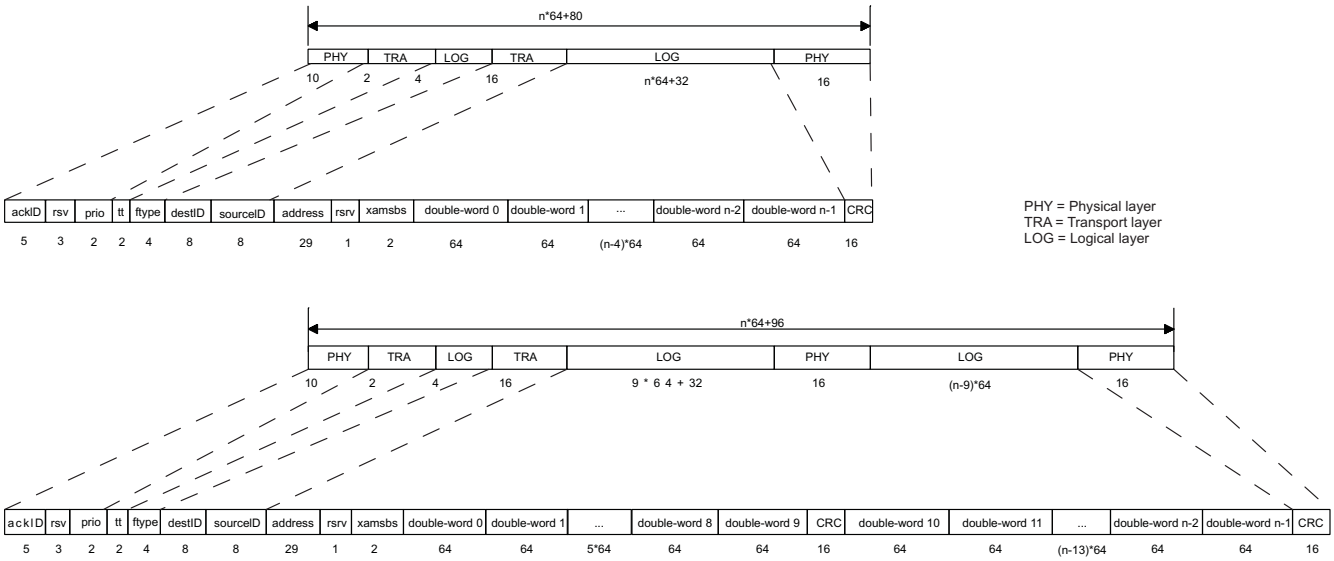
Figure 5. Operation Sequence



2.1.2.2 Example Packet - Streaming Write

An example packet is shown as two data streams in Figure 6. The first is for payload sizes of 80 bytes or less, while the second applies to payload sizes of 80 to 256 bytes. SRIO packets must have a length that is an even integer of 32 bits. If the combination of physical, logical and transport layers has a length that is an integer of 16 bits, a 16-bit pad of value 0000h is added to the end of the packet, after the CRC (not shown). Bit fields that are defined as reserved are assigned to logic 0s when generated and ignored when received. All request and response packet formats are described in the RapidIO *Input/Output Logical Specification* and *Message Passing Logical Specification*.

Figure 6. 1x/4x RapidIO Packet Data Stream (Streaming-Write Class)



NOTE: Figure 6 assumes that addresses are 32-bit and device IDs are 8-bit.

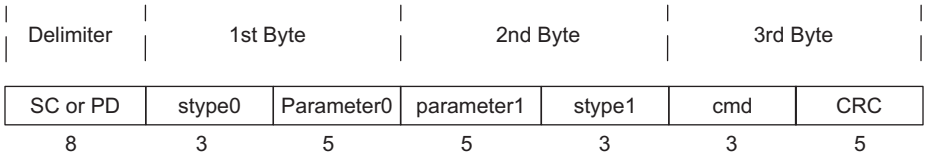
The device ID, being an 8-bit field, will address up to 256 nodes in the system. If 16-bit addresses were used, the system could accommodate up to 64k nodes.

The data stream includes a Cyclic Redundancy Code (CRC) field to ensure the data was correctly received. The CRC value protects the entire packet except the ackID and one bit of the reserved PHY field. The peripheral checks the CRC automatically in hardware. If the CRC is correct, a Packet-Accepted control symbol is sent by the receiving device. If the CRC is incorrect, a Packet-Not-Accepted control symbol is sent so that transmission may be retried.

2.1.2.3 Control Symbols

Control symbols are physical layer message elements used to manage link maintenance, packet delimiting, packet acknowledgment, error reporting, and error recovery. All transmitted data packets are delimited by start-of-packet and end-of-packet delimiters. SRIO control symbols are 24 bits long and are protected by their own CRC (see Figure 7). Control symbols provide two functions: stype0 symbols convey the status of the port transmitting the symbol, and stype1 symbols are requests to the receiving port or transmission delimiters. They have the following format, which is detailed in Section 3 of the RapidIO *Physical Layer 1x/4x LP-Serial Specification*.

Figure 7. Serial RapidIO Control Symbol Format



Control symbols are delimited by special characters at the beginning of the symbol. If the control symbol contains a packet delimiter(start-of-packet, end-of-packet, etc.), the special character PD (K28.3) is used. If the control symbol does not contain a packet delimiter, the special character SC (K28.0) is used. This use of special characters provides an early warning of the contents of the control symbol. The CRC does not protect the special characters, but an illegal or invalid character is recognized and flagged as Packet-Not-Accepted. Since control symbols are known length, they do not need end delimiters.

The type of received packet determines how the packet routing is handled. Reserved or undefined packet types are destroyed before being processed by the logical layer functional blocks. This prevents erroneous allocation of resources to them. Unsupported packet types are responded to with an error response packet. Section 2.1.2.4 details the handling of such packets.

2.1.2.4 SRIO Packet Type

The type of SRIO packet is determined by the combination of Ftype and Ttype fields in the packet. Table 3 lists all supported combinations of Ftype/Ttype and the corresponding decoded actions on the packets.

Table 3. Packet Types

Ftype	Ttype	Packet Type
Ftype = 0	Ttype = don't care	
Ftype = 2	Ttype = 0100b	NREAD
	Ttype = 1100b	Atomic increment
	Ttype = 1101b	Atomic decrement
	Ttype = 1110b	Atomic set
	Ttype = 1111b	Atomic clear
	Ttype = others	
Ftype = 5	Ttype = 0100b	NWRITE
	Ttype = 0101b	NWRITE_R
	Ttype = 1110b	Atomic test and swap
	Ttype = others	
Ftype = 6	Ttype = don't care	SWRITE
Ftype = 7 (1)	Ttype = don't care	Congestion control
Ftype = 8	Ttype = 0000b	Maintenance read
	Ttype = 0001b	Maintenance write
	Ttype = 0010b	Maintenance read response
	Ttype = 0011b	Maintenance write response
	Ttype = 0100b	Maintenance port-write
	Ttype = others	
Ftype = 10	Ttype = don't care	Doorbell
Ftype = 11	Ttype = don't care	Message
Ftype = 13	Ttype = 0000b	Response(+Doorbell Resp)
	Ttype = 0001b	Message Response
	Ttype = 1000b	Response w/payload
	Ttype = other	
Undefined Ftypes: 1,3,4,9,12,14,15		

(1) The TMS320C6457 device only supports receive operation of Ftype 7 packets; sending Ftype 7 packets is not supported.

2.2 SRIO Pins

The SRIO device pins are high-speed differential signals based on Current-Mode Logic (CML) switching levels. The transmit and receive buffers are self-contained within the clock recovery blocks. The reference clock input is not incorporated into the SERDES macro. It uses a differential input buffer that is compatible with the LVDS and LVPECL interfaces available from crystal oscillator manufacturers. Table 4 describes the device pins for the SRIO peripheral.

Table 4. Pin Description

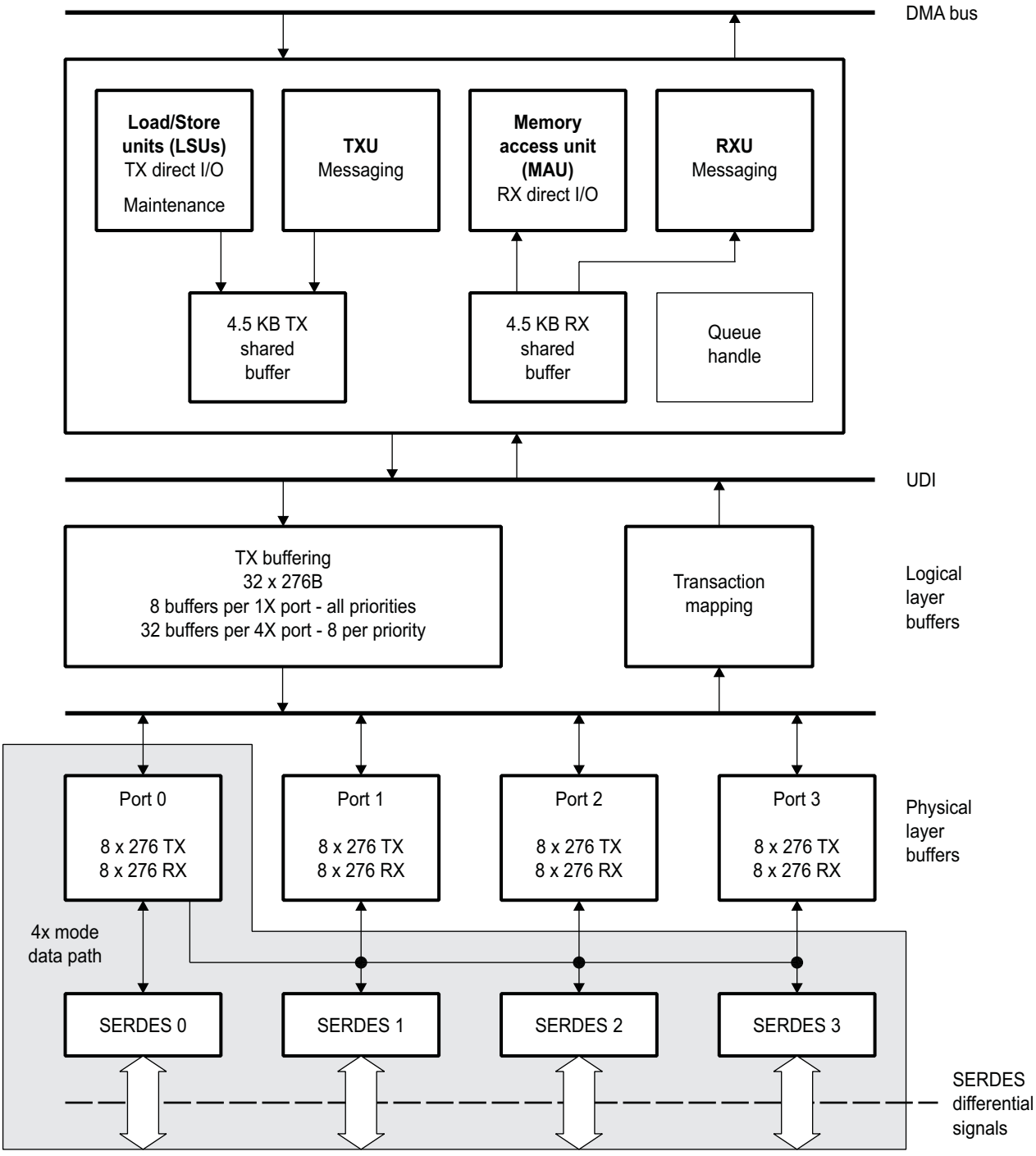
Pin Name	Pin Count	Signal Direction	Description
RIOTX3/$\overline{\text{RIOTX}}$3	2	Output	Transmit Data - Differential point-to-point unidirectional bus. Transmits packet data to a receiving device's RX pins. Most significant bits in 1 port 4X device. Used in 4 port 1X device.
RIOTX2/$\overline{\text{RIOTX}}$2	2	Output	Transmit Data - Differential point-to-point unidirectional bus. Transmits packet data to a receiving device's RX pins. Bit used in 4 port 1x device and 1 port 4X device.
RIOTX1/$\overline{\text{RIOTX}}$1	2	Output	Transmit Data - Differential point-to-point unidirectional bus. Transmits packet data to a receiving device's RX pins. Bit used in 4 port 1x device and 1 port 4X device.
RIOTX0/$\overline{\text{RIOTX}}$0	2	Output	Transmit Data - Differential point-to-point unidirectional bus. Transmits packet data to a receiving device's RX pins. Bit used in 1 port 1X device, 4 port 1x device, and 1 port 4X device.
RIORX3/$\overline{\text{RIORX}}$3	2	Input	Receive Data - Differential point-to-point unidirectional bus. Receives packet data for a transmitting device's TX pins. Most significant bits in 1 port 4X device. Used in 4 port 1X device.
RIORX2/$\overline{\text{RIORX}}$2	2	Input	Receive Data - Differential point-to-point unidirectional bus. Receives packet data for a transmitting device's TX pins. Bit used in 4 port 1x device and 1 port 4X device.
RIORX1/$\overline{\text{RIORX}}$1	2	Input	Receive Data - Differential point-to-point unidirectional bus. Receives packet data for a transmitting device's TX pins. Bit used in 4 port 1x device and 1 port 4X device.
RIORX0/$\overline{\text{RIORX}}$0	2	Input	Receive Data - Differential point-to-point unidirectional bus. Receives packet data for a transmitting device's TX pins. Bit used in 1 port 1X device, 4 port 1x device, and 1 port 4X device.
RIOCLK/$\overline{\text{RIOCLK}}$	2	Input	Reference Clock Input Buffer for peripheral clock recovery circuitry.

2.3 Functional Operation

2.3.1 Component Block Diagram

Figure 8 shows the SRIO peripheral component block diagram. The load/store unit (LSU) controls the transmission of direct I/O packets, and the memory access unit (MAU) controls the reception of direct I/O packets. The LSU also controls the transmission of maintenance packets. Message packets are transmitted by the TXU and received by the RXU. These four units use the internal DMA to communicate with internal memory, and they use buffers and receive/transmit ports to communicate with external devices. Serializer/deserializer (SERDES) macros support the ports by performing the parallel-to-serial coding for transmission and serial-to-parallel decoding for reception.

Figure 8. SRIO Component Block Diagram



2.3.2 SERDES Macro and Its Configurations

SRIO offers many benefits to customers by allowing a scalable non-proprietary interface. With the use of TI's SERDES macros, the peripheral is very adaptable and bandwidth scalable. The same peripheral can be used for all three frequency nodes specified in V1.2 of the *RapidIO Interconnect Specification* (1.25, 2.5, and 3.125 Gbps). This allows you to design to only one protocol throughout the system and selectively choose the bandwidth, thus eliminating the need for user's proprietary protocols in many instances, and providing a faster design turn and production ramp. Since this interface is serial, the application space is not limited to a single board. It will propagate into backplane applications as well. Integration of these macros on an ASIC or DSP allows you to reduce the number of discrete components on the board and eliminates the need for bus driver chips.

Additionally, there are some valuable features built into TI SERDES. System optimization can be uniquely managed to meet individual customer applications. For example, control registers within the SERDES allow you to adjust the TX differential output voltage (Vod) on a per driver basis. This allows power savings on short trace links (on the same board) by reducing the TX swing. Similarly, data edge rates can be adjusted through the control registers to help reduce any EMI affects. Unused links can be individually powered down without affecting the working links.

The SERDES macro is a self-contained macro which includes transmitter (TX), receiver (RX), phase-locked-loop (PLL), clock recovery, serial-to-parallel (S2P), and parallel-to-serial (P2S) blocks. The internal PLL multiplies a user-supplied reference clock. All loop filter components of the PLL are on-chip. Likewise, the differential TX and RX buffers contain on-chip termination resistors. The only off-chip component requirement is for DC blocking capacitors.

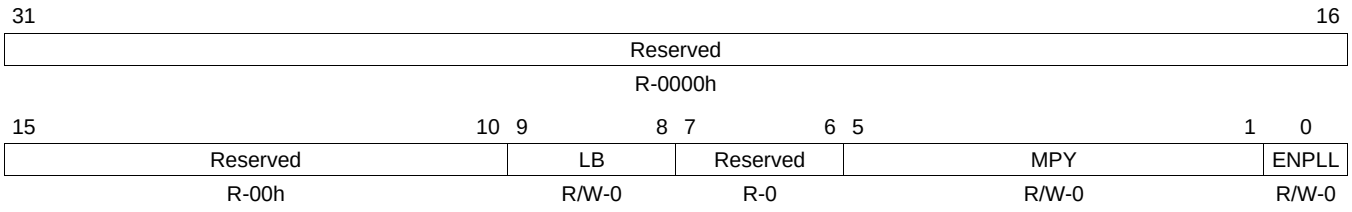
2.3.2.1 Enabling the PLL

The Physical layer SERDES has a built-in PLL, which is used for the clock recovery circuitry. The PLL is responsible for clock multiplication of a slow speed reference clock. This reference clock has no timing relationship to the serial data and is asynchronous to any CPU system clock. The multiplied high-speed clock is only routed within the SERDES block; it is not distributed to the remaining blocks of the peripheral, nor is it a boundary signal to the core of the device. It is extremely important to have a good quality reference clock, and to isolate it and the PLL from all noise sources. Since RapidIO requires 8-bit/10-bit encoded data, the 8-bit mode of the SERDES PLL is not be used.

The SERDES macro is configured with the register SERDES_CFG0_CNTL, SERDES_CFGRX*n*_CNTL, and SERDES_CFGTX*n*_CNTL, where *n* is the number of the macro. To enable the internal PLL, the ENPLL bit of SERDES_CFG0_CNTL (see Figure 9 and Table 5) must be set. After setting this bit, it is necessary to allow 1µs for the regulator to stabilize. Thereafter, the PLL will take no longer than 200 reference clock cycles to lock to the required frequency, provided RIOCLK and $\overline{\text{RIOCLK}}$ are stable.

Registers SERDES_CFG1_CNTL, SERDES_CFG2_CNTL, and SERDES_CFG3_CNTL are not used.

Figure 9. SERDES Macro Configuration Register 0 (SERDES_CFG0_CNTL)



LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 5. SERDES Macro Configuration Register 0 (SERDES_CFG0_CNTL) Field Descriptions

Bit	Field	Value	Description
31-10	Reserved	0000h	Reserved
9-8	LB		Loop bandwidth. Specify loop bandwidth settings. Jitter on the reference clock degrades both the transmit eye and receiver jitter tolerance, thereby, impairing system performance. Performance of the integrated PLL is optimized according to the jitter characteristics of the reference clock via the LB field.
		00b	Frequency-dependent bandwidth. The PLL bandwidth is set to 1/12 of the RIOCLK/$\overline{\text{RIOCLK}}$ frequency. This setting is suitable for most systems that input the reference clock via a low-jitter input cell and is required for standards compliance.
		01b	Reserved
		10b	Low bandwidth. The PLL bandwidth is set to 1/20 of the RIOCLK/$\overline{\text{RIOCLK}}$ frequency or 3 MHz, whichever is larger. In systems where the reference clock is directly input via a low-jitter input cell, but is of lower quality, this setting may offer better performance. It reduces the amount of reference clock jitter transferred through the PLL. However, it also increases the susceptibility to loop noise generated within the PLL itself. It is difficult to predict whether the improvement in the former will more than offset the degradation in the latter.
		11b	High bandwidth. The PLL bandwidth is set to 1/8 of the RIOCLK/$\overline{\text{RIOCLK}}$ frequency. This is the setting appropriate for systems where the reference clock is cleaned through an ultra-low-jitter LC-based PLL. Standards compliance is achieved even if the reference clock input to the cleaner PLL is outside the specification for the standard.
7-6	Reserved	00h	Reserved
5-1	MPY		PLL multiply. Select PLL multiply factors between 4 and 60.
		00000b	4x
		00001b	5x
		00010b	6x
		00011b	Reserved
		00100b	8x
		00101b	10x
		00110b	12x
		00111b	12.5x
		01000b	15x
		01001b	20x
		01010b	25x
		01011b	Reserved
		01100b	Reserved
		01111b	Reserved
		1xxxxb	Reserved
0	ENPLL		Enable PLL
		0	PLL disabled
		1	PLL enabled

Based on the MPY value, the line rate versus PLL output clock frequency can be calculated. This is summarized in Table 6.

Table 6. Line Rate versus PLL Output Clock Frequency

Rate	Line Rate	PLL Output Frequency	RATESCALE
Full	x Gbps	0.5x GHz	0.5
Half	x Gbps	x GHz	1
Quarter	x Gbps	2x GHz	2

$$\text{RIOCLK and } \overline{\text{RIOCLK}}_{FREQ} = \frac{LINERATE \times RATESCALE}{MPY}$$

The rate is defined by the RATE bits of the SERDES_CFGRX*n*_CNTL register and the SERDES_CFGTX*n*_CNTL register, respectively.

The primary operating frequency of the SERDES macro is determined by the reference clock frequency and PLL multiplication factor. However, to support lower frequency applications, each receiver and transmitter can also be configured to operate at a half or quarter of this rate via the RATE bits of the SERDES_CFGRX*n*_CNTL and SERDES_CFGTX*n*_CNTL registers as described in Table 7.

Table 7. Effect of the RATE Bits

RATE	Description
00b	Full rate. Two data samples taken per PLL output clock cycle.
01b	Half rate. One data sample taken per PLL output clock cycle.
10b	Quarter rate. One data sample taken every two PLL output clock cycles.
11b	Reserved.

Table 8 shows the frequency range versus the multiplication factor (MPY).

Table 8. Frequency Range versus MPY Value

MPY	RIOCLK and $\overline{\text{RIOCLK}}$ Range (MHz)	Line Rate Range (Gbps)		
		Full	Half	Quarter
4	312.5	2.50	1.25	N/A
5	312.5	3.125	N/A	N/A
8	156.25	2.50	1.25	N/A
10	156.25	3.125	N/A	N/A
10	125	2.50	1.25	N/A
12.5	125	3.125	N/A	N/A

2.3.2.2 Enabling the Receiver

To enable a receiver for deserialization, the ENRX bit of the associated SERDES_CFGRX*n*_CNTL registers (100h-10Ch) must be set high. The fields of SERDES_CFGRX*n*_CNTL are shown in Figure 10 and described in Table 9.

When ENRX is low, all digital circuitry within the receiver will be disabled, and clocks will be gated off. All current sources within the receiver will be fully powered down, with the exception of those associated with the loss of signal detector and IEEE1149.6 boundary scan comparators. Loss of signal power down is independently controlled via the LOS bits of SERDES_CFGRX*n*_CNTL. When enabled, the differential signal amplitude of the received signal is monitored. Whenever loss of signal is detected, the clock recovery algorithm is frozen to prevent the phase and frequency of the recovered clock from being modified by low level signal noise.

The clock recovery algorithms listed in the CDR bits operate to adjust the clocks used to sample the received message so that the data samples are taken midway between data transitions. The second order algorithm can be optionally disabled, and both can be configured to optimize their dynamics. Both algorithms use the same basic technique for determining whether the sampling clock is ideally placed, and if not whether it needs to be moved earlier or later. When two contiguous data samples are different, the phase sample between the two is examined. Eight data samples and nine phase samples are taken with each result counted as a vote to move the sample point either earlier or later. These eight data bits constitute the voting window. The eight votes are then counted, and an action to adjust the position of the sampling clock occurs if there is a majority of early or late votes. The first order algorithm makes a single phase adjustment per majority vote. The second order algorithm acts repeatedly according to the net difference between early and late majority votes, thereby adjusting for the rate of change of phase.

Setting the ALIGN field to 01 enables alignment to the K28 comma symbols included in the 8b:10b data encoding scheme defined by the IEEE and employed by numerous transmission standards. For systems which cannot use comma based symbol alignment, the single bit alignment jog capability provides a means to control the symbol realignment features of the receiver directly from logic implemented in the ASIC core. This logic can be designed to support whatever alignment detection protocol is required.

The EQ bits allow for enabling and configuring the adaptive equalizer incorporated in all of the receive channels, which can compensate for channel insertion loss by attenuating the low frequency components with respect to the high frequency components of the signal, thereby reducing inter-symbol interference. Above the zero frequency, the gain increases at 6dB/octave until it reaches the high frequency gain. When enabled, the receiver equalization logic analyzes data patterns and transition times to determine whether the low frequency gain of the equalizer should be increased or decreased. For the fully adaptive setting (EQ = 0001), if the low frequency gain reaches the minimum value, the zero frequency is then reduced. Likewise, if it reaches the maximum value, the zero frequency is then increased. This decision logic is implemented as a voting algorithm with a relatively long analysis interval. The slow time constant that results reduces the probability of incorrect decisions but allows the equalizer to compensate for the relatively stable response of the channel.

- No adaptive equalization. The equalizer provides a flat response at the maximum gain. This setting may be appropriate if jitter at the receiver occurs predominantly as a result of crosstalk rather than frequency dependent loss.
- Fully adaptive equalization. Both the low frequency gain and zero position of the equalizer are determined algorithmically by analyzing the data patterns and transition positions in the received data. This setting should be used for most applications.
- Partially adaptive equalization. The low frequency gain of the equalizer is determined algorithmically by analyzing the data patterns and transition positions in the received data. The zero position is fixed in one of eight zero positions. For any given application, the optimal setting is a function of the loss characteristics of the channel and the spectral density of the signal as well as the data rate, which means it is not possible to identify the best setting by data rate alone, although generally speaking, the lower the line rate, the lower the zero frequency that will be required.

Figure 10. SERDES Receive Channel Configuration Register *n* (SERDES_CFGRX*n*_CNTL)

31–26	25–24	23	22–19	18–16
Reserved	Reserved (write 0s)	-	EQ	CDR
R-0	R/W-0	R-0	R/W-0	R/W-0

15–14	13–12	11	10–8	7	6–5	4–2	1	0
LOS	ALIGN	-	TERM (write 001b)	INVPAIR	RATE	BUSWIDTH	- (write 0)	ENRX
R/W-0	R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 9. SERDES Receive Channel Configuration Register *n* (SERDES_CFGRX*n*_CNTL) Field Descriptions

Bit	Field	Value	Description
31-26	Reserved	000000b	These read-only bits return 0s when read.
25-24	Reserved	00b	Always write 0s to these reserved bits.
23	Reserved	0	This read-only bit returns 0 when read.
22-19	EQ	0000b-1111b	Equalizer. Enables and configures the adaptive equalizer to compensate for loss in the transmission media. For the selectable values, see Table 10.
18-16	CDR		Clock/data recovery. Configures the clock/data recovery algorithm.
		000b	First order. Phase offset tracking up to ±488 ppm.
		001b	Second order. Highest precision frequency offset matching but poorest response to changes in frequency offset, and longest lock time. Suitable for use in systems with fixed frequency offset.
		010b	Second order. Medium precision frequency offset matching, frequency offset change response, and lock time.
		011b	Second order. Best response to changes in frequency offset and fastest lock time, but lowest precision frequency offset matching. Suitable for use in systems with spread spectrum clocking.
		100b	First order with fast lock. Phase offset tracking up to ±1953 ppm in the presence of ..10101010.. training pattern and ±448 ppm, otherwise.
		101b	Second order with fast lock. As per setting 001, but with improved response to changes in frequency offset when not close to lock.
		110b	Second order with fast lock. As per setting 010, but with improved response to changes in frequency offset when not close to lock.
		111b	Second order with fast lock. As per setting 011, but with improved response to changes in frequency offset when not close to lock.
15-14	LOS		Loss of signal. Enables loss of signal detection with 2 selectable thresholds.
		00b	Disabled. Loss of signal detection disabled.
		01b	High threshold. Loss of signal detection threshold in the range 85 to 195mV$_{dfpp}$. This setting is suitable for Infiniband.
		10b	Low threshold. Loss of signal detection threshold in the range 65 to 175mV$_{dfpp}$. This setting is suitable for PCI-E and S-ATA.
		11b	Reserved
13-12	ALIGN		Symbol alignment. Enables internal or external symbol alignment.
		00b	Alignment disabled. No symbol alignment will be performed while this setting is selected, or when switching to this selection from another.
		01b	Comma alignment enabled. Symbol alignment will be performed whenever a misaligned comma symbol is received.
		10b	Alignment jog. The symbol alignment will be adjusted by one bit position when this mode is selected (that is, the ALIGN value changes from 0xb to 1xb).
		11b	Reserved
11	Reserved	0	This read-only bit returns 0 when read.

Table 9. SERDES Receive Channel Configuration Register *n* (SERDES_CFGRX*n*_CNTL) Field Descriptions (continued)

Bit	Field	Value	Description
10-8	TERM	001b	Input termination. The only valid value for this field is 001b; all other values are reserved. The value 001b sets the common point to 0.8 VDDT and supports AC coupled systems using CML transmitters. The transmitter has no effect on the receiver common mode, which is set to optimize the input sensitivity of the receiver. Common mode termination is via a 50 pF capacitor to VSSA.
7	INVPAIR		Invert polarity. Inverts polarity of RIORXn and $\overline{\text{RIORXn}}$.
		0	Normal polarity. RIORXn is considered to be positive data and $\overline{\text{RIORXn}}$ negative.
		1	Inverted polarity. RIORXn is considered to be negative data and $\overline{\text{RIORXn}}$ positive.
6-5	RATE		Operating rate. Selects full, half, or quarter rate operation.
		00b	Full rate. Two data samples taken per PLL output clock cycle.
		01b	Half rate. One data sample taken per PLL output clock cycle.
		10b	Quarter rate. One data sample taken every two PLL output clock cycles.
		11b	Reserved
4-2	BUSWIDTH	000b	Bus width. Always write 000b to this field, to indicate a 10-bit-wide parallel bus to the clock. All other values are reserved. See Section 2.3.2.1 for an explanation of the bus.
1	Reserved	0	Always write 0 to this reserved bit.
0	ENRX		Enable receiver
		0	Disable this receiver.
		1	Enable this receiver.

Table 10. EQ Bits

CFGRX[22-19]	Low Frequency Gain	Zero Frequency (at e_{28} (min))
0000b	Maximum	-
0001b	Adaptive	Adaptive
001xb	Reserved	
01xxb	Reserved	
1000b	Adaptive	1084 MHz
1001b		805 MHz
1010b		573 MHz
1011b		402 MHz
1100b		304 MHz
1101b		216 MHz
1110b		156 MHz
1111b		135 MHz

2.3.2.3 Enabling the Transmitter

To enable a transmitter for serialization, the ENTX bit of the associated SERDES_CFGTX*n*_CNTL registers (110h-10Ch) must be set high. When ENTX is low, all digital circuitry within the transmitter will be disabled, and clocks will be gated off, with the exception of the transmit clock (TXBCLK[*n*]) output, which will continue to operate normally. All current sources within the transmitter will be fully powered down, with the exception of the current mode logic (CML) driver, which will remain powered up if boundary scan is selected. Figure 11 shows the fields of SERDES_CFGTX*n*_CNTL and Table 11 describes them.

Figure 11. SERDES Transmit Channel Configuration Register *n* (SERDES_CFGTX*n*_CNTL)

31	17	16
Reserved		ENFTP
R-0		R/W-1

15 12	11 9	8	7	6 5	4 2	1	0
DE	SWING	CM	INVPAIR	RATE	BUSWIDTH	(write 0)	ENTX
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 11. SERDES Transmit Channel Configuration Register *n* (SERDES_CFGTX*n*_CNTL) Field Descriptions

Bit	Field	Value	Description
31-17	Reserved	0	These read-only bits return 0s when read.
16	ENFTP	1	Enables fixed phase relationship of transmit input clock with respect to transmit output clock. The only valid value for this field is 1b; all other values are reserved.
15-12	DE	0000b-1111b	De-emphasis. Selects one of 15 output de-emphasis settings from 4.76 to 71.42%. De-emphasis provides a means to compensate for high frequency attenuation in the attached media. It causes the output amplitude to be smaller for bits which are not preceded by a transition than for bits which are. See Table 12.
11-9	SWING	000b-111b	Output swing. Selects one of 8 outputs amplitude settings between 125 and 1250mV$_{dfpp}$. See Table 13.
8	CM		Common mode. Adjusts the common mode to suit the termination at the attached receiver. For output swing settings above 750mV, this reduced common mode can cause distortion of the waveform. Under these conditions, this bit should be set high to offset some of the common mode reduction.
		0	Normal common mode. Common mode not adjusted.
		1	Raised common mode. Common mode raised by 5% of difference between RIOTXn and $\overline{\text{RIOTXn}}$
7	INVPAIR		Invert polarity. Inverts the polarity of RIOTXn and $\overline{\text{RIOTXn}}$.
		0	Normal polarity. RIOTXn is considered to be positive data and $\overline{\text{RIOTXn}}$ negative.
		1	Inverted polarity. RIOTXn is considered to be negative data and $\overline{\text{RIOTXn}}$ positive.
6-5	RATE		Operating rate. Selects full, half, or quarter rate operation.
		00b	Full rate. Two data samples taken per PLL output clock cycle.
		01b	Half rate. One data sample taken per PLL output clock cycle.
		10b	Quarter rate. One data sample taken every two PLL output clock cycles.
		11b	Reserved
4-2	BUSWIDTH	000b	Bus width. Always write 000b to this field, to indicate a 10-bit-wide parallel bus to the clock. All other values are reserved. See Section 2.3.2.1 for an explanation of the bus.
1	Reserved	0	Always write 0 to this reserved bit.
0	ENTX		Enable transmitter
		0	Disable this transmitter.
		1	Enable this transmitter.

Table 12. DE Bits of SERDES_CFGTX*n*_CNTL

DE Bits	Amplitude Reduction	
	%	dB
0000b	0	0
0001b	4.76	-0.42
0010b	9.52	-0.87
0011b	14.28	-1.34
0100b	19.04	-1.83
0101b	23.8	-2.36
0110b	28.56	-2.92
0111b	33.32	-3.52
1000b	38.08	-4.16
1001b	42.85	-4.86
1010b	47.61	-5.61
1011b	52.38	-6.44
1100b	57.14	-7.35
1101b	61.9	-8.38
1110b	66.66	-9.54
1111b	71.42	-10.87

Table 13. SWING Bits of SERDES_CFGTX*n*_CNTL

SWING Bits	Amplitude (mV_{dfpp})
000b	125
001b	250
010b	500
011b	625
100b	750
101b	1000
110b	1125
111b	1250

2.3.2.4 SERDES Configuration Example

```
//full sample rate at 3.125 Gbps
//SERDES reference clock (RIOCLK) 125 MHz
//MPY = 12.5       125MHz = ((3.125 Gbps)(.5))/MPY

SRIO_REGS->SERDES_CFG0_CNTL = 0x0000000F;
SRIO_REGS->SERDES_CFG1_CNTL = 0x00000000;
SRIO_REGS->SERDES_CFG2_CNTL = 0x00000000;
SRIO_REGS->SERDES_CFG3_CNTL = 0x00000000;
// SRIO_REGS->SERDES_CFG1_CNTL not used
// SRIO_REGS->SERDES_CFG2_CNTL not used
// SRIO_REGS->SERDES_CFG3_CNTL not used

//four ports enabled
SRIO_REGS->SERDES_CFGRX0_CNTL  = 0x00081101 ;
SRIO_REGS->SERDES_CFGRX1_CNTL  = 0x00081101 ;
SRIO_REGS->SERDES_CFGRX2_CNTL  = 0x00081101 ;
SRIO_REGS->SERDES_CFGRX3_CNTL  = 0x00081101 ;
SRIO_REGS->SERDES_CFGTX0_CNTL  = 0x00010801 ;
SRIO_REGS->SERDES_CFGTX1_CNTL  = 0x00010801 ;
SRIO_REGS->SERDES_CFGTX2_CNTL  = 0x00010801 ;
SRIO_REGS->SERDES_CFGTX3_CNTL  = 0x00010801 ;
```

For the number of ports, see Table 1.

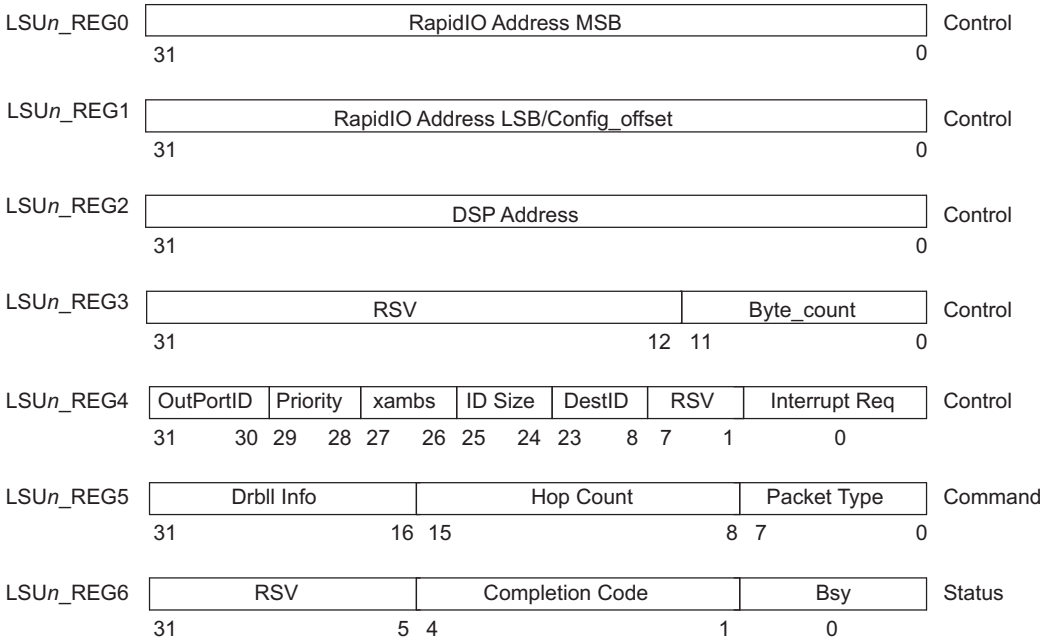
For an SRIO initialization example, see Section A.1.

2.3.3 Direct I/O Operation

The direct I/O (Load/Store) module serves as the source of all outgoing direct I/O packets. With direct I/O, the RapidIO packet contains the specific address where the data should be stored or read in the destination device. Direct I/O requires that a RapidIO source device keep a local table of addresses for memory within the destination device. Once these tables are established, the RapidIO source controller uses this data to compute the destination address and insert it into the packet header. The RapidIO destination peripheral extracts the destination address from the received packet header and transfers the payload to memory via the DMA.

When a CPU wants to send data from memory to an external processing element (PE) or read data from an external PE, it provides the RIO peripheral vital information about the transfer such as DSP memory address, target device ID, target destination address, packet priority, etc. Essentially, a means must exist to fill all the header fields of the RapidIO packet. The Load/Store module provides a mechanism to handle this information exchange via a set of MMRs acting as transfer descriptors. These registers, shown in Figure 12, are addressable by the CPU through the configuration bus. Upon completion of a write to LSU*n*_REG5, a data transfer is initiated for either an NREAD, NWRITE, NWRITE_R, SWRITE, ATOMIC, or MAINTENANCE RapidIO transaction. Some fields, such as the RapidIO *srcTID/targetTID* field, are assigned by hardware and do not have a corresponding command register field.

Figure 12. Load/Store Registers for RapidIO (Address Offset: LSU1 400h-418h, LSU2 420h-438h, LSU3 440h-458h, LSU4 460h-478h)



The mapping of LSU register fields to RapidIO packet header fields is explained in Table 14 and Table 15. Table 14 has the fields of the control and command registers (LSU*n*_REG0 through LSU*n*_REG5), and Table 15 has the fields of the status register (LSU*n*_REG6).

Table 14. LSU Control/Command Register Fields

LSU Register Field	RapidIO Packet Header Field
RapidIO Address MSB	32-bit Extended Address Fields - Packet Types 2, 5, and 6
RapidIO Address LSB/Config_offset	• 32-bit Address - Packet Types 2, 5, and 6 (Will be used in conjunction with BYTE_COUNT to create 64-bit aligned RapidIO packet header address) • 24-bit Config_offset Field - Maintenance Packets Type 8 (Will be used in conjunction with BYTE_COUNT to create 64-bit aligned RapidIO packet header Config_offset). The 2 LSBs of this field must be zero since the smallest configuration access is 4 bytes.
DSP Address	32-bit DSP byte address. Not available in RapidIO Header.

Table 14. LSU Control/Command Register Fields (continued)

LSU Register Field	RapidIO Packet Header Field
Byte_Count	Number of data bytes to Read/Write - up to 4K bytes. (Used in conjunction with RapidIO address to create WRSIZE/RDSIZE and WDPTR in RapidIO packet header.)
	000000000000b - 4K bytes
	000000000001b - 1 byte
	000000000010b - 2 bytes
	. . .
	111111111111b - 4095 bytes
	(Maintenance requests are limited to 4 bytes)
ID Size	RapidIO tt field specifying 8- or 16-bit DeviceIDs.
	00b - 8-bit deviceIDs
	01b - 16-bit deviceIDs
	10b - reserved
	11b - reserved
Priority	RapidIO prio field specifying packet priority (0 = lowest, 3 = highest). Request packets should not be sent at a priority level of 3 to avoid system deadlock. It is the responsibility of the software to assign the appropriate outgoing priority.
Xamsbs	RapidIO xamsb field specifying the extended address MSBs.
DESTID	RapidIO destinationID field specifying the target device.
Packet Type	4 MSBs: 4-bit ftype field for all packets
	4 LSBs: 4-bit trans field for packet types 2, 5, and 8
OutPortID	Not available in RapidIO header.
	Indicates the output port number for the packet to be transmitted from. Specified by the CPU along with NodeID.
Drbll Info	RapidIO doorbell info field for type 10 packets (see Table 23).
Hop Count	RapidIO hop_count field specified for Type 8 Maintenance packets.
Interrupt Req	Not available in RapidIO header.
	CPU controlled request bit used for interrupt generation. Typically used in conjunction with non-posted commands to alert the CPU when the requested data/status is present.
	0 - An interrupt is not requested upon completion of command
	1- An interrupt is requested upon completion of command

Table 15. LSU Status Register Fields

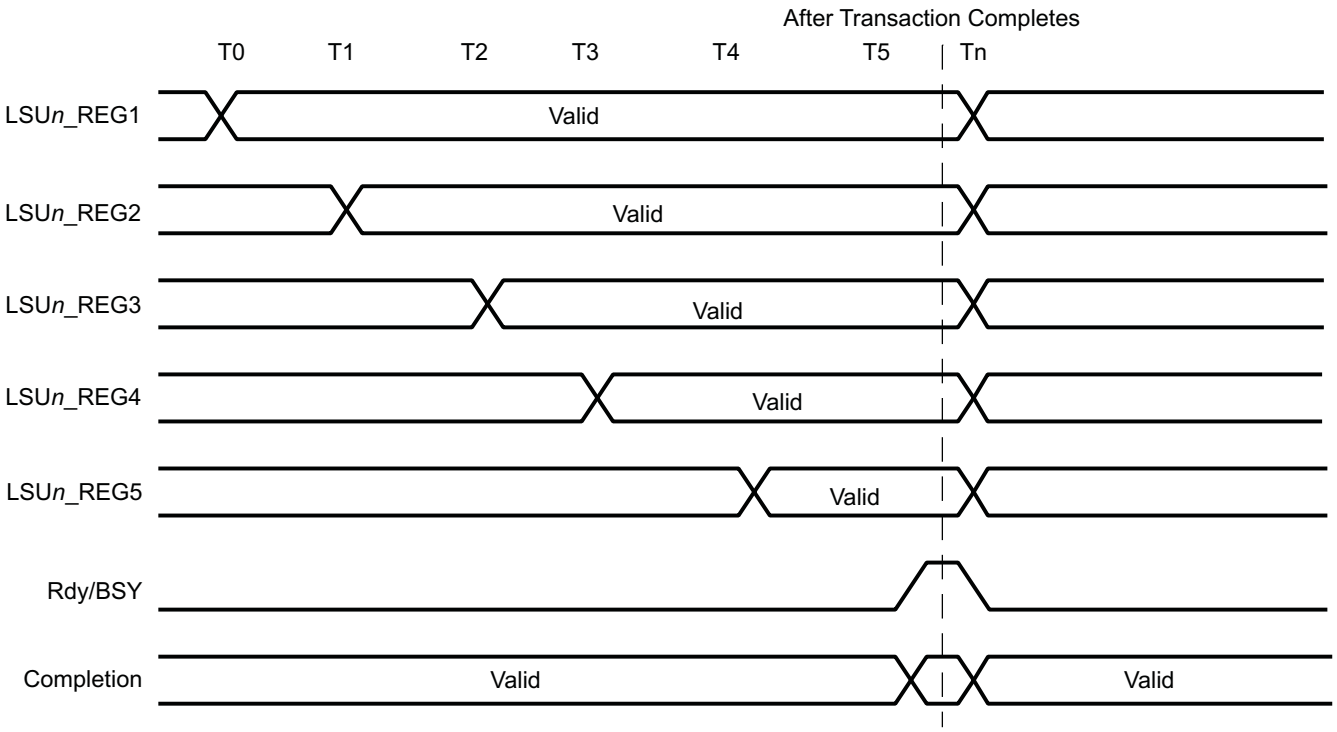
LSU Register Field	Function
BSY	Indicates status of the command registers.
	0 - Command registers are available (writable) for next set of transfer descriptors
	1 - Command registers are busy with current transfer
Completion Code	Indicates the status of the pending command.
	000b - Transaction complete, no errors (Posted/Non-posted)
	001b - Transaction timeout occurred on Non-posted transaction
	010b - Transaction complete, packet not sent due to flow control blockade (Xoff)
	011b - Transaction complete, non-posted response packet (type 8 and 13) contained ERROR status, or response payload length was in error
	100b - Transaction complete, packet not sent due to unsupported transaction type or invalid programming encoding for one or more LSU register fields
	101b - DMA data transfer error
	110b - Retry DOORBELL response received, or Atomic Test-and-swap was not allowed (semaphore in use)
	111b - Transaction complete, packet not sent due to unavailable outbound credit at given priority (1)

(1) Status available only when busy (BSY) signal = 0.

Four LSU register sets exist. This allows four outstanding requests for all transaction types that require a response (i.e., non-posted). For multi-core devices, software manages the usage of the registers. A shared configuration bus accesses all register sets. A single core device can utilize all four LSU blocks.

Figure 13 shows the timing diagram for accessing the LSU registers. The busy (BSY) signal is de-asserted. LSU*n*_REG1 is written on configuration bus clock cycle T0, LSU*n*_REG2 is written on cycle T1, LSU*n*_REG3 is written on cycle T2, and LSU*n*_REG4 is written on cycle T3. The command register LSU*n*_REG5 is written on cycle T4. The extended address field in LSU*n*_REG0 is assumed to be constant in this example. Upon completion of the write to the command register (next clock cycle T5), the BSY signal is asserted, at which point the preceding completion code is invalid and accesses to the LSU registers are not allowed. Once the transaction completes (either as a successful transmission, or unsuccessfully, such as flow control prevention or response timeout) and any required interrupt service routine is completed, the BSY signal is de-asserted and the completion code becomes valid and the registers are accessible again.

Figure 13. LSU Registers Timing

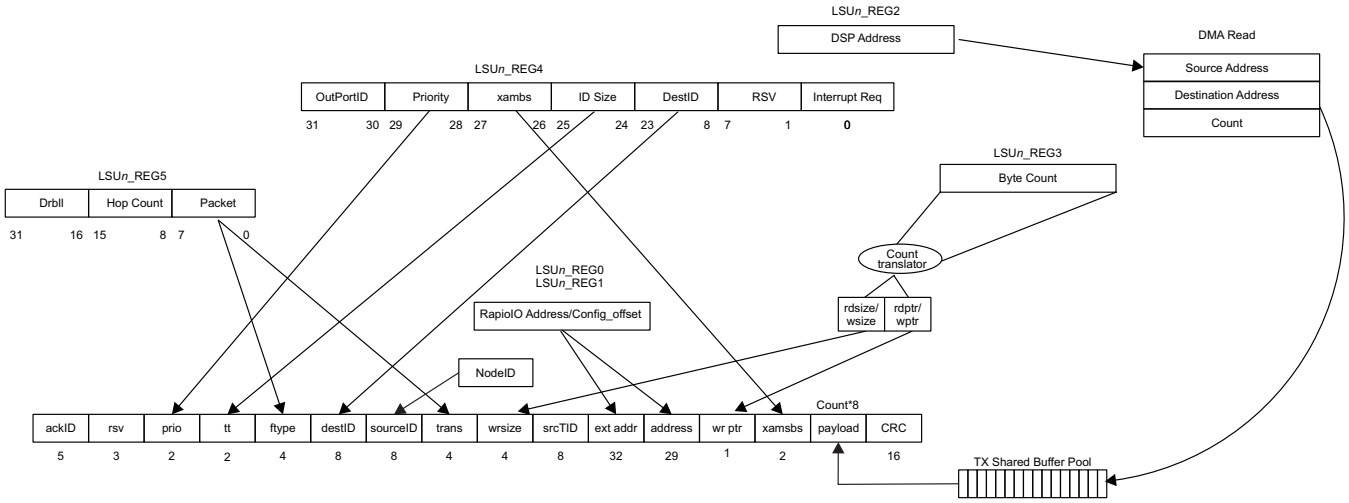


The following code illustrates an LSU registers programming example.

```
SRIO_REGS->LSU1_REG0 =     CSL_FMK( SRIO_LSU1_REG0_RAPIDIO_ADDRESS_MSB,0 );
SRIO_REGS->LSU1_REG1 =     CSL_FMK( SRIO_LSU1_REG1_ADDRESS_LSB_CONFIG_OFFSET,(int)&rcvBuff1[0] );
SRIO_REGS->LSU1_REG2 =     CSL_FMK( SRIO_LSU1_REG2_DSP_ADDRESS, (int)&xmtBuff1[0]);
SRIO_REGS->LSU1_REG3 =     CSL_FMK( SRIO_LSU1_REG3_BYTE_COUNT,byte_count );
SRIO_REGS->LSU1_REG4 =     CSL_FMK( SRIO_LSU1_REG4_OUTPORTID,0 )  |
                           CSL_FMK( SRIO_LSU1_REG4_PRIORITY,0 )   |
                           CSL_FMK( SRIO_LSU1_REG4_XAMSB,0 )      |
                           CSL_FMK( SRIO_LSU1_REG4_ID_SIZE,1 )    |
                           CSL_FMK( SRIO_LSU1_REG4_DESTID,0xBEEF )|
                           CSL_FMK( SRIO_LSU1_REG4_INTERRUPT_REQ,1 );
SRIO_REGS->LSU1_REG5 =     CSL_FMK( SRIO_LSU1_REG5_DRBLL_INFO,0x0000 )|
                           CSL_FMK( SRIO_LSU1_REG5_HOP_COUNT,0x00 )   |
                           CSL_FMK( SRIO_LSU1_REG5_PACKET_TYPE,type );
```

Figure 14 gives an example of the data flow and field mappings for a burst NWRITE_R transaction.

Figure 14. Example Burst NWRITE_R



For WRITE commands, the payload is combined with the header information from the control/command registers and buffered in the shared TX buffer resource pool. Finally, it is forwarded to the TX FIFO for transmission. READ commands have no payload. In this case, only the control/command register fields are buffered and used to create a RapidIO NREAD packet, which is forwarded to the TX FIFO. Corresponding response packet payloads from READ transactions are buffered in the shared RX buffer resource pool when forwarded from the receive ports. Both posted and non-posted operations rely on the OutPortID command register field to specify the appropriate output port/FIFO.

The data is burst internally to the Load/Store module at the DMA clock rate.

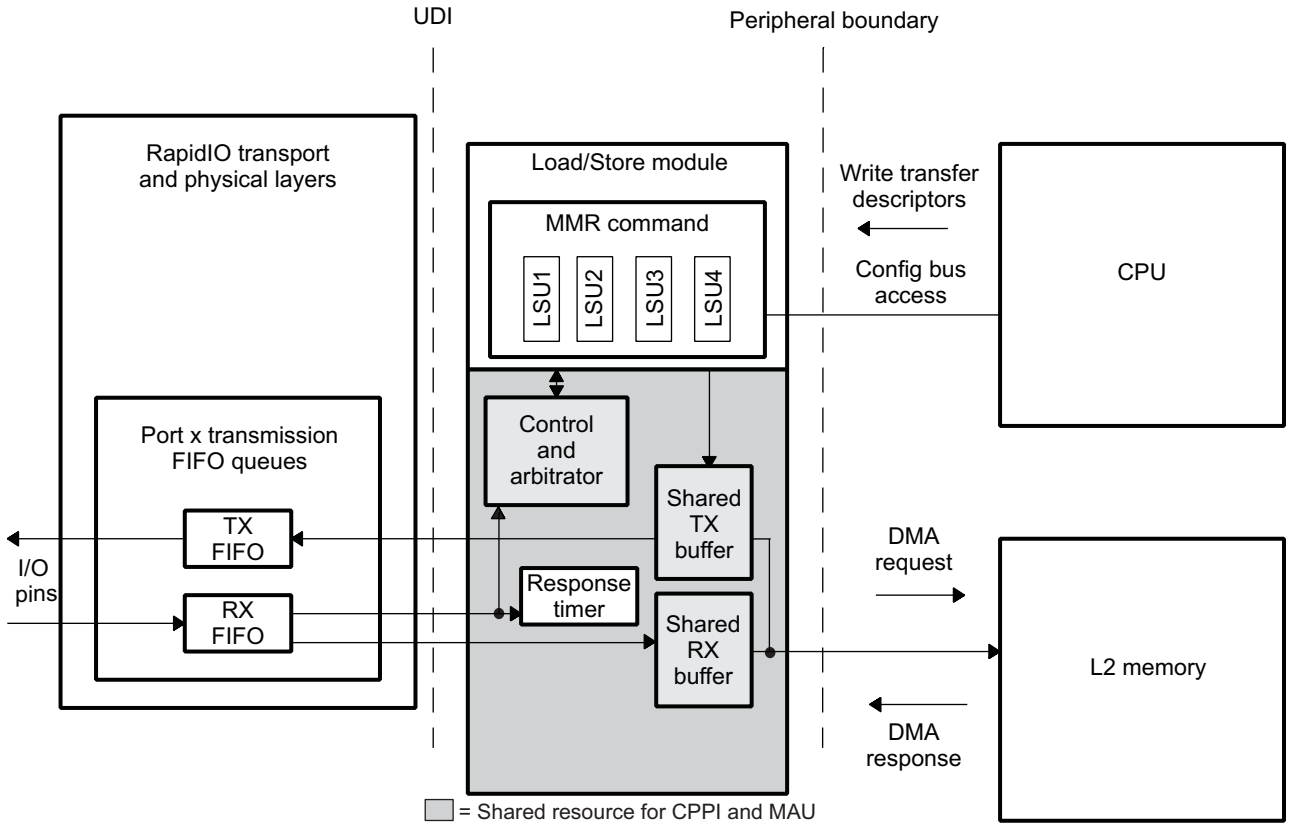
2.3.3.1 Detailed Data Path Description

The Load/Store module is for generating all outgoing RapidIO direct I/O packets. Any read or write transaction, other than the messaging protocol, uses this interface. In addition, outgoing DOORBELL packets are generated through this interface.

The data path for this module uses DMA bus as the DMA interface. The configuration bus is used by the CPU to access the control/command registers. The registers contain transfer descriptors that are needed to initiate READ and WRITE packet generation. After the transfer descriptors are written, flow control status is queried. The unit examines the DESTID and PRIORITY fields of LSU*n*_REG4 to determine if that flow has been Xoffd. Additionally, the free buffer status of the TX FIFO is checked (based on the OutPortID register field). Only after the flow control access is granted, and a TX FIFO buffer has been allocated, can a DMA bus read command be issued for payload data to be moved into the shared TX buffer. Data is moved from the shared TX buffer to the appropriate output TX FIFO in simple sequential order based on completion of the DMA bus transaction. However, if fabric congestion occurs, priority can affect the order in which the data leaves the TX FIFOs.

Here a reordering mechanism exists, which transmits the highest priority packets first if RETRY acknowledges. Once in the FIFO, the data is guaranteed to be transmitted through the pins. Alternatively, if an intended flow has been shut down, the peripheral signals the CPU with an interrupt to notify that the packet was not sent and sets the completion code to 010b in the status register. The registers are held until the interrupt service routine is complete before the BSY signal is released (BSY=0 in LSU*n*_REG6) and the CPU can then rewrite or overwrite the transfer descriptors with new data. Figure 15 illustrates the data path and buffering that is required to support the Load/Store module.

Figure 15. Load/Store Module Data Flow Diagram



2.3.3.2 Direct I/O TX Operation

WRITE Transactions:

The TX buffers are implemented in a single SRAM and shared between multiple cores. A state machine arbitrates and assigns available buffers between the LSUs. When the DMA bus read request is transmitted, the appropriate TX buffer address is specified within it. The data payload is written to that buffer through the DMA bus response transaction. Depending on the architecture of the device, interleaving of multi-segmented DMA bus responses from the DMA is possible. Upon receipt of a DMA bus read response segment, the unit checks the completion status of the payload. Note that only one payload can be completed in any single DMA bus cycle. The Load/Store module can only forward the packet to the TX FIFO after the final payload byte from the DMA bus response has been written into the shared TX buffer. Once the packet is forwarded to the TX FIFO, the shared TX buffer can be released and made available for a new transaction.

The TX buffer space is dynamically shared among all outgoing sources, including the Load/Store module and the TX CPPI, as well as the response packets from RX CPPI and the memory access unit (MAU). Thus, the buffer space memory is partitioned to handle packets with and without payloads. A 4.5K-byte buffer space is configured to support 16 packets with payloads up to 256 bytes, in addition to 16 packets without payloads. The SRAM is configured as a 128-bit wide two port, which matches the UDI width of the TX FIFOs.

NOTE: The "UDI" ("User Defined Interface") is a reference to the interface between (a) the SERDES and the FIFO queues and (b) the logical buffers, shared buffers, LSU and MAU modules, response timer, and controllers (together known as the "User Application"). UDI could also be known as the "logical/physical interface". No action is required to "define" this interface.

Data leaves the shared TX buffer sequentially in order of receipt, not based on the packet priority. However, if fabric congestion occurs, priority can affect the order in which the data leaves the TX FIFOs. A reordering mechanism exists here, which transmits the highest priority packets first if RETRY acknowledges.

For posted WRITE operations, which do not require a RapidIO response packet, a core may submit multiple outstanding requests. For instance, a single core may have many streaming write packets buffered at any given time, given outgoing resources. In this application, the control/command registers can be released (BSY = 0) to the CPU as soon as the header info is written into the shared TX buffer. If the request has been flow controlled, the peripheral will set the completion code status register and appropriate interrupt bit of the ICSR. The control/command registers can be released after the interrupt service routine completes.

For non-posted WRITE operations, which do require a RapidIO response packet, there can be only one outstanding request per core at any given time. The payload data and header information is written to the shared TX buffer as described above; however, the command registers cannot be released (BSY = 1) until the response packet is routed back to the module and the appropriate completion code is set in the status register. One special case exists for outgoing test-and-swap packets (Ftype 5, Transaction 1110b). This is the only WRITE class packet that expects a response with payload. This response payload is routed to the LSU, where it is examined to verify whether the semaphore was accepted, and then the appropriate completion code is set. The payload is not transferred out of the peripheral via the DMA bus.

So the general flow is as follows:

- LSU registers are written using the configuration bus
- Flow control is determined
- TX FIFO free buffer availability is determined
- DMA bus read request for data payload
- DMA bus response writes data to specified module buffer in the shared TX buffer space
- DMA bus read response is monitored for last byte of payload
- Header data in the LSU registers is written to the shared TX buffer space
- Payload and header are transferred to the TX FIFO
- The LSU registers are released if no RapidIO response is needed
- Transfer from the TX FIFO to external device based on priority

READ Transactions:

The flow for generating READ transactions is similar to non-posted WRITE with response transactions. There are two main differences: READ packets contain no data payload, and READ responses have a payload. So READ commands simply require a non-payload shared TX buffer. In addition, they require a shared RX buffer. This buffer is not pre-allocated before transmitting the READ request packet, since doing so could cause traffic congestion of other in-bound packets destined to other functional blocks.

Again, the control/command registers cannot be released (BSY = 1) until the response packet is routed back to the module and appropriate completion code is set in the status register.

So the general flow would be:

- LSU registers are written using the configuration bus
- Flow control is determined
- TX FIFO free buffer availability is determined
- Header data in the LSU registers is written to the shared TX buffer
- Payload and header are transferred to the TX FIFO
- The LSU registers are released if no RapidIO response is needed
- Transfer from the TX FIFO to external device based on priority

For all transactions, the shared TX buffers are released as soon as the packet is forwarded to the TX FIFOs. If an ERROR or RETRY response is received for a non-posted transaction, the CPU must either reinitiate the process by writing to the LSU register, or initiate a new transaction altogether.

Segmentation:

The LSU handles two types of segmentation of outbound requests. The first type is when the Byte_Count of Read/Write requests exceeds 256 bytes (up to 4K bytes). The second type is when Read/Write request RapidIO address is non-64-bit aligned. In both cases, the outgoing request is broken up into multiple RapidIO request packets. For example, assume that the CPU wants to perform a 1K-byte store operation to an external RapidIO device. After setting up the LSU registers, the CPU performs one write to the LSU*n*_REG5 register. The peripheral hardware then segments the store operation into four RapidIO write packets of 256 bytes each, and calculates the 64-bit-aligned RapidIO address, WRSIZE, and WDPTR as required for each packet. This example requires four outbound handles to be assigned and four DMA transmit requests. The LSU registers cannot be released until all posted request packets are passed to the TX FIFOs. Alternatively, for non-posted operations, such as CPU loads, all packet responses must be received before the LSU registers are released.

2.3.3.3 Direct I/O RX Operation

Response packets are always type 13 RapidIO packets. All response packets with transaction types not equal to 0001b are routed to the LSU block sequentially in order of reception. These packets may have a payload, depending on the type of corresponding request packet that was originally sent. Due to the nature of RapidIO switch fabric systems, response packets can arrive in any order. The data payload, if any, and header data is moved from the RX FIFO to the shared RX buffer. The targetTID field of the packet is examined to determine which core and corresponding set of registers are waiting for the response. Remember, there can be only one outstanding request per core. Any payload data is moved from the shared RX buffer into memory through normal DMA bus operations.

Registers for all non-posted operations should only be held for a finite amount of time to avoid blocking resources when a request or response packet is somehow lost in the switch fabric. This time correlates to the 24-bit Port Response Time-out Control CSR value discussed in sections 5.10.1 and 6.1.2.4 of the RapidIO *Physical Layer 1x/4x LP-Serial Specification*. If the time expires, control/command register resources should be released, and an error is logged in the error-management RapidIO registers. The *RapidIO Interconnect Specification* states that the maximum time interval (all 1s) is between 3 and 6 seconds. A logical layer timeout occurs if the response packet is not received before a countdown timer (initialized to this CSR value) reaches zero.

Each outstanding packet response timer requires a 4-bit register. The register is loaded with the current timecode when the transaction is sent. The timecode comes from a 4 bit counter associated with the 24 bit down counter that continually counts down and is re-loaded with the value of SP_RT_CTL (address offset 1124h) when it reaches 0. Each time the timecode changes, a 4-bit compare is done to the register. If the register becomes equal to the timecode again, without a response being seen, then the transaction has timed out. Essentially, instead of the 24-bit value representing the period of the response timer, the period is now defined as P = (2^24 x 16)/F. This means the countdown timer frequency needs to be 44.7 - 89.5Mhz for a 6 - 3 second response timeout. Because the needed timer frequency is derived from the DMA bus clock (which is device dependent), the hardware supports a programmable configuration register field to properly scale the clock frequency. This configuration register field is described in the Peripheral Setting Control register (address offset 0020h).

If a response packet indicates ERROR status, the Load/Store module notifies the CPU by generating an error interrupt for the pending non-posted transaction. If the response has completed successfully, and the Interrupt Req bit is set in the control register, the module generates a CPU servicing interrupt to notify the CPU that the response is available. The control/command registers can be released as soon as the response packet is received by the logical layer. The hardware is not responsible for attempting a retransmission of the non-posted transaction.

If a Doorbell response packet indicates Retry status, the Load/Store module notifies the CPU by generating an interrupt. The control/command registers can be released as soon as the response packet is received by the logical layer. The hardware is not responsible for attempting retransmission of the Doorbell transaction.

So the general flow is as follows:

- Previously, the control/command registers were written and the request packet was sent
- Response Packet Type13, Trans != 0001b arrives at module interface, and is handled sequentially (not based on priority)
- The targetTID is examined to determine routing of a response to the appropriate core
- The status field of the response packet is checked for ERROR, RETRY or DONE
- If the field is DONE, it submits DMA bus request and transmits the payload (if any) to DSP address. If the field is ERROR/RETRY, it sets an interrupt
- Command registers are released (BSY = 0)
- Optional Interrupt to CPU notifying packet reception

2.3.3.4 Reset and Power Down State

Upon reset, the Load/Store module clears the command register fields and wait for a write by the CPU.

The Load/Store module can be powered down if the direct I/O protocol is not being supported in the application. For example, if the messaging protocol is being used for data transfers, powering down the Load/Store module will save power. In this situation, the command registers should be powered down and inaccessible. Clocks should be gated to these blocks while in the power down state.

2.3.4 Message Passing

The Communications Port Programming Interface (CPPI) module is the incoming and outgoing message-passing protocol engine of the RapidIO peripheral. Messages contain application specific data that is pushed to the receiving device comparable to a streaming write. Messages do not contain read operations, but do have response packets.

With message passing, a destination address is not specified. Instead, a mailbox identifier is used within the RapidIO packet. The mailbox is controlled and mapped to memory by the local (destination) device. For RapidIO message passing, four mailbox locations are specified. Each mailbox can contain 4 separate transactions (or letters), effectively providing 16 messages. Single packet messages provide 64 mailboxes with 4 letters, effectively providing 256 messages. Mailboxes can be defined for different data types or priorities. The advantage of message passing is that the source device does not require any knowledge of the destination device's memory map. The DSP contains buffer description tables for each mailbox. These tables define a memory map and pointers for each mailbox. Messages are transferred to the appropriate memory locations via the DMA.

The data path for this module uses the DMA bus as the DMA interface. The ftype header field of the received RapidIO message packets are decoded by the logical layer of the peripheral. Only Type 11 and Type 13 (transaction type 1) packets are routed to this module. Data is routed from the priority-based RX FIFOs to the CPPI module's data buffer within the shared buffer pool. The mbox (mailbox) header fields are examined by the mailbox mapper block of the CPPI module. Based on the mailbox and message length, the data is assigned memory addresses within memory. Data is transferred via DMA bus commands to memory from the buffer space of the peripheral. The maximum buffer space should accommodate 256 bytes of data, as that is the maximum payload size of a RapidIO packet. Each message in memory will be represented by a buffer descriptor in the queue.

The following rules exist for all CPPI traffic:

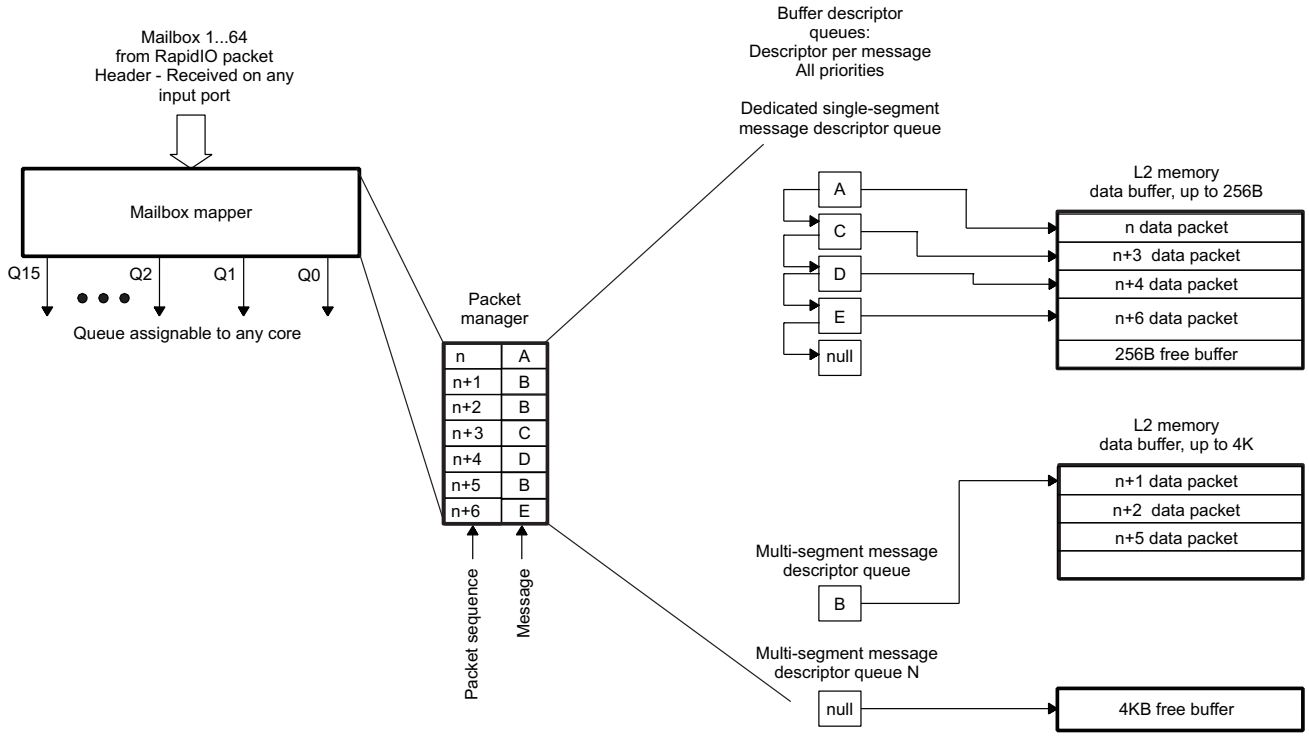
- One buffer descriptor is provided per message (each buffer descriptor consists of 4 words or 16 bytes).
- Contiguous memory space is required for multi-segment read and write operations.
  - There are fixed buffer sizes (configured to handle the application's maximum message size).
- An ERROR response is sent if the RX message is too big for the allotted buffer space.
  - ERROR responses are sent for all subsequent segments of that message.
- An ERROR response is sent if the mailbox is not mapped, or if it is mapped to a non-existent queue.
- An ERROR response is sent if the mailbox is mapped but the queue is not initialized (the head descriptor pointer is not written), or if the queue is disabled (due to a teardown).
- An ERROR response is sent if the RX buffer descriptor queue has no empty buffers (there is an overflow) .

- Out-of-order responses are allowed.
- A RETRY response is issued to the first received segment of a multi-segment message when the RX queue is busy servicing another request.
  - Subsequent RETRY responses may have to be sent for received pipeline segments or additional pipelined messages to the same queue.
- In-order message reception for dedicated flows is mode programmable.
- A queue is needed for each supported simultaneous multi-segment RX message.
- A minimum of 1.25K bytes of SRAM (64 buffer descriptors) is supported.
- The transmit source must be able to retry any given segment of a message.
- DESTID is equal to port for TX operations, and the same DESTID is not accessible from multiple ports.

2.3.4.1 RX Operation

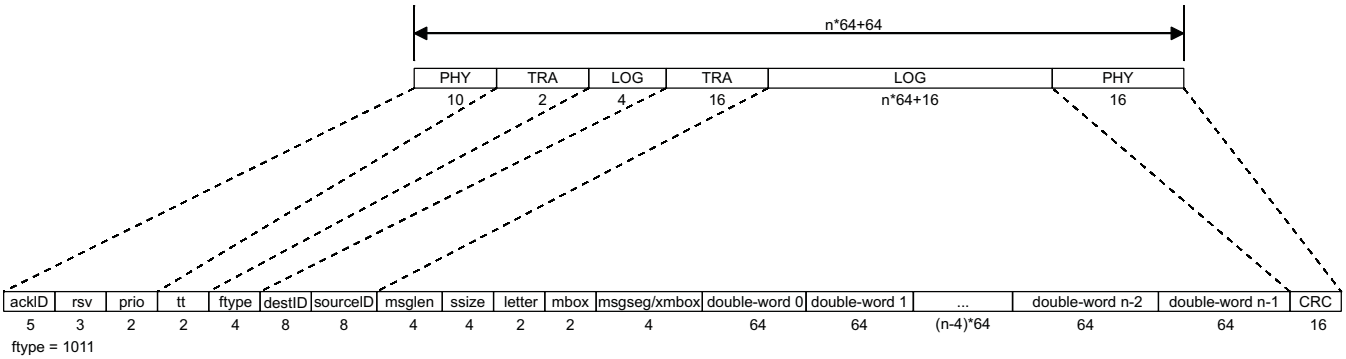
As message packets are received by the RapidIO ports, the data is written into memory while maintaining accurate state information that is needed for future processing. For instance, if a message spans multiple packets, information is saved that allows re-assembly of those packets by the CPU. The CPPI module provides a scheme for tracking single and multi-packet messages, linking messages in queues, and generating interrupts. Figure 16 illustrates the scheme.

Figure 16. CPPI RX Scheme for RapidIO



Messages addressed to any of the 64 mailbox locations can be received on any of the RapidIO ports simultaneously. Packets are handled sequentially in order of receipt. The function of the mailbox mapper block is to direct the inbound messages to the appropriate queue and finally to the correct core. The queue mapping is programmable and must be configured after device reset. RapidIO originally supported only 4 mailboxes with 4 letters/mailbox. Letters allow concurrent message traffic between sender and receiver. However, for messages that consist of only single packets, the unused 4-bit packet field normally indicating the message segment extends the available number of mailboxes. Figure 17 shows the packet header fields for message requests.

Figure 17. Message Request Packet



This enables the letter and mailbox fields to instead allow four concurrent single-segment messages to sixty-four possible mailboxes (256 total locations) for a source and destination pair. The mailbox mapper directs the inbound messages to the appropriate queue based on a pre-programmed routing table. It bases the decision on the SOURCEID, MSGLEN (the size indicates whether the message is segmented), MBOX, LETTER, and XMBOX fields of the RapidIO packet.

There are 32 programmable look-up table entries for mapping mailboxes to queues. Each entry consists of two registers, RXU_MAP_L*n* and RXU_MAP_H*n*, which are shown in Figure 18. A detailed summary of these register's field is in Section 5.52. In total, there are 64 registers, at address offsets 0800h-08FCh. Each entry stores the queue number associated with the message's intended mailbox/letter. If a mailbox/letter is not supported or does not have a mapping table entry, the message is discarded and an ERROR response sent. The mapping entries can explicitly call out a mailbox and letter combination, or alternatively, the mask fields can be used to grant multiple mailbox/letter combinations access to a queue using the same table entry. A masking value of 0 in the mailbox or letter mask fields indicates that the corresponding bit in the mailbox or letter field will not be used to match for this queue mapping entry. For example, a mailbox mask of all zeros would allow a mapping entry to be used for all incoming mailboxes.

The mapping table entry also provides a security feature to enable or disable access from specific external devices to local mailboxes. The sourceID field indicates which external device has access to the mapping entry and corresponding queue. A compare is performed between the sourceID of the incoming message packet and each relevant mailbox/letter table mapping entry SOURCEID field. If they do not match, an ERROR response is sent back to the sender, and the transaction is logged in the logical layer error management capture registers, which sets an interrupt. A PROMISCUOUS bit allows this security feature to be disabled. When the PROMISCUOUS bit is set, full access to the mapping entry from any sourceID is allowed. Note that when the PROMISCUOUS bit is set, the mailbox/letter and corresponding mask bits are still in effect. When the PROMISCUOUS bit is cleared, it equals a mask value of FFFFh, and only a request with the matching sourceID is allowed access to the mailbox.

Each table entry also indicates if it used for single or multi-segment message mapping. Single segment message mapping entries utilize all six bits of the mailbox and corresponding mask fields. Multi-segment entries uses only the 2 LSBs. The number of simultaneous supported multi-segment messages is determined by the number of dedicated RX queues as discussed further below. It is recommended to dedicate a multi-segment mapping entry for each supported simultaneous letter. Essentially, letter masks should be avoided for multi-segment mapping to reduce excessive retries. Note that it is possible to configure the table entries such that incoming single segment and multi-segment messages are directed to the same queue. To avoid this condition, properly program the mapping table entries.

Figure 18. Mailbox to Queue Mapping Register Pair

Mailbox to Queue Mapping Register L *n* (RXU_MAP_L *n*)

31 30	29 24	23 22	21 16
LETTER_MASK	MAILBOX_MASK	LETTER	MAILBOX
R/W-11	R/W-111111	R/W-00	R/W-000000

15 0
SOURCEID
R/W-0000h

Mailbox to Queue Mapping Register H *n* (RXU_MAP_H *n*)

31
Reserved
R-0

10	9 8	7 6	5 2	1	0
Reserved	TT	Reserved	QUEUE_ID	PROMISCUOUS	SEGMENT MAPPING
R-0	R/W-01	R-00	R/W-0000	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

The packet manager maintains the RX DMA state of free and used data buffers within the memory space. It directs the data to specific addresses within the memory and maintains and updates the buffer descriptor queues. There is a single buffer descriptor per RapidIO message. For example, single segment messages have one buffer descriptor, as do multi-segment messages with up to 4K-byte payloads.

There can be multiple RX buffer descriptor queues per core. It is suggested that one queue be dedicated to single segment messages and additional queues be dedicated to multi-segment messages. Each multi-segment message queue can support only one incoming message at a time. Depending on the application, it may be necessary to support multiple simultaneous segmentation and reassembly (SAR) operations per core. In this case, a buffer descriptor queue is allocated for each desired simultaneous message. The peripheral supports a total of 16 assignable RX queues and their associated RX DMA state registers. Each of the queues can be assigned to single or multi-segment messages.

Table 16 and Table 17 describe the RX DMA State Registers.

Table 16. RX DMA State Head Descriptor Pointer (HDP) - Address Offset 600h-63Ch

Bit	Name	Description
31-0	RX Queue Head Descriptor Pointer	RX Queue Head Descriptor Pointer: This field is the memory address for the first buffer descriptor in the channel receive queue. This field is written by the DSP core to initiate queue receive operations and is zeroed by the port when all free buffers have been used. An error condition results if the DSP core writes this field when the current field value is nonzero. The address must be 32-bit word aligned.

Table 17. RX DMA State Completion Pointer (CP) - Address Offset 680h-6BCh

Bit	Name	Description
31-0	RX Queue Completion Pointer	RX Queue Completion Pointer: This field is the memory address for the receive queue completion pointer. This register is written by the DSP core with the buffer descriptor address for the last buffer processed by the DSP core during interrupt processing. The port uses the value written to determine if the interrupt should be de-asserted.

If a multi-segment buffer descriptor queue is not currently free, and an RX port receives another multi-segment message that is destined for that queue, the RX CPPI sends a RETRY RESPONSE packet (type 13) to the sender, indicating that an internal buffering problem exists. If a multi-segment buffer descriptor queue is busy and there is another incoming multi-segment message with the same SOURCEID, MAILBOX, and LETTER, an ERROR response is sent. This usually indicates that a TX programming error has occurred, where duplicate segments or segments outside the MSGLEN were sent. Upon successful reception of any message segment, the RX CPPI is responsible for sending a DONE response to the sender.

If a RX message's length is greater than that of the targeted buffer descriptor, an ERROR response is sent back to the source device. In addition, the DSP is notified with the use of the CC field of the RX CPPI buffer descriptor, described as follows. This situation can result from a DSP software error (misallocating a buffer for the queue), or as a result of sender error (sending to a wrong mailbox).

An RX transaction timeout is used by all multi-segment queues, in order to not hang receive mailbox resources in the event that a message segment is lost in the fabric. This response-to-request timer controls the time-out period for sending a response packet and receiving the next request packet of a given multi-segment message. It has the same value and is analogous to the request-to-response timer discussed in the TX CPPI and LSU sections, which is defined by the 24-bit value in the port response time-out CSR (See Section 2.3.3.3). The *RapidIO Interconnect Specification* states that the maximum time interval (all 1s) is between 3 and 6 seconds. Each multi-segment receive timer requires a 4-bit register. The register is loaded with the current timecode when the response is sent. Each time the timecode changes, a 4-bit compare is done to the register. If the register becomes equal to the timecode again, without the next message segment being seen, then the transaction has timed out. If this happens, the RX buffer resources can be released.

The buffer descriptor points to the corresponding data buffer in memory and also points to the next buffer descriptor in the queue. As segments of a received message arrive, the msgseg field of each segment is monitored to detect the completion of the received message. Once a full message is received, the OWNERSHIP bit is cleared in the packet's buffer descriptor to give control to the host. At this point, a host interrupt is issued. This interface works with programmable interrupt rate control. There is an ICSR bit for each supported queue. On interrupt, the CPU processes the RX buffer queue, detecting received packets by the status of the OWNERSHIP bit in each buffer descriptor. The host processes the RX queue until it reaches a buffer descriptor with a set OWNERSHIP bit, or set EOQ bit. Once processing is complete, the host updates the RX DMA State Completion Pointer, allowing the peripheral to reuse the buffer.

Figure 19 shows the RX buffer descriptor fields and Table 18 describes them. A RX buffer descriptor is a contiguous block of four 32-bit data words aligned on a 32-bit boundary. Accesses to these registers are restricted to 32-bit boundaries.

Figure 19. RX Buffer Descriptor Fields

Word Offset	Bit Fields																													
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	next_descriptor_pointer																															
1	buffer_pointer																															
2	src_id																pri		tt		reserved								mailbox			
3	sop	eop	ownership	eoq					teardown	reserved											cc			message_length								

Table 18. RX Buffer Descriptor Field Descriptions

Field	Value	Description
next_descriptor_pointer		Next Descriptor Pointer: The 32-bit word aligned memory address of the next buffer descriptor in the RX queue. This references the next buffer descriptor from the current buffer descriptor. If the value of this pointer is zero, then the current buffer is the last buffer in the queue. The DSP core sets the next_descriptor_pointer.
buffer_pointer		Buffer Pointer: The byte aligned memory address of the buffer associated with the buffer descriptor. The DSP core sets the buffer_pointer.
sop = 1		Start of Message: Indicates that the descriptor buffer is the first buffer in the message. This bit is always set, as this device only supports one buffer per message.
eop = 1		End of Message: Indicates that the descriptor buffer is the last buffer in the message. This bit is always set, as this device only supports one buffer per message.

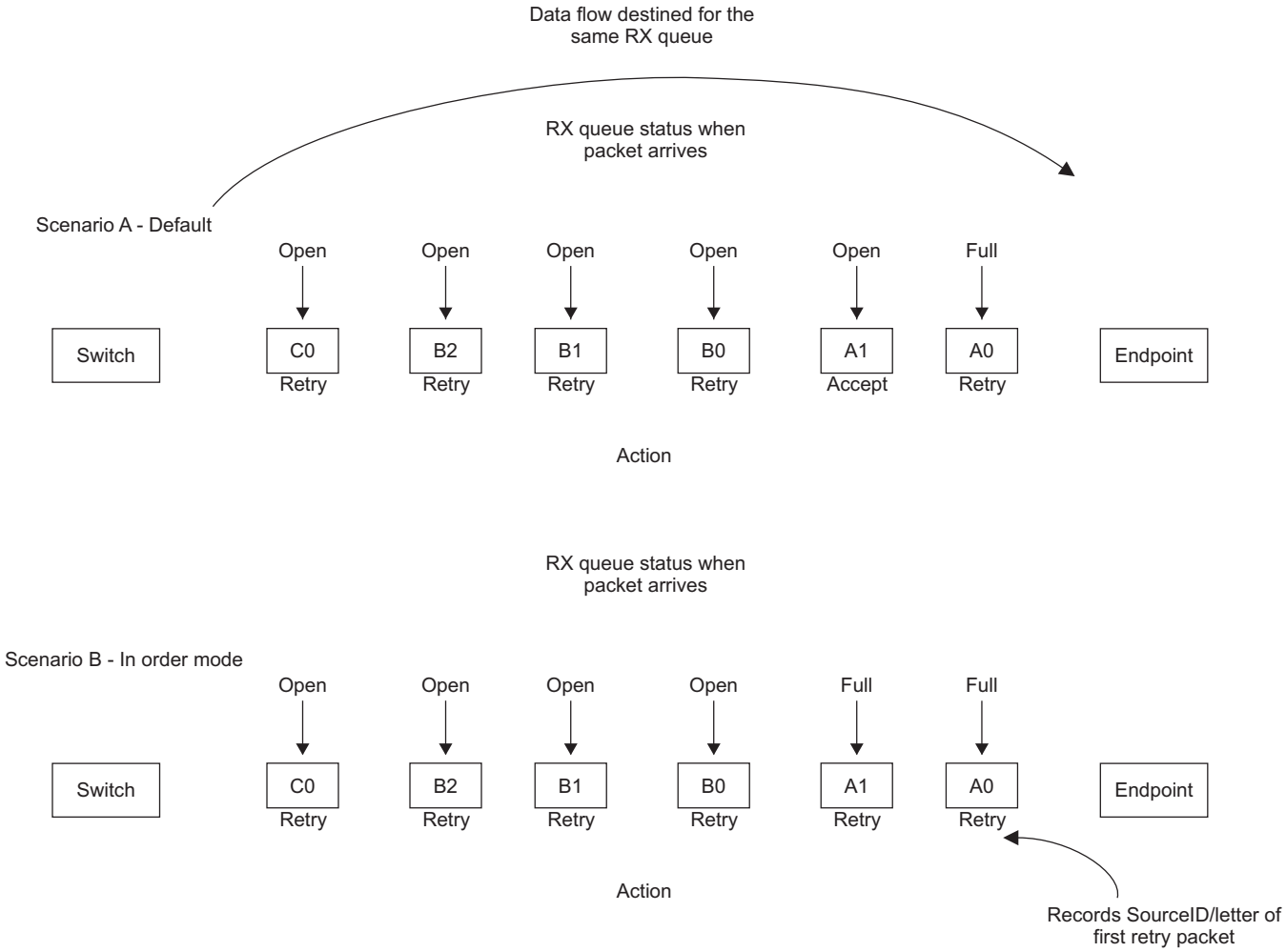
Table 18. RX Buffer Descriptor Field Descriptions (continued)

Field	Value	Description
ownership		Ownership: Indicates ownership of the message and is valid only on sop. This bit is set by the DSP core and cleared by the port when the message has been transmitted. The DSP core uses this bit to reclaim buffers.
	0	The message is owned by the DSP core
	1	The message is owned by the port
eoq		End Of Queue: Set by the port to indicate that the RX queue empty condition exists. This bit is valid only on eop. The port determines the end of queue condition by a zero next_descriptor_pointer.
	0	The RX queue has more buffers available for reception.
	1	The Descriptor buffer is the last buffer in the last message in the queue.
teardown_complete		Teardown Complete: Set by the port to indicate that the host commanded teardown process is complete, and the channel buffers may be reclaimed by the host.
	0	The port has not completed the teardown process.
	1	The port has completed the commanded teardown process.
message_length		Message Length: Initially written by the DSP core to specify the maximum number of double-words the buffer can receive. Updated by the peripheral (after receiving a message) to indicate the actual number of double-words in the entire message. Message payloads are limited to a maximum size of 512 double-words (4096 bytes).
	000000000b	512 double words
	000000001b	1 double word
	000000010b	2 double words
	. . .	
	111111111b	511 double words
src_id		Source Node ID: Unique node identifier of the source of the message. Written by the DSP core.
tt		RapidIO tt field specifying 8- or 16-bit DeviceIDs. Written by the DSP core.
	00b	8-bit deviceIDs
	01b	16-bit deviceIDs
	10b	Reserved
	11b	Reserved
pri		Message Priority: Specifies the SRIO priority at which the message was sent. Written by the DSP core.
cc		Completion Code: Written by the port.
	000	Good completion. Message received.
	001	Error, RX message length greater than supported buffer descriptor message_length
	010	Error, TimeOut on receiving one of the segments
	011	DMA transfer error on one or more segments
	100	Queue teardown completed, data invalid
	101-111	Reserved
mailbox		Destination Mailbox: Specifies the mailbox to which the message was sent. Written by the DSP core.
	000000b	Mailbox 0
	000001b	Mailbox 1
	. . .	
	000100b	Mailbox 4
	. . .	
	111111b	Mailbox 63 For multi-segment messages, only the two LSBs of this mailbox are valid. Hardware ignores the four MSBs if the incoming message has multiple segments.

Although the switch fabric delivers the segments of multi-packet messages in the order they were sent, buffer resources at the receiving endpoint may only become available after the initial segment(s) of a message have had to be retried. The peripheral can accept out-of-order segments and track completion of the overall message. Scenario A in Figure 20 shows this concept.

For applications that are set up for specific message flows between a single source and destination, it may require in-order delivery of messages. This is described in scenario B of Figure 20. This scenario is similar to scenario A, although one message may be retried due to a lack of receiver resources, subsequent pipelined messages may arrive just as resources are freed up. This is a problem for systems requiring in-order message delivery. In this case, the peripheral needs to record the Src_id/mailbox/letter of the first retried message and retry all subsequent new requests until resources are available and a segment for that Src_id/mailbox/letter is received. As long as all messages are from the same source and that source sends (and retries) packets in order, then all messages will be received in order. Note that this solution is less effective when multiple sources share an RX queue. The RX CPPI Control register (address offset 0744h) sets this mode of operation on all receive queues. Once this mode is set and a retry is issued, the queue will continue to wait for an incoming message that matches the Src_id/mailbox/letter combination. If no such packet arrives, the RX queue is unusable in a locked state. To reenable the queue, the in-order bit in the RX CPPI Control register must be disabled by software for that queue, after which it may be enabled again if desired. The in-order mode of operation is only valid on multi-segment queues because single-segment messages will never generate RETRY responses.

Figure 20. RX CPPI Mode Explanation



In addition, multiple messages can be interleaved at the receive port due to ordering within a connected switch's output queue. This can occur when using a single or multiple priorities. The RX CPPI block can handle simultaneous interleaved multi-segment messages. This implies that state information (write pointers and sourceID) is maintained on each simultaneous message to properly store the segments in memory. The number of simultaneous transactions supported directly impacts the number of states to be stored, and the size of the buffer descriptor memory outside the peripheral. With this in mind, the peripheral's supported buffer descriptor SRAM is parameterizable. A minimum size of 1.25K bytes is recommended, which will allow up to 64 buffer descriptors to be stored at any given time for one core. These buffer descriptors can be configured to support any combination of single and multi-segment messages. For example, if the application only handles single-segment messages, all 64 buffers can be allotted to that queue. Note that a given RX queue can contain packets of all priorities which have been directed from any of the receive ports.

A CPU may wish to stop receiving messages and reclaim buffers belonging to a specific queue. This is called queue teardown. The CPU initiates a RX queue teardown by writing to the RX Queue Teardown command register (address Offset 0740h).

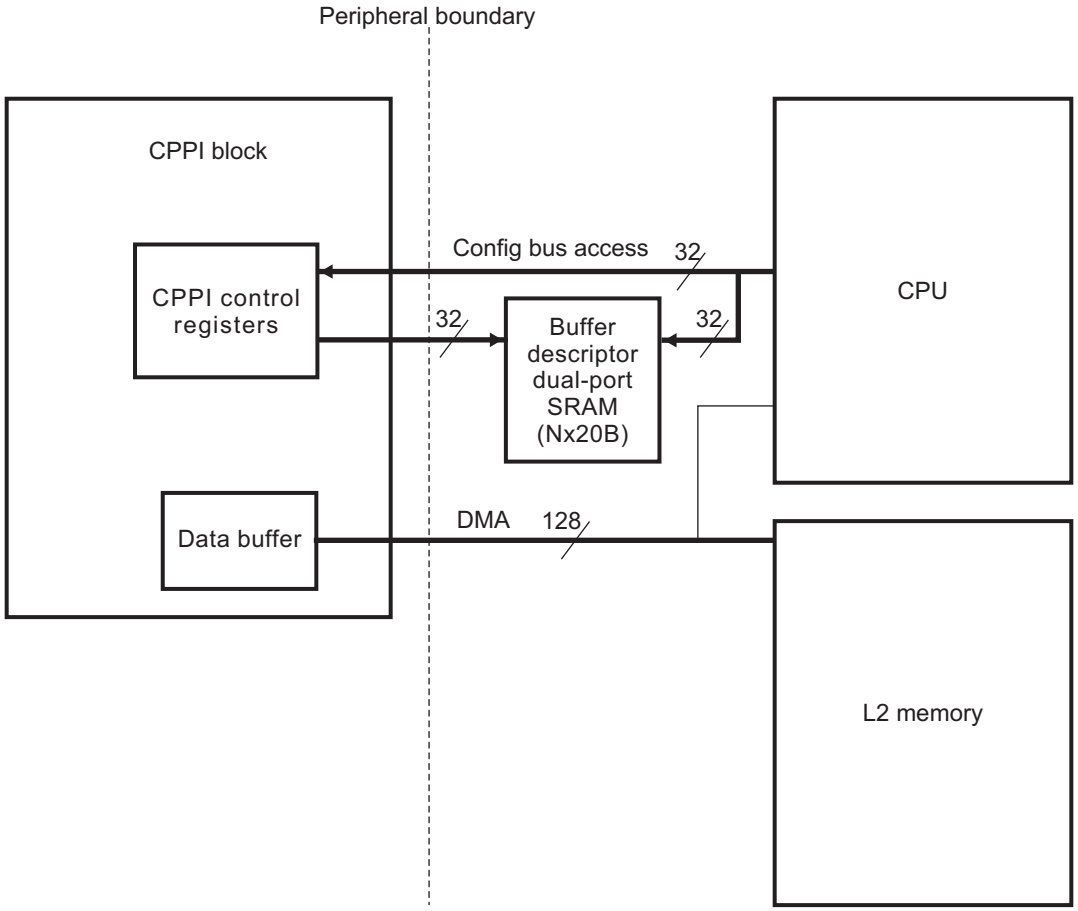
Teardown of an RX queue causes the following actions:

- If teardown is issued by software during the time when the RX state machine is idle, then the state machine will immediately start the teardown procedure:
  - If the queue to be torn down is in-message (waiting for one or more segments), then the queue will be torn down and reported with the current buffer descriptor (teardown bit set, ownership bit cleared, CC = 100b). All other fields of the buffer descriptor are invalid. The peripheral completes the teardown procedure by clearing the HDP register, setting the CP register to FFFFFFFCh, and issuing an interrupt for the given queue. The teardown command register bit is automatically cleared by the peripheral.
  - If the queue is not in-message, and active (next descriptor available), then the next descriptor will be fetched and updated to report teardown (teardown bit set, ownership bit cleared, CC = 100b). All other fields of the buffer descriptor are invalid. The peripheral completes the teardown procedure by clearing the HDP register, setting the CP register to FFFFFFFCh, and issuing an interrupt for the given queue. The teardown command register bit is automatically cleared by the peripheral.
  - If the queue is not in-message, but inactive (next descriptor unavailable), then no additional buffer descriptor will be written. The HDP register and the CP register remain unchanged. An interrupt is not issued. The teardown command register bit is automatically cleared by the peripheral.
- If teardown is issued by software during the time when the RXU state machine is busy, the teardown procedure will be postponed until the state machine is idle.

After the teardown process is complete and the interrupt is serviced by the CPU, the software must re-initialize the RX queue to restart normal operation.

The buffer descriptor queues are maintained in local SRAM just outside of the peripheral, as shown in Figure 21. This allows the quickest access time, while maintaining a level of configurability for device implementation. The SRAM is accessible by the CPU through the configuration bus. Alternatively, the buffer descriptors could use L2 memory as well.

Figure 21. CPPI Boundary Diagram



2.3.4.2 TX Operation

Outgoing messages are handled similarly, with buffer descriptor queues that are assigned by the CPUs. The queues are configured and initialized upon reset. When a CPU wants to send a message to an external RapidIO device, it writes the buffer descriptor information via the configuration bus into the SRAM. Again, there is a single buffer descriptor per RapidIO message. Upon completion of writing the buffer descriptor, the OWNERSHIP bit is set to give control to the peripheral. The CPU then writes the TX DMA State HDP register to initiate the queue transmit. For TX operation, PortID is specified to direct the outgoing packet to the appropriate port. Table 19 and Table 20 describe the TX DMA state registers. Figure 22 shows the TX buffer descriptor fields and Table 21 describes them. A TX buffer descriptor is a contiguous block of four 32-bit data words aligned on a 32-bit boundary.

Table 19. TX DMA State Head Descriptor Pointer (HDP) - Address Offset 500h-53Ch

Bit	Name	Description
31-0	TX Queue Head Descriptor Pointer	TX Queue Head Descriptor Pointer: This field is the DSP core memory address for the first buffer descriptor in the transmit queue. This field is written by the DSP core to initiate queue transmit operations and is zeroed by the port when all packets in the queue have been transmitted. An error condition results if the DSP core writes this field when the current field value is nonzero. The address must be 32-bit word aligned.

Table 20. TX DMA State Completion Pointer (CP) - Address Offset 58h-5BCh

Bit	Name	Description
31-0	TX Queue Completion Pointer	TX Queue Completion Pointer: This field is the DSP core memory address for the transmit queue completion pointer. This register is written by the DSP core with the buffer descriptor address for the last buffer processed by the DSP core during interrupt processing. The port uses the value written to determine if the interrupt should be de-asserted.

Figure 22. TX Buffer Descriptor Fields

Word Offset	Bit Fields																													
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	next_descriptor_pointer																															
1	buffer_pointer																															
2	dest_id																		pri		tt		port_id		ssize				mailbox			
3	sop	eop	ownership	eoq	teardown	reserved											retry_count						cc		message_length							

Table 21. TX Buffer Descriptor Field Definitions

Field	Value	Description
next_descriptor_pointer		Next Descriptor Pointer: The 32-bit word aligned memory address of the next buffer descriptor in the TX queue. This references the next buffer descriptor from the current buffer descriptor. If the value of this pointer is zero, then the current buffer is the last buffer in the queue. The DSP core sets the next_descriptor_pointer.
buffer_pointer		Buffer Pointer: The byte-aligned memory address of the buffer associated with the buffer descriptor. The DSP core sets the buffer_pointer.
sop = 1		Start of Message: Indicates that the descriptor buffer is the first buffer in the message. This bit is always set, as this device only supports one buffer per message.
eop = 1		End of Message: Indicates that the descriptor buffer is the last buffer in the message. This bit is always set, as this device only supports one buffer per message.
ownership		Ownership: Indicates ownership of the message and is valid only on sop. This bit is set by the DSP core and cleared by the port when the message has been transmitted. The DSP core uses this bit to reclaim buffers.
	0	The message is owned by the DSP core
	1	The message is owned by the port
eoq		End Of Queue: Set by the port to indicate that the TX queue empty condition exists. This bit is valid only on eop. The port determines the end-of-queue condition by a zero next_descriptor_pointer.
	0	The TX queue has more messages available to transfer.
	1	The Descriptor buffer is the last buffer in the last message in the queue.
teardown_complete		Teardown Complete: Set by the port to indicate that the DSP-commanded teardown process is complete and the channel buffers may be reclaimed by the DSP core.
	0	The port has not completed the teardown process.
	1	The port has completed the commanded teardown process.
retry_count		Message Retry Count: Set by the DSP core to indicate the total number of retries allowed for this message, including all segments. Decremented by the port each time a message is retried.
	000000b	Infinite retries
	000001b	Retry message 1 time
	000002b	Retry message 2 times
	. . .	
	111111b	Retry message 63 times

Table 21. TX Buffer Descriptor Field Definitions (continued)

Field	Value	Description
cc		Completion Code: Set by the port.
	000	Good completion. Message received a completion response.
	001	Transaction error. Message received an error response.[1]
	010	Excessive retries. Message received more than retry_count retry responses.
	011	Transaction timeout. Transaction timer elapsed without any message response being received.
	100	DMA data transfer error.
	101	Descriptor programming error.
	110	TX queue teardown complete.
	111	Outbound credit not available.
message_length		Message Length: Written by the DSP core to specify the number of double-words to transmit. Message payloads are limited to a maximum size of 512 double-words (4096 bytes).
	000000000b	512 double words
	000000001b	1 double word
	000000010b	2 double words
	. . .	
	111111111b	511 double words
dest_id		Destination Node Id: Unique Node identifier for the Destination of the message. Written by the DSP core.
pri		Message Priority: Specifies the SRIO priority at which the message will be sent. Messages should not be sent at a priority level of 3 because the message response is required to promote the priority to avoid system deadlock. It is the responsibility of the software to assign the appropriate outgoing priority.
tt		RapidIO *tt* field specifying 8- or 16-bit DeviceIDs. Written by the host.
	00b	8-bit deviceIDs
	01b	16-bit deviceIDs
	10b	Reserved
	11b	Reserved
port_id		Port number for routing outgoing packet. Written by the DSP core.
ssize		RapidIO standard message payload size. Indicates how the hardware should segment the outgoing message by specifying the maximum number of double-words per packet. If the message is a multi-segment message, this field remains the same for all outgoing segments. All segments of the message, except for the last segment, have payloads equal to this size. The last message segment may be equal or less than this size. Maximum message size for a 16-segment message is shown below. Message_length/16 must be less than or equal to Ssize, if not, the message is not sent and CC 101b is set. Written by the DSP core.
	0000b - 1000b	Reserved
	1001b	1 Double-word payload (Supports up to a 128-byte message)
	1010b	2 Double-word payload (Supports up to a 256-byte message)
	1011b	4 Double-word payload (Supports up to a 512-byte message)
	1100b	8 Double-word payload (Supports up to a 1024-byte message)
	1101b	16 Double-word payload (Supports up to a 2048-byte message)
	1110b	32 Double-word payload (Supports up to a 4096-byte message)
	1111b	Reserved

(1) An error transfer completion code indicates an error in one or more segments of a transmitted multi-segment message.

Table 21. TX Buffer Descriptor Field Definitions (continued)

Field	Value	Description
mailbox		Destination Mailbox: Specifies the mailbox to which the message was sent. Written by the DSP core.
	000000b	Mailbox 0
	000001b	Mailbox 1
	. . .	
	000100b	Mailbox 4
	. . .	
	111111b	Mailbox 63 For multi-segment messages, only the two LSBs of this mailbox are valid. Hardware ignores the four MSBs if the incoming message has multiple segments.

Once the port controls the buffer descriptor, the DEST_ID field can be queried to determine flow control. If the transaction has been flow controlled, the DMA bus READ request is postponed so that the TX buffer space is not wasted. Because buffer descriptors cannot be reordered in the link list, if the transaction at the head of the buffer descriptor queue is flow controlled, head-of-line (HOL) blocking will occur on that queue. When this occurs, all transactions located in that queue are stalled. To counter the effects and reduce back-up of more TX packets, multiple queues are available. The peripheral supports a total of 16 assignable TX queues and their associated TX DMA state registers. The transmission order between queues is based on a programmable weighted round-robin scheme at the message level. The programmable control registers are shown in Figure 23. This scheme allows configurability of the queue transmission order, as well as the weight of each queue within the round robin.

The TX state machine begins by processing the current TX_Queue_Map(n). It will attempt to process the queue and number of buffer descriptors from that queue programmed in this mapping entry. Then it will move to TX_Queue_Map(n+1), followed by TX_Queue_Map(n+2) and so forth. It is important to note that this mapping order is fixed in a circular pattern. Each mapper can point to any queue and multiple mappers can point to a single queue. If a mapper points to an inactive queue, the peripheral recognizes this and moves to the next mapper. In order for an active queue to transmit packets, at least one mapper must be pointing to that queue. The default settings dictate an equally weighted round-robin that starts on queue0 and increments by one until reaching queue15.

The round-robin scheme does not provide precise control over the order of data sent out of the device. The ordering of the messages provided by the entries in the Weighted Round Robin Programming Registers is not an absolute guarantee of the actual transmission order or receive order of the messages. For example, take a case where there are two active queues and the TX_Queue_Map registers are setup to continuously send 2 messages from Queue 0, followed by 1 message from Queue 1. If the first message from Queue 0 attempts to reuse a mailbox/letter combination already in use (Content Addressable Memory (CAM) violation), or fails to gain outbound credit due to buffer congestion at a given priority, then the state machine will re-evaluate the TX_Queue_Map to decide on the next step. Since the TX_Queue_Map has been programmed to send two messages from Queue 0 before moving to Queue 1, it will re-attempt to send the same message from Queue 0 before moving on. Whether it is successful or not, the next attempt will come from Queue 1. Within a given queue, the hardware will always try to send the head buffer descriptor and can not move to the next buffer descriptor in the queue until a completion code is written. The weighted round robin control advocates, that statistically over many transmissions, the messages will be transmitted in accordance with the percentages programmed into the registers .

Network traffic can also affect the packet delivery order. The physical layer of the RapidIO peripheral can re-order packets of different priorities when fabric congestion occurs.

If message ordering is needed, the following must be obeyed:

- Multi Segmented Messages
  - If there are only two devices A sending to B where ordering has to be guaranteed:
    - - Use one TX queue
    - - Use the same priority
    - - Map all messages to the same RX queue
  - If there are multiple devices A and B both sending to C, and ordering has to be guaranteed for both:

- - Use one TX queue in each sending device
- - Use the same priority within each TX queue
- - Map all A messages to the same RX queue and all B messages to another queue by disabling the promiscuous mode and programming allowable sourceIDs.

- Single Segmented Messages
  - There will never be a retry so even if there are multiple senders:
    - - Use one TX queue in each sending device
    - - Use the same priority within each TX queue
    - - Map all messages to the same RX queue

Figure 23. Weighted Round-Robin Programming Registers - Address Offset 7E0h-7ECh

TX_QUEUE_CNTL0 - Address Offset 7E0h

<---------- TX_Queue_Map3 ---------->		<---------- TX_Queue_Map2 ---------->	
31 28	27 24	23 20	19 16
Number of Msgs	Queue Pointer	Number of Msgs	Queue Pointer
R/W-0h	R/W-3h	R/W-0	R/W-2h

<---------- TX_Queue_Map1 ---------->		<---------- TX_Queue_Map0 ---------->	
15 12	11 8	7 4	3 0
Number of Msgs	Queue Pointer	Number of Msgs	Queue Pointer
R/W-0h	R/W-1h	R/W-0h	R/W-0h

TX_QUEUE_CNTL1 - Address Offset 7E4h

<---------- TX_Queue_Map7 ---------->		<---------- TX_Queue_Map6 ---------->	
31 28	27 24	23 20	19 16
Number of Msgs	Queue Pointer	Number of Msgs	Queue Pointer
R/W-0	R/W-7h	R/W-0h	R/W-6h

<---------- TX_Queue_Map5 ---------->		<---------- TX_Queue_Map4 ---------->	
15 12	11 8	7 4	3 0
Number of Msgs	Queue Pointer	Number of Msgs	Queue Pointer
R/W-0h	R/W-5h	R/W-0h	R/W-4h

TX_QUEUE_CNTL2 - Address Offset 7E8h

<---------- TX_Queue_Map11 ---------->		<---------- TX_Queue_Map10 ---------->	
31 28	27 24	23 20	19 16
Number of Msgs	Queue Pointer	Number of Msgs	Queue Pointer
R/W-0h	R/W-Bh	R/W-0h	R/W-Ah

<---------- TX_Queue_Map9 ---------->		<---------- TX_Queue_Map8 ---------->	
15 12	11 8	7 4	3 0
Number of Msgs	Queue Pointer	Number of Msgs	Queue Pointer
R/W-0h	R/W-9h	R/W-0h	R/W-8h

TX_QUEUE_CNTL3 - Address Offset 7ECh

<---------- TX_Queue_Map15 ---------->		<---------- TX_Queue_Map14 ---------->	
31 28	27 24	23 20	19 16
Number of Msgs	Queue Pointer	Number of Msgs	Queue Pointer
R/W-0h	R/W-Fh	R/W-0h	R/W-Eh

<---------- TX_Queue_Map13 ---------->		<---------- TX_Queue_Map12 ---------->	
15 12	11 8	7 4	3 0
Number of Msgs	Queue Pointer	Number of Msgs	Queue Pointer
R/W-0h	R/W-Dh	R/W-0h	R/W-Ch

Table 22. Weighted Round-Robin Programming Registers - Address Offset 7E0h-7ECh

Field Pair	Register[Bits]	Field	Value	Description
TX_Queue_Map0	TX_QUEUE_CNTL0[3-0]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL0[7-4]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map1.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages
TX_Queue_Map1	TX_QUEUE_CNTL0[11-8]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL0[15-12]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map2.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages
TX_Queue_Map2	TX_QUEUE_CNTL0[19-16]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL0[23-20]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map3.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages
TX_Queue_Map3	TX_QUEUE_CNTL0[27-24]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL0[31-28]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map4.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages
TX_Queue_Map4	TX_QUEUE_CNTL1[3-0]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL1[7-4]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map5.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages

Table 22. Weighted Round-Robin Programming Registers - Address Offset 7E0h-7ECh (continued)

Field Pair	Register[Bits]	Field	Value	Description
TX_Queue_Map5	TX_QUEUE_CNTL1[11-8]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL1[15-12]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map6.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages
TX_Queue_Map6	TX_QUEUE_CNTL1[19-16]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL1[23-20]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map7.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages
TX_Queue_Map7	TX_QUEUE_CNTL1[27-24]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL1[31-28]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map8.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages
TX_Queue_Map8	TX_QUEUE_CNTL2[3-0]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL2[7-4]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map9.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages
TX_Queue_Map9	TX_QUEUE_CNTL2[11-8]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL2[15-12]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map10.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages

Table 22. Weighted Round-Robin Programming Registers - Address Offset 7E0h-7ECh (continued)

Field Pair	Register[Bits]	Field	Value	Description
TX_Queue_Map10	TX_QUEUE_CNTL2[19-16]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL2[23-20]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map11.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages
TX_Queue_Map11	TX_QUEUE_CNTL2[27-24]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL2[31-28]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map12.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages
TX_Queue_Map12	TX_QUEUE_CNTL3[3-0]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL3[7-4]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map13.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages
TX_Queue_Map13	TX_QUEUE_CNTL3[11-8]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL3[15-12]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map14.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages
TX_Queue_Map14	TX_QUEUE_CNTL3[19-16]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL3[23-20]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map15.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages

Table 22. Weighted Round-Robin Programming Registers - Address Offset 7E0h-7ECh (continued)

Field Pair	Register[Bits]	Field	Value	Description
TX_Queue_Map15	TX_QUEUE_CNTL3[27-24]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL3[31-28]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map0.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages

The TX queues are treated differently than the RX queues. A TX queue can mix single and multi-segment message buffer descriptors. The software manages the queue usage.

All outgoing message segments have responses that indicate the status of the transaction. Responses may indicate DONE, ERROR or RETRY. A buffer descriptor may be released back to CPU control (OWNERSHIP = 0), only after all segment responses are received, or alternatively if a response timeout occurs. Timeouts and response evaluation have high priority in the state-machine since they are the only means to release TX packet resources. The CC is set in the buffer descriptor to indicate the response status to the CPU. If there is a RETRY response, the TX CPPI module will immediately retry the packet before continuing to the next queue in the round-robin loop, as long as the RETRY_COUNT is not exceeded. Once this limit is exceeded, the buffer can be released back to CPU control with the appropriate CC set. Retry of a message segment does not imply retrying a whole message. Only segments for which a RETRY response is received should be re-transmitted. This will involve calculating the correct starting point within the TX data buffer based on the failed segment number and message length. To achieve respectable performance, the peripheral must not wait for a message/segment response before sending out the next packet.

Since RapidIO allows for out-of-order responses, the TX CPPI hardware must support this functionality. As responses are received, the hardware updates the corresponding TX buffer descriptor to reflect the status. However, if the response is out-of-order, the hardware does not update the CP or set the corresponding interrupt. Only after all preceding outstanding message responses are received, will the CP and interrupt be updated. This ensures that a contiguous block of buffer descriptors, starting at the oldest outstanding descriptor, has been processed by the hardware and is ready for the CPU to reclaim the buffers.

A transaction timeout is used by all outgoing message and direct I/O packets. It has the same value and is analogous to the request-to-response timer discussed in the RX CPPI and LSU sections, which is defined by the 24-bit value in the port response time-out CSR (See Section 2.3.3.3). The *RapidIO Interconnect Specification* states that the maximum time interval (all 1s) is between 3 and 6 seconds. A logical layer timeout occurs if the response packet is not received before a countdown timer (initialized to this CSR value) reaches zero. Since transaction responses can be acknowledged out-of-order, a timer is needed for each supported outstanding packet in the TX queue. Each outstanding packet response timer requires a 4-bit register. The register is loaded with the current timecode when the transaction is sent. Each time the timecode changes, a 4-bit compare is done to the 16 outstanding packet registers. If the register becomes equal to the timecode again, without a response being seen, then the transaction has timed out and the buffer descriptor is written.

Essentially, instead of the 24-bit value representing the period of the response timer, the period is now defined as $P = (2^{24} \times 16)/F$. This means the countdown timer frequency needs to be 44.7-89.5 MHz for a 6-3 second response timeout. Since the needed timer frequency is derived from the DMA bus clock (which is device dependent), the hardware supports a programmable configuration register field to properly scale the clock frequency. This configuration register field is described in the Peripheral Setting Control register (address offset 0020h).

The CPU initiates a TX queue teardown by writing to the TX Queue Teardown command register. Teardown of a TX queue will cause the following actions:

- No new messages will be sent.
- All messages (single and multi-segment) already started will be completed.

  - Failing to complete the message TX would leave an active receiver blocked waiting for the final segments until the transaction eventually times-out.
  - Note that normal TX State Machine operation is to not send any more segments once an error response has been received on any segment. So if the receiver has also been torn-down (and is receiving error responses) multi-segment transmit will complete as soon as all in-transit segments have been responded to.

- When all in-transit messages/segments have been responded to, teardown will be completed as follows:
  - If the queue is active, the teardown bit will be set in the next buffer descriptor in the queue. The peripheral completes the teardown procedure by clearing the HDP register, setting the CP register to FFFFFFFCh, and issuing an interrupt for the given queue. The teardown command register bit is automatically cleared by the peripheral.
  - If the queue is in-active (no additional buffer descriptors available), or becomes inactive after a message in transmission is completed, no buffer descriptor fields are written. The HDP register and the CP register remain unchanged. An interrupt is not issued. The teardown command register bit is automatically cleared by the peripheral.
  - Because of topology differences between flow's response, packets may arrive in a different order to the order of requests.

After the teardown process is complete and the interrupt is serviced by the CPU, software must re-initialize the TX queue to restart normal operation.

2.3.4.3 Reset and Power Down State

Upon reset, the CPPI module must be configured by the CPU. The CPU sets up the receive and transmit queues in memory. Then the CPU updates the CPPI module with the appropriate RX/TX DMA state head descriptor pointer, so the peripheral knows with which buffer descriptor address to start. Additionally, the CPU must provide the CPPI module with initial buffer descriptor values for each data buffer.

The CPPI module can be powered down if the message passing protocol is not being supported in the application. For example, if the direct I/O protocol is being used for data transfers, powering down the CPPI module will save power. In this situation, the buffer descriptor queue SRAMs and mailbox mapper logic should be powered down. Clocks should be gated to these blocks while in the power down state. Section 2.3.10 describes this in detail.

2.3.4.4 Message Passing Software Requirements

Software performs the following functions for messaging:

RX Operation

- Assigns Mailbox-to-queue mapping and allowable SourceIDs/mailbox- Queue Mapping
- Sets up associated buffer descriptor memory - CPPI RAM or L2 RAM
- Link-lists the buffer descriptors, next_descriptor_pointer
- Assigns single segment (256-byte payload) and multi-segment (4K-byte payload) buffers to queues buffer_length
- Assigns buffer descriptor to data buffer, buffer_pointer
- Gives control of the buffer to the peripheral, ownership = 1

Configures and initiates RX queues

- Assigns Head Descriptor Pointer, HDP, for up to 16 queues: RX DMA State HDP
- Port begins to consume buffers beginning with HDP descriptor and sets ownership = 0 for each buffer descriptor used. Writes Completion Pointer, CP, RX DMA State CP and moves to next buffer.
- Port hardware generates pending interrupt when CP is written. Physical interrupt generated when Interrupt Pacing Count down timer = 0.

Processes interrupt

- Determines ICSR bit and process corresponding queue until ownership = 1 or eoq = 1
- Sets processed buffer descriptor ownership = 1
- Writes CP value of last buffer descriptor processed

- Port hardware clears ICSR bit only if the CP value written by CPU equals port written value in the RX DMA State CP register
- Resets interrupt pacing value

TX Operation

Sets up associated buffer descriptor memory - CPPI RAM or L2 RAM

- Link-lists the buffer descriptors, next_descriptor_pointer
- Assigns buffer descriptor to data buffer, buffer_pointer
- CPU writes buffer descriptors and sets ownership = 1 for each used.
- Specifies RIO fields: Dest_id, Pri, tt, Mailbox
- Sets parameters: PortID, Message_length
- Port starts queue transmit on CPU write to HDP for up to 16 queues - TX DMA State HDP
- Port processes corresponding queues until ownership = 0 or next_descriptor_pointer = all 0s. Port sets eoq = 1 and writes all 0s to the HDP.
- When each packet transmission is complete, the port sets ownership = 0 and issues an interrupt to the CPU by writing the last processed buffer descriptor address to the CP, TX DMA State CP

Processes interrupt

- The CPU processes the buffer queue to reclaim buffers. If ownership = 0, the packet has been transmitted and the buffer is reclaimed.
- CPU processes the queue until eoq = 1 or ownership = 1
- CPU determines all packets have been transmitted if ownership = 0, eoq = 1, and next_descriptor_pointer = all 0s in last processed buffer descriptor
- CPU acknowledges the interrupt after re-claiming all available buffer descriptors.
- CPU acknowledges the interrupt by writing the CP value
- This value is compared against the port written value in the TX DMA State CP register, if equal, the interrupt is de-asserted.

Initialization Example

```
SRIO_REGS->Queue0_RXDMA_HDP   = 0 ;
SRIO_REGS->Queue1_RXDMA_HDP   = 0 ;
SRIO_REGS->Queue2_RXDMA_HDP   = 0 ;
SRIO_REGS->Queue3_RXDMA_HDP   = 0 ;
SRIO_REGS->Queue4_RXDMA_HDP   = 0 ;
SRIO_REGS->Queue5_RXDMA_HDP   = 0 ;
SRIO_REGS->Queue6_RXDMA_HDP   = 0 ;
SRIO_REGS->Queue7_RXDMA_HDP   = 0 ;
SRIO_REGS->Queue8_RXDMA_HDP   = 0 ;
SRIO_REGS->Queue9_RXDMA_HDP   = 0 ;
SRIO_REGS->Queue10_RXDMA_HDP  = 0 ;
SRIO_REGS->Queue11_RXDMA_HDP  = 0 ;
SRIO_REGS->Queue12_RXDMA_HDP  = 0 ;
SRIO_REGS->Queue13_RXDMA_HDP  = 0 ;
SRIO_REGS->Queue14_RXDMA_HDP  = 0 ;
SRIO_REGS->Queue15_RXDMA_HDP  = 0 ;
```

Queue Mapping

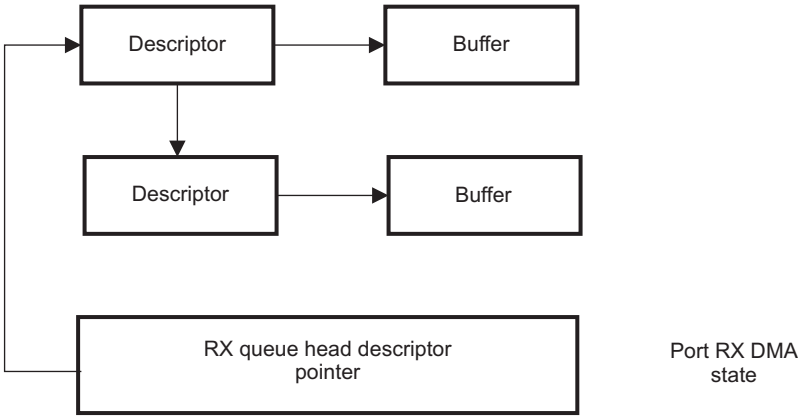
```
SRIO_REGS->RXU_MAP01_L = CSL_FMK( SRIO_RXU_MAP01_L_LETTER_MASK, 3)    |
                         CSL_FMK( SRIO_RXU_MAP01_L_MAILBOX_MASK, 0x3F)|
                         CSL_FMK( SRIO_RXU_MAP01_L_LETTER, 0)         |
                         CSL_FMK( SRIO_RXU_MAP01_L_MAILBOX, 1)        |
                         CSL_FMK( SRIO_RXU_MAP01_L_SOURCEID, 0xBEEF);
SRIO_REGS->RXU_MAP01_H = CSL_FMK( SRIO_RXU_MAP01_H_TT, 1)         |
                         CSL_FMK( SRIO_RXU_MAP01_H_QUEUE_ID, 0)   |
                         CSL_FMK( SRIO_RXU_MAP01_H_PROMISCUOUS, 1)|
                         CSL_FMK( SRIO_RXU_MAP01_H_SEGMENT_MAPPING, 1);
```

RX Buffer Descriptor

```
RX_DESCP0_0->RXDESC0 = CSL_FMK( SRIO_RXDESC0_N_POINTER,(int )RX_DESCP0_1 ); //link to RX_DESCP0_1

RX_DESCP0_0->RXDESC1 = CSL_FMK( SRIO_RXDESC1_B_POINTER,(int )&rcvBuff1[0] );

RX_DESCP0_0->RXDESC2 = CSL_FMK( SRIO_RXDESC2_SRC_ID, 0xBEEF)|
```

```
                          CSL_FMK( SRIO_RXDESC2_PRI, 1)        |
                          CSL_FMK( SRIO_RXDESC2_TT, 1)         |
                          CSL_FMK( SRIO_RXDESC2_MAILBOX, 0);

RX_DESCP0_0->RXDESC3 = CSL_FMK( SRIO_RXDESC3_SOP,1 )      |
                          CSL_FMK( SRIO_RXDESC3_EOP,1 )      |
                          CSL_FMK( SRIO_RXDESC3_OWNERSHIP,1 )|
                          CSL_FMK( SRIO_RXDESC3_EOQ,1 )      |
                          CSL_FMK( SRIO_RXDESC3_TEARDOWN,0 ) |
                          CSL_FMK( SRIO_RXDESC3_CC,0 )       |
                          CSL_FMK( SRIO_RXDESC3_MESSAGE_LENGTH,MLEN_512DW);

RX_DESCP0_1->RXDESC0 = CSL_FMK( SRIO_RXDESC0_N_POINTER, 0); //end of message

RX_DESCP0_1->RXDESC1 = CSL_FMK( SRIO_RXDESC1_B_POINTER,(int )&rcvBuff2[0] );

RX_DESCP0_1->RXDESC2 = CSL_FMK( SRIO_RXDESC2_SRC_ID, 0xBEEF)|
                          CSL_FMK( SRIO_RXDESC2_PRI, 1)        |
                          CSL_FMK( SRIO_RXDESC2_TT, 1)         |
                          CSL_FMK( SRIO_RXDESC2_MAILBOX, 1);

RX_DESCP0_1->RXDESC3 = CSL_FMK( SRIO_RXDESC3_SOP,1 )      |
                          CSL_FMK( SRIO_RXDESC3_EOP,1 )      |
                          CSL_FMK( SRIO_RXDESC3_OWNERSHIP,1 )|
                          CSL_FMK( SRIO_RXDESC3_EOQ,1 )      |
                          CSL_FMK( SRIO_RXDESC3_TEARDOWN,0 ) |
                          CSL_FMK( SRIO_RXDESC3_CC,0 )       |
                          CSL_FMK( SRIO_RXDESC3_MESSAGE_LENGTH,MLEN_512DW );
```

Figure 24. RX Buffer Descriptors

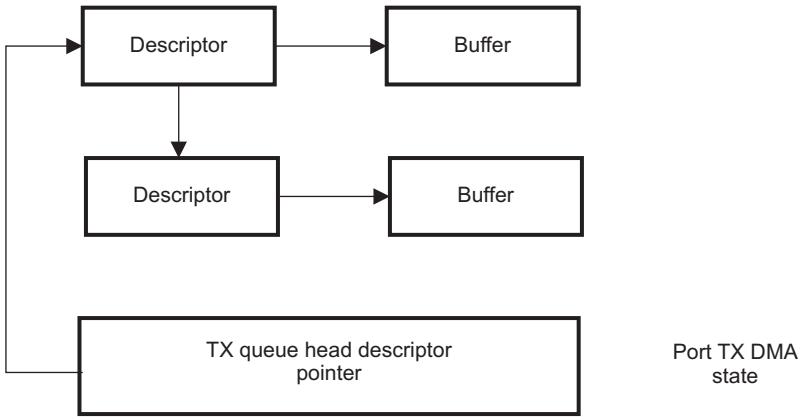


TX Buffer Descriptor

```
TX_DESCP0_0->TXDESC0 = CSL_FMK( SRIO_TXDESC0_N_POINTER,(int )TX_DESCP0_1 );    //link to
TX_DESCP0_1

TX_DESCP0_0->TXDESC1 = CSL_FMK( SRIO_TXDESC1_B_POINTER,(int )&xmtBuff1[0] );   //Buffer Pointer

TX_DESCP0_0->TXDESC2 = CSL_FMK( SRIO_TXDESC2_DESTID, 0xBEEF)   |
                          CSL_FMK( SRIO_TXDESC2_PRI, 1)           |
                          CSL_FMK( SRIO_TXDESC2_TT, 1)            |
                          CSL_FMK( SRIO_TXDESC2_PORTID, 3)        |
                          CSL_FMK( SRIO_TXDESC2_SSIZE, SSIZE_256B)|
                          CSL_FMK( SRIO_TXDESC2_MAILBOX, 0);

TX_DESCP0_0->TXDESC3 = CSL_FMK( SRIO_TXDESC3_SOP,1 )        |
                          CSL_FMK( SRIO_TXDESC3_EOP,1 )        |
                          CSL_FMK( SRIO_TXDESC3_OWNERSHIP,1 )  |
                          CSL_FMK( SRIO_TXDESC3_EOQ,1 )        |
                          CSL_FMK( SRIO_TXDESC3_TEARDOWN,0 )   |
                          CSL_FMK( SRIO_TXDESC3_RETRY_COUNT,0 )|
                          CSL_FMK( SRIO_TXDESC3_MESSAGE_LENGTH,MLEN_512DW );

TX_DESCP0_1->TXDESC0 = CSL_FMK( SRIO_TXDESC0_N_POINTER, 0);      //end of message

TX_DESCP0_1->TXDESC1 = CSL_FMK( SRIO_TXDESC1_B_POINTER,(int )&xmtBuff2[0] );
```

```
TX_DESCP0_1->TXDESC2 = CSL_FMK( SRIO_TXDESC2_DESTID, 0xBEEF)   |
                       CSL_FMK( SRIO_TXDESC2_PRI, 1)           |
                       CSL_FMK( SRIO_TXDESC2_TT, 1)            |
                       CSL_FMK( SRIO_TXDESC2_PORTID, 3)        |
                       CSL_FMK( SRIO_TXDESC2_SSIZE, SSIZE_256B)|
                       CSL_FMK( SRIO_TXDESC2_MAILBOX, 1);

TX_DESCP0_1->TXDESC3 = CSL_FMK( SRIO_TXDESC3_SOP,1 )        |
                       CSL_FMK( SRIO_TXDESC3_EOP,1 )        |
                       CSL_FMK( SRIO_TXDESC3_OWNERSHIP,1 )  |
                       CSL_FMK( SRIO_TXDESC3_EOQ,1 )        |
                       CSL_FMK( SRIO_TXDESC3_TEARDOWN,0 )   |
                       CSL_FMK( SRIO_TXDESC3_RETRY_COUNT,0 )|
                       CSL_FMK( SRIO_TXDESC3_MESSAGE_LENGTH,MLEN_512DW );
```

Figure 25. TX Buffer Descriptors



Start Message Passing

```
SRIO_REGS->Queue0_RXDMA_HDP = (int )RX_DESCP0_0 ;

SRIO_REGS->Queue0_TXDMA_HDP = (int )TX_DESCP0_0 ;
```

For a message passing software programming example, see Section A.3.

2.3.5 Maintenance

The type 8 MAINTENANCE packet format accesses the RapidIO capability registers (CARs), command and status registers (CSRs), and data structures. Unlike other request formats, the type 8 packet format serves as both the request and the response format for maintenance operations. Type 8 packets contain no addresses and only contain data payloads for write requests and read responses. All configuration register read accesses are word (4-byte) accesses. All configuration register write accesses are also word (4-byte) accesses.

The wrsize field specifies the maximum size of the data payload for multiple double-word transactions. The data payload may not exceed that size but may be smaller if desired. Both the maintenance read and the maintenance write request generate the appropriate maintenance response.

The maintenance port-write operation is a write operation that does not have guaranteed delivery and does not have an associated response. This maintenance operation is useful for sending messages such as error indicators or status information from a device that does not contain an endpoint, such as a switch. The data payload is typically placed in a queue in the targeted endpoint and an interrupt is typically generated to a local processor. A port-write request to a queue that is full or busy servicing another request may be discarded.

```
SRIO_REGS->LSU1_REG0 = CSL_FMK( SRIO_LSU1_REG0_RAPIDIO_ADDRESS_MSB,0 );
SRIO_REGS->LSU1_REG1 = CSL_FMK( SRIO_LSU1_REG1_ADDRESS_LSB_CONFIG_OFFSET, (int )car_csr );
SRIO_REGS->LSU1_REG2 = CSL_FMK( SRIO_LSU1_REG2_DSP_ADDRESS, (int )&xmtBuff[0]);
SRIO_REGS->LSU1_REG3 = CSL_FMK( SRIO_LSU1_REG3_BYTE_COUNT,byte_count );
SRIO_REGS->LSU1_REG4 = CSL_FMK( SRIO_LSU1_REG4_OUTPORTID,0 )      |
                       CSL_FMK( SRIO_LSU1_REG4_PRIORITY,0 )       |
                       CSL_FMK( SRIO_LSU1_REG4_XAMSB,0 )          |    //no extended address
```

```
                        CSL_FMK( SRIO_LSU1_REG4_ID_SIZE,1 )         |
                        CSL_FMK( SRIO_LSU1_REG4_DESTID,0xBEEF )     |
                        CSL_FMK( SRIO_LSU1_REG4_INTERRUPT_REQ,0 );
SRIO_REGS->LSU1_REG5 = CSL_FMK( SRIO_LSU1_REG5_DRBLL_INFO,0x0000 )|
                        CSL_FMK( SRIO_LSU1_REG5_HOP_COUNT,0x03 )    |
                        CSL_FMK( SRIO_LSU1_REG5_PACKET_TYPE,type );   //type = REQ_MAINT_RD
```

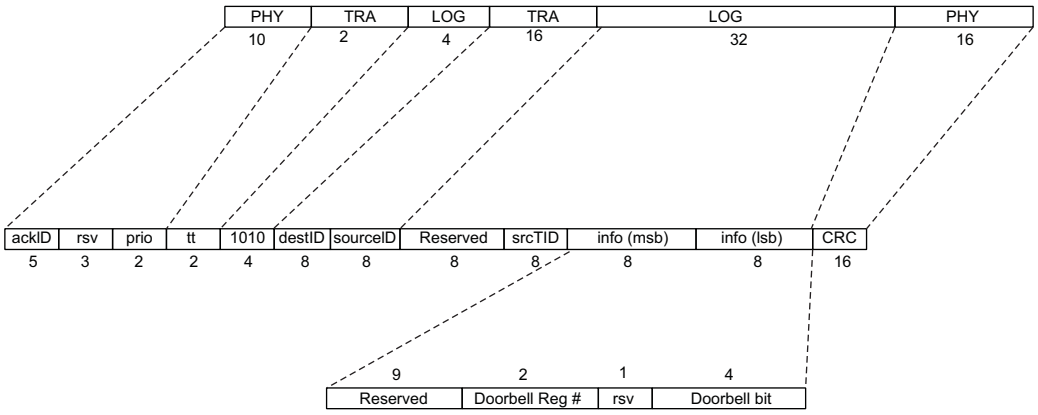
For a LSU programming example, see Section A.2.

2.3.6 Doorbell Operation

The doorbell operation is shown in Figure 26. It consists of the DOORBELL and RESPONSE transactions (typically a DONE response), and it is used by a processing element to send a very short message to another processing element through the interconnect fabric. The DOORBELL transaction contains the info field to hold information and does not have a data payload. This field is software-defined and can be used for any desired purpose; see the *RapidIO Interconnect Specification*, Section 3.1.4, *Type 10 Packet Formats (Doorbell Class)*, for information about the info field. A processing element that receives a doorbell transaction takes the packet and puts it in a doorbell message queue within the processing element. This queue may be implemented in hardware or in local memory. This behavior is similar to that of typical message passing mailbox hardware. The local processor is expected to read the queue to determine the sending processing element and the info field, and determine what action to take.

The DOORBELL functionality is user-defined, but this packet type is commonly used to initiate DSP core (CPU) interrupts. A DOORBELL packet is not associated with a particular data packet that was previously transferred, so the info field of the packet must be configured to reflect the DOORBELL bit to be serviced for the correct TID (Transfer Information Descriptor) information to be processed.

Figure 26. Doorbell Operation



The DOORBELL packet's 16-bit INFO field indicates which DOORBELL register interrupt bit to set. There are four DOORBELL registers, each currently with 16 bits, allowing 64 interrupt sources or circular buffers. For assignment of the 16 bits of DOORBELL_INFO field, see Table 23. Each bit can be assigned to any core as described below by the Interrupt Condition Routing Registers. Additionally, each status bit is user-defined for the application. For instance, it may be desirable to support multiple priorities with multiple TID circular buffers per core if control data uses a high priority (for example, priority = 2), while data packets are sent on priority 0 or 1. This allows the control packets to have preference in the switch fabric and arrive as quickly as possible. Since it may be required to interrupt the CPU for both data and control packet processing separately, separate circular buffers are used, and DOORBELL packets need to distinguish between them for interrupt servicing. If any reserved bit in the DOORBELL_INFO field is set, an error response is sent. If the DOORBELL_INFO field indicates an interrupt bit (ICSR) which is already set, a retry response is sent.

Table 23. Examples of DOORBELL_INFO Designations (See Figure 26)

DOORBELL_INFO Field Segments				Value Written To DOORBELL_INFO Field Of LSUn_REG5	Associated Doorbell Interrupt Routing Bits	Mapped To This Doorbell Interrupt Status Bit
Reserved	Doorbell Reg #	Rsv	Doorbell Bit			
000000000b	00b	0b	0000b	0000h	DOORBELL0_ICRR[3:0]	DOORBELL0_ICSR[0]
000000000b	00b	0b	1001b	0009h	DOORBELL0_ICRR2[7:4]	DOORBELL0_ICSR[9]
000000000b	01b	0b	0111b	0027h	DOORBELL1_ICRR[31:28]	DOORBELL1_ICSR[7]
000000000b	01b	0b	1100b	002Ch	DOORBELL1_ICRR2[19:16]	DOORBELL1_ICSR[12]
000000000b	10b	0b	0101b	0045h	DOORBELL2_ICRR[23:20]	DOORBELL2_ICSR[5]
000000000b	10b	0b	1111b	004Fh	DOORBELL2_ICRR2[31:28]	DOORBELL2_ICSR[15]
000000000b	11b	0b	0110b	0066h	DOORBELL3_ICRR[27:24]	DOORBELL3_ICSR[6]
000000000b	11b	0b	1011b	006Bh	DOORBELL3_ICRR2[15:12]	DOORBELL3_ICSR[11]

```
SRIO_REGS->LSU1_REG0 = CSL_FMK( SRIO_LSU1_REG0_RAPIDIO_ADDRESS_MSB,0 );
SRIO_REGS->LSU1_REG1 = CSL_FMK( SRIO_LSU1_REG1_ADDRESS_LSB_CONFIG_OFFSET, 0);
SRIO_REGS->LSU1_REG2 = CSL_FMK( SRIO_LSU1_REG2_DSP_ADDRESS, 0);
SRIO_REGS->LSU1_REG3 = CSL_FMK( SRIO_LSU1_REG3_BYTE_COUNT, 0 );
SRIO_REGS->LSU1_REG4 = CSL_FMK( SRIO_LSU1_REG4_OUTPORTID,1 )  |
                       CSL_FMK( SRIO_LSU1_REG4_PRIORITY,0 )   |
                       CSL_FMK( SRIO_LSU1_REG4_XAMSB,0 )      |
                       CSL_FMK( SRIO_LSU1_REG4_ID_SIZE,1 )    |
                       CSL_FMK( SRIO_LSU1_REG4_DESTID,0xBEEF )|
                       CSL_FMK( SRIO_LSU1_REG4_INTERRUPT_REQ,0 );
SRIO_REGS->LSU1_REG5 = CSL_FMK( SRIO_LSU1_REG5_DRBLL_INFO,0x0000 )|
                       CSL_FMK( SRIO_LSU1_REG5_HOP_COUNT,0x03 )   |
                       CSL_FMK( SRIO_LSU1_REG5_PACKET_TYPE,type );//type = DOORBELL
```

For a doorbell programming example, see Section A.2.

2.3.7 Atomic Operations

The Atomic operation is a combination read and write operation. The destination reads the data at the specified address, returns the read data to the requestor, performs the required operation to the data, and then writes the modified data back to the specified address without allowing any intervening activity to that address. Defined operations are increment, decrement, test-and-swap, set, and clear (see Table 3, *Packet Type*). Of these, only test-and-swap requires the requesting processing element to supply data. Incoming Atomic operations which target the device are not supported for internal L2 memory or registers. Atomic request operations to external devices are supported and have a response packet.

Request Atomic operations (Ftype 2) never contain a data payload. These operations are like NREAD (24h) transactions. The data payload size for the response to an Atomic transaction is 8 bytes. The addressing scheme defined for the read portion of the Atomic transaction also controls the size of the atomic operation in memory so that the bytes are contiguous and of size byte, half-word (2 bytes), or word (4 bytes), and are aligned to that boundary and byte lane as with a regular read transaction. Double-word (8-byte), 3-byte, 5-byte, 6-byte, and 7-byte Atomic transactions are not allowed.

Atomic test-and-swap operations (Ftype 5) to external devices are limited to a payload of one double-word (8 bytes). These operations are like NWRITE with response (55h) transactions. The addressing scheme defined for the write transactions also controls the size of the Atomic operation in memory so that the bytes are contiguous and of size byte, half-word (2 bytes), or word (4 bytes), and are aligned to that boundary and byte lane as with a regular write transaction. Double-word (8-byte), 3-byte, 5-byte, 6-byte, and 7-byte Atomic test-and-swap transactions are not allowed. Upon receipt of the request, the targeted device swaps the contents of the specified memory location and the payload if the contents of the memory location are all 0s. The contents of the memory location are returned, and the appropriate completion code is set in the LSU status register (LSU*n*_REG6). The completion codes are listed in Table 15.

2.3.8 Congestion Control

The RapidIO *Logical Layer Flow Control Extensions Specification*. This section describes the requirements and implementation of congestion control within the peripheral.

The peripheral is notified of switch fabric congestion through type 7 RapidIO packets. The packets are referred to as Congestion Control Packets (CCPs). The purpose of these packets is to turn off (Xoff), or turn on (Xon) specific flows defined by DESTID and PRIORITY of outgoing packets. CCPs are sent at the highest priority in an attempt to address fabric congestion as quickly as possible. CCPs do not have a response packet and they do not have guaranteed delivery.

When the peripheral receives an Xoff CCP, the peripheral must block outgoing LSU and CPPI packets that are destined for that flow. When the peripheral receives an Xon, the flow may be enabled. Since CCPs may arrive from different switches within the fabric, it is possible to receive multiple Xoff CCPs for the same flow. For this reason, the peripheral must maintain a table and count of Xoff CCPs for each flow. For example, if two Xoff CCPs are received for a given flow, two Xon CCPs must be received before the flow is enabled.

Since CCPs do not have guaranteed delivery and can be dropped by the fabric, an implicit method of enabling an Xoff'd flow must exist. A simple timeout method is used. Additionally, flow control checks can be enabled or disabled through the Transmit Source Flow Control Masks. Received CCPs are not passed through the DMA bus interface.

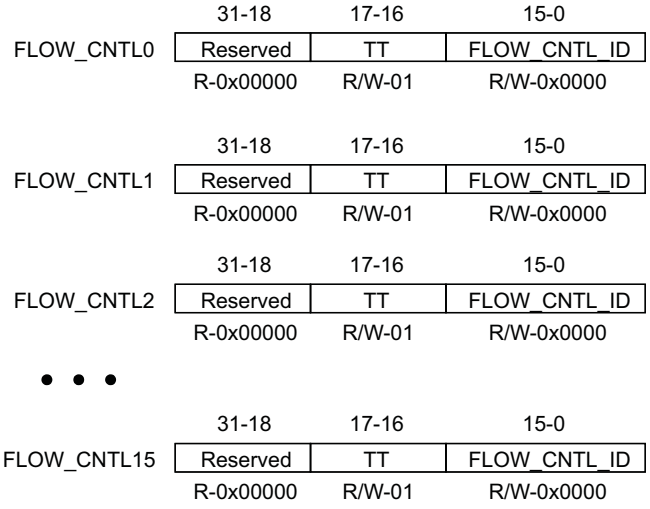
2.3.8.1 Detailed Description

To avoid large and complex table management, a basic scheme is implemented for congestion management. The primary goal is to avoid large parallel searches of a centralized congested route table for each outgoing packet request. The congested route table requirements and subsequent searches would be overwhelming if each possible DESTID and PRIORITY combination had its own entry. To implement a more basic scheme, the following assumptions have been made:

- A small number of flows constitute the majority of traffic, and these flows are most likely to cause congestion
- HOL blocking is undesired, but allowable for TX CPPI queues
- Flow control will be based on DESTID only, regardless of PRIORITY

The congested route table is therefore more static in nature. Instead of dynamically updating a table with each CCP's flow information as it arrives, a small finite-entry table is set up and configured by software to reflect the more critical flows it is using. Only these flows have a discrete table entry. A 16 entry table reflects 15 critical flows, leaving the sixteenth entry for general other flows, which are categorized together. Figure 27 and Table 24 summarize the DESTID table entries that are programmable by the CPU through dedicated flow control registers. A 3-bit hardware counter is implemented for table entries 0 through 14, to maintain a count of Xoff CCPs for that flow. The other flows table entry counts Xoff CCPs for all flows other than the discrete entries. The counter for this table entry has 5 bits. All outgoing flows with non-zero Xoff counts are disabled. The counter value is decremented for each corresponding Xon CCP that is received, but it is not decrement below zero. Additionally, a hardware timer exists for each table entry to turn on flows that may have been abandoned by lost Xon CCPs. The timer value is of an order of magnitude larger than the 32-bit Port Response Time-out CSR value. For this reason, each transmission source adds 2 bits to its 4-bit response time-out counter. Descriptions of this type of time-out counter are in Section 2.3.3.3 and Section 2.3.4.2. The additional 2 bits count three timecode revolutions and provide an implicit Xon timer equal to 3x the Response time-out counter value.

Figure 27. Flow Control Table Entry Registers - Address Offset 0900h-093Ch



LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

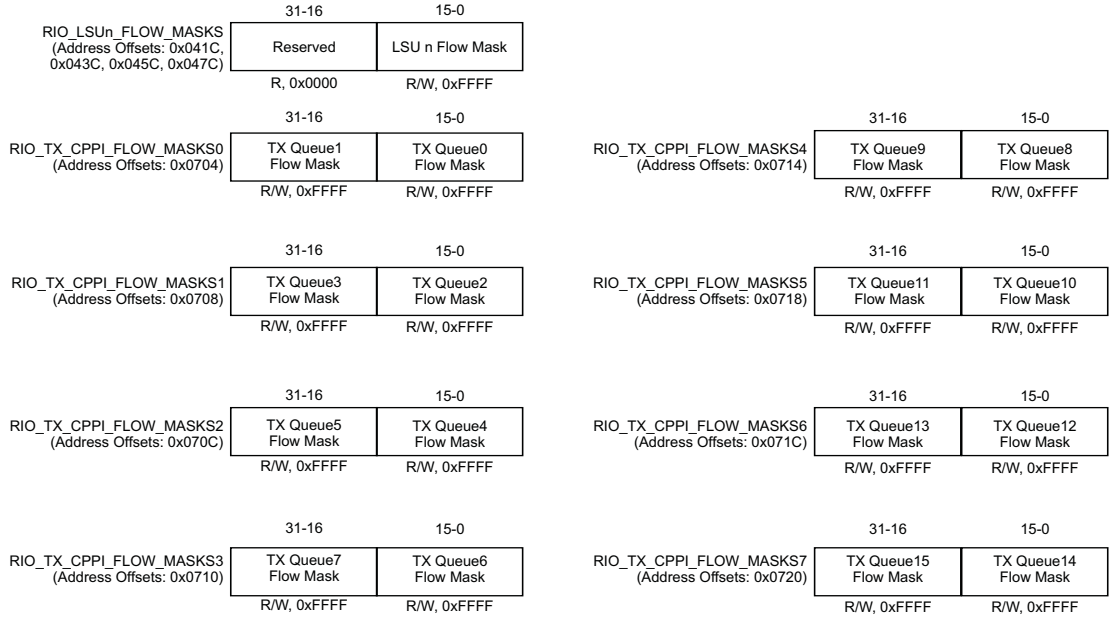
Table 24. Flow Control Table Entry Register *n* (FLOW_CNTL*n*) Field Descriptions

Bit	Field	Value	Description
31-18	Reserved	0	These read-only bits return 0s when read.
17-16	TT		Transfer type for flow *n*
		00b	8-bit destination IDs
		01b	16-bit destination IDs
		1xb	Reserved
15-0	FLOW_CNTL_ID	0000h-FFFFh	Destination ID for flow *n*. When 8-bit destination IDs are used (TT = 00b), the 8 MSBs of this field are don't care bits.

Each transmit source, including any LSU and any TX CPPI queue, indicates which of the 16 flows it uses with a 16-bit flow mask. Figure 28 illustrates the registers that contain the flow masks, and Figure 29 illustrates the general form of an individual flow mask. As can be seen from Table 25, bits 0 through 15 of the flow mask correspond to flows 0 through 15, respectively.

The CPU must configure the flow masks upon reset. The default setting is all 1s, indicating that the transmit source supports all flows. If the register is set to all 0s, the transmit source does not support any flow, and consequently, that source is never flow-controlled. If any of the table entry counters that a transmit source supports have a corresponding non-zero Xoff count, the transmit source is flow-controlled. A simple 16-bit bus indicates the Xoff state of all 16 flows and is compared to the transmit source mask register. Each source interprets this result and performs flow control accordingly. For example, an LSU module that is flow-controlled can reload its registers and attempt to send a packet to another flow, while a TX CPPI queue that is flow-controlled may create HOL blocking issues on that queue.

Figure 28. Transmit Source Flow Control Masks



LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Figure 29. Fields Within Each Flow Mask

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FL15	FL14	FL13	FL12	FL11	FL10	FL9	FL8	FL7	FL6	FL5	FL4	FL3	FL2	FL1	FL0
R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1

LEGEND: R/W = Read/Write; -*n* = Value after reset

Table 25. Fields Within Each Flow Mask

Bit	Field	Value	Description
15	FL15	0	TX source does not support Flow 15 from table entry
		1	TX source supports Flow 15 from table entry
14	FL14	0	TX source does not support Flow 14 from table entry
		1	TX source supports Flow 14 from table entry
13	FL13	0	TX source does not support Flow 13 from table entry
		1	TX source supports Flow 13 from table entry
12	FL12	0	TX source does not support Flow 12 from table entry
		1	TX source supports Flow 12 from table entry
11	FL11	0	TX source does not support Flow 11 from table entry
		1	TX source supports Flow 11 from table entry
10	FL10	0	TX source does not support Flow 10 from table entry
		1	TX source supports Flow 10 from table entry
9	FL9	0	TX source does not support Flow 9 from table entry
		1	TX source supports Flow 9 from table entry
8	FL8	0	TX source does not support Flow 8 from table entry
		1	TX source supports Flow 8 from table entry
7	FL7	0	TX source does not support Flow 7 from table entry
		1	TX source supports Flow 7 from table entry
6	FL6	0	TX source does not support Flow 6 from table entry
		1	TX source supports Flow 6 from table entry

Table 25. Fields Within Each Flow Mask (continued)

Bit	Field	Value	Description
5	FL5	0	TX source does not support Flow 5 from table entry
		1	TX source supports Flow 5 from table entry
4	FL4	0	TX source does not support Flow 4 from table entry
		1	TX source supports Flow 4 from table entry
3	FL3	0	TX source does not support Flow 3 from table entry
		1	TX source supports Flow 3 from table entry
2	FL2	0	TX source does not support Flow 2 from table entry
		1	TX source supports Flow 2 from table entry
1	FL1	0	TX source does not support Flow 1 from table entry
		1	TX source supports Flow 1 from table entry
0	FL0	0	TX source does not support Flow 0 from table entry
		1	TX source supports Flow 0 from table entry

2.3.9 Endianness

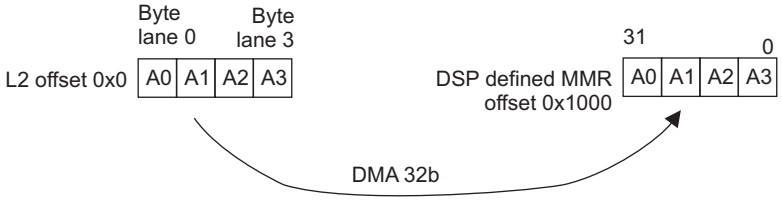
RapidIO is based on Big Endian. This is discussed in detail in Section 2.4 of the *RapidIO Interconnect Specification*. Essentially, Big Endian specifies the address ordering as the most significant bit/byte first. For example, in the 29-bit address field of a RapidIO packet (shown in Figure 6) the left-most bit that is transmitted first in the serial bit stream is the MSB of the address. Likewise, the data payload of the packet is double-word aligned Big Endian, which means the MSB is transmitted first. Bit 0 of all the RapidIO-defined MMR registers is the MSB.

All Endian-specific conversion is handled within the peripheral. For double-word aligned payloads, the data should be written contiguously into memory beginning at the specified address. Any unaligned payloads will be padded and properly aligned within the 8-byte boundary. In this case, WDPTR, RDSIZE, and WRSIZE RapidIO header fields indicate the byte position of the data within the double-word boundary. An example of an unaligned transfer is shown in Section 2.4 of the *RapidIO Interconnect Specification*.

2.3.9.1 Translation for MMR space

There are no Endian translation requirements for accessing the local MMR space. Regardless of the device memory Endian configuration, all configuration bus accesses are performed on 32-bit values at a fixed address position. The bit positions in the 32-bit word are defined by this specification. This means that a memory image which will be copied to a MMR is identical between Little Endian and Big Endian configurations. Configuration bus reads are performed in the same manner. Figure 30 illustrates the concept. The desired operation is to locally update a serial RapidIO MMR (offset 1000h) with a value of A0A1A2A3h, using the configuration bus.

Figure 30. Configuration Bus Example



When accessing RapidIO defined MMR within an external device, RapidIO allows 4 bytes, 8 bytes, or any multiple of a double-word access (up to 64 bytes) for type 8 (maintenance) packets. The peripheral only supports 4-byte accesses as the target, but can generate all sizes of request packets. RapidIO is defined as Big Endian only, and has double-word aligned Big Endian packet payloads.

2.3.9.2 Endian Conversion

The DMA also supports byte-wide accesses. The peripheral performs endian conversion on the payload if little endian is used on the device. This conversion is not only applicable for type 8 packets, but is relevant for all outgoing payloads of NWRITE, NWRITE_R, SWRITE, NREAD, and message passing.

Based on the application model, when run in little endian mode, swapping the original big-endian data based on different boundaries can be done: swap on 8-byte boundary, swap on 4-byte boundary, swap on 2-byte boundary, and swap on 1-byte boundary as shown in Table 26.

	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Original data in big-endian	0	1	2	3	4	5	6	7

Table 26. DMA Little-Endian Swapping Modes

Swapping Sequence	Mode	Byte 7	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0
Swap on 8-byte	D	0	1	2	3	4	5	6	7
Swap on 4-byte	C	4	5	6	7	0	1	2	3
Swap on 2-byte	B	6	7	4	5	2	3	0	1
Swap on 1-byte	A	7	6	5	4	3	2	1	0

The modes in Table 26 refer to the setting of the three bits in the PER_SET_CNTL (address offset 0020h) as in Table 27:

Table 27. Bits for Little-Endian Swapping

Field	Definition
lend_swap_mode[31:30]	MAU little-endian swapping mode: 00b: Mode A 01b: Mode B 10b: Mode C 11b: Mode D
lend_swap_mode[29:28]	LSU little-endian swapping mode: 00b: Mode A 01b: Mode B 10b: Mode C 11b: Mode D
lend_swap_mode[23:22]	TXU/RXU little-endian swapping mode: 00b: Mode A 01b: Mode B 10b: Mode C 11b: Mode D

2.3.10 Reset and Power Down

The RapidIO peripheral allows independent software controlled shutdown for the logical blocks listed in Table 28. With the exception of BLK0_EN for the memory-mapped registers (MMRs), when the BLK*n*_EN signals are de-asserted, the clocks are gated to these blocks, effectively providing a shutdown function.

Table 28. Reset Hierarchy

Logical Block	Bus Reset	GBL _EN	BLK0 _EN	BLK1 _EN	BLK2 _EN	BLK3 _EN	BLK4 _EN	BLK5 _EN	BLK6 _EN	BLK7 _EN	BLK8 _EN
DMA interface	√	√									
MMRs: Reset/power-down control registers	√	√									
MMRs: Non-reset/power-down control registers	√	√	√								
Interrupt handling unit (IHU)	√	√									
Traffic flow logic	√	√									
Congestion control unit (CCU)	√	√									
LSU (Direct I/O initiator)	√	√		√							
MAU (Direct I/O target)	√	√			√						
TXU (message passing initiator)	√	√				√					
RXU (message passing target)	√	√					√				
Port 0 datapath	√	√						√			
Port 1 datapath	√	√							√		
Port 2 datapath	√	√								√	
Port 3 datapath	√	√									√

Reset of the SERDES macros is handled independently of the registers discussed in this section. The SERDES can be configured to shutdown unused links or fully shutdown. SERDES TX and RX channels may be enabled/disabled by writing to bit 0 of the SERDES_CFGTX*n*_CNTL and SERDES_CFGRX*n*_CNTL registers. The PLL and remaining SERDES functional blocks can be controlled by writing to the ENPLL signal in the SERDES_CFG0_CNTL register. This bit will drive the SERDES signal input, which will gate the reference clock to these blocks internally. This reference clock is sourced from a device pin specifically for the SERDES and is not derived from the CPU clock, thus it resets asynchronously. ENPLL will disable all SERDES high-speed output clocks. Since these clocks are distributed to all the links, ENPLL should only be used to completely shutdown the peripheral. It should be noted that shutdown of SERDES links in between normal packet transmissions is not permissible for two reasons. First, the serial RapidIO sends idle packets between data packets to maintain synchronization and lane alignment. Without this mechanism, the RapidIO RX logic can be mis-aligned for both 1X and 4X ports. Second, the lock time of the SERDES PLL would need to reoccur, which would slow down the operation.

When the SERDES ENTX signal is held low, the corresponding transmitter is powered down. In this state, both outputs, TXP and TXN, will be pulled high to VDDT.

2.3.10.1 Reset and Power Down Summary

After reset, the state of the peripheral depends on the default register values.

Software can also perform a hard reset of each logical block within the peripheral via the GBL_EN and BLK*n*_EN bits. The GBL_EN bit resets the peripheral, while the rest of the device is not reset. The BLK*n*_EN bits shut down unused portions of the peripheral, which minimizes power by resetting the appropriate logical block(s) and gating off the clock to the appropriate logical block(s). This should be considered an abrupt reset that is independent of the state of the peripheral and that resets the peripheral to its original state.

Upon reset of the peripheral, the device must reestablish communication with its link partner. Depending on the system, this may include a discovery phase in which a host processor reads the peripheral's CAR/CSR registers to determine its capabilities. In its simplest form, it involves retraining the SERDES and going through the initialization phase to synchronize on bit and word boundaries by using idle and control symbols, as described in Section 5.5.2 of the Part VI of the *RapidIO Interconnect Specification*. Until the peripheral and its partner are fully initialized and ready for normal operation, the peripheral will not send any data packets or non-status control symbols.

- GBL_EN: Resets all MMRs, excluding Reset Ctl Values (0000h-01FCh). Resets all logical blocks except MMR configuration bus i/f. While asserted, the slave configuration bus is operational.
- BLK_EN0: Resets all MMRs, excluding Reset Ctl Values (0000h-01FCh). Other logical blocks are unaffected, including MMR configuration bus i/f.
- BLK_EN[n:1]: Single enable/reset per logical block. See Table 28.

2.3.10.2 Enable and Enable Status Registers

The enable and enable status registers are comprised of two global registers and nine pairs of block-specific registers. The global registers are summarized by Figure 31, Figure 32, Table 28, and Table 29. The GBL_EN register is implemented with a single enable bit. This bit is logically ORed with the reset input to the module and is fanned out to all logical blocks within the peripheral.

Figure 31. GBL_EN (Address 0030h)

31	1	0
Reserved		EN
R-0		R/W-1

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Figure 32. GBL_EN_STAT (Address 0034h)

31	24
Reserved	
R-0	

23	16
Reserved	
R-0	

15	10	9	8
Reserved		BLK8_EN_STAT	BLK7_EN_STAT
R-0		R-1	R-1

7	6	5	4	3	2	1	0
BLK6_EN_STAT	BLK5_EN_STAT	BLK4_EN_STAT	BLK3_EN_STAT	BLK2_EN_STAT	BLK1_EN_STAT	BLK0_EN_STAT	GBL_EN_STAT
R-1	R-1	R-1	R-1	R-1	R-1	R-1	R-1

LEGEND: R = Read only; -*n* = Value after reset

Table 29. Global Enable and Global Enable Status Field Descriptions

Register (Bit)	Field	Value	Description
GBL_EN(31-1)	Reserved	0	These read-only bits return 0s when read.
GBL_EN(0)	EN		Global enable. This bit controls reset to all clock domains within the peripheral.
		0	The peripheral is to be disabled (held in reset with clocks disabled).
		1	The peripheral is to be enabled.
GBL_EN_STAT(31-10)	Reserved	0	These read-only bits return 0s when read.
GBL_EN_STAT(9)	BLK8_EN_STAT		Block 8 enable status. Logical block 8 is SRIO port 3.
		0	Logical block 8 is in reset with its clock off.
		1	Logical block 8 is enabled with its clock running.
GBL_EN_STAT(8)	BLK7_EN_STAT		Block 7 enable status. Logical block 7 is SRIO port 2.
		0	Logical block 7 is in reset with its clock off.
		1	Logical block 7 is enabled with its clock running.
GBL_EN_STAT(7)	BLK6_EN_STAT		Block 6 enable status. Logical block 6 is SRIO port 1.
		0	Logical block 6 is in reset with its clock off.
		1	Logical block 6 is enabled with its clock running.
GBL_EN_STAT(6)	BLK5_EN_STAT		Block 5 enable status. Logical block 5 is SRIO port 0.
		0	Logical block 5 is in reset with its clock off.
		1	Logical block 5 is enabled with its clock running.
GBL_EN_STAT(5)	BLK4_EN_STAT		Block 4 enable status. Logical block 4 is the message receive unit (RXU).
		0	Logical block 4 is in reset with its clock off.
		1	Logical block 4 is enabled with its clock running.
GBL_EN_STAT(4)	BLK3_EN_STAT		Block 3 enable status. Logical block 3 is the message transmit unit (TXU).
		0	Logical block 3 is in reset with its clock off.
		1	Logical block 3 is enabled with clock running.
GBL_EN_STAT(3)	BLK2_EN_STAT		Block 2 enable status. Logical block 2 is the memory access unit (MAU).
		0	Logical block 2 is in reset with its clock off.
		1	Logical block 2 is enabled with its clock running.
GBL_EN_STAT(2)	BLK1_EN_STAT		Block 1 enable status. Logical block 1 is the Load/Store module, which is comprised of the four Load/Store units (LSU1, LSU2, LSU3, and LSU4).
		0	Logical block 1 is in reset with its clock off.
		1	Logical block 1 is enabled with its clock running.
GBL_EN_STAT(1)	BLK0_EN_STAT		Block 0 enable status. Logical block 0 is the set of memory-mapped control registers for the SRIO peripheral.
		0	Logical block 0 is in reset with its clock off.
		1	Logical block 0 is enabled with its clock running.
GBL_EN_STAT(0)	GBL_EN_STAT		Global enable status
		0	The peripheral is in reset with all its clocks off.
		1	The peripheral is enabled with all its clocks running.

The 18 block-specific registers are represented by Figure 33 through Figure 38. These register pairs have bits with the same functions, which are described in Table 30.

Figure 33. BLK0_EN (Address 0038h)

31	1	0
Reserved		EN
R-0		R/W-1

LEGEND: R = Read, W = Write, -*n* = Value after reset

Figure 34. BLK0_EN_STAT (Address 003Ch)

31	1	0
Reserved		EN_STAT
R-0		R-1

LEGEND: R = Read, W = Write, -*n* = Value after reset

Figure 35. BLK1_EN (Address 0040h)

31	1	0
Reserved		EN
R-0		R/W-1

LEGEND: R = Read, W = Write, -*n* = Value after reset

Figure 36. BLK1_EN_STAT (Address 0044h)

31	1	0
Reserved		EN_STAT
R-0		R-1

LEGEND: R = Read, W = Write, -*n* = Value after reset

- •
- •
- •

Figure 37. BLK8_EN (Address 0078h)

31	1	0
Reserved		EN
R-0		R/W-1

LEGEND: R = Read, W = Write, -*n* = Value after reset

Figure 38. BLK8_EN_STAT (Address 007Ch)

31	1	0
Reserved		EN_STAT
R-0		R-1

LEGEND: R = Read, W = Write, -*n* = Value after reset

Table 30. Block Enable and Block Enable Status Field Descriptions

Register(Bit)	Field	Value	Description
BLK*n*_EN(31-1)	Reserved	0	These read-only bits return 0s when read.
BLK*n*_EN(0)	EN		Block *n* enable
		0	Logical block *n* is to be reset with its clock off.
		1	Logical block *n* is to be enabled with its clock running.
BLK*n*_EN_STAT(31-1)	Reserved	0	These read-only bits return 0s when read.
BLK*n*_EN_STAT(0)	EN_STAT		Block *n* enable status
		0	Logical block *n* is reset with its clock off.
		1	Logical block *n* is enabled with its clock running.

2.3.10.3 Software Shutdown Details

Power consumption is minimized for all logical blocks that are in shutdown. In addition to simply asserting the appropriate reset signal to each logical block within the peripheral, clocks are gated off to the corresponding logical block as well. Clocks are allowed to run for 32 clock cycles, which is necessary to fully reset each logical block. When the appropriate logical block is fully reset, the clock input to that subblock is gated off. When software asserts GBL_EN/BLKn_EN to release the logical block from reset, the clocks are un-gated and the GBL_EN_STAT/BLKn_EN_STAT bit(s) indicate a value of 1b.

NOTE: The BLK_EN bits allow you to shut down and gate clocks to unused portions of the logic, while other parts of the peripheral continue to operate. When shutting down an individual block, if TXU and RXU queues are not torn down correctly, the DMA bus could hang. For example, setting BLK3_EN = 0 (disabling the TXU) before a teardown of the queue could cause any outstanding DMA request returned to the peripheral for the TXU to hang the bus.

When using the GBL_EN to shutdown/reset the entire peripheral, it is important to first stop all master-initiated commands on the DMA bus interface. For example, if the GBL_EN is asserted in the middle of a DMA transfer from the peripheral, this could hang the bus. The procedure to follow is:

1. Stop all RapidIO source transactions, including LSU and TXU operations. The four LSU blocks should indicate a BSY status of 0b (offsets 0418h, 0438h, 0458h, 0478h). If an EDMA channel is used for driving the LSU, it must be stopped to prevent new/additional transfers. This procedure is outside the scope of this specification. Teardown of the TXU queues is accomplished by writing 0000FFFFh to RIO_TX_QUEUE_TEAR_DOWN (offset 0700h). Hardware will then tear down the queues and clear these bits automatically when the teardown is complete.
2. Stop all RapidIO message receive, RXU, operations. Teardown of the RXU queues is accomplished by writing 0000FFFFh to RIO_RX_QUEUE_TEAR_DOWN (offset 0740h). Hardware will then tear down the queues and clear these bits automatically when complete.
3. Once teardown is complete, clear the PEREN bit of the RIO_PCR (offset 0004h) to stop all new logical layer transactions.
4. Wait 1 second to finish any current DMA transfer.
5. De-assert GBL_EN (offset 0030h).

2.3.11 Emulation

Expected behavior during emulation halt is controlled within the peripheral by the SOFT and FREE bits of the peripheral control register (PCR). These bits are shown in Figure 39 and described in Table 31.

Figure 39. Peripheral Control Register (PCR) - Address Offset 0004h

31	16
Reserved	
R-0	

15	3	2	1	0
Reserved		PEREN	SOFT	FREE
R-0		R/W-0	R/W-0	R/W-1

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 31. Peripheral Control Register (PCR) Field Descriptions

Bit	Field	Value	Description
31-3	Reserved	0	These read-only bits return 0s when read.
2	PEREN		Peripheral enable. Controls the flow of data in the logical layer of the peripheral. As an initiator, it will prevent TX transaction generation; as a target, it will disable incoming requests. This should be the last enable bit to toggle when bringing the device out of reset to begin normal operation.
		0	Data flow control is disabled.
		1	Data flow control is enabled.
1	SOFT		Soft stop. This bit and the FREE bit determine how the SRIO peripheral behaves during emulation halts.
		0	Hard stop. All status registers are frozen in default state. (This mode is not supported on the SRIO peripheral.)
		1	Soft stop
0	FREE		Free run
		0	The SOFT bit takes effect.
		1	Free run. Peripheral ignores the emulation suspend signal and functions normally.

Free Run Mode: (default mode) Peripheral does not respond to an emulation suspend assertion. The peripheral functions normally, irrespective of the CPU emulation state.

Soft Stop Mode: The peripheral gracefully halts operations. The peripheral halts operation at a point that makes sense both to the internal DMA/data access operation and to the pin interface as described below, after finishing packet reception or transmission in progress:

- **DMA bus DMA master:** DMA bus requests in progress are allowed to complete (DMA bus has no means to throttle command in progress from the master). DMA bus requests that correspond to the same network packet are allowed to complete. No new DMA bus requests will be generated on the next new packet.
- **Configuration bus MMR interface:** All memory-mapped register (MMR) configuration bus requests are serviced as normal.
- **Events/interrupts:** New events/interrupts are not generated to the CPU for newly arriving packets. Current transactions are allowed to finish and may cause an interrupt upon completion.
- **Slave pin interface:** The pin interface functions as normal. If buffering is available in the peripheral, the peripheral services externally generated requests as long as possible. When the internal buffers are consumed, the peripheral will retry incoming network packets in the physical layer.
- **Master pin interface**: No new master requests are generated. Master requests in progress are allowed to complete, including all packets located in the physical layer transmit buffers.

Hard Stop Mode: The peripheral halts immediately. This mode is not supported in the peripheral.

2.3.12 TX Buffers, Credit, and Packet Reordering

Packets to be transmitted by the SRIO peripheral travel to logical layer buffers. The packets are then moved from the logical layer buffers to physical layer buffers. From the physical layer buffers, the packets are transmitted through a port to a connected device.

2.3.12.1 Multiple Ports With 1x Operation

With multiple ports in 1x mode, logical layer buffers are grouped per port and contain all priorities. Each group is 8 buffers deep. A counter is maintained for each port to track available buffer credit across the UDI. The count is initialized to 8 credits per port. The count is decremented each time a packet is sent across the UDI for a port. Each port buffer group has a buffer release signal which indicates the release of a packet from the logical layer buffer to the port's physical buffer, thus indicating the freeing up of space in the port's logical buffer.

Thresholds are used to govern outbound credit when requested by the protocol units (MAU, RXU, TXU, and the LSUs). These thresholds are programmable in the peripheral settings control register (PER_SET_CNTL at address offset 0020h).

The physical layer buffer tries to process all packets in the order they were sent across the UDI. However, it is also governed by a re-ordering algorithm to decide which packets may be sent to the physical layer buffer depending on credit availability there.

The physical layer buffers act like a FIFO unless there is a retry of a packet from the connected device, in which case a re-ordering algorithm is used. The algorithm searches backward through the buffer group for the first packet with the highest priority. If there are no higher priority packets in the queue, the current packet is sent again. As an example of the re-ordering algorithm, suppose a physical layer buffer group contains packets with the following priorities:

0 0 1 2 3 3 1 0

where the leftmost 0 represents the packet that was the first in, or the head of the queue. If this packet is retried, the next packet to be sent is the earliest packet with priority 3 (the lefthand 3). If that packet is sent successfully, the physical layer attempts to send the original retried packet again; otherwise, the physical layer repeats the re-ordering algorithm.

2.3.12.2 Single Port With 1x or 4x Operation

In the case when only one port is used, logical layer buffers are grouped per priority. Each priority is 8 buffers deep. A counter is maintained for each priority to track available buffer credit across the UDI. The count is initialized to 8 credits per port. The count is decremented each time a packet is sent across the UDI for a port. Each port buffer group has a buffer release signal which indicates the release of a packet from the logical layer buffer to the port's physical buffer, thus indicating the freeing up of space in the port's logical buffer.

A priority arbiter empties the logical layer buffer with the highest priority available first. For example, it empties all available priority 3 buffers before priority 2, 1, or 0.

The physical layer buffers act like a FIFO unless there is a retry of a packet from the connected device, in which case a re-ordering algorithm is used. The algorithm searches backward through the buffer group for the first packet with the highest priority. If there are no higher priority packets in the queue, the current packet is sent again. As an example of the re-ordering algorithm, suppose a physical layer buffer group contains packets with the following priorities:

0 0 1 2 3 3 1 0

where the leftmost 0 represents the packet that was the first in, or head of the queue. If this packet is retried, the next packet to be sent is the earliest packet with priority 3 (the lefthand 3). If that packet is sent successfully, the physical layer attempts to send the original retried packet again; otherwise, the physical layer repeats the re-ordering algorithm.

2.3.12.3 Unavailable Outbound Credit

At any time, if one of the credit counters reaches 0, no more buffer credit is available. The following describes how the protocol units deal with this case.

MAU or RXU. In the case of the MAU or the RXU, all outbound packets are response packets. As a result, the MAU or RXU is free to promote a packet's priority level until priority 3 is reached. If priority 3 cannot warrant a credit, the MAU or RXU keeps retrying on priority 3 until credit is available. The assumption is that if all priority levels become backed up, the physical layer re-ordering mechanism will be implemented to send out the highest priority packets first.

LSUs. For single-packet transfers, if the transfer is unsuccessful after 256 times of credit request, a completion code of 111b is indicated in the LSU status register (LSUn_REG6). After reading this status, software must determine whether to try again, increase the priority, or try a different control flow.

For transfers (with up to 4K-byte payloads) requiring multiple packets, if the transfer is unsuccessful after 256 times of credit request for the first packet, a completion code of 111b is indicated in LSUn_REG6. After the first packet is successfully completed, subsequent packets are given more retry attempts. The LSU makes up to 64K attempts to gain outbound credit for the subsequent packets. If the LSU is unsuccessful after the 64K attempts, a completion code of 111b is indicated in LSUn_REG6.

TXU. The TXU cannot change state to handle inbound responses while it is requesting outbound credit. To avoid deadlock situations, the TXU tries for outbound credit in the following manner.

For single-segment messages, if the transfer is unsuccessful after 256 times of credit request, the TXU moves to the next queue in the round-robin loop of TX buffer descriptor queues. The TXU tries to send the unbent message again the next time the round-robin scheduler returns to the given queue.

For multi-segment messages, if the transfer is unsuccessful after 256 times of credit request for the first segment, the TXU moves to the next queue in the round-robin loop. The TXU tries to send the unsent message again the next time around the loop. After the first segment is granted outbound credit and is sent to the physical layer for transmission, all subsequent segments are given 64K attempts to gain outbound credit. If the TXU is unsuccessful after the 64K attempts, a completion code of 111b is written to the buffer descriptor, and the message is cancelled with no attempt to resend.

2.3.13 Initialization Example

2.3.13.1 Enabling the SRIO Peripheral

When the device is powered on, the SRIO peripheral is in a disabled state. Before any SRIO specific initialization can take place, the peripheral needs to be enabled; otherwise, its registers cannot be written, and the reads will all return a value of zero.

```
/* Glb enable srio */
SRIO_REGS->GBL_EN = 0x00000001 ;
SRIO_REGS->BLK0_EN = 0x00000001 ;  //MMR_EN
SRIO_REGS->BLK5_EN = 0x00000001 ;  //PORT0_EN
SRIO_REGS->BLK1_EN = 0x00000001 ;  //LSU_EN
SRIO_REGS->BLK2_EN = 0x00000001 ;  //MAU_EN
SRIO_REGS->BLK3_EN = 0x00000001 ;  //TXU_EN
SRIO_REGS->BLK4_EN = 0x00000001 ;  //RXU_EN
SRIO_REGS->BLK6_EN = 0x00000001 ;  //PORT1_EN
SRIO_REGS->BLK7_EN = 0x00000001 ;  //PORT2_EN
SRIO_REGS->BLK8_EN = 0x00000001 ;  //PORT3_EN
```

2.3.13.2 PLL, Ports, and Data Rate Initializations

To change from 1 lane to 4 lanes there are 2 registers that need to be programmed (see Table 32).

Table 32. Port Mode Register Settings

SP_IP_MODE (offset 0x12004) Bits 31-30 [1]	PER_SET_CNTL (offset 0x0020) Bit 8	Port Mode
0x00	0x0	1x/4p[2]
0x01	0x1	1x/1x[3]

(1) Bits 31-30 are read only. To enable writing to read-only registers, change the PER_SET_CNTL register (offset 0020h) bit 24 to 0.
(2) UDI buffers are priority based.
(3) UDI buffers are port based.

For example, Enable PLL, 333MHz, 1x/4p (srio4p1x_mode = 1), x20, 125MHz ref. clock, 2.5 Gbps, half rate:

```
if (srio4p1x_mode){
       rdata = SRIO_REGS->PER_SET_CNTL;
       wdata = 0x0000014F; //4p1x
       mask = 0x000001FF;
       mdata = (wdata & mask) | (rdata & ~mask);
       SRIO_REGS->PER_SET_CNTL = mdata ; // enable PLL
}
else{
       wdata = 0x0000004F; // enable PLL, 1p4x
       rdata = SRIO_REGS->PER_SET_CNTL;
       mask = 0x000001FF;
       mdata = (wdata & mask) | (rdata & ~mask);
       SRIO_REGS->PER_SET_CNTL = mdata ; // enable PLL, 1p1x/4x
}

//INIT_MAC0
if (srio4p1x_mode){
          SRIO_REGS->SP_IP_MODE = 0x4400003F; // mltc/rst/pw enable, clear
}
else{
          SRIO_REGS->SP_IP_MODE = 0x0400003F; // mltc/rst/pw enable, clear
}
SRIO_REGS->SERDES_CFG0_CNTL = 0x00000013;

SRIO_REGS->SERDES_CFG1_CNTL = 0x00000000;
SRIO_REGS->SERDES_CFG2_CNTL = 0x00000000;
SRIO_REGS->SERDES_CFG3_CNTL = 0x00000000;

SRIO_REGS->SERDES_CFGRX0_CNTL = 0x00081121 ; // enable rx, half rate
SRIO_REGS->SERDES_CFGRX1_CNTL = 0x00081121 ; // enable rx, half rate
SRIO_REGS->SERDES_CFGRX2_CNTL = 0x00081121 ; // enable rx, half rate
SRIO_REGS->SERDES_CFGRX3_CNTL = 0x00081121 ; // enable rx, half rate
SRIO_REGS->SERDES_CFGTX0_CNTL = 0x00010821 ; // enable tx, half rate
SRIO_REGS->SERDES_CFGTX1_CNTL = 0x00010821 ; // enable tx, half rate
SRIO_REGS->SERDES_CFGTX2_CNTL = 0x00010821 ; // enable tx, half rate
SRIO_REGS->SERDES_CFGTX3_CNTL = 0x00010821 ; // enable tx, half rate
```

For the number of ports, see Table 1.

2.3.13.3 Peripheral Initializations

Set Device ID Registers

```
  rdata = SRIO_REGS->DEVICEID_REG1;
  wdata = 0x00ABBEEF;
  mask = 0x00FFFFFF;
  mdata = (wdata & mask) | (rdata & ~mask);
  SRIO_REGS->DEVICEID_REG1 = mdata ; // id-16b=BEEF, id-08b=AB

  rdata = SRIO_REGS->DEVICEID_REG2;
  wdata = 0x00ABBEEF;
  mask = 0x00FFFFFF;
  mdata = (wdata & mask) | (rdata & ~mask);
```

```
  SRIO_REGS->DEVICEID_REG2 = mdata ; // id-16b=BEEF, id-08b=AB

  rdata = SRIO_REGS->PER_SET_CNTL;
  data = 0x00000000;
  mask = 0x01000000;
  mdata = (wdata & mask) | (rdata & ~mask);
  SRIO_REGS->PER_SET_CNTL = mdata; // bootcmpl=0

SRIO_REGS->DEV_ID            = 0xBEEF0030 ;   // id=BEEF, ti=0x0030
SRIO_REGS->DEV_INFO          = 0x00000000 ;   // 0
SRIO_REGS->ASBLY_ID          = 0x00000030 ;   // ti=0x0030
SRIO_REGS->ASBLY_INFO        = 0x00000000;    // 0x0000, next ext=0x0100
SRIO_REGS->PE_FEAT           = 0x20000019 ;   // proc, bu ext, 16-bit ID, 34-bit addr
SRIO_REGS->SRC_OP            = 0x0000FDF4;    // all
SRIO_REGS->DEST_OP           = 0x0000FC04;    // all except atomic
SRIO_REGS->PE_LL_CTL         = 0x00000001;    // 34-bit addr
SRIO_REGS->LCL_CFG_HBAR      = 0x00000000 ;   // 0
SRIO_REGS->LCL_CFG_BAR       = 0x00000000;    // 0
SRIO_REGS->BASE_ID           = 0x00ABBEEF;    // 16b-id=BEEF, 08b-id=AB
SRIO_REGS->HOST_BASE_ID_LOCK = 0x0000BEEF;    // id=BEEF, lock
SRIO_REGS->COMP_TAG          = 0x00000000;    // not touched
SRIO_REGS->SP_IP_DISCOVERY_TIMER = 0x90000000;// 0, short cycles for sim

SRIO_REGS->IP_PRESCAL = 0x00000021; // srv_clk prescalar=0x21 (333MHz)
SRIO_REGS->SP0_SILENCE_TIMER = 0x20000000;
SRIO_REGS->SP1_SILENCE_TIMER = 0x20000000;
SRIO_REGS->SP2_SILENCE_TIMER = 0x20000000;
SRIO_REGS->SP3_SILENCE_TIMER = 0x20000000;

  rdata = SRIO_REGS->PER_SET_CNTL;
  wdata = 0x01000000;
  mask = 0x01000000;
  mdata = (wdata & mask) | (rdata & ~mask);
  SRIO_REGS->PER_SET_CNTL = mdata; // bootcmpl=1

RIO_REGS->SP_LT_CTL          = 0xFFFFFF00;  // long
SRIO_REGS->SP_RT_CTL         = 0xFFFFFF00;  // long
SRIO_REGS->SP_GEN_CTL        = 0x40000000;  // agent, master, undiscovered
SRIO_REGS->SP0_CTL           = 0x00600000;  // enable i/o
SRIO_REGS->SP1_CTL           = 0x00600000;  // enable i/o
SRIO_REGS->SP2_CTL           = 0x00600000;  // enable i/o
SRIO_REGS->SP3_CTL           = 0x00600000;  // enable i/o

SRIO_REGS->ERR_DET           = 0x00000000 ;  // clear
SRIO_REGS->ERR_EN            = 0x00000000 ;  // disable
SRIO_REGS->H_ADDR_CAPT       = 0x00000000 ;  // clear
SRIO_REGS->ADDR_CAPT         = 0x00000000 ;  // clear
SRIO_REGS->ID_CAPT           = 0x00000000 ;  // clear
SRIO_REGS->CTRL_CAPT         = 0x00000000 ;  // clear


SRIO_REGS->SP_IP_PW_IN_CAPT0 = 0x00000000 ;  // clear
SRIO_REGS->SP_IP_PW_IN_CAPT1 = 0x00000000 ;  // clear
SRIO_REGS->SP_IP_PW_IN_CAPT2 = 0x00000000 ;  // clear
SRIO_REGS->SP_IP_PW_IN_CAPT3 = 0x00000000 ;  // clear

//INIT_WAIT wait for lane initialization
```

For the number of ports, see Table 1.

Read register to check portx(1-4) OK bit (4 port example)

```
// polling SRIO_MAC's port_ok bit
rdata = SRIO_REGS->P0_ERR_STAT ;
while ((rdata & 0x00000002) != 0x00000002)
{
         rdata = SRIO_REGS->P0_ERR_STAT ;
}
if (srio4p1x_mode){
         rdata = SRIO_REGS->P1_ERR_STAT;
         while ((rdata & 0x00000002) != 0x00000002)
{
               rdata = SRIO_REGS->P1_ERR_STAT;
}
```

```
        rdata = SRIO_REGS->P2_ERR_STAT;
        while ((rdata & 0x00000002) != 0x00000002)
{
              rdata = SRIO_REGS->P2_ERR_STAT;
}
        rdata = SRIO_REGS->P3_ERR_STAT;
        while ((rdata & 0x00000002) != 0x00000002)
{
              rdata = SRIO_REGS->P3_ERR_STAT;
}
}
```

Assert the PEREN bit to enable logical layer data flow

```
SRIO_REGS->PCR = 0x00000004; // peren
```

For an SRIO initialization example, see Section A.1.

Recover the port if it is already in the error state. For information on software-assisted error recovery, see Appendix B.

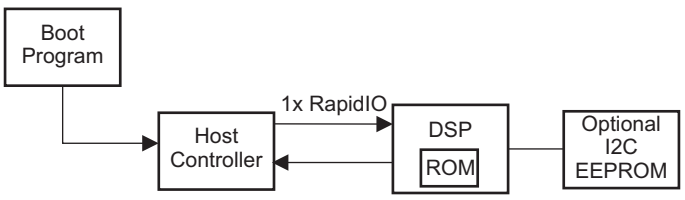
2.3.14 Bootload Capability

2.3.14.1 Configuration and Operation

Figure 40 illustrates the system components involved in bootload operation. It is assumed that an external device will initiate the bootload data transfer and master the DMA interface. Upon reset, the following sequence of events must occur:

1. DSP is placed in SRIO boot mode by HW mode pins.
2. Host takes DSP out of reset ($\overline{\text{POR}}$ or $\overline{\text{RST}}$). The peripheral's state machines and registers are reset.
3. Internal boot-strap ROM configures device registers, including SERDES, and DMA. DSP executes internal ROM code to initialize SRIO.
   - Choice of 4 pin selectable configurations
   - Optionally, I2C boot can be used to configure SRIO
4. DSP executes idle instruction.
5. RapidIO ports send Idle control symbols to train PHYs.
6. Host enabled to explore system with RapidIO Maintenance packets.
7. Host identifies, enumerates and initializes the RapidIO device.
8. Host controller configures DSP peripherals through maintenance packets.
   - SRIO Device IDs are set for DSPs (either by pin strapping or by host manipulation)
9. Boot Code sent from host controller to DSP L2 memory base address via NWRITE.
10. DSP CPU is awakened by an interrupt such as a RapidIO DOORBELL packet.
11. Boot Code is executed and normal operation follows.

Figure 40. Bootload Operation



2.3.14.2 Bootload Data Movement

The system host is responsible for writing the bootload data into the DSP's L2 memory. As such, bootload is only supported using the direct I/O model, and not the message passing model. Bootload data must be sent in packets with explicit L2 memory addresses indicating proper destination within the DSP. As part of the peripheral's configuration, it should be set up to transfer the desired bootload program to the DSP's memory through normal DMA bus commands.

2.3.14.3 Device Wakeup

Upon completion of the bootload data transfer, the system host issues a DOORBELL interrupt to the DSP. The RapidIO peripheral processes this interrupt in a manner similar to that described in Section 4, monitoring the DMA bus write-with-response commands to ensure that the data has been completely transferred through the DMA. This interrupt wakes up the CPUs by pulling them out of their reset state.

The 16-bit data field of the DOORBELL packet should be configured to interrupt Core 0 by setting a corresponding ICSR bit as described in Figure 43.

2.3.15 RX Multicast and Multiple DESTID Support

The RapidIO peripheral supports RX multicast, as well as, unicast operations. To support multicast, an endpoint must either be able to accept discrete multiple DESTIDs from incoming packets or, alternatively, accept all DESTIDs without performing a check on the incoming packet. In the latter case, the system relies on the fabric to deliver the appropriate packets to the endpoint (correct switch routing tables). The RapidIO peripheral supports both of these approaches by utilizing three configuration bits (described in Table 33) to control DESTID checking. Table 33 shows the various configurations. The device's main BASE_ID (0x1060) is automatically copied into the DEVICEID_REG1 (0x0080) register by the hardware, and does not require a separate write by the software. The DEVICEID_REG1 register should be ignored since writes have no effect and reads do not return the correct value.

Table 33. DESTID Checking Control Modes

Mode	Operation	LOG_TGT_ID_DIS (PER_SET_CNTL, Bit 27, 0x0020)	SRC_TGT_ID_DIS (SP_IP_MODE Bit 26, 0x12004)	F8_TGT_ID_DIS (SP_IP_MODE Bit 24, 0x12004)
A	• Host performs maintenance enumeration, discovery, and assignment using only DESTID = BASE_ID (0x1060) value. • Unicast requires DESTID = BASE_ID (0x1060). • No multicast. • No packet forwarding.	X	0	0
B	• Host performs maintenance enumeration, discovery, and assignment using any DESTID. – Allows hard coding (pin strapping) of BASE_ID, then maintenance is read back by the host. • Unicast requires DESTID = BASE_ID (0x1060). • No multicast. • No packet forwarding.	X	0	1
C	• Host performs maintenance enumeration, discovery, and assignment using only DESTID = BASE_ID (0x1060) value. • Unicast requires DESTID = BASE_ID (0x1060). • Supports three local multicast groups, DESTID = 0x0084, 0x0088, or 0x008C. • Supports packet forwarding for all packet types.	0	1	0

Table 33. DESTID Checking Control Modes (continued)

Mode	Operation	LOG_TGT_ID_DIS (PER_SET_CNTL, Bit 27, 0x0020)	SRC_TGT_ID_DIS (SP_IP_MODE Bit 26, 0x12004)	F8_TGT_ID_DIS (SP_IP_MODE Bit 24, 0x12004)
D	• Host performs maintenance enumeration, discovery, and assignment using any DESTID. – Allows hard coding (pin strapping) of BASE_ID, then maintenance is read back by the host. • Unicast requires DESTID = BASE_ID (0x1060). • Supports three local multicast groups, DESTID = 0x0084, 0x0088, or 0x008C. • No packet forwarding of FTYPE 8 maintenance packets.	0	1	1
E	• Host performs maintenance enumeration, discovery, and assignment using only DESTID = BASE_ID (0x1060) value. • Supports infinite multicast/unicast groups. • No packet forwarding.	1	1	0
F	• Host performs maintenance enumeration, discovery, and assignment using any DESTID. – Allows hard coding (pin strapping) of BASE_ID, then maintenance is read back by the host. • Supports infinite multicast/unicast groups. • No packet forwarding.	1	1	1

Modes A and C are our legacy modes. Mode B is a superset of Mode A and Mode F is a superset of Mode E. The most common modes are Mode C, Mode D, and Mode F.

- **F8_TGT_ID_DIS:** The physical layer DESTID checking, for FTYPE 8 packets, is controlled with the F8_TGT_ID_DIS bit. This bit only disables maintenance packet checking at the physical layer. That is, if this bit is active, then, regardless of the FTYPE 8 packet's DESTID, the physical layer accepts and handles the maintenance request. If this bit is inactive, then the non-matching FTYPE 8 packets are forwarded to the logical layer, where they can be packet forwarded or destroyed, accordingly.
- **SRC_TGT_ID_DIS:** This is the MAC layer disable bit. This bit is the legacy control bit for disabling DESTID checking on all other packet types. If this bit is active, then all packets regardless of DESTID will be forwarded to the logical layer (except FTYPE 8 packets that are intercepted/handled by the physical layer). If this bit is inactive, then the non-matching packets are destroyed before reaching the logical layer.
- **LOG_TGT_ID_DIS:** This is the logical layer disable bit. This bit disables DESTID checking in the logical layer. If this bit is active, then all packets, regardless of the DESTID, are forwarded to the appropriate functional block of the peripheral. If this bit is inactive, then all non-matching packets are destroyed. This bit allows unlimited multicast/unicast IDs for all supported packet types, not just posted operations. This means that packet-forwarding features cannot be used. Due to the current RTL implementation, there are some considerations when this bit is active:
  - **MAU and RXU:** There are no side affects for request packets that are targeted for MAU and RXU. This means that if the LOG_TGT_ID_DIS bit is active, all packets, regardless of DESTID, are accepted and forwarded to the appropriate functional block.
  - **LSU:** Due to the current LSU implementation (LSU scoreboards the DESTID for response packets), those direct I/O response packets that are not targeted to our endpoint device are not accepted.
  - **TXU:** Due to the current TXU implementation (TXU does NOT scoreboard the DESTID for response packets), those message response packets that are not targeted to our endpoint device are accepted.
  - **CCU:** Due to the current CCU implementation (CCU does NOT verify the DESTID for request packets), those congestion control packets that are not targeted to our endpoint device are accepted.

2.3.15.1 Discrete Multicast ID Support

In operation modes C and D (see Table 33), the allowable multicast DEVICEIDs are stored in the DEVICEID_REG2, DEVICEID_REG3, DEVICEID_REG4 (0x0084, 0x0088, 0x008C) registers.

When a packet is received, the packet's TT field and DESTID are checked against the DEVICEID (0x0080) and the MULTICASTIDs (see Table 34). If no match is found and packet forwarding is disabled, the packet is destroyed and not forwarded to the logical layer. If a match to one of these IDs is found, it is forwarded to the logical layer. Since multicast operations are defined to be operations that do not require responses, they are limited to NWRITE and SWRITE operations and forwarded to the MAU. The multicast mode is disabled by writing the main BASE_ID (0x1060) into the multicast ID registers.

Table 34. Multicast DeviceID Operation

Local DeviceID Register Offset	Multicast DeviceID Register Offset	Multicast DeviceID Register Offset	Multicast DeviceID Register Offset
0080h	0084h	0088h	008Ch

Multicast transactions are I/O packets that specify a destination memory address within the header. For this reason, multicast support is limited to groups containing devices with the same memory map or other devices that can perform address translation. It is the responsibility of the system designer to pre-determine valid multicast address ranges.

2.3.15.2 Unlimited Multicast and DESTID Support

Unlimited multicast and unicast DESTIDs can be supported in modes E and F (see Table 33). In these modes, all supported packet types are accepted and sent to the appropriate peripheral functional blocks regardless of incoming DestID. This means that non-posted transaction types such as NWRITE_R and messages requests can be sent to DESTIDs other than the BASE_ID value. When a response packet is sent out, the DESTID and SOURCEID of the incoming request packet are swapped. Packet forwarding can not be used in these modes.

RapidIO now requires that devices support a promiscuous mode of operation for device discovery and enumeration. This means that devices should be able to respond to all incoming DESTIDs for FTYPE 8 maintenance after boot. A mode F supports this requirement and is more likely to be used.

2.3.15.3 Daisy Chain Operation and Packet Forwarding

Some applications may require daisy chaining of devices together versus using a switch fabric. Typically, these applications are low-cost implementations. Daisy chains have variable system latency depending on device position within the chain. Daisy chain implementations also have reduced bandwidth capabilities, since the link bandwidth does not change, the bandwidth allocated to each device in the chain is limited (sum of devices' individual bandwidth needs cannot exceed link bandwidth). To support daisy chain or ring topologies, the peripheral features a hardware packet forwarding function. This feature eliminates the need for software to be involved in routing a packet to the next device in the chain. The basic idea behind the hardware packet forwarding logic is to provide an input port to output port path such that the packets never leave the peripheral (no DMA transfer). Mode C (see Table 33) supports hardware packet forwarding. A simple check of an incoming packet's DESTID versus the device's DEVICEID and MULTICASTIDs is done to determine if the packet should be forwarded. If the packet's DESTID matches DEVICEID, the packet is accepted and processed by the device. If the packet's DESTID matches any MULTICASTID, the packet is accepted by the device and forwarded based on the rules outlined below. If the packet's DESTID does not match either, the packet is destroyed or forwarded, depending on the hardware packet forwarding setup.

Additionally, it is beneficial to be able to only forward a packet if the destination ID is one of the devices in the chain/ring. Otherwise, a rogue packet may be forwarded endlessly using up valuable bandwidth. The hardware packet forwarding uses a 4 entry mapping table shown in Figure 72 and Figure 73. These mapping entries allow programmable selection of output port based on the in-coming packets DESTID range.

The algorithm is as follows:

- The packet's DESTID is compared against the mapping entries based on the packet's TT field. If TT = 2'b00, then the 8-bit version of ID boundaries is used. If TT = 2'b01, then the 16-bit version of ID boundaries is used.
- If any packet's DESTID = DEVICEID, it is handled normally and not forwarded. This is the highest priority check.
- If any packet's DESTID = MULTICASTID, it is accepted. If the packet's DESTID also falls into one of the ranges specified in the hardware packet forwarding mapping entries, the packet is also forwarded to the outbound port programmed.
- If the packet's DESTID ≠ DEVICEID and DESTID ≠ MULTICASTID, but falls into one of the ranges specified in the hardware packet forwarding mapping entries, the packet is forwarded.
- If multiple table entry ranges are matched, then table entry 0 has highest priority, followed by table entry 1, and so on.
- If the packet's DESTID ≠ DEVICEID and DESTID ≠ MULTICASTID, and does not fall into one of the ranges specified in the hardware packet forwarding mapping entries, the packet is destroyed.
- Hardware packet forwarding can be disabled by assigning all the table entry upper and lower DEVICEID boundaries equal to the local BASEID value.

Table 35 details the behavior of the hardware packet forwarding and multicast support, with respect to these specific RapidIO packet types:

1. nread (Ftype=2, Ttype=4'b0100)
2. atomic inc (Ftype=2, Ttype=4'b1100)
3. atomic dec (Ftype=2, Ttype=4'b1101)
4. atomic set (Ftype=2, Ttype=4'b1110)
5. atomic clr (Ftype=2, Ttype=4'b1111)
6. nwrite (Ftype=5, Ttype=4'b0100)
7. nwrite_r (Ftype=5, Ttype=4'b0101)
8. atomic t&s (Ftype=5, Ttype=4'b1110)
9. swrite (Ftype=6)
10. congestion (Ftype=7)
11. maint read (Ftype=8, Ttype=4'b0000)
12. maint write (Ftype=8, Ttype=4'b0001)
13. maint rd resp (Ftype=8, Ttype=4'b0010)
14. maint wr resp (Ftype=8, Ttype=4'b0011)
15. maint port wr (Ftype=8, Ttype=4'b0100)
16. doorbell (Ftype=10)
17. message (Ftype=11)
18. resp w/o payload (Ftype=13, Ttype=4'b0000)
19. message resp (Ftype=13, Ttype=4'b0001)
20. resp w/ payload (Ftype=13, Ttype=4'b1000)

Table 35. Hardware Packet Forwarding and Multicast Operation

Incoming Packet DESTID Matches Local DEVICEID	Incoming Packet DESTID Matches MULTICASTID	Incoming Packet DESTID Matches One of the Packet Forwarding Table Entry Ranges	Behavior
Yes	No	No	All packet types 1 to 20 will be handled accordingly by the local corresponding logical layer protocol unit (LSU, MAU, TXU, RXU, and CCU). Physical layer handles 11, 12, and 15.
Yes	No	Yes	All packet types 1 to 20 will be handled accordingly by the local corresponding logical layer protocol unit (LSU, MAU, TXU, RXU, and CCU). Physical layer handles 11, 12, and 15.

Table 35. Hardware Packet Forwarding and Multicast Operation (continued)

Incoming Packet DESTID Matches Local DEVICEID	Incoming Packet DESTID Matches MULTICASTID	Incoming Packet DESTID Matches One of the Packet Forwarding Table Entry Ranges	Behavior
Yes	Yes	No	All packet types 1 to 20 will be handled accordingly by the local corresponding logical layer protocol unit (LSU, MAU, TXU, RXU, and CCU). Physical layer handles 11, 12, and 15.
Yes	Yes	Yes	All packet types 1 to 20 will be handled accordingly by the local corresponding logical layer protocol unit (LSU, MAU, TXU, RXU, and CCU). Physical layer handles 11, 12, and 15.
No	Yes	No	All packet types 1 to 20 are forwarded to the MAU. The MAU only supports 1, 6, 7, 9, and 16. All other types are not supported and may cause undefined behavior, including the generation of an ERROR response.
No	Yes	Yes	All packet types 1 to 20 are forwarded to the MAU. The MAU supports types 1, 6, 7, 9, and 16. All other types are not supported and may cause undefined behavior, including the generation of an ERROR response. The MAU will forward support packet types by copying inbound to outbound. The port ID, specified in the packet forwarding table entry, will be used for the outbound forwarded packet, as well as, any outbound response packets.(1)
No	No	Yes	All packet types 1 to 20 are forwarded to the MAU. The MAU copies the packet from inbound to outbound. The outbound port ID is specified by the packet forwarding table entry.
No	No	No	All packet types 1 to 20 will be destroyed internally.

(1) Having response packets transmitted from a different port than the incoming request packet is NOT standard RapidIO practice. If this is prohibited due to system topology, only packet types 6 and 9 should be used.

Since the packet forwarding is done at the logical layer and not the physical layer, CRCs are regenerated for each forwarded packet. For more information on the programmable packet forwarding entries, see Figure 72 and Figure 73.

3 Logical/Transport Error Handling and Logging

Error management registers allow detection and logging of logical/transport layer errors. The detectable errors are captured in the logical layer error detect CSR (see Figure 41). Table 36 names the functional block(s) involved for each detectable error condition, and includes brief descriptions of the errors captured.

Figure 41. Logical/Transport Layer Error Detect CSR (ERR_DET)

31	30	29	28	27	26	25	24
IO_ERR_RSPNS	MSG_ERR_RSPNS	Reserved	ERR_MSG_FORMAT	ILL_TRANS_DECODE	Reserved	MSG_REQ_TIMEOUT	PKT_RSPNS_TIMEOUT
R/W-0	R/W-0	R-0	R/W-0	R/W-0	R-0	R/W-0	R/W-0

23	22	21	16
UNSOLICITED_RSPNS	UNSUPPORTED_TRANS	Reserved	
R/W-0	R/W-0	R-0	

15	8
Reserved	
R-0	

7	6	5	0
RX_CPPI_SECURITY	RX_IO_DMA_ACCESS	Reserved	
R/W-0	R/W-0	R-0	

LEGEND: R = Read; W = Write; -*n* = Value after reset

Table 36. Logical/Transport Layer Error Detect CSR (ERR_DET) Field Descriptions

Bit	Field	Value	Description
31	IO_ERR_RSPNS		IO error response (endpoint device only)
		0	An LSU did not receive an ERROR response to an IO logical layer request.
		1	An LSU received an ERROR response to an IO logical layer request. To clear this bit, write 0 to it.
30	MSG_ERR_RSPNS		Message error response (endpoint device only)
		0	The TXU did not receive an ERROR response to a message logical layer request.
		1	The TXU received an ERROR response to a message logical layer request. To clear this bit, write 0 to it.
29	Reserved	0	This read-only bit returns 0 when read.
28	ERR_MSG_FORMAT		Error in message format (endpoint device only)
		0	The RXU did not receive a message data payload with an invalid size or segment.
		1	The RXU received a message data payload with an invalid size or segment. To clear this bit, write 0 to it.
27	ILL_TRANS_DECODE		Illegal transaction decode (switch or endpoint device)
			For an LSU or the TXU:
		0	The LSU/TXU did not receive illegal fields in the response packet for an IO/message transaction.
		1	The LSU/TXU received illegal fields in the response packet for an IO/message transaction. To clear this bit, write 0 to it.
			For the MAU or the RXU:
		0	The MAU/RXU did not receive illegal fields in the request packet for an IO/message transaction.
		1	The MAU/RXU received illegal fields in the request packet for an IO/message transaction. To clear this bit, write 0 to it.
26	Reserved	0	This read-only bit returns 0 when read.

Table 36. Logical/Transport Layer Error Detect CSR (ERR_DET) Field Descriptions (continued)

Bit	Field	Value	Description
25	MSG_REQ_TIMEOUT		Message request timeout (endpoint device only)
		0	A timeout has not been detected by RXU.
		1	A timeout has been detected by the RXU. A required message request has not been received by the RXU within the specified time-out interval. To clear this bit, write 0 to it.
24	PKT_RSPNS_TIMEOUT		Packet response timeout (endpoint device only)
		0	A timeout has not been detected by an LSU or the TXU.
		1	A timeout has been detected by an LSU or the TXU. A required response has not been received by the LSU/TXU within the specified timeout interval. To clear this bit, write 0 to it.
23	UNSOLICITED_RSPNS		Unsolicited response (switch or endpoint device)
		0	An unsolicited response packet has not been received by an LSU or the TXU.
		1	An unsolicited response packet has been received by an LSU or the TXU. To clear this bit, write 0 to it.
22	UNSUPPORTED_TRANS		Unsupported transaction (switch or endpoint device)
		0	The MAU has not received an unsupported transaction.
		1	The MAU has received an unsupported transaction. That is, the MAU received a transaction that is not supported in the destination operations CAR. To clear this bit, write 0 to it.
21-8	Reserved	0	These read-only bits return 0 when read.
7	RX_CPPI_SECURITY		RX CPPI security error
		0	The RXU has not detected an access block.
		1	The RXU has detected an access block. That is, access to one of the RX queues was blocked. To clear this bit, write 0 to it.
6	RX_IO_DMA_ACCESS		RX IO DMA access error
		0	A DMA access to the MAU has not been blocked.
		1	A DMA access to the MAU was blocked. To clear this bit, write 0 to it.
5-0	Reserved	0	These read-only bits return 0 when read.

4 Interrupt Conditions

This section defines the CPU interrupt capabilities and requirements of the peripheral.

4.1 CPU Interrupts

The following interrupts are supported by the RIO peripheral.

- Error status: Event indicating that a run-time error was reached. The CPU should reset/resynchronize the peripheral.
- Critical error: Event indicating that a critical error state was reached. The CPU should reset the system.
- CPU servicing: Event indicating that the CPU should service the peripheral.

4.2 General Description

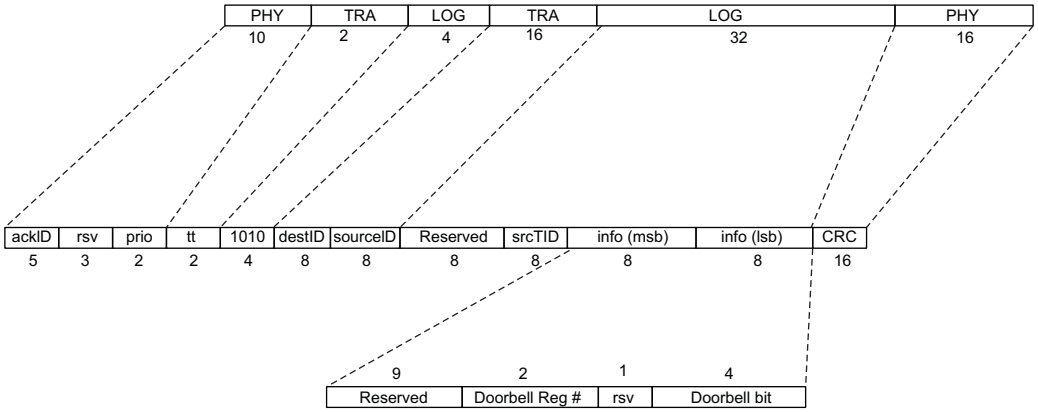
The RIO peripheral is capable of generating various types of CPU interrupts. The interrupts serve two general purposes: error indication and servicing requests.

Since RapidIO is a packet oriented interface, the peripheral must recognize and respond to inbound signals from the serial interface. There are no GPIO or external pins used to indicate an interrupt request. Thus, the interrupt requests are signaled either by an external RapidIO device through the packet protocols discussed as follows, or are generated internally by the RIO peripheral.

CPU servicing interrupts lag behind the corresponding data, which was generally transferred from an external processing element into local L2 memory. This transfer can use a messaging or direct I/O protocol. When the single or multi-packet data transfer is complete, the external PE, or the peripheral itself, must notify the local processor that the data is available for processing. To avoid erroneous data being processed by the local CPU, the data transfer must complete through the DMA before the CPU interrupt is serviced. This condition could occur since the data and interrupt queues are independent of each other, and DMA transfers can stall. To avoid this condition, all data transfers from the peripheral through the DMA use write-with-response DMA bus commands, allowing the peripheral to always be aware that outstanding transfers have completed. Interrupts are generated only after all DMA bus responses are received. Since all RapidIO packets are handled sequentially, and submitted on the same DMA priority queue, the peripheral must keep track of the number of DMA requests submitted and the number of responses received. Thus, a simple counter within the peripheral ensures that data packets have arrived in memory before submitting an interrupt.

The sending device initiates the interrupt by using the RapidIO defined DOORBELL message. The DOORBELL packet format is shown in Figure 42. The DOORBELL functionality is user-defined. This packet type is commonly used to initiate CPU interrupts. A DOORBELL packet is not associated with a particular data packet that was previously transferred, so the INFO field of the packet must be configured to reflect the DOORBELL bit to be serviced for the correct TID info to be processed.

Figure 42. RapidIO DOORBELL Packet for Interrupt Use



The DOORBELL packet's 16-bit INFO field indicates which DOORBELL register interrupt bit to set. There are four DOORBELL registers, each currently with 16 bits, allowing 64 interrupt sources or circular buffers (see Table 23 for assignment of the 16 bits of DOORBELL_INFO field). Each bit can be assigned to any core as described by the Interrupt Condition Routing Registers. Additionally, each status bit is user-defined for the application. For instance, it may be desirable to support multiple priorities with multiple TID circular buffers per core if control data uses a high priority (for example, priority = 2), while data packets are sent on priority 0 or 1. This allows the control packets to have preference in the switch fabric and arrive as quickly as possible. Since it may be required to interrupt the CPU for both data and control packet processing separately, separate circular buffers are used, and DOORBELL packets must distinguish between them for interrupt servicing. If any reserved bit in the DOORBELL info field is set, an error response is sent.

The interrupt approach to the messaging protocol is somewhat different. Since the source device is unaware of the data's physical location in the destination device, and since each messaging packet contains size and segment information, the peripheral can automatically generate the interrupt after it has successfully received all packet segments comprising the complete message. This DMA interface uses the Communications Port Programming Interface (CPPI). This interface is a link-listed approach versus a circular buffer approach. Data buffer descriptors which contain information such as start of Packet (SOP), end of packet (EOP), end of queue (EOQ), and packet length are built from the RapidIO header fields. The data buffer descriptors also contain the address of the corresponding data buffer as assigned by the receive device. The data buffer descriptors are then link-listed together as multiple packets are received. Interrupts are generated by the peripheral after all segments of the messages are received and successfully transferred through the DMA bus with the write-with-response commands. Interrupt pacing is also implemented at the peripheral level to manage the interrupt rate, as described in Section 4.7.

Error handling on the RapidIO link is handled by the peripheral, and as such, does not require the intervention of software for recovery. This includes CRC errors due to bit rate errors that may cause erroneous or invalid operations. The exception to this statement is the use of the RapidIO error management extended features. This specification monitors and tabulates the errors that occur on a per port basis. If the number of errors exceeds a pre-determined configurable amount, the peripheral should interrupt the CPU software and notify that an error condition exits. Alternatively, if a system host is used, the peripheral may issue a port-write operation to notify the system software of a bad link.

A system reset, or Critical Error interrupt, can be initialized through the RapidIO link. This procedure allows an external device to reset the local device, causing all state machine and configuration registers to reset to their original values. This is executed with the Reset-Device command described in Part VI, Section 3.4.5 of the RapidIO *Physical Layer 1x/4x LP-Serial Specification*. Four sequential Reset-Device control symbols are needed to avoid inadvertent resetting of a device.

4.3 Interrupt Condition Status and Clear Registers

Interrupt condition status and clear registers configure which CPU interrupts are to be generated and how, based on the peripheral activity. All peripheral conditions that result in a CPU interrupt are grouped so that the interrupt can be accessed in the minimum number of register reads possible.

For each of the three types of interrupts (CPU servicing, error status, and critical error), there are two sets of registers:

- Interrupt Condition Status Register (ICSR): Status register that reflects the state of each condition that can trigger the interrupt. The general description of each interrupt condition status bit (ICSx) is given in Table 37.
- Interrupt Condition Clear Register (ICCR): Command register that allows each condition to be cleared. This is typically required prior to enabling a condition, so that spurious interrupts are not generated. Table 37 shows the general description of an interrupt condition clear bit (ICCx).

These registers are accessible in the memory map of the CPU. The CPU controls the clear register. The status register is readable by the CPU to determine the peripheral condition.

Table 37. Interrupt Condition Status and Clear Bits

Field	Access	Reset Value	Value	Function
ICSx	R	0	0	Condition not present
			1	Condition present
ICCx	W	0	0	No effect
			1	Clear the condition status bit (ICSx)

4.3.1 Doorbell Interrupt Condition Status and Clear Registers

The interrupt condition status registers (ICSRs) and the interrupt condition clear registers (ICCRs) for the four doorbells are shown in Figure 43 through Figure 46. These registers are used when the SRIO peripheral receives doorbell packets. The 16 ICS bits of each interrupt condition status register (ICSR) indicate the incoming doorbell information packet. For example, the bits ICS15, ICS8, and ICS0 of DOORBELL0_ICSR correspond to Doorbell 0 information bits 15, 8, and 0. The 16 ICC bits of each interrupt condition clear register (ICCR) are used to clear the corresponding bits in the ICSR. For example, the ICC7 bit of DOORBELL2_ICCR is used to clear the ICS7 bit of DOORBELL2_ICSR.

Figure 43. Doorbell 0 Interrupt Condition Status and Clear Registers

Doorbell 0 Interrupt Condition Status Register (DOORBELL0_ICSR) (Address Offset 0200h)

31	16
Reserved	
R-0	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICS15	ICS14	ICS13	ICS12	ICS11	ICS10	ICS9	ICS8	ICS7	ICS6	ICS5	ICS4	ICS3	ICS2	ICS1	ICS0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Doorbell 0 Interrupt Condition Clear Register (DOORBELL0_ICCR) (Address Offset 0208h)

31	16
Reserved	
R-0	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICC15	ICC14	ICC13	ICC12	ICC11	ICC10	ICC9	ICC8	ICC7	ICC6	ICC5	ICC4	ICC3	ICC2	ICC1	ICC0
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: R = Read only; W = Write only; -*n* = Value after reset

Figure 44. Doorbell 1 Interrupt Condition Status and Clear Registers

Doorbell 1 Interrupt Condition Status Register (DOORBELL1_ICSR) (Address Offset 0210h)

31	16
Reserved	
R-0	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICS15	ICS14	ICS13	ICS12	ICS11	ICS10	ICS9	ICS8	ICS7	ICS6	ICS5	ICS4	ICS3	ICS2	ICS1	ICS0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Doorbell 1 Interrupt Condition Clear Register (DOORBELL1_ICCR) (Address Offset 0218h)

31	16
Reserved	
R-0	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICC15	ICC14	ICC13	ICC12	ICC11	ICC10	ICC9	ICC8	ICC7	ICC6	ICC5	ICC4	ICC3	ICC2	ICC1	ICC0
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: R = Read only; W = Write only; -*n* = Value after reset

Figure 45. Doorbell 2 Interrupt Condition Status and Clear Registers

Doorbell 2 Interrupt Condition Status Register (DOORBELL2_ICSR) (Address Offset 0220h)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICS15	ICS14	ICS13	ICS12	ICS11	ICS10	ICS9	ICS8	ICS7	ICS6	ICS5	ICS4	ICS3	ICS2	ICS1	ICS0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Doorbell 2 Interrupt Condition Clear Register (DOORBELL2_ICCR) (Address Offset 0228h)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICC15	ICC14	ICC13	ICC12	ICC11	ICC10	ICC9	ICC8	ICC7	ICC6	ICC5	ICC4	ICC3	ICC2	ICC1	ICC0
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: R = Read only; W = Write only; -*n* = Value after reset

Figure 46. Doorbell 3 Interrupt Condition Status and Clear Registers

Doorbell 3 Interrupt Condition Status Register (DOORBELL3_ICSR) (Address Offset 0230h)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICS15	ICS14	ICS13	ICS12	ICS11	ICS10	ICS9	ICS8	ICS7	ICS6	ICS5	ICS4	ICS3	ICS2	ICS1	ICS0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Doorbell 3 Interrupt Condition Clear Register (DOORBELL3_ICCR) (Address Offset 0238h)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICC15	ICC14	ICC13	ICC12	ICC11	ICC10	ICC9	ICC8	ICC7	ICC6	ICC5	ICC4	ICC3	ICC2	ICC1	ICC0
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: R = Read only; W = Write only; -*n* = Value after reset

4.3.2 CPPI Interrupt Condition Status and Clear Registers

The ICSRs and the ICCRs for the RXU and the TXU are shown in Figure 47 and Figure 48. These interrupt condition registers are used when the SRIO peripheral receives and transmits data message packets. Each ICS bit corresponds to the interrupt for one of the buffer descriptor queues. For example, the bits ICS15, ICS8, and ICS0 of RX_CPPI_ICSR correspond to RX buffer descriptor queues 15, 8, and 0. Similarly, the bits ICS15, ICS8, and ICS0 of TX_CPPI_ICSR support TX buffer descriptor queues 15, 8, and 0. The 16 ICC bits of each interrupt condition clear register (ICCR) are used to clear the corresponding bits in the ICSR.

For reception, the clearing of any ICSR bit depends on the CPU writing the value of the last buffer descriptor processed to the completion pointer (CP) register for the queue (QUEUE*n*_RXDMA_CP). Port hardware clears the ICSR bit only if the CP value written by the CPU equals the port written value in the CP register.

For transmission, the clearing of any ICSR bit is dependent on the CPU writing to the CP register for the queue (QUEUE*n*_TXDMA_CP). The CPU acknowledges the interrupt after reclaiming all available buffer descriptors by writing the CP value. This value is compared against the port written value in the CP register. If the values are equal, the interrupt is de-asserted.

Figure 47. RX CPPI Interrupt Condition Status and Clear Registers

RX CPPI Interrupt Condition Status Register (RX_CPPI_ICSR) (Address Offset 0240h)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICS15	ICS14	ICS13	ICS12	ICS11	ICS10	ICS9	ICS8	ICS7	ICS6	ICS5	ICS4	ICS3	ICS2	ICS1	ICS0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

RX CPPI Interrupt Condition Clear Register (RX_CPPI_ICCR) (Address Offset 0248h)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICC15	ICC14	ICC13	ICC12	ICC11	ICC10	ICC9	ICC8	ICC7	ICC6	ICC5	ICC4	ICC3	ICC2	ICC1	ICC0
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: R = Read only; W = Write only; -*n* = Value after reset

Figure 48. TX CPPI Interrupt Condition Status and Clear Registers

TX CPPI Interrupt Condition Status Register (TX_CPPI_ICSR) (Address Offset 0250h)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICS15	ICS14	ICS13	ICS12	ICS11	ICS10	ICS9	ICS8	ICS7	ICS6	ICS5	ICS4	ICS3	ICS2	ICS1	ICS0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

TX CPPI Interrupt Condition Clear Register (TX_CPPI_ICCR) (Address Offset 0258h)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICC15	ICC14	ICC13	ICC12	ICC11	ICC10	ICC9	ICC8	ICC7	ICC6	ICC5	ICC4	ICC3	ICC2	ICC1	ICC0
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: R = Read only; W = Write only; -*n* = Value after reset

4.3.3 LSU Interrupt Condition Status and Clear Registers

The ICSR and the ICCR for the LSUs are shown in Figure 49. These interrupt condition registers are used when the SRIO peripheral transmits direct I/O packets. As described in Table 38, each of the status and clear bits corresponds to a particular type of transaction interrupt condition for a particular LSU. The ICS bits of LSU_ICSR indicate the occurrence of the conditions. The ICC bits of LSU_ICCR are used to clear the corresponding ICS bits.

Figure 49. LSU Interrupt Condition Status and Clear Registers

LSU Interrupt Condition Status Register (LSU_ICSR) (Address Offset 0260h)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ICS31	ICS30	ICS29	ICS28	ICS27	ICS26	ICS25	ICS24	ICS23	ICS22	ICS21	ICS20	ICS19	ICS18	ICS17	ICS16
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICS15	ICS14	ICS13	ICS12	ICS11	ICS10	ICS9	ICS8	ICS7	ICS6	ICS5	ICS4	ICS3	ICS2	ICS1	ICS0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LSU Interrupt Condition Clear Register (LSU_ICCR) (Address Offset 0268h)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ICC31	ICC30	ICC29	ICC28	ICC27	ICC26	ICC25	ICC24	ICC23	ICC22	ICC21	ICC20	ICC19	ICC18	ICC17	ICC16
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICC15	ICC14	ICC13	ICC12	ICC11	ICC10	ICC9	ICC8	ICC7	ICC6	ICC5	ICC4	ICC3	ICC2	ICC1	ICC0
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: R = Read only; W = Write only; -*n* = Value after reset

Table 38. Interrupt Conditions Shown in LSU_ICSR and Cleared With LSU_ICCR

Bit	Associated LSU	Interrupt Condition
31	LSU4	Packet not sent due to unavailable outbound credit at given priority
30	LSU4	Retry Doorbell response received or Atomic test-and-swap was not allowed (semaphore in use)
29	LSU4	Transaction was not sent due to DMA data transfer error
28	LSU4	Transaction was not sent due to unsupported transaction type or invalid field encoding
27	LSU4	Non-posted transaction received ERROR response, or error in response payload
26	LSU4	Transaction was not sent due to Xoff condition
25	LSU4	Transaction timeout occurred
24	LSU4	Transaction complete, No errors (posted/non-posted)[(1)]
23	LSU3	Packet not sent due to unavailable outbound credit at given priority
22	LSU3	Retry Doorbell response received or Atomic test-and-swap was not allowed (semaphore in use)
21	LSU3	Transaction was not sent due to DMA data transfer error
20	LSU3	Transaction was not sent due to unsupported transaction type or invalid field encoding
19	LSU3	Non-posted transaction received ERROR response, or error in response payload
18	LSU3	Transaction was not sent due to Xoff condition
17	LSU3	Transaction timeout occurred Non-posted transaction received ERROR response, or error in response payload
16	LSU3	Transaction complete, No errors (posted/non-posted)[(1)]
15	LSU2	Packet not sent due to unavailable outbound credit at given priority
14	LSU2	Retry Doorbell response received or Atomic test-and-swap was not allowed (semaphore in use)
13	LSU2	Transaction was not sent due to DMA data transfer error

(1) Enable for this interrupt is ultimately controlled by the Interrupt Req register bit of LSU*n*_REG4. This allows enabling/disabling on a per request basis. For optimum LSU performance, interrupt pacing should not be used on the LSU interrupts. Section 4.7 describes interrupt pacing.

Table 38. Interrupt Conditions Shown in LSU_ICSR and Cleared With LSU_ICCR (continued)

Bit	Associated LSU	Interrupt Condition
12	LSU2	Transaction was not sent due to unsupported transaction type or invalid field encoding
11	LSU2	Non-posted transaction received ERROR response, or error in response payload
10	LSU2	Transaction was not sent due to Xoff condition
9	LSU2	Transaction timeout occurred
8	LSU2	Transaction complete, No errors (posted/non-posted)(1)
7	LSU1	Packet not sent due to unavailable outbound credit at given priority
6	LSU1	Retry Doorbell response received or Atomic test-and-swap was not allowed (semaphore in use)
5	LSU1	Transaction was not sent due to DMA data transfer error
4	LSU1	Transaction was not sent due to unsupported transaction type or invalid field encoding
3	LSU1	Non-posted transaction received ERROR response, or error in response payload
2	LSU1	Transaction was not sent due to Xoff condition
1	LSU1	Transaction timeout occurred
0	LSU1	Transaction complete, No errors (posted/non-posted)(1)

4.3.4 Error, Reset, and Special Event Interrupt Condition Status and Clear Registers

The ICSR and the ICCR for the SRIO ports are shown in Figure 50. As described in Table 39, each of the non-reserved status and clear bits corresponds to a particular interrupt condition in one or more of the SRIO ports. The ICS bits of ERR_RST_EVNT_ICSR indicate the occurrence of the conditions. The ICC bits of ERR_RST_EVNT_ICCR are used to clear the corresponding ICS bits.

Figure 50. Error, Reset, and Special Event Interrupt Condition Status and Clear Registers

Error, Reset, and Special Event Interrupt Condition Status Register (ERR_RST_EVNT_ICSR) (Address Offset 0270h)

31															17	16
Reserved																ICS16
R-0																R-0

15	12	11	10	9	8	7	3	2	1	0
Reserved		ICS11	ICS10	ICS9	ICS8	Reserved		ICS2	ICS1	ICS0
R-0		R-0	R-0	R-0	R-0	R-0		R-0	R-0	R-0

Error, Reset, and Special Event Interrupt Condition Clear Register (ERR_RST_EVNT_ICCR) (Address Offset 0278h)

31															17	16
Reserved																ICC16
R-0																W-0

15	12	11	10	9	8	7	3	2	1	0
Reserved		ICC11	ICC10	ICC9	ICC8	Reserved		ICC2	ICC1	ICC0
R-0		W-0	W-0	W-0	W-0	R-0		W-0	W-0	W-0

LEGEND: R = Read only; W = Write only; -*n* = Value after reset

Table 39. Interrupt Conditions Shown in ERR_RST_EVNT_ICSR and Cleared With ERR_RST_EVNT_ICCR

Bit	Interrupt Condition
31-17	Reserved
16	Device reset interrupt from any port
15-12	Reserved
11	Port 3 error
10	Port 2 error
9	Port 1 error
8	Port 0 error
7-3	Reserved
2	Logical layer error management event capture
1	Port-write-in request received on any port
0	Multi-cast event control symbol interrupt received on any port

The interrupt status bits found in the ERR_RST_EVNT (0x0270) can be cleared by writing to the ICCR register (0x0278) in the same manner as other interrupts. However, in order for new event detection and interrupt generation to occur for these special interrupts, additional register bits must be cleared. The following table notes the additional interrupt source register bits that need to be cleared and the appropriated sequence. These are all the bits that can cause the ERR_RST_EVNT status bits to be set

Table 40. Interrupt Clearing Sequence for Special Event Interrupts

Interrupt Function	First Step	Second Step	Third Step
Multicast Event Control Symbol received on any port	Write 1 to clear: Offset 0x0278 ERR_RST_EVNT_ICCR[0]	Write 1 to clear: Offset 0x12004 SP_IP_MODE[4]	
Port Write In Request received on any port	Write 1 to clear: Offset 0x0278	Write 1 to clear: Offset 0x12004	

Table 40. Interrupt Clearing Sequence for Special Event Interrupts (continued)

Interrupt Function	First Step	Second Step	Third Step
	ERR_RST_EVNT_ICCR[1]	SP_IP_MODE[0]	
Port 0 Error	Write 1 to clear:	Write 1 to clear any of the following possible bits:	Write 1 to clear:
	Offset 0x0278	Offset 0x2040	Offset 0x14004
	ERR_RST_EVNT_ICCR[8]	SP0_ERR_STAT[2] - Fatal error	SP0_CTL_INDEP[6]
		SP0_ERR_STAT[25] - Failed Threshold	
		SP0_ERR_STAT[24] - Degraded Threshold	
		Offset 0x14004	
		SP0_CTL_INDEP[20] - Illegal Transaction	
		SP0_CTL_INDEP[16] - Max Retry Error	
Port 1 Error	Write 1 to clear:	Write 1 to clear any of the following possible bits:	Write 1 to clear:
	Offset 0x0278	Offset 0x2080	Offset 0x14104
	ERR_RST_EVNT_ICCR[9]	SP1_ERR_STAT[2] - Fatal error	SP1_CTL_INDEP[6]
		SP1_ERR_STAT[25] - Failed Threshold	
		SP1_ERR_STAT[24] - Degraded Threshold	
		Offset 0x14104	
		SP1_CTL_INDEP[20] - Illegal Transaction	
		SP1_CTL_INDEP[16] - Max Retry Error	
Port 2 Error	Write 1 to clear:	Write 1 to clear any of the following possible bits:	Write 1 to clear:
	Offset 0x0278	Offset 0x20C0	Offset 0x14204
	ERR_RST_EVNT_ICCR[10]	SP2_ERR_STAT[2] - Fatal error	SP2_CTL_INDEP[6]
		SP2_ERR_STAT[25] - Failed Threshold	
		SP2_ERR_STAT[24] - Degraded Threshold	
		Offset 0x14204	
		SP2_CTL_INDEP[20] - Illegal Transaction	
		SP2_CTL_INDEP[16] - Max Retry Error	
Port 3 Error	Write 1 to clear:	Write 1 to clear any of the following possible bits:	Write 1 to clear:
	Offset 0x0278	Offset 0x2100	Offset 0x14304
	ERR_RST_EVNT_ICCR[11]	SP3_ERR_STAT[2] - Fatal error	SP_CTL_INDEP[6]
		SP3_ERR_STAT[25] - Failed Threshold	
		SP3_ERR_STAT[24] - Degraded Threshold	
		Offset 0x14304	
		SP3_CTL_INDEP[20] - Illegal Transaction	

Table 40. Interrupt Clearing Sequence for Special Event Interrupts (continued)

Interrupt Function	First Step	Second Step	Third Step
		SP3_CTL_INDEP[16] - Max Retry Error	
Device Reset	Write 1 to clear:	Write 1 to clear:	
	Offset 0x0278	Offset 0x12004	
	ERR_RST_EVNT_ICCR[16]	SP_IP_MODE[2]	

4.4 Interrupt Condition Routing Registers

The interrupt conditions are programmable to select the interrupt output that will be driven. Using the interrupt condition routing registers (ICRRs), software can independently route each interrupt request to any of the interrupt destinations supported by the device. For example, a quad core device may support four CPU servicing interrupt destinations, one per core (INTDST0 for Core0, INTDST1 for Core1, INTDST2 for Core2, and INTDST3 for Core3). In addition, INTDST4 may be globally routed to all cores and provide notification of a change in the one ICSR, while INTDST5 may be globally routed to all cores and provide notification of a change in a different ICSR. The routing defaults for an interrupt condition routing bit (ICRx) are given in Table 41.

Table 41. Interrupt Condition Routing Options

Field	Access	Reset Value	Value	Function
ICRx	R	0000b	0000b	Routed to INTDST0
			0001b	Routed to INTDST1
			0010b	Routed to INTDST2
			0011b	Routed to INTDST3
			0100b	Routed to INTDST4
			0101b	Routed to INTDST5
			0110b	Routed to INTDST6
			0111b	Routed to INTDST7
			1111b	No interrupt destination, interrupt source disabled
			other	Reserved

4.4.1 Doorbell Interrupt Condition Routing Registers

Figure 51 shows the interrupt condition routing registers for Doorbell 0. The other doorbell ICRRs have the same bit field map, with the following addresses:

- DOORBELL1_ICRR and DOORBELL1_ICCR2 (address offsets 0290h and 0294h)
- DOORBELL2_ICRR and DOORBELL2_ICRR2 (address offset 02A0h and 02A4h)
- DOORBELL3_ICRR and DOORBELL3_ICRR2 (address offset 02B0h and 02B4h)

When doorbell packets are received by the SRIO peripheral, these ICRRs route doorbell interrupt requests to interrupt destinations. For example, if ICS6 = 1 in DOORBELL2_ICSR and ICR6 = 0010b in DOORBELL2_ICRR, the interrupt request from Doorbell 2, bit 6 is sent to interrupt destination 2.

Figure 51. Doorbell 0 Interrupt Condition Routing Registers

Doorbell 0 Interrupt Condition Routing Register (DOORBELL0_ICRR) (Address Offset 0280h)

31–28	27–24	23–20	19–16
ICR7	ICR6	ICR5	ICR4
R/W-0000	R/W-0000	R/W-0000	R/W-0000

15–12	11–8	7–4	3–0
ICR3	ICR2	ICR1	ICR0
R/W-0000	R/W-0000	R/W-0000	R/W-0000

Doorbell 0 Interrupt Condition Routing Register 2 (DOORBELL0_ICRR2) (Address Offset 0284h)

31–28	27–24	23–20	19–16
ICR15	ICR14	ICR13	ICR12
R/W-0000	R/W-0000	R/W-0000	R/W-0000

15–12	11–8	7–4	3–0
ICR11	ICR10	ICR9	ICR8
R/W-0000	R/W-0000	R/W-0000	R/W-0000

LEGEND: R/W = Read/Write; -*n* = Value after reset

4.4.1.1 CPPI Interrupt Condition Routing Registers

Figure 52 shows the ICRRs for the RXU, and Figure 53 shows the ICRRs for the TXU. These registers route queue interrupts to interrupt destinations. For example, if ICS6 = 1 in RX_CPPI_ICSR and ICR6 = 0010b in RX_CPPI_ICRR, the interrupt request from RX buffer descriptor queue 6 is sent to interrupt destination 2. Similarly, if ICS6 = 1 in TX_CPPI_ICSR and ICR6 = 0011b in TX_CPPI_ICRR, the interrupt request from TX buffer descriptor queue 6 is sent to interrupt destination 3.

Figure 52. RX CPPI Interrupt Condition Routing Registers

RX CPPI Interrupt Condition Routing Register (RX_CPPI_ICRR) (Address Offset 02C0h)

31–28	27–24	23–20	19–16
ICR7	ICR6	ICR5	ICR4
R/W-0000	R/W-0000	R/W-0000	R/W-0000

15–12	11–8	7–4	3–0
ICR3	ICR2	ICR1	ICR0
R/W-0000	R/W-0000	R/W-0000	R/W-0000

RX CPPI Interrupt Condition Routing Register 2 (RX_CPPI_ICRR2) (Address Offset 02C4h)

31–28	27–24	23–20	19–16
ICR15	ICR14	ICR13	ICR12
R/W-0000	R/W-0000	R/W-0000	R/W-0000

15–12	11–8	7–4	3–0
ICR11	ICR10	ICR9	ICR8
R/W-0000	R/W-0000	R/W-0000	R/W-0000

LEGEND: R/W = Read/Write; -*n* = Value after reset

Figure 53. TX CPPI Interrupt Condition Routing Registers

TX CPPI Interrupt Condition Routing Register (TX_CPPI_ICRR) (Address Offset 02D0h)

31–28	27–24	23–20	19–16
ICR7	ICR6	ICR5	ICR4
R/W-0000	R/W-0000	R/W-0000	R/W-0000

15–12	11–8	7–4	3–0
ICR3	ICR2	ICR1	ICR0
R/W-0000	R/W-0000	R/W-0000	R/W-0000

TX CPPI Interrupt Condition Routing Register 2 (TX_CPPI_ICRR2) (Address Offset 02D4h)

31–28	27–24	23–20	19–16
ICR15	ICR14	ICR13	ICR12
R/W-0000	R/W-0000	R/W-0000	R/W-0000

15–12	11–8	7–4	3–0
ICR11	ICR10	ICR9	ICR8
R/W-0000	R/W-0000	R/W-0000	R/W-0000

LEGEND: R/W = Read/Write; -*n* = Value after reset

4.4.1.2 LSU Interrupt Condition Routing Registers

Figure 54 shows the ICRRs for the LSU interrupt requests. These registers route LSU interrupt requests to interrupt destinations. For example, if ICS4 = 1 in LSU_ICSR and ICR4 = 0000b in LSU_ICRR0, LSU1 has generated a transaction-timeout interrupt request, and that request is routed to interrupt destination 0.

Figure 54. LSU Interrupt Condition Routing Registers

LSU Interrupt Condition Routing Register 0 (LSU_ICRR0) (Address Offset 02E0h)

31–28	27–24	23–20	19–16
ICR7	ICR6	ICR5	ICR4
R/W-0000	R/W-0000	R/W-0000	R/W-0000

15–12	11–8	7–4	3–0
ICR3	ICR2	ICR1	ICR0
R/W-0000	R/W-0000	R/W-0000	R/W-0000

LSU Interrupt Condition Routing Register 1 (LSU_ICRR1) (Address Offset 02E4h)

31–28	27–24	23–20	19–16
ICR15	ICR14	ICR13	ICR12
R/W-0000	R/W-0000	R/W-0000	R/W-0000

15–12	11–8	7–4	3–0
ICR11	ICR10	ICR9	ICR8
R/W-0000	R/W-0000	R/W-0000	R/W-0000

LSU Interrupt Condition Routing Register 2 (LSU_ICRR2) (Address Offset 02E8h)

31–28	27–24	23–20	19–16
ICR23	ICR22	ICR21	ICR20
R/W-0000	R/W-0000	R/W-0000	R/W-0000

15–12	11–8	7–4	3–0
ICR19	ICR18	ICR17	ICR16
R/W-0000	R/W-0000	R/W-0000	R/W-0000

LSU Interrupt Condition Routing Register 3 (LSU_ICRR3) (Address Offset 02ECh)

31–28	27–24	23–20	19–16
ICR31	ICR30	ICR29	ICR28
R/W-0000	R/W-0000	R/W-0000	R/W-0000

15–12	11–8	7–4	3–0
ICR27	ICR26	ICR25	ICR24
R/W-0000	R/W-0000	R/W-0000	R/W-0000

LEGEND: R/W = Read/Write; -*n* = Value after reset

4.4.1.3 Error, Reset, and Special Event Interrupt Condition Routing Registers

The ICRRs shown in Figure 55 route port interrupt requests to interrupt destinations. For example, if ICS8 = 1 in ERR_RST_EVNT_ICSR and ICR8 = 0001b in ERR_RST_EVNT_ICRR2, port 0 has generated an error interrupt request, and that request is routed to interrupt destination 1.

Figure 55. Error, Reset, and Special Event Interrupt Condition Routing Registers

Error, Reset, and Special Event ICRR (ERR_RST_EVNT_ICRR) (Address Offset 02F0h)

31		12	11	8	7	4	3	0
Reserved								
R-0								
Reserved			ICR2		ICR1		ICR0	
R-0			R/W-0000		R/W-0000		R/W-0000	

Error, Reset, & Special Event ICRR 2 (ERR_RST_EVNT_ICRR2) (Address Offset 02F4h)

31		16	15	12	11	8	7	4	3	0
Reserved										
R-0										
			ICR11		ICR10		ICR9		ICR8	
			R/W-0000		R/W-0000		R/W-0000		R/W-0000	

Error, Reset, and Special Event ICRR 3 (ERR_RST_EVNT_ICRR3) (Address Offset 02F8h)

31		4	3	0
Reserved				
R-0				
Reserved			ICR16	
R-0			R/W-0000	

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

4.5 Interrupt Status Decode Registers

There are 8 blocks of the ICSRs to indicate the source of a pending interrupt.

```
0x0200: Doorbell0 interrupts
0x0210: Doorbell1 interrupts
0x0220: Doorbell2 interrupts
0x0230: Doorbell3 interrupts
0x0240: RX CPPI interrupts
0x0250: TX CPPI interrupts
0x0260: LSU interrupts
0x0270: Error, Reset, and Special Event interrupts
```

To reduce the number of reads (up to 5 reads) required to find the source bit, an Interrupt Status Decode Register (ISDR) is implemented for each supported physical interrupt destination. The device supports up to eight interrupt destinations, INTDST0-INTDST7. The names of the ISDRs and their address offsets are:

- INTDST0_DECODE (address offset 0300h)
- INTDST1_DECODE (address offset 0304h)
- INTDST2_DECODE (address offset 0308h)
- INTDST3_DECODE (address offset 030Ch)
- INTDST4_DECODE (address offset 0310h)
- INTDST5_DECODE (address offset 0314h)
- INTDST6_DECODE (address offset 0318h)
- INTDST7_DECODE (address offset 031Ch)

Aside from supporting different interrupt destinations, the ISDRs are the same in content and functionality. The register fields are shown in Figure 56. Figure 57 shows which interrupt sources can be mapped to

each bit in the ISDR. Bits within the LSU interrupt condition status register (ICSR) are logically grouped for a given core and ORed together into a single bit (bit 31) of the decode register. Similarly, the bits within the Error, Reset, and Special Event ICSR are ORed together into bit 30 of the decode register. The TX CPPI and RX CPPI interrupt sources (one for each buffer descriptor queue) can be mapped to bits 31-16 as shown in Figure 57. The doorbell interrupt sources can be mapped to bits 15-0.

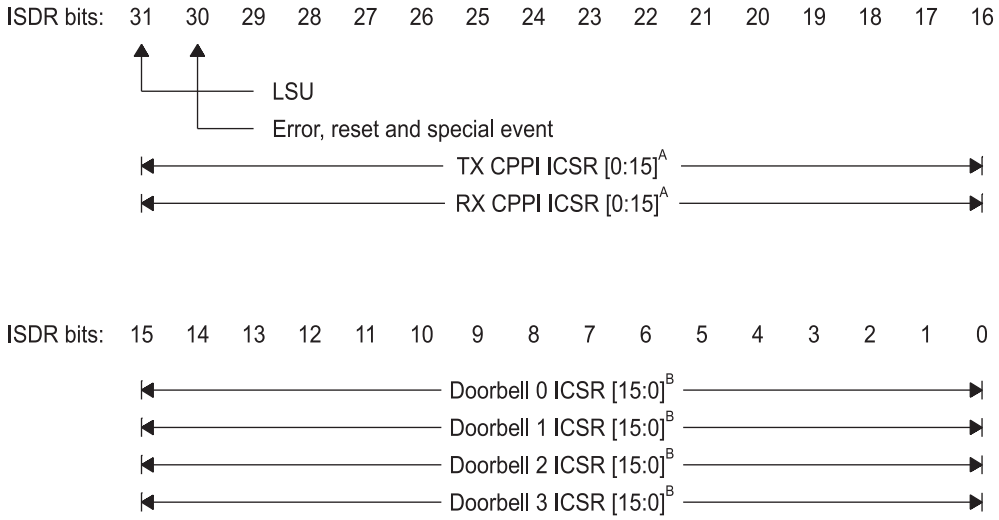
An interrupt source is mapped to ISDR bits only if the ICRR for that interrupt source routes it to the corresponding interrupt destination. When multiple interrupt sources are mapped to the same bit, the bit status is a logical OR of those interrupt sources. The mapping of interrupt source bits to decode bits is fixed and is not programmable.

Figure 56. Interrupt Status Decode Register (INTDST*n*_DECODE)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ISD31	ISD30	ISD29	ISD28	ISD27	ISD26	ISD25	ISD24	ISD23	ISD22	ISD21	ISD20	ISD19	ISD18	ISD17	ISD16
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ISD15	ISD14	ISD13	ISD12	ISD11	ISD10	ISD9	ISD8	ISD7	ISD6	ISD5	ISD4	ISD3	ISD2	ISD1	ISD0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = Value after reset

Figure 57. Interrupt Sources Assigned to ISDR Bits

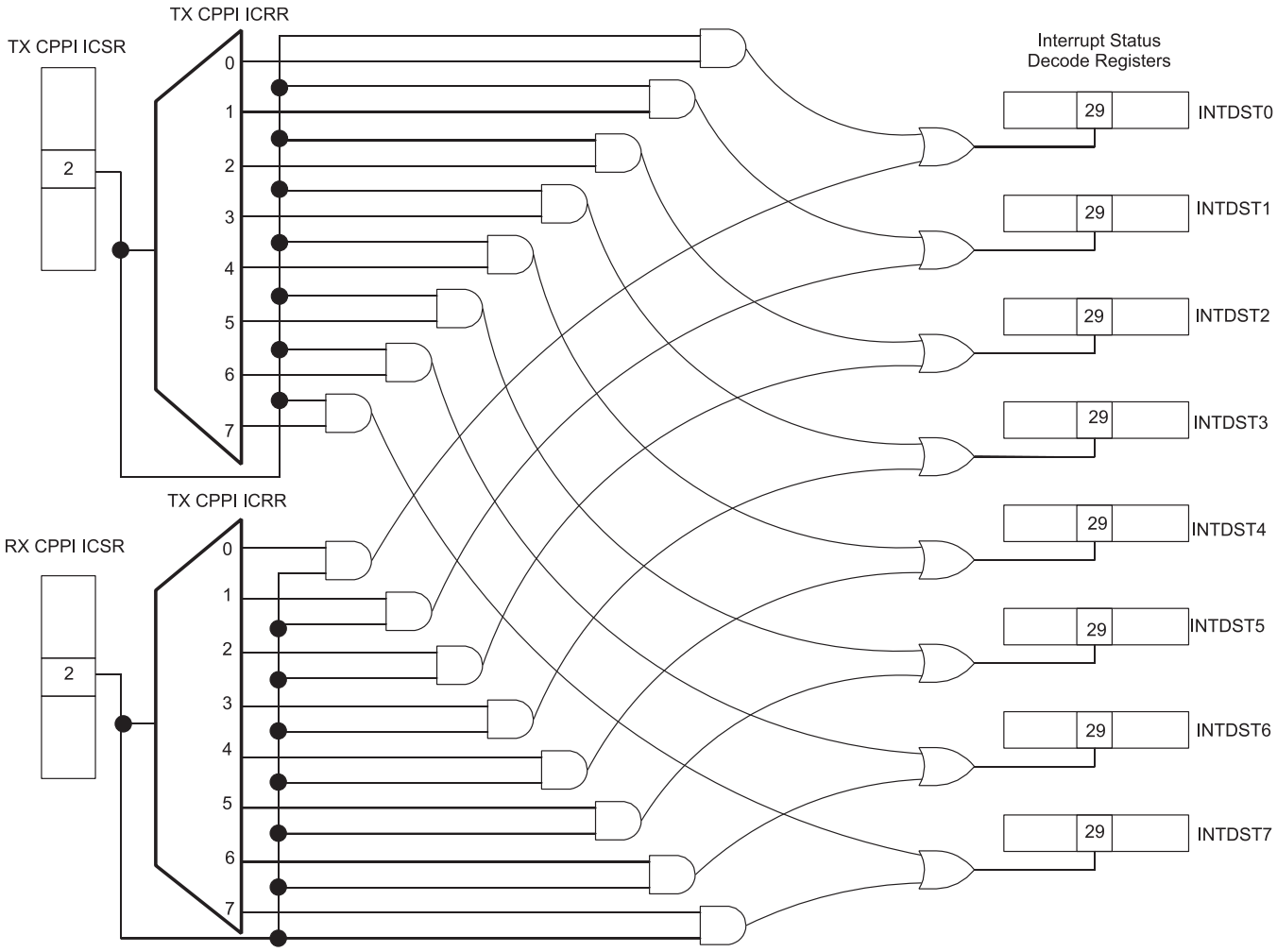


A Note that bits 0 through 15 of the ICSR correspond to bits 31 through 16 of the ISDR. For example, bit 15 of the ICSR corresponds to bit 31 of the ISDR, and so on.

B Note that bits 15 through 0 of the ICSR correspond to bits 15 through 0 of the ISDR. For example, bit 15 of the ICSR corresponds to bit 15 of the ISDR, and so on.

As an example of reading an ISDR, if bit 29 of the ISDR is set, this indicates that there is a pending interrupt on either the TX CPPI queue 2 or RX CPPI queue 2. Figure 58 illustrates the decode routing for this example.

Figure 58. Example Diagram of Interrupt Status Decode Register Mapping



The following are suggestions for minimizing the number of register reads to identifying the interrupt source:

- Dedicate each doorbell ICSR to one core. The CPU can then determine the interrupt source from a single read of the decode register.
- Assign the RX and TX CPPI queues orthogonally to different cores. The CPU can then determine the interrupt source from a single read of the decode registers. The only exceptions to this are bits 31 and 30, which are also logically ORed with LSU and port interrupt sources.

4.6 Interrupt Generation

Interrupts are triggered on a 0-to-1 logic-signal transition. Regardless of the interrupt sources, the physical interrupts are set only when the total number of set ICSR bits transitions from none to one or more. The peripheral is responsible for setting the correct bit within the ICSR. The ICRR register maps the pending interrupt request to the appropriate physical interrupt line. The corresponding CPU is interrupted and reads the ISDR and ICSR registers to determine the interrupt source and appropriate action. Interrupt generation is governed by the interrupt pacing discussed Section 4.7.

4.7 Interrupt Pacing

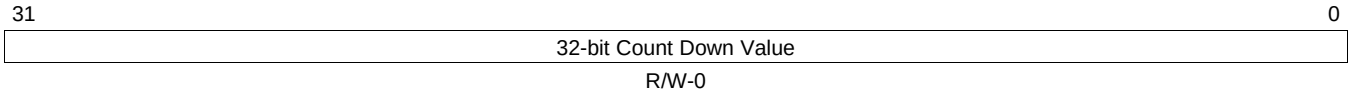
The rate at which an interrupt can be generated is controllable for each physical interrupt destination. Rate control is implemented with a programmable down-counter. The load value of the counter is written by the CPU into the appropriate interrupt rate control register (see Figure 59). The counter reloads and immediately starts down-counting each time the CPU writes these registers. When the rate control counter register is written, and the counter value reaches zero (note that the CPU may write zero immediately for a zero count), the interrupt pulse generation logic is allowed to fire a single pulse if any bits in the corresponding ICSR register bits are set (or become set after the zero count is reached). The counter remains at zero. When the single pulse is generated, the logic will not generate another pulse, regardless of interrupt status changes, until the rate control counter register is written again.

An interrupt rate control register (INTDST*n*_RATE_CNTL) is implemented for each supported physical interrupt destination. The device supports up to eight interrupt destinations, INTDST0-INTDST7. The names of the registers and their address offsets are:

- INTDST0_RATE_CNTL (address offset 0320h)
- INTDST1_RATE_CNTL (address offset 0324h)
- INTDST2_RATE_CNTL (address offset 0328h)
- INTDST3_RATE_CNTL (address offset 032Ch)
- INTDST4_RATE_CNTL (address offset 0330h)
- INTDST5_RATE_CNTL (address offset 0334h)
- INTDST6_RATE_CNTL (address offset 0338h)
- INTDST7_RATE_CNTL (address offset 033Ch)

If interrupt pacing is not desired for a particular interrupt destination, the CPU must still write 00000000h into the INTDST*n*_RATE_CNTL register after clearing the corresponding ICSR bits to acknowledge the physical interrupt. If an ICSR is not mapped to an interrupt destination, pending interrupt bits within the ICSR maintain current status. When enabled, the interrupt logic re-evaluates all pending interrupts and re-pulses the interrupt signal if any interrupt conditions are pending. The down-counter is based on the DMA clock cycle.

Figure 59. INTDST*n*_RATE_CNTL Interrupt Rate Control Register



LEGEND: R/W = Read/Write; -*n* = Value after reset

4.8 Interrupt Handling

Interrupts are either signaled externally through RapidIO packets, or internally by state machines in the peripheral. CPU servicing interrupts are signaled externally by the DOORBELL RapidIO packet in direct I/O mode, or internally by the CPPI module in the message passing mode. Error Status interrupts are signaled when error counting logic within the peripheral have reached their thresholds. In either case, it is the peripheral that signals the interrupt and sets the corresponding status bits.

When the CPU is interrupted, it reads the ICSR registers to determine the source of the interrupt and appropriate action to take. For example, if it is a DOORBELL interrupt, the CPU will read from an L2 address that is specified by its circular buffer read pointer that is managed by software. There may be more than one circular buffer for each core. The correct circular buffer to read from and increment depends on the bit set in the ICSR register. The CPU then clears the status bit.

For Error Status interrupts, the peripheral must indicate to all the CPUs that one of the link ports has reached the error threshold. In this case, the peripheral sets the status bit indicating degraded or failed limits have been reached, and an interrupt is generated to each core through the ICRR mapping. The cores can then scan the ICSR registers to determine the port with the error problems. Further action can then be taken as determined by the application.

```
Interrupt Handler
temp1 = SRIO_REGS->TX_CPPI_ICSR;
  if ((temp1 & 0x00000001) == 0x00000001)
  {
           SRIO_REGS->Queue0_TXDMA_CP = (int )TX_DESCP0_0;
}

temp2 = SRIO_REGS->RX_CPPI_ICSR;
  if ((temp2 & 0x00000001) == 0x00000001)
  {
           SRIO_REGS->Queue0_RXDMA_CP = (int )RX_DESCP0_0;
}

SRIO_REGS->DOORBELL0_ICCR=0xFFFFFFFF;
SRIO_REGS->DOORBELL1_ICCR=0xFFFFFFFF;
SRIO_REGS->DOORBELL2_ICCR=0xFFFFFFFF;
SRIO_REGS->DOORBELL3_ICCR=0xFFFFFFFF;
SRIO_REGS->INTDST0_Rate_CNTL=1;
```

For an SRIO interrupt handling example, see Section A.4.

5 SRIO Registers

Table 42 lists the names and address offsets of the memory-mapped registers for the Serial RapidIO (SRIO) peripheral. For the exact memory addresses of these registers, see the *TMS320C6457 Fixed-Point Digital Signal Processor* data manual (SPRS582).

Table 42. SRIO Registers

Offset	Acronym	Register Description	See
0000h	PID	Peripheral Identification Register	Section 5.1
0004h	PCR	Peripheral Control Register	Section 5.2
0020h	PER_SET_CNTL	Peripheral Settings Control Register	Section 5.3
0024h	PER_SET_CNTL1	Peripheral Settings Control Register 1	Section 5.4
0030h	GBL_EN	Peripheral Global Enable Register	Section 5.5
0034h	GBL_EN_STAT	Peripheral Global Enable Status	Section 5.6
0038h	BLK0_EN	Block Enable 0	Section 5.7
003Ch	BLK0_EN_STAT	Block Enable Status 0	Section 5.8
0040h	BLK1_EN	Block Enable 1	Section 5.7
0044h	BLK1_EN_STAT	Block Enable Status 1	Section 5.8
0048h	BLK2_EN	Block Enable 2	Section 5.7
004Ch	BLK2_EN_STAT	Block Enable Status 2	Section 5.8
0050h	BLK3_EN	Block Enable 3	Section 5.7
0054h	BLK3_EN_STAT	Block Enable Status 3	Section 5.8
0058h	BLK4_EN	Block Enable 4	Section 5.7
005Ch	BLK4_EN_STAT	Block Enable Status 4	Section 5.8
0060h	BLK5_EN	Block Enable 5	Section 5.7
0064h	BLK5_EN_STAT	Block Enable Status 5	Section 5.8
0068h	BLK6_EN	Block Enable 6	Section 5.7
006Ch	BLK6_EN_STAT	Block Enable Status 6	Section 5.8
0070h	BLK7_EN	Block Enable 7	Section 5.7
0074h	BLK7_EN_STAT	Block Enable Status 7	Section 5.8
0078h	BLK8_EN	Block Enable 8	Section 5.7
007Ch	BLK8_EN_STAT	Block Enable Status 8	Section 5.8
0080h	DEVICEID_REG1	RapidIO DEVICEID1 Register	Section 5.9
0084h	DEVICEID_REG2	RapidIO DEVICEID2 Register	Section 5.10
0088h	DEVICEID_REG3	RapidIO DEVICEID3 Register	Section 5.11
008Ch	DEVICEID_REG4	RapidIO DEVICEID4 Register	Section 5.12
0090h	PF_16B_CNTL0	Packet Forwarding Register 0 for 16-bit DeviceIDs	Section 5.13
0094h	PF_8B_CNTL0	Packet Forwarding Register 0 for 8-bit DeviceIDs	Section 5.14
0098h	PF_16B_CNTL1	Packet Forwarding Register 1 for 16-bit DeviceIDs	Section 5.13
009Ch	PF_8B_CNTL1	Packet Forwarding Register 1 for 8-bit DeviceIDs	Section 5.14
00A0h	PF_16B_CNTL2	Packet Forwarding Register 2 for 16-bit DeviceIDs	Section 5.13
00A4h	PF_8B_CNTL2	Packet Forwarding Register 2 for 8-bit DeviceIDs	Section 5.14
00A8h	PF_16B_CNTL3	Packet Forwarding Register 3 for 16-bit DeviceIDs	Section 5.13
00ACh	PF_8B_CNTL3	Packet Forwarding Register 3 for 8-bit DeviceIDs	Section 5.14
0100h	SERDES_CFGRX0_CNTL	SERDES Receive Channel Configuration Register 0	Section 5.15
0104h	SERDES_CFGRX1_CNTL	SERDES Receive Channel Configuration Register 1	Section 5.15
0108h	SERDES_CFGRX2_CNTL	SERDES Receive Channel Configuration Register 2	Section 5.15
010Ch	SERDES_CFGRX3_CNTL	SERDES Receive Channel Configuration Register 3	Section 5.15
0110h	SERDES_CFGTX0_CNTL	SERDES Transmit Channel Configuration Register 0	Section 5.16
0114h	SERDES_CFGTX1_CNTL	SERDES Transmit Channel Configuration Register 1	Section 5.16
0118h	SERDES_CFGTX2_CNTL	SERDES Transmit Channel Configuration Register 2	Section 5.16

Table 42. SRIO Registers (continued)

Offset	Acronym	Register Description	See
011Ch	SERDES_CFGTX3_CNTL	SERDES Transmit Channel Configuration Register 3	Section 5.16
0120h	SERDES_CFG0_CNTL	SERDES Macro Configuration Register 0	Section 5.17
0124h	SERDES_CFG1_CNTL	SERDES Macro Configuration Register 1	Section 5.17
0128h	SERDES_CFG2_CNTL	SERDES Macro Configuration Register 2	Section 5.17
012Ch	SERDES_CFG3_CNTL	SERDES Macro Configuration Register 3	Section 5.17
0200h	DOORBELL0_ICSR	DOORBELL Interrupt Condition Status Register 0	Section 5.18
0208h	DOORBELL0_ICCR	DOORBELL Interrupt Condition Clear Register 0	Section 5.19
0210h	DOORBELL1_ICSR	DOORBELL Interrupt Condition Status Register 1	Section 5.18
0218h	DOORBELL1_ICCR	DOORBELL Interrupt Condition Clear Register 1	Section 5.19
0220h	DOORBELL2_ICSR	DOORBELL Interrupt Condition Status Register 2	Section 5.18
0228h	DOORBELL2_ICCR	DOORBELL Interrupt Condition Clear Register 2	Section 5.19
0230h	DOORBELL3_ICSR	DOORBELL Interrupt Condition Status Register 3	Section 5.18
0238h	DOORBELL3_ICCR	DOORBELL Interrupt Condition Clear Register 3	Section 5.19
0240h	RX_CPPI_ICSR	RX CPPI Interrupt Condition Status Register	Section 5.20
0248h	RX_CPPI_ICCR	RX CPPI Interrupt Condition Clear Register	Section 5.21
0250h	TX_CPPI_ICSR	TX CPPI Interrupt Condition Status Register	Section 5.22
0258h	TX_CPPI_ICCR	TX CPPI Interrupt Condition Clear Register	Section 5.23
0260h	LSU_ICSR	LSU Interrupt Condition Status Register	Section 5.24
0268h	LSU_ICCR	LSU Interrupt Condition Clear Register	Section 5.25
0270h	ERR_RST_EVNT_ICSR	Error, Reset, and Special Event Interrupt Condition Status Register	Section 5.26
0278h	ERR_RST_EVNT_ICCR	Error, Reset, and Special Event Interrupt Condition Clear Register	Section 5.27
0280h	DOORBELL0_ICRR	DOORBELL0 Interrupt Condition Routing Register	Section 5.28
0284h	DOORBELL0_ICRR2	DOORBELL 0 Interrupt Condition Routing Register 2	Section 5.28
0290h	DOORBELL1_ICRR	DOORBELL1 Interrupt Condition Routing Register	Section 5.28
0294h	DOORBELL1_ICRR2	DOORBELL 1 Interrupt Condition Routing Register 2	Section 5.28
02A0h	DOORBELL2_ICRR	DOORBELL2 Interrupt Condition Routing Register	Section 5.28
02A4h	DOORBELL2_ICRR2	DOORBELL 2 Interrupt Condition Routing Register 2	Section 5.28
02B0h	DOORBELL3_ICRR	DOORBELL3 Interrupt Condition Routing Register	Section 5.28
02B4h	DOORBELL3_ICRR2	DOORBELL 3 Interrupt Condition Routing Register 2	Section 5.28
02C0h	RX_CPPI _ICRR	Receive CPPI Interrupt Condition Routing Register	Section 5.29
02C4h	RX_CPPI _ICRR2	Receive CPPI Interrupt Condition Routing Register 2	Section 5.29
02D0h	TX_CPPI _ICRR	Transmit CPPI Interrupt Condition Routing Register	Section 5.30
02D4h	TX_CPPI _ICRR2	Transmit CPPI Interrupt Condition Routing Register 2	Section 5.30
02E0h	LSU_ICRR0	LSU Interrupt Condition Routing Register 0	Section 5.31
02E4h	LSU_ICRR1	LSU Interrupt Condition Routing Register 1	Section 5.31
02E8h	LSU_ICRR2	LSU Interrupt Condition Routing Register 2	Section 5.31
02ECh	LSU_ICRR3	LSU Interrupt Condition Routing Register 3	Section 5.31
02F0h	ERR_RST_EVNT_ICRR	Error, Reset, and Special Event Interrupt Condition Routing Register	Section 5.32
02F4h	ERR_RST_EVNT_ICRR2	Error, Reset, and Special Event Interrupt Condition Routing Register 2	Section 5.32
02F8h	ERR_RST_EVNT_ICRR3	Error, Reset, and Special Event Interrupt Condition Routing Register 3	Section 5.32
0300h	INTDST0_DECODE	INTDST Interrupt Status Decode Register 0	Section 5.33
0304h	INTDST1_DECODE	INTDST Interrupt Status Decode Register 1	Section 5.33
0308h	INTDST2_DECODE	INTDST Interrupt Status Decode Register 2	Section 5.33
030Ch	INTDST3_DECODE	INTDST Interrupt Status Decode Register 3	Section 5.33

Table 42. SRIO Registers (continued)

Offset	Acronym	Register Description	See
0310h	INTDST4_DECODE	INTDST Interrupt Status Decode Register 4	Section 5.33
0314h	INTDST5_DECODE	INTDST Interrupt Status Decode Register 5	Section 5.33
0318h	INTDST6_DECODE	INTDST Interrupt Status Decode Register 6	Section 5.33
031Ch	INTDST7_DECODE	INTDST Interrupt Status Decode Register 7	Section 5.33
0320h	INTDST0_RATE_CNTL	INTDST Interrupt Rate Control Register 0	Section 5.34
0324h	INTDST1_RATE_CNTL	INTDST Interrupt Rate Control Register 1	Section 5.34
0328h	INTDST2_RATE_CNTL	INTDST Interrupt Rate Control Register 2	Section 5.34
032Ch	INTDST3_RATE_CNTL	INTDST Interrupt Rate Control Register 3	Section 5.34
0330h	INTDST4_RATE_CNTL	INTDST Interrupt Rate Control Register 4	Section 5.34
0334h	INTDST5_RATE_CNTL	INTDST Interrupt Rate Control Register 5	Section 5.34
0338h	INTDST6_RATE_CNTL	INTDST Interrupt Rate Control Register 6	Section 5.34
033Ch	INTDST7_RATE_CNTL	INTDST Interrupt Rate Control Register 7	Section 5.34
0400h	LSU1_REG0	LSU1 Control Register 0	Section 5.35
0404h	LSU1_REG1	LSU1 Control Register 1	Section 5.36
0408h	LSU1_REG2	LSU1 Control Register 2	Section 5.37
040Ch	LSU1_REG3	LSU1 Control Register 3	Section 5.38
0410h	LSU1_REG4	LSU1 Control Register 4	Section 5.39
0414h	LSU1_REG5	LSU1 Control Register 5	Section 5.40
0418h	LSU1_REG6	LSU1 Control Register 6	Section 5.41
041Ch	LSU1_FLOW_MASKS	LSU1 Congestion Control Flow Mask Register	Section 5.42
0420h	LSU2_REG0	LSU2 Control Register 0	Section 5.35
0424h	LSU2_REG1	LSU2 Control Register 1	Section 5.36
0428h	LSU2_REG2	LSU2 Control Register 2	Section 5.37
042Ch	LSU2_REG3	LSU2 Control Register 3	Section 5.38
0430h	LSU2_REG4	LSU2 Control Register 4	Section 5.39
0434h	LSU2_REG5	LSU2 Control Register 5	Section 5.40
0438h	LSU2_REG6	LSU2 Control Register 6	Section 5.41
043Ch	LSU2_FLOW_MASKS1	LSU2 Congestion Control Flow Mask Register	Section 5.42
0440h	LSU3_REG0	LSU3 Control Register 0	Section 5.35
0444h	LSU3_REG1	LSU3 Control Register 1	Section 5.36
0448h	LSU3_REG2	LSU3 Control Register 2	Section 5.37
044Ch	LSU3_REG3	LSU3 Control Register 3	Section 5.38
0450h	LSU3_REG4	LSU3 Control Register 4	Section 5.39
0454h	LSU3_REG5	LSU3 Control Register 5	Section 5.40
0458h	LSU3_REG6	LSU3 Control Register 6	Section 5.41
045Ch	LSU3_FLOW_MASKS2	LSU3 Congestion Control Flow Mask Register	Section 5.42
0460h	LSU4_REG0	LSU4 Control Register 0	Section 5.35
0464h	LSU4_REG1	LSU4 Control Register 1	Section 5.36
0468h	LSU4_REG2	LSU4 Control Register 2	Section 5.37
046Ch	LSU4_REG3	LSU4 Control Register 3	Section 5.38
0470h	LSU4_REG4	LSU4 Control Register 4	Section 5.39
0474h	LSU4_REG5	LSU4 Control Register 5	Section 5.40
0478h	LSU4_REG6	LSU4 Control Register 6	Section 5.41
047Ch	LSU4_FLOW_MASKS3	LSU4 Congestion Control Flow Mask Register	Section 5.42
0500h	QUEUE0_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 0	Section 5.43
0504h	QUEUE1_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 1	Section 5.43

Table 42. SRIO Registers (continued)

Offset	Acronym	Register Description	See
0508h	QUEUE2_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 2	Section 5.43
050Ch	QUEUE3_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 3	Section 5.43
0510h	QUEUE4_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 4	Section 5.43
0514h	QUEUE5_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 5	Section 5.43
0518h	QUEUE6_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 6	Section 5.43
051Ch	QUEUE7_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 7	Section 5.43
0520h	QUEUE8_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 8	Section 5.43
0524h	QUEUE9_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 9	Section 5.43
0528h	QUEUE10_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 10	Section 5.43
052Ch	QUEUE11_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 11	Section 5.43
0530h	QUEUE12_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 12	Section 5.43
0534h	QUEUE13_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 13	Section 5.43
0538h	QUEUE14_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 14	Section 5.43
053Ch	QUEUE15_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 15	Section 5.43
0580h	QUEUE0_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 0	Section 5.44
0584h	QUEUE1_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 1	Section 5.44
0588h	QUEUE2_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 2	Section 5.44
058Ch	QUEUE3_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 3	Section 5.44
0590h	QUEUE4_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 4	Section 5.44
0594h	QUEUE5_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 5	Section 5.44
0598h	QUEUE6_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 6	Section 5.44
059Ch	QUEUE7_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 7	Section 5.44
05A0h	QUEUE8_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 8	Section 5.44
05A4h	QUEUE9_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 9	Section 5.44
05A8h	QUEUE10_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 10	Section 5.44
05ACh	QUEUE11_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 11	Section 5.44
05B0h	QUEUE12_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 12	Section 5.44
05B4h	QUEUE13_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 13	Section 5.44
05B8h	QUEUE14_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 14	Section 5.44
05BCh	QUEUE15_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 15	Section 5.44
0600h	QUEUE0_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 0	Section 5.45
0604h	QUEUE1_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 1	Section 5.45
0608h	QUEUE2_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 2	Section 5.45
060Ch	QUEUE3_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 3	Section 5.45

Table 42. SRIO Registers (continued)

Offset	Acronym	Register Description	See
0610h	QUEUE4_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 4	Section 5.45
0614h	QUEUE5_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 5	Section 5.45
0618h	QUEUE6_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 6	Section 5.45
061Ch	QUEUE7_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 7	Section 5.45
0620h	QUEUE8_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 8	Section 5.45
0624h	QUEUE9_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 9	Section 5.45
0628h	QUEUE10_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 10	Section 5.45
062Ch	QUEUE11_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 11	Section 5.45
0630h	QUEUE12_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 12	Section 5.45
0634h	QUEUE13_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 13	Section 5.45
0638h	QUEUE14_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 14	Section 5.45
063Ch	QUEUE15_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 15	Section 5.45
0680h	QUEUE0_RXDMA_CP	Queue Receive DMA Completion Pointer Register 0	Section 5.46
0684h	QUEUE1_RXDMA_CP	Queue Receive DMA Completion Pointer Register 1	Section 5.46
0688h	QUEUE2_RXDMA_CP	Queue Receive DMA Completion Pointer Register 2	Section 5.46
068Ch	QUEUE3_RXDMA_CP	Queue Receive DMA Completion Pointer Register 3	Section 5.46
0690h	QUEUE4_RXDMA_CP	Queue Receive DMA Completion Pointer Register 4	Section 5.46
0694h	QUEUE5_RXDMA_CP	Queue Receive DMA Completion Pointer Register 5	Section 5.46
0698h	QUEUE6_RXDMA_CP	Queue Receive DMA Completion Pointer Register 6	Section 5.46
069Ch	QUEUE7_RXDMA_CP	Queue Receive DMA Completion Pointer Register 7	Section 5.46
06A0h	QUEUE8_RXDMA_CP	Queue Receive DMA Completion Pointer Register 8	Section 5.46
06A4h	QUEUE9_RXDMA_CP	Queue Receive DMA Completion Pointer Register 9	Section 5.46
06A8h	QUEUE10_RXDMA_CP	Queue Receive DMA Completion Pointer Register 10	Section 5.46
06ACh	QUEUE11_RXDMA_CP	Queue Receive DMA Completion Pointer Register 11	Section 5.46
06B0h	QUEUE12_RXDMA_CP	Queue Receive DMA Completion Pointer Register 12	Section 5.46
06B4h	QUEUE13_RXDMA_CP	Queue Receive DMA Completion Pointer Register 13	Section 5.46
06B8h	QUEUE14_RXDMA_CP	Queue Receive DMA Completion Pointer Register 14	Section 5.46
06BCh	QUEUE15_RXDMA_CP	Queue Receive DMA Completion Pointer Register 15	Section 5.46
0700h	TX_QUEUE_TEAR_DOWN	Transmit Queue Teardown Register	Section 5.47
0704h	TX_CPPI_FLOW_MASKS0	Transmit CPPI Supported Flow Mask Register 0	Section 5.48
0708h	TX_CPPI_FLOW_MASKS1	Transmit CPPI Supported Flow Mask Register 1	Section 5.48
070Ch	TX_CPPI_FLOW_MASKS2	Transmit CPPI Supported Flow Mask Register 2	Section 5.48
0710h	TX_CPPI_FLOW_MASKS3	Transmit CPPI Supported Flow Mask Register 3	Section 5.48
0714h	TX_CPPI_FLOW_MASKS4	Transmit CPPI Supported Flow Mask Register 4	Section 5.48
0718h	TX_CPPI_FLOW_MASKS5	Transmit CPPI Supported Flow Mask Register 5	Section 5.48
071Ch	TX_CPPI_FLOW_MASKS6	Transmit CPPI Supported Flow Mask Register 6	Section 5.48
0720h	TX_CPPI_FLOW_MASKS7	Transmit CPPI Supported Flow Mask Register 7	Section 5.48
0740h	RX_QUEUE_TEAR_DOWN	Receive Queue Teardown Register	Section 5.49
0744h	RX_CPPI_CNTL	Receive CPPI Control Register	Section 5.50

Table 42. SRIO Registers (continued)

Offset	Acronym	Register Description	See
07E0h	TX_QUEUE_CNTL0	Transmit CPPI Weighted Round Robin Control Register 0	Section 5.51
07E4h	TX_QUEUE_CNTL1	Transmit CPPI Weighted Round Robin Control Register 1	Section 5.51
07E8h	TX_QUEUE_CNTL2	Transmit CPPI Weighted Round Robin Control Register 2	Section 5.51
07ECh	TX_QUEUE_CNTL3	Transmit CPPI Weighted Round Robin Control Register 3	Section 5.51
0800h	RXU_MAP_L0	MailBox-to-Queue Mapping Register L0	Section 5.52
0804h	RXU_MAP_H0	MailBox-to-Queue Mapping Register H0	Section 5.52
0808h	RXU_MAP_L1	MailBox-to-Queue Mapping Register L1	Section 5.52
080Ch	RXU_MAP_H1	MailBox-to-Queue Mapping Register H1	Section 5.52
0810h	RXU_MAP_L2	MailBox-to-Queue Mapping Register L2	Section 5.52
0814h	RXU_MAP_H2	MailBox-to-Queue Mapping Register H2	Section 5.52
0818h	RXU_MAP_L3	MailBox-to-Queue Mapping Register L3	Section 5.52
081Ch	RXU_MAP_H3	MailBox-to-Queue Mapping Register H3	Section 5.52
0820h	RXU_MAP_L4	MailBox-to-Queue Mapping Register L4	Section 5.52
0824h	RXU_MAP_H4	MailBox-to-Queue Mapping Register H4	Section 5.52
0828h	RXU_MAP_L5	MailBox-to-Queue Mapping Register L5	Section 5.52
082Ch	RXU_MAP_H5	MailBox-to-Queue Mapping Register H5	Section 5.52
0830h	RXU_MAP_L6	MailBox-to-Queue Mapping Register L6	Section 5.52
0834h	RXU_MAP_H6	MailBox-to-Queue Mapping Register H6	Section 5.52
0838h	RXU_MAP_L7	MailBox-to-Queue Mapping Register L7	Section 5.52
083Ch	RXU_MAP_H7	MailBox-to-Queue Mapping Register H7	Section 5.52
0840h	RXU_MAP_L8	MailBox-to-Queue Mapping Register L8	Section 5.52
0844h	RXU_MAP_H8	MailBox-to-Queue Mapping Register H8	Section 5.52
0848h	RXU_MAP_L9	MailBox-to-Queue Mapping Register L9	Section 5.52
084Ch	RXU_MAP_H9	MailBox-to-Queue Mapping Register H9	Section 5.52
0850h	RXU_MAP_L10	MailBox-to-Queue Mapping Register L10	Section 5.52
0854h	RXU_MAP_H10	MailBox-to-Queue Mapping Register H10	Section 5.52
0858h	RXU_MAP_L11	MailBox-to-Queue Mapping Register L11	Section 5.52
085Ch	RXU_MAP_H11	MailBox-to-Queue Mapping Register H11	Section 5.52
0860h	RXU_MAP_L12	MailBox-to-Queue Mapping Register L12	Section 5.52
0864h	RXU_MAP_H12	MailBox-to-Queue Mapping Register H12	Section 5.52
0868h	RXU_MAP_L13	MailBox-to-Queue Mapping Register L13	Section 5.52
086Ch	RXU_MAP_H13	MailBox-to-Queue Mapping Register H13	Section 5.52
0870h	RXU_MAP_L14	MailBox-to-Queue Mapping Register L14	Section 5.52
0874h	RXU_MAP_H14	MailBox-to-Queue Mapping Register H14	Section 5.52
0878h	RXU_MAP_L15	MailBox-to-Queue Mapping Register L15	Section 5.52
087Ch	RXU_MAP_H15	MailBox-to-Queue Mapping Register H15	Section 5.52
0880h	RXU_MAP_L16	MailBox-to-Queue Mapping Register L16	Section 5.52
0884h	RXU_MAP_H16	MailBox-to-Queue Mapping Register H16	Section 5.52
0888h	RXU_MAP_L17	MailBox-to-Queue Mapping Register L17	Section 5.52
088Ch	RXU_MAP_H17	MailBox-to-Queue Mapping Register H17	Section 5.52
0890h	RXU_MAP_L18	MailBox-to-Queue Mapping Register L18	Section 5.52
0894h	RXU_MAP_H18	MailBox-to-Queue Mapping Register H18	Section 5.52
0898h	RXU_MAP_L19	MailBox-to-Queue Mapping Register L19	Section 5.52
089Ch	RXU_MAP_H19	MailBox-to-Queue Mapping Register H19	Section 5.52
08A0h	RXU_MAP_L20	MailBox-to-Queue Mapping Register L20	Section 5.52

Table 42. SRIO Registers (continued)

Offset	Acronym	Register Description	See
08A4h	RXU_MAP_H20	MailBox-to-Queue Mapping Register H20	Section 5.52
08A8h	RXU_MAP_L21	MailBox-to-Queue Mapping Register L21	Section 5.52
08ACh	RXU_MAP_H21	MailBox-to-Queue Mapping Register H21	Section 5.52
08B0h	RXU_MAP_L22	MailBox-to-Queue Mapping Register L22	Section 5.52
08B4h	RXU_MAP_H22	MailBox-to-Queue Mapping Register H22	Section 5.52
08B8h	RXU_MAP_L23	MailBox-to-Queue Mapping Register L23	Section 5.52
08BCh	RXU_MAP_H23	MailBox-to-Queue Mapping Register H23	Section 5.52
08C0h	RXU_MAP_L24	MailBox-to-Queue Mapping Register L24	Section 5.52
08C4h	RXU_MAP_H24	MailBox-to-Queue Mapping Register H24	Section 5.52
08C8h	RXU_MAP_L25	MailBox-to-Queue Mapping Register L25	Section 5.52
08CCh	RXU_MAP_H25	MailBox-to-Queue Mapping Register H25	Section 5.52
08D0h	RXU_MAP_L26	MailBox-to-Queue Mapping Register L26	Section 5.52
08D4h	RXU_MAP_H26	MailBox-to-Queue Mapping Register H26	Section 5.52
08D8h	RXU_MAP_L27	MailBox-to-Queue Mapping Register L27	Section 5.52
08DCh	RXU_MAP_H27	MailBox-to-Queue Mapping Register H27	Section 5.52
08E0h	RXU_MAP_L28	MailBox-to-Queue Mapping Register L28	Section 5.52
08E4h	RXU_MAP_H28	MailBox-to-Queue Mapping Register H28	Section 5.52
08E8h	RXU_MAP_L29	MailBox-to-Queue Mapping Register L29	Section 5.52
08ECh	RXU_MAP_H29	MailBox-to-Queue Mapping Register H29	Section 5.52
08F0h	RXU_MAP_L30	MailBox-to-Queue Mapping Register L30	Section 5.52
08F4h	RXU_MAP_H30	MailBox-to-Queue Mapping Register H30	Section 5.52
08F8h	RXU_MAP_L31	MailBox-to-Queue Mapping Register L31	Section 5.52
08FCh	RXU_MAP_H31	MailBox-to-Queue Mapping Register H31	Section 5.52
0900h	FLOW_CNTL0	Flow Control Table Entry Register 0	Section 5.53
0904h	FLOW_CNTL1	Flow Control Table Entry Register 1	Section 5.53
0908h	FLOW_CNTL2	Flow Control Table Entry Register 2	Section 5.53
090Ch	FLOW_CNTL3	Flow Control Table Entry Register 3	Section 5.53
0910h	FLOW_CNTL4	Flow Control Table Entry Register 4	Section 5.53
0914h	FLOW_CNTL5	Flow Control Table Entry Register 5	Section 5.53
0918h	FLOW_CNTL6	Flow Control Table Entry Register 6	Section 5.53
091Ch	FLOW_CNTL7	Flow Control Table Entry Register 7	Section 5.53
0920h	FLOW_CNTL8	Flow Control Table Entry Register 8	Section 5.53
0924h	FLOW_CNTL9	Flow Control Table Entry Register 9	Section 5.53
0928h	FLOW_CNTL10	Flow Control Table Entry Register 10	Section 5.53
092Ch	FLOW_CNTL11	Flow Control Table Entry Register 11	Section 5.53
0930h	FLOW_CNTL12	Flow Control Table Entry Register 12	Section 5.53
0934h	FLOW_CNTL13	Flow Control Table Entry Register 13	Section 5.53
0938h	FLOW_CNTL14	Flow Control Table Entry Register 14	Section 5.53
093Ch	FLOW_CNTL15	Flow Control Table Entry Register 15	Section 5.53
1000h	DEV_ID	Device Identity CAR	Section 5.54
1004h	DEV_INFO	Device Information CAR	Section 5.55
1008h	ASBLY_ID	Assembly Identity CAR	Section 5.56
100Ch	ASBLY_INFO	Assembly Information CAR	Section 5.57
1010h	PE_FEAT	Processing Element Features CAR	Section 5.58
1018h	SRC_OP	Source Operations CAR	Section 5.59
101Ch	DEST_OP	Destination Operations CAR	Section 5.60
104Ch	PE_LL_CTL	Processing Element Logical Layer Control CSR	Section 5.61

Table 42. SRIO Registers (continued)

Offset	Acronym	Register Description	See
1058h	LCL_CFG_HBAR	Local Configuration Space Base Address 0 CSR	Section 5.62
105Ch	LCL_CFG_BAR	Local Configuration Space Base Address 1 CSR	Section 5.63
1060h	BASE_ID	Base Device ID CSR	Section 5.64
1068h	HOST_BASE_ID_LOCK	Host Base Device ID Lock CSR	Section 5.65
106Ch	COMP_TAG	Component Tag CSR	Section 5.66
1100h	SP_MB_HEAD	1x/4x LP_Serial Port Maintenance Block Header	Section 5.67
1120h	SP_LT_CTL	Port Link Time-Out Control CSR	Section 5.68
1124h	SP_RT_CTL	Port Response Time-Out Control CSR	Section 5.69
113Ch	SP_GEN_CTL	Port General Control CSR	Section 5.70
1140h	SP0_LM_REQ	Port 0 Link Maintenance Request CSR	Section 5.71
1144h	SP0_LM_RESP	Port 0 Link Maintenance Response CSR	Section 5.72
1148h	SP0_ACKID_STAT	Port 0 Local AckID Status CSR	Section 5.73
1158h	SP0_ERR_STAT	Port 0 Error and Status CSR	Section 5.74
115Ch	SP0_CTL	Port 0 Control CSR	Section 5.75
1160h	SP1_LM_REQ	Port 1 Link Maintenance Request CSR	Section 5.71
1164h	SP1_LM_RESP	Port 1 Link Maintenance Response CSR	Section 5.72
1168h	SP1_ACKID_STAT	Port 1 Local AckID Status CSR	Section 5.73
1178h	SP1_ERR_STAT	Port 1 Error and Status CSR	Section 5.74
117Ch	SP1_CTL	Port 1 Control CSR	Section 5.75
1180h	SP2_LM_REQ	Port 2 Link Maintenance Request CSR	Section 5.71
1184h	SP2_LM_RESP	Port 2 Link Maintenance Response CSR	Section 5.72
1188h	SP2_ACKID_STAT	Port 2 Local AckID Status CSR	Section 5.73
1198h	SP2_ERR_STAT	Port 2 Error and Status CSR	Section 5.74
119Ch	SP2_CTL	Port 2 Control CSR	Section 5.75
11A0h	SP3_LM_REQ	Port 3 Link Maintenance Request CSR	Section 5.71
11A4h	SP3_LM_RESP	Port 3 Link Maintenance Response CSR	Section 5.72
11A8h	SP3_ACKID_STAT	Port 3 Local AckID Status CSR	Section 5.73
11B8h	SP3_ERR_STAT	Port 3 Error and Status CSR	Section 5.74
11BCh	SP3_CTL	Port 3 Control CSR	Section 5.75
2000h	ERR_RPT_BH	Error Reporting Block Header	Section 5.76
2008h	ERR_DET	Logical/Transport Layer Error Detect CSR	Section 5.77
200Ch	ERR_EN	Logical/Transport Layer Error Enable CSR	Section 5.78
2010h	H_ADDR_CAPT	Logical/Transport Layer High Address Capture CSR	Section 5.79
2014h	ADDR_CAPT	Logical/Transport Layer Address Capture CSR	Section 5.80
2018h	ID_CAPT	Logical/Transport Layer Device ID Capture CSR	Section 5.81
201Ch	CTRL_CAPT	Logical/Transport Layer Control Capture CSR	Section 5.82
2028h	PW_TGT_ID	Port-Write Target Device ID CSR	Section 5.83
2040h	SP0_ERR_DET	Port 0 Error Detect CSR	Section 5.84
2044h	SP0_RATE_EN	Port 0 Error Enable CSR	Section 5.85
2048h	SP0_ERR_ATTR_CAPT_DBG0	Port 0 Attributes Error Capture CSR 0	Section 5.86
204Ch	SP0_ERR_CAPT_DBG1	Port 0 Packet/Control Symbol Error Capture CSR 1	Section 5.87
2050h	SP0_ERR_CAPT_DBG2	Port 0 Packet/Control Symbol Error Capture CSR 2	Section 5.88
2054h	SP0_ERR_CAPT_DBG3	Port 0 Packet/Control Symbol Error Capture CSR 3	Section 5.89
2058h	SP0_ERR_CAPT_DBG4	Port 0 Packet/Control Symbol Error Capture CSR 4	Section 5.90
2068h	SP0_ERR_RATE	Port 0 Error Rate CSR 0	Section 5.91
206Ch	SP0_ERR_THRESH	Port 0 Error Rate Threshold CSR	Section 5.92
2080h	SP1_ERR_DET	Port 1 Error Detect CSR	Section 5.84

Table 42. SRIO Registers (continued)

Offset	Acronym	Register Description	See
2084h	SP1_RATE_EN	Port 1 Error Enable CSR	Section 5.85
2088h	SP1_ERR_ATTR_CAPT_DBG0	Port 1 Attributes Error Capture CSR 0	Section 5.86
208Ch	SP1_ERR_CAPT_DBG1	Port 1 Packet/Control Symbol Error Capture CSR 1	Section 5.87
2090h	SP1_ERR_CAPT_DBG2	Port 1 Packet/Control Symbol Error Capture CSR 2	Section 5.88
2094h	SP1_ERR_CAPT_DBG3	Port 1 Packet/Control Symbol Error Capture CSR 3	Section 5.89
2098h	SP1_ERR_CAPT_DBG4	Port 1 Packet/Control Symbol Error Capture CSR 4	Section 5.90
20A8h	SP1_ERR_RATE	Port 1 Error Rate CSR	Section 5.91
20ACh	SP1_ERR_THRESH	Port 1 Error Rate Threshold CSR	Section 5.92
20C0h	SP2_ERR_DET	Port 2 Error Detect CSR	Section 5.84
20C4h	SP2_RATE_EN	Port 2 Error Enable CSR	Section 5.85
20C8h	SP2_ERR_ATTR_CAPT_DBG0	Port 2 Attributes Error Capture CSR 0	Section 5.86
20CCh	SP2_ERR_CAPT_DBG1	Port 2 Packet/Control Symbol Error Capture CSR 1	Section 5.87
20D0h	SP2_ERR_CAPT_DBG2	Port 2 Packet/Control Symbol Error Capture CSR 2	Section 5.88
20D4h	SP2_ERR_CAPT_DBG3	Port 2 Packet/Control Symbol Error Capture CSR 3	Section 5.89
20D8h	SP2_ERR_CAPT_DBG4	Port 2 Packet/Control Symbol Error Capture CSR 4	Section 5.90
20E8h	SP2_ERR_RATE	Port 2 Error Rate CSR	Section 5.91
20ECh	SP2_ERR_THRESH	Port 2 Error Rate Threshold CSR	Section 5.92
2100h	SP3_ERR_DET	Port 3 Error Detect CSR	Section 5.84
2104h	SP3_RATE_EN	Port 3 Error Enable CSR	Section 5.85
2108h	SP3_ERR_ATTR_CAPT_DBG0	Port 3 Attributes Error Capture CSR 0	Section 5.86
210Ch	SP3_ERR_CAPT_DBG1	Port 3 Packet/Control Symbol Error Capture CSR 1	Section 5.87
2110h	SP3_ERR_CAPT_DBG2	Port 3 Packet/Control Symbol Error Capture CSR 2	Section 5.88
2114h	SP3_ERR_CAPT_DBG3	Port 3 Packet/Control Symbol Error Capture CSR 3	Section 5.89
2118h	SP3_ERR_CAPT_DBG4	Port 3 Packet/Control Symbol Error Capture CSR 4	Section 5.90
2128h	SP3_ERR_RATE	Port 3 Error Rate CSR	Section 5.91
212Ch	SP3_ERR_THRESH	Port 3 Error Rate Threshold CSR	Section 5.92
12000h	SP_IP_DISCOVERY_TIMER	Port IP Discovery Timer in 4x mode	Section 5.93
12004h	SP_IP_MODE	Port IP Mode CSR	Section 5.94
12008h	IP_PRESCAL	Port IP Prescaler Register	Section 5.95
12010h	SP_IP_PW_IN_CAPT0	Port-Write-In Capture CSR Register 0	Section 5.96
12014h	SP_IP_PW_IN_CAPT1	Port-Write-In Capture CSR Register 1	Section 5.96
12018h	SP_IP_PW_IN_CAPT2	Port-Write-In Capture CSR Register 2	Section 5.96
1201Ch	SP_IP_PW_IN_CAPT3	Port-Write-In Capture CSR Register 3	Section 5.96
14000h	SP0_RST_OPT	Port 0 Reset Option CSR	Section 5.97
14004h	SP0_CTL_INDEP	Port 0 Control Independent Register	Section 5.98
14008h	SP0_SILENCE_TIMER	Port 0 Silence Timer Register	Section 5.99
1400Ch	SP0_MULT_EVNT_CS	Port 0 Multicast-Event Control Symbol Request Register	Section 5.100
14014h	SP0_CS_TX	Port 0 Control Symbol Transmit Register	Section 5.101
14100h	SP1_RST_OPT	Port 1 Reset Option CSR	Section 5.97
14104h	SP1_CTL_INDEP	Port 1 Control Independent Register	Section 5.98
14108h	SP1_SILENCE_TIMER	Port 1 Silence Timer Register	Section 5.99
1410Ch	SP1_MULT_EVNT_CS	Port 1 Multicast-Event Control Symbol Request Register	Section 5.100
14114h	SP1_CS_TX	Port 1 Control Symbol Transmit Register	Section 5.101
14200h	SP2_RST_OPT	Port 2 Reset Option CSR	Section 5.97
14204h	SP2_CTL_INDEP	Port 2 Control Independent Register	Section 5.98
14208h	SP2_SILENCE_TIMER	Port 2 Silence Timer Register	Section 5.99

Table 42. SRIO Registers (continued)

Offset	Acronym	Register Description	See
1420Ch	SP2_MULT_EVNT_CS	Port 2 Multicast-Event Control Symbol Request Register	Section 5.100
14214h	SP2_CS_TX	Port 2 Control Symbol Transmit Register	Section 5.101
14300h	SP3_RST_OPT	Port 3 Reset Option CSR	Section 5.97
14304h	SP3_CTL_INDEP	Port 3 Control Independent Register	Section 5.98
14308h	SP3_SILENCE_TIMER	Port 3 Silence Timer Register	Section 5.99
1430Ch	SP3_MULT_EVNT_CS	Port 3 Multicast-Event Control Symbol Request Register	Section 5.100
14314h	SP3_CS_TX	Port 3 Control Symbol Transmit Register	Section 5.101

5.1 Peripheral Identification Register (PID)

The peripheral identification register (PID) is a read-only register that contains the ID and ID revision number for that peripheral. The PID stores version information used to identify the peripheral. Writes have no effect to this register. The values are hard coded and will not change from their reset state.

The peripheral ID register (PID) is shown in Figure 60 and described in Table 43.

Figure 60. Peripheral ID Register (PID) - Address Offset 0000h

31	30	29	28	27	16
SCHEME		Reserved		FUNC	
R-1h		R-0h		(A)	

15	11	10	8	7	6	5	0
RTL		MAJOR		CUSTOM		MINOR	
(B)		R-1h		R-0h		R-02h	

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset
A. Reset value is R-4A7h for TCI6484 device; R-4A2h for TCI6486/C6472 devices; R-4A4h for TCI6487/88 devices.
B. Reset value is R-60h for TCI6484 device; R-30h for TCI6486/87/88/C6472 devices.

Table 43. Peripheral ID Register (PID) Field Descriptions

Bit	Field	Value	Description
31-30	SCHEME		Peripheral scheme field. Fixed to 0x1.
29-28	Reserved		Reserved field
27-16	FUNC		Peripheral Functional class.
15-11	RTL		RTL revision ID
10-8	MAJOR		Major revision ID
7-6	CUSTOM		Custom revision ID
5-0	MINOR		Minor revision ID

5.2 Peripheral Control Register (PCR)

The peripheral control register (PCR) contains a bit that enables or disables data flow in the logical layer of the entire peripheral. In addition, the PCR has emulation control bits that control the peripheral behavior during emulation halts. PCR is shown in Figure 61 and described in Table 44. For additional programming information, see Section 2.3.11.

Figure 61. Peripheral Control Register (PCR) - Address Offset 0004h

31	16
Reserved	
R-0	

15	3	2	1	0
Reserved		PEREN	SOFT	FREE
R-0		R/W-0	R/W-0	R/W-1

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 44. Peripheral Control Register (PCR) Field Descriptions

Bit	Field	Value	Description
31-3	Reserved	0	These read-only bits return 0s when read.
2	PEREN		Peripheral flow control enable. Controls the flow of data in the logical layer of the peripheral. As an initiator, it will prevent TX transaction generation; as a target, it will disable incoming requests. This should be the last enable bit to toggle when bringing the device out of reset to begin normal operation.
		0	Data flow control is disabled.
		1	Data flow control is enabled.

Table 44. Peripheral Control Register (PCR) Field Descriptions (continued)

Bit	Field	Value	Description
1	SOFT		Soft stop. This bit and the FREE bit determine how the SRIO peripheral behaves during emulation halts.
		0	Hard stop. All status registers are frozen in default state. (This mode is not supported on the SRIO peripheral.)
		1	Soft stop
0	FREE		Free run
		0	The SOFT bit takes effect.
		1	Free run. Peripheral ignores the emulation suspend signal and functions normally.

5.3 Peripheral Settings Control Register (PER_SET_CNTL)

The peripheral settings control register (PER_SET_CNTL) is shown in Figure 62 and described in Table 45. For additional programming information, see Section 2.3.12.

Figure 62. Peripheral Settings Control Register (PER_SET_CNTL) - Address Offset 0020h

31	30	29	28	27	26	25	24
LEND_SWAP_MODE		LEND_SWAP_MODE		LOG_TGT_ID_DIS	SW_MEM_SLEEP_OVERRIDE	LOOPBACK	BOOT_COMPLETE
R/W-0		R/W-0		R/W-0	R/W-1	R/W-0	R/W-0

23	22	21	20	18	17	16
LEND_SWAP_MODE		Reserved	TX_PRI2_WM		TX_PRI1_WM	
R/W-0		R-0	R/W-01h		R/W-02h	

15	14	12	11	9	8
TX_PRI1_WM	TX_PRI0_WM		CBA_TRANS_PRI		1X_MODE
R/W-02h	R/W-03h		R/W-0		R/W-1

7	4	3	2	1	0
PRESCALER_SELECT		ENPLL4	ENPLL3	ENPLL2	ENPLL1
R/W-0		R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 45. Peripheral Settings Control Register (PER_SET_CNTL) Field Descriptions

Bit	Field	Value	Description
31-30	LEND_SWAP_MODE		MAU little-endian swapping mode (See Section 2.3.9.2 for details.)
		00b	Mode A
		01b	Mode B
		10b	Mode C
		11b	Mode D
29-28	LEND_SWAP_MODE		LSU little-endian swapping mode (See Section 2.3.9.2 for details.)
		00b	Mode A
		01b	Mode B
		10b	Mode C
		11b	Mode D
27	LOG_TGT_ID_DIS		Logical layer disable. This bit disables all the packet types at the logical layer.
		0	All non-matching packets are destroyed.
		1	All packets, regardless of the destination ID, are forwarded to the application.
26	SW_MEM_SLEEP_OVERRIDE		Software memory sleep override
		0	Memories are put in sleep mode while in shutdown
		1	Memories are not put in sleep mode while in shutdown
25	LOOPBACK		Loopback mode
		0	Normal operation
		1	Loop back mode. Transmit data to receive on the same port. Packet data is looped back in the digital domain before the SERDES macros.

Table 45. Peripheral Settings Control Register (PER_SET_CNTL) Field Descriptions (continued)

Bit	Field	Value	Description
24	BOOT_COMPLETE		Controls ability to write any register during initialization. It also includes read only registers during normal mode of operation that have application defined reset value.
		0	Write to read-only registers enabled
		1	Write to read-only registers disabled. Usually the boot_complete is asserted once after reset to define power on configuration.
			As soon as the value becomes 1, it triggers the physical layer to start the state machine, which initiates the port initialization process. Once initialization has successfully completed and the port is exchanging control symbols and packets, the BOOT_COMPLETE signal has no effect on traffic. De-asserting it permits writing of normally read-only registers. The effect of BOOT_COMPLETE only becomes available when the reset is applied once again using the GBL_EN register.
23-22	LEND_SWAP_MODE		TXU/RXU little-endian swapping mode (See Section 2.3.9.2 for details.)
		00b	Mode A
		01b	Mode B
		10b	Mode C
		11b	Mode D
21	Reserved	000b	These read-only bits return 0s when read.
20-18	TX_PRI2_WM	000b-111b	Transmit credit threshold. Sets the required number of logical layer TX buffers needed to send priority 2 packets across the UDI. This is valid for all ports in 1x mode only. Required buffer count for transmit credit threshold 2 value (TX_PRI2_WM): • 000→8, 7, 6, 5, 4, 3, 2, 1 (effectively lets all of this priority pass) • 001→8, 7, 6, 5, 4, 3, 2 • 010→8, 7, 6, 5, 4, 3 • 011→8, 7, 6, 5, 4 • 100→8, 7, 6, 5 • 101→8, 7, 6 • 110→8, 7 • 111→8
17-15	TX_PRI1_WM	000b-111b	Transmit credit threshold. Sets the required number of logical layer TX buffers needed to send priority 1 packets across the UDI. This is valid for all ports in 1x mode only. Required buffer count for transmit credit threshold 1 value (TX_PRI1_WM): • 000→8, 7, 6, 5, 4, 3, 2, 1 (effectively lets all of this priority pass) • 001→8, 7, 6, 5, 4, 3, 2 • 010→8, 7, 6, 5, 4, 3 • 011→8, 7, 6, 5, 4 • 100→8, 7, 6, 5 • 101→8, 7, 6 • 110→8, 7 • 111→8

Table 45. Peripheral Settings Control Register (PER_SET_CNTL) Field Descriptions (continued)

Bit	Field	Value	Description
14-12	TX_PRI0_WM	000b-111b	Transmit credit threshold. Sets the required number of logical layer TX buffers needed to send priority 0 packets across the UDI. This is valid for all ports in 1x mode only. Required buffer count for transmit credit threshold 0 value (TX_PRI0_WM): • 000→8, 7, 6, 5, 4, 3, 2, 1 (effectively lets all of this priority pass) • 001→8, 7, 6, 5, 4, 3, 2 • 010→8, 7, 6, 5, 4, 3 • 011→8, 7, 6, 5, 4 • 100→8, 7, 6, 5 • 101→8, 7, 6 • 110→8, 7 • 111→8
11-9	CBA_TRANS_PRI	000b-111b	DSP system transaction priority. 000b - Highest Priority … 111b - Lowest Priority.
8	1X_MODE		This register bit determines the UDI buffering setup (priority versus port). For additional programming information, see Section 2.3.13.2.
		0	UDI buffers are priority based
		1	UDI buffers are port based. This mode must be selected when using more than one 1x port
7-4	PRESCALER_SELECT		Internal frequency prescaler, used to drive the request to response timers. These 4 bits are the prescaler reload value allowing division of the DMA clock by a range from 1 up to 16. Setting should reflect the device DMA frequency in MHz.
		0000b	Sets the internal clock frequency Min 44.7 and Max 89.5
		0001b	Sets the internal clock frequency Min 89.5 and Max 179.0
		0010b	Sets the internal clock frequency Min 134.2 and Max 268.4
		0011b	Sets the internal clock frequency Min 180.0 and Max 360.0
		0100b	Sets the internal clock frequency Min 223.7 and Max 447.4
		0101b	Sets the internal clock frequency Min 268.4 and Max 536.8
		0110b	Sets the internal clock frequency Min 313.2 and Max 626.4
		0111b	Sets the internal clock frequency Min 357.9 and Max 715.8
		1000b	sets the internal clock frequency Min 402.7 and Max 805.4
		1001b	Sets the internal clock frequency Min 447.4 and Max 894.8
		1010b	Sets the internal clock frequency Min 492.1 and Max 984.2
		1011b	Sets the internal clock frequency Min 536.9 and Max 1073.8
		1100b	Sets the internal clock frequency Min 581.6 and Max 1163.2
		1101b	Sets the internal clock frequency Min 626.3 and Max 1252.6
		1110b	Sets the internal clock frequency Min 671.1 and Max 1342.2
		1111b	Sets the internal clock frequency Min 715.8 and Max 1431.6
3	ENPLL4	0	Not used. Should always be programmed as "0". See Section 2.3.2.1 to enable SERDES PLL.
2	ENPLL3	0	Not used. Should always be programmed as "0". See Section 2.3.2.1 to enable SERDES PLL.
1	ENPLL2	0	Not used. Should always be programmed as "0". See Section 2.3.2.1 to enable SERDES PLL.
0	ENPLL1	0	Not used. Should always be programmed as "0". See Section 2.3.2.1 to enable SERDES PLL.

5.4 Peripheral Settings Control Register 1 (PER_SET_CNTL1)

PER_SET_CNTL1 is shown in Figure 63 and described in Table 46.

Figure 63. Peripheral Settings Control Register 1 (PER_SET_CNTL1) - Address Offset 0024h

31	24	23	16
TXU_MIDSEG_CREDIT_MULTI		TXU_1STSEG_CREDIT_MULTI	
R/W-01h		R/W-01h	

15	8	7	0
LSU_MIDSEG_CREDIT_MULTI		LSU_1STSEG_CREDIT_MULTI	
R/W-01h		R/W-01h	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

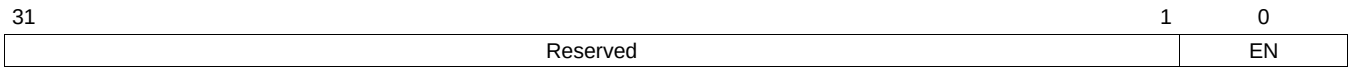
Table 46. Peripheral Settings Control Register 1 (PER_SET_CNTL1) Field Descriptions

Bit	Field	Value	Description
31-24	TXU_MIDSEG_CREDIT_MULTI		Multiplier for TXU credit attempts on middle packets of a transfer. Reset value is set to a multiply of 1, so credit will be attempted 1x64K times.
23-16	TXU_1STSEG_CREDIT_MULTI		Multiplier for TXU credit attempts on 1st packet of a transfer. Reset value is set to a multiply of 1, so credit will be attempted 1x256 times.
15-8	LSU_MIDSEG_CREDIT_MULTI		Multiplier for LSU credit attempts on middle packets of a transfer. Reset value is set to a multiply of 1, so credit will be attempted 1x64K times.
7-0	LSU_1STSEG_CREDIT_MULTI		Multiplier for LSU credit attempts on 1st packet of a transfer. Reset value is set a multiply of 1, so credit will be attempted 1x256 times.

5.5 Peripheral Global Enable Register (GBL_EN)

GBL_EN is implemented with a single enable bit for the entire SRIO peripheral. This bit is logically ORed with the reset input to the module and is fanned out to all logical blocks within the peripheral. GBL_EN is shown in Figure 64 and described in Table 47. For additional programming information, see Section 2.3.10.

Figure 64. Peripheral Global Enable Register (GBL_EN) - Address Offset 0030h



LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

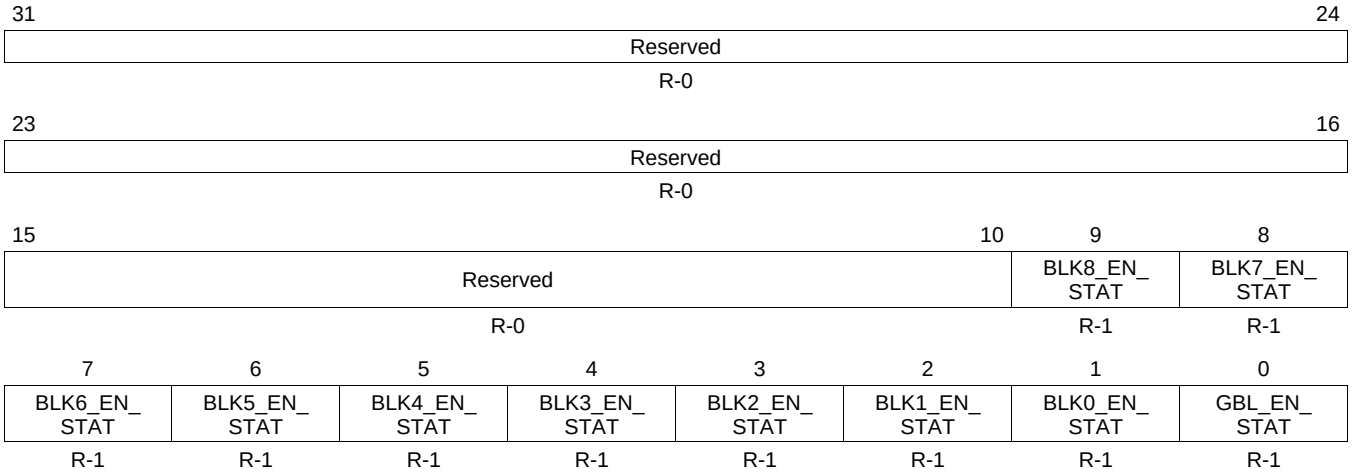
Table 47. Peripheral Global Enable Register (GBL_EN) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	00000000h	These read-only bits return 0s when read.
0	EN		Global enable. This bit controls reset to all clock domains within the peripheral.
		0	The peripheral is to be disabled (held in reset with clocks disabled).
		1	The peripheral is to be enabled.

5.6 Peripheral Global Enable Status Register (GBL_EN_STAT)

The peripheral global enable status register (GBL_EN_STAT) is shown in Figure 65 and described in Table 48. For additional programming information, see Section 2.3.10.

Figure 65. Peripheral Global Enable Status Register (GBL_EN_STAT) - Address Offset 0034h



LEGEND: R = Read only; -*n* = Value after reset

Table 48. Peripheral Global Enable Status Register (GBL_EN_STAT) Field Descriptions

Bit	Field	Value	Description
31-10	Reserved	0	These read-only bits return 0s when read.
9	BLK8_EN_STAT		Block 8 enable status. Logical block 8 is SRIO port 3.
		0	Logical block 8 is in reset with its clock off.
		1	Logical block 8 is enabled with its clock running.
8	BLK7_EN_STAT		Block 7 enable status. Logical block 7 is SRIO port 2.
		0	Logical block 7 is in reset with its clock off.
		1	Logical block 7 is enabled with its clock running.
7	BLK6_EN_STAT		Block 6 enable status. Logical block 6 is SRIO port 1.
		0	Logical block 6 is in reset with its clock off.
		1	Logical block 6 is enabled with its clock running.
6	BLK5_EN_STAT		Block 5 enable status. Logical block 5 is SRIO port 0.
		0	Logical block 5 is in reset with its clock off.
		1	Logical block 5 is enabled with its clock running.
5	BLK4_EN_STAT		Block 4 enable status. Logical block 4 is the message receive unit (RXU).
		0	Logical block 4 is in reset with its clock off.
		1	Logical block 4 is enabled with its clock running.
4	BLK3_EN_STAT		Block 3 enable status. Logical block 3 is the message transmit unit (TXU).
		0	Logical block 3 is in reset with its clock off.
		1	Logical block 3 is enabled with clock running.
3	BLK2_EN_STAT		Block 2 enable status. Logical block 2 is the memory access unit (MAU).
		0	Logical block 2 is in reset with its clock off.
		1	Logical block 2 is enabled with its clock running.
2	BLK1_EN_STAT		Block 1 enable status. Logical block 1 is the Load/Store module, which is comprised of the four Load/Store units (LSU1, LSU2, LSU3, and LSU4).
		0	Logical block 1 is in reset with its clock off.
		1	Logical block 1 is enabled with its clock running.

Table 48. Peripheral Global Enable Status Register (GBL_EN_STAT) Field Descriptions (continued)

Bit	Field	Value	Description
1	BLK0_EN_STAT		Block 0 enable status. Logical block 0 is the set of memory-mapped registers (MMRs) for the SRIO peripheral.
		0	Logical block 0 is in reset with its clock off.
		1	Logical block 0 is enabled with its clock running.
0	GBL_EN_STAT		Global enable status
		0	The peripheral is in reset with all its clocks off.
		1	The peripheral is enabled with all its clocks running.

5.7 Block *n* Enable Register (BLK*n*_EN)

There are nine of these registers, one for each of nine logical blocks in the peripheral. The registers and the blocks they support are listed in Table 49. The general form for a block *n* enable register (BLK*n*_EN) is shown in Figure 66 and described in Table 50. For additional programming information, see Section 2.3.10.

Table 49. Block *n* Enable Registers and the Associated Blocks

Register	Address Offset	Associated Block
BLK0_EN	0038h	Logical block 0: the set of memory-mapped control registers for the SRIO peripheral
BLK1_EN	0040h	Logical block 1: the Load/Store module (the four LSUs and supporting logic)
BLK2_EN	0048h	Logical block 2: the memory access unit (MAU)
BLK3_EN	0050h	Logical block 3: the message transmit unit (TXU)
BLK4_EN	0058h	Logical block 4: the message receive unit (RXU).
BLK5_EN	0060h	Logical block 5: SRIO port 0
BLK6_EN	0068h	Logical block 6: SRIO port 1.
BLK7_EN	0070h	Logical block 7: SRIO port 2.
BLK8_EN	0078h	Logical block 8: SRIO port 3.

Figure 66. Block *n* Enable Register (BLK*n*_EN)

31	1	0
Reserved		EN
R-0		R/W-1

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 50. Block *n* Enable Register (BLK*n*_EN) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	EN		Block *n* enable
		0	Logical block *n* is to be reset with its clock off.
		1	Logical block *n* is to be enabled with its clock running.

5.8 Block *n* Enable Status Register (BLK*n*_EN_STAT)

There are nine of these registers, one for each of nine logical blocks in the peripheral. The registers and the blocks they support are listed in Table 51. The general form for a block *n* enable status register (BLK*n*_EN_STAT) is shown in Figure 67 and described in Table 52. For additional programming information, see Section 2.3.10.

Table 51. Block *n* Enable Status Registers and the Associated Blocks

Register	Address Offset	Associated Block
BLK0_EN_STAT	003Ch	Logical block 0: the set of memory-mapped control registers for the SRIO peripheral
BLK1_EN_STAT	0044h	Logical block 1: the Load/Store module (the four LSUs and supporting logic)
BLK2_EN_STAT	004Ch	Logical block 2: the memory access unit (MAU)
BLK3_EN_STAT	0054h	Logical block 3: the message transmit unit (TXU)
BLK4_EN_STAT	005Ch	Logical block 4: the message receive unit (RXU).
BLK5_EN_STAT	0064h	Logical block 5: SRIO port 0
BLK6_EN_STAT	006Ch	Logical block 6: SRIO port 1.
BLK7_EN_STAT	0074h	Logical block 7: SRIO port 2.
BLK8_EN_STAT	007Ch	Logical block 8: SRIO port 3.

Figure 67. Block *n* Enable Status Register (BLK*n*_EN)

31	1	0
Reserved		EN_STAT
R-0		R/W-1

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 52. Block *n* Enable Status Register (BLK*n*_EN_STAT) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	These read-only bits return 0s when read.
0	EN_STAT		Block *n* enable status
		0	Logical block *n* is reset with its clock off.
		1	Logical block *n* is enabled with its clock running.

5.9 RapidIO DEVICEID1 Register (DEVICEID_REG1)

The RapidIO DEVICEID1 register (DEVICEID_REG1) is shown in Figure 68 and described in Table 53.

Figure 68. RapidIO DEVICEID1 Register (DEVICEID_REG1) - Address Offset 0080h

31	24	23	16
Reserved		8BNODEID	
R-00h		R/W-FFh	

15	0
16BNODEID	
R/W-FFFFh	

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 53. RapidIO DEVICEID1 Register (DEVICEID_REG1) Field Descriptions

Bit	Field	Value	Description
31-24	Reserved	0	Reserved
23-16	8BNODEID	00h-FFh	This value is equal to the value of the RapidIO Base Device ID CSR. The CPU must read the CSR value and set this register, so that outgoing packets contain the correct SOURCEID value
15-0	16BNODEID	0000h-FFFFh	This value is equal to the value of the RapidIO Base Device ID CSR. The CPU must read the CSR value and set this register, so that outgoing packets contain the correct SOURCEID value

5.10 RapidIO DEVICEID2 Register (DEVICEID_REG2)

The RapidIO DEVICEID2 register (DEVICEID_REG2 is shown in Figure 69 and described in Table 54.

Figure 69. RapidIO DEVICEID2 Register (DEVICEID_REG2) - Address Offset 0084h

31	24	23	16
Reserved		8BNODEID	
R-00h		R/W-FFh	

15	0
16BNODEID	
R-FFFFh	

LEGEND: R/W = Read/Write; R = Read only; *-n* = Value after reset

Table 54. RapidIO DEVICEID2 Register (DEVICEID_REG2) Field Descriptions

Bit	Field	Value	Description
31-24	Reserved	0	Reserved
23-16	8BNODEID	00h-FFh	This is a secondary supported DeviceID checked against an incoming packet's DESTID field. Typically used for Multi-cast support.
15-0	16BNODEID	0000h-FFFFh	This is a secondary supported DeviceID checked against an incoming packet's DESTID field. Typically used for Multi-cast support.

5.11 RapidIO DEVICEID3 Register (DEVICEID_REG3)

The RapidIO DEVICEID3 register (DEVICEID_REG3) is shown in Figure 70 and described in Table 55.

Figure 70. RapidIO DEVICEID3 Register (DEVICEID_REG3) - Address Offset 0088h

31	24	23	16
Reserved		8BNODEID	
R-00h		R/W-FFh	

15	0
16BNODEID	
R-FFFFh	

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 55. RapidIO DEVICEID3 Register (DEVICEID_REG3) Field Descriptions

Bit	Field	Value	Description
31-24	Reserved	0	Reserved
23-16	8BNODEID	00h-FFh	This is a secondary supported DeviceID checked against an incoming packet's DESTID field. Typically used for Multi-cast support.
15-0	16BNODEID	0000h-FFFFh	This is a secondary supported DeviceID checked against an incoming packet's DESTID field. Typically used for Multi-cast support.

5.12 RapidIO DEVICEID4 Register (DEVICEID_REG4)

The RapidIO DEVICEID2 register (DEVICEID_REG4) is shown in Figure 71 and described in Table 56.

Figure 71. RapidIO DEVICEID4 Register (DEVICEID_REG4) - Address Offset 008Ch

31	24	23	16
Reserved		8BNODEID	
R-00h		R/W-FFh	

15	0
16BNODEID	
R-FFFFh	

LEGEND: R/W = Read/Write; R = Read only; *-n* = Value after reset

Table 56. RapidIO DEVICEID4 Register (DEVICEID_REG4) Field Descriptions

Bit	Field	Value	Description
31-24	Reserved	0	Reserved
23-16	8BNODEID	00h-FFh	This is a secondary supported DeviceID checked against an incoming packet's DESTID field. Typically used for Multi-cast support.
15-0	16BNODEID	0000h-FFFFh	This is a secondary supported DeviceID checked against an incoming packet's DESTID field. Typically used for Multi-cast support.

5.13 Packet Forwarding Register *n* for 16-Bit Device IDs (PF_16B_CNTL*n*)

There are four of these registers (see Table 57). The general form of a packet forwarding register for 16-bit DeviceIDs is shown in Figure 72 and described in Table 58. For additional programming information, see Section 2.3.15.

Table 57. PF_16B_CNTL Registers

Register	Address Offset
PF_16B_CNTL0	0090h
PF_16B_CNTL1	0098h
PF_16B_CNTL2	00A0h
PF_16B_CNTL3	00A8h

Figure 72. Packet Forwarding Register *n* for 16-Bit Device IDs (PF_16B_CNTL*n*)

31 16	15 0
16BIT_DEVID_UP_BOUND	16BIT_DEVID_LOW_BOUND
R/W-FFFFh	R/W-FFFFh

LEGEND: R/W = Read/Write; -*n* = Value after reset

Table 58. Packet Forwarding Register *n* for 16-Bit DeviceIDs (PF_16B_CNTL*n*) Field Descriptions

Bit	Field	Value	Description
31-16	16BIT_DEVID_UP_BOUND	0000h-FFFFh	Upper 16-bit DeviceID boundary. DESTID above this range cannot use the table entry.
15-0	16BIT_DEVID_LOW_BOUND	0000h-FFFFh	Lower 16-bit DeviceID boundary. DESTID lower than this number cannot use the table entry.

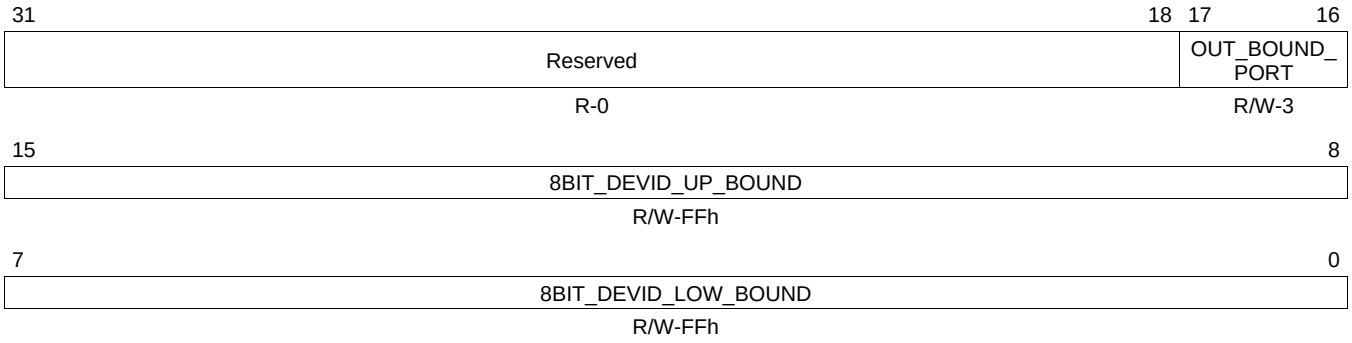
5.14 Packet Forwarding Register *n* for 8-Bit Device IDs (PF_8B_CNTL*n*)

There are four of these registers (see Table 59). The general form of a packet forwarding register for 16-bit DeviceIDs is shown in Figure 73 and described in Table 60. For additional programming information see Section 2.3.15.

Table 59. PF_8B_CNTL Registers

Register	Address Offset
PF_8B_CNTL0	0094h
PF_8B_CNTL1	009Ch
PF_8B_CNTL2	00A4h
PF_8B_CNTL3	00ACh

Figure 73. Packet Forwarding Register *n* for 8-Bit Device IDs (PF_8B_CNTL*n*)



LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 60. Packet Forwarding Register *n* for 8-Bit DeviceIDs (PF_8B_CNTL*n*) Field Descriptions

Bit	Field	Value	Description
31-18	Reserved	0	Reserved
17-16	OUT_BOUND_PORT	0-3	Output port number for packets whose DESTID falls within the 8-bit or 16-bit range for this table entry.
15-8	8BIT_DEVID_UP_BOUND	00h-FFh	Upper 8-bit DeviceID boundary. DESTID above this range cannot use the table entry.
7-0	8BIT_DEVID_LOW_BOUND	00h-FFh	Lower 8-bit DeviceID boundary. DESTID lower than this number cannot use the table entry.

5.15 SERDES Receive Channel Configuration Register *n* (SERDES_CFGRX*n*_CNTL)

There are four of these registers, to support four ports (see Table 61). The general form for a SERDES receive channel configuration register is summarized by Figure 74 and Table 62. See Section 2.3.2.2 for a complete explanation of the programming of these registers.

Table 61. SERDES_CFGRX*n*_CNTL Registers and the Associated Ports

Register	Address Offset	Associated Port
SERDES_CFGRX0_CNTL	0100h	Port 0
SERDES_CFGRX1_CNTL	0104h	Port 1
SERDES_CFGRX2_CNTL	0108h	Port 2
SERDES_CFGRX3_CNTL	010Ch	Port 3

Figure 74. SERDES Receive Channel Configuration Register *n* (SERDES_CFGRX*n*_CNTL)

31–26	25–24	23	22–19	18–16
Reserved	Reserved (write 0s)	-	EQ	CDR
R-0	R/W-0	R-0	R/W-0	R/W-0

15–14	13–12	11	10–8	7	6–5	4–2	1	0
LOS	ALIGN	-	TERM (write 001b)	INVPAIR	RATE	BUSWIDTH	- (write 0)	ENRX
R/W-0	R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 62. SERDES Receive Channel Configuration Register *n* (SERDES_CFGRX*n*_CNTL) Field Descriptions

Bit	Field	Value	Description
31-26	Reserved	000000b	These read-only bits return 0s when read.
25-24	Reserved	00b	Always write 0s to these reserved bits.
23	Reserved	0	This read-only bit returns 0 when read.
22-19	EQ	0000b-1111b	Equalizer. Enables and configures the adaptive equalizer to compensate for loss in the transmission media. For the selectable values, see Table 63.
18-16	CDR		Clock/data recovery. Configures the clock/data recovery algorithm.
		000b	First order. Phase offset tracking up to ±488 ppm.
		001b	Second order. Highest precision frequency offset matching but poorest response to changes in frequency offset, and longest lock time. Suitable for use in systems with fixed frequency offset.
		010b	Second order. Medium precision frequency offset matching, frequency offset change response, and lock time.
		011b	Second order. Best response to changes in frequency offset and fastest lock time, but lowest precision frequency offset matching. Suitable for use in systems with spread spectrum clocking.
		100b	First order with fast lock. Phase offset tracking up to ±1953 ppm in the presence of ..10101010.. training pattern, and ±448 ppm otherwise.
		101b	Second order with fast lock. As per setting 001, but with improved response to changes in frequency offset when not close to lock.
		110b	Second order with fast lock. As per setting 010, but with improved response to changes in frequency offset when not close to lock.
		111b	Second order with fast lock. As per setting 011, but with improved response to changes in frequency offset when not close to lock.

Table 62. SERDES Receive Channel Configuration Register *n* (SERDES_CFGRX*n*_CNTL) Field Descriptions (continued)

Bit	Field	Value	Description
15-14	LOS		Loss of signal. Enables loss of signal detection with 2 selectable thresholds.
		00b	Disabled. Loss of signal detection disabled.
		01b	High threshold. Loss of signal detection threshold in the range 85 to 195mV$_{dfpp}$. This setting is suitable for Infiniband.
		10b	Low threshold. Loss of signal detection threshold in the range 65 to 175mV$_{dfpp}$. This setting is suitable for PCI-E and S-ATA.
		11b	Reserved
13-12	ALIGN		Symbol alignment. Enables internal or external symbol alignment.
		00b	Alignment disabled. No symbol alignment will be performed while this setting is selected, or when switching to this selection from another.
		01b	Comma alignment enabled. Symbol alignment will be performed whenever a misaligned comma symbol is received.
		10b	Alignment jog. The symbol alignment will be adjusted by one bit position when this mode is selected (that is, the ALIGN value changes from 0xb to 1xb).
		11b	Reserved
11	Reserved	0	This read-only bit returns 0 when read.
10-8	TERM	001b	Input termination. The only valid value for this field is 001b. This value sets the common point to 0.8 VDDT and supports AC coupled systems using CML transmitters. The transmitter has no effect on the receiver common mode, which is set to optimize the input sensitivity of the receiver. Common mode termination is via a 50 pF capacitor to VSSA.
7	INVPAIR		Invert polarity. Inverts polarity of RIORXn and $\overline{\text{RIORXn}}$.
		0	Normal polarity. RIORXn is considered to be positive data and $\overline{\text{RIORXn}}$ negative.
		1	Inverted polarity. RIORXn is considered to be negative data and $\overline{\text{RIORXn}}$ positive.
6-5	RATE		Operating rate. Selects full, half, or quarter rate operation.
		00b	Full rate. Two data samples taken per PLL output clock cycle.
		01b	Half rate. One data sample taken per PLL output clock cycle.
		10b	Quarter rate. One data sample taken every two PLL output clock cycles.
		11b	Reserved
4-2	BUSWIDTH	000b	Bus width. Always write 000b to this field, to indicate a 10-bit-wide parallel bus to the clock. All other values are reserved. See Section 2.3.2.1 for an explanation of the bus.
1	Reserved	0	Always write 0 to this reserved bit.
0	ENRX		Enable receiver
		0	Disable this receiver.
		1	Enable this receiver.

Table 63. EQ Bits

CFGRX[22-19]	Low Freq Gain	Zero Freq (at e_{28} (min))
0000b	Maximum	-
0001b	Adaptive	Adaptive
001xb	Reserved	
01xxb	Reserved	

Table 63. EQ Bits (continued)

CFGRX[22-19]	Low Freq Gain	Zero Freq (at e_{28} (min))
1000b	Adaptive	1084MHz
1001b		805MHz
1010b		573MHz
1011b		402MHz
1100b		304MHz
1101b		216MHz
1110b		156MHz
1111b		135MHz

5.16 SERDES Transmit Channel Configuration Register *n* (SERDES_CFGTX*n*_CNTL)

There are four of these registers, to support four ports (see Table 64). The general form for a SERDES transmit channel configuration register is summarized by Figure 75 and Table 65. See Section 2.3.2.3 for a complete explanation of the programming for these registers.

Table 64. SERDES_CFGTX*n*_CNTL Registers and the Associated Ports

Register	Address Offset	Associated Port
SERDES_CFGTX0_CNTL	0110h	Port 0
SERDES_CFGTX1_CNTL	0114h	Port 1
SERDES_CFGTX2_CNTL	0118h	Port 2
SERDES_CFGTX3_CNTL	011Ch	Port 3

Figure 75. SERDES Transmit Channel Configuration Register *n* (SERDES_CFGTX*n*_CNTL)

31 – 17	16
Reserved	ENFTP
R-0	R/W-0

15 – 12	11 – 9	8	7	6 – 5	4 – 2	1	0
DE	SWING	CM	INVPAIR	RATE	BUSWIDTH	- (write 0)	ENTX
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 65. SERDES Transmit Channel Configuration Register *n* (SERDES_CFGTX*n*_CNTL) Field Descriptions

Bit	Field	Value	Description
31-17	Reserved	0	These read-only bits return 0s when read.
16	ENFTP	1	Enables fixed phase relationship of transmit input clock with respect to transmit output clock. The only valid value for this field is 1b; all other values are reserved.
15-12	DE	0000b-1111b	De-emphasis. Selects one of 15 output de-emphasis settings from 4.76 to 71.42%. De-emphasis provides a means to compensate for high frequency attenuation in the attached media. It causes the output amplitude to be smaller for bits which are not preceded by a transition than for bits which are. See Table 66.
11-9	SWING	000b-111b	Output swing. Selects one of 8 output amplitude settings between 125 and 1250mV$_{dfpp}$. See Table 67.
8	CM		Common mode. Adjusts the common mode to suit the termination at the attached receiver.
		0	Normal common mode. Common mode not adjusted.
		1	Raised common mode. Common mode raised by 5% of e_{54}.
7	INVPAIR		Invert polarity. Inverts the polarity of RIOTXn and $\overline{\text{RIOTXn}}$.
		0	Normal polarity. RIOTXn is considered to be positive data and $\overline{\text{RIOTXn}}$ negative.
		1	Inverted polarity. RIOTXn is considered to be negative data and $\overline{\text{RIOTXn}}$ positive.
6-5	RATE		Operating rate. Selects full, half, or quarter rate operation.
		00b	Full rate. Two data samples taken per PLL output clock cycle.
		01b	Half rate. One data sample taken per PLL output clock cycle.
		10b	Quarter rate. One data sample taken every two PLL output clock cycles.
		11b	Reserved
4-2	BUSWIDTH	000b	Bus width. Always write 000b to this field, to indicate a 10-bit-wide parallel bus to the clock. All other values are reserved. See Section 2.3.2.1 for an explanation of the bus.
1	Reserved	0	Always write 0 to this reserved bit.
0	ENTX		Enable transmitter
		0	Disable this transmitter.
		1	Enable this transmitter.

Table 66. DE Bits of SERDES_CFGTX*n*_CNTL

DE Bits	Amplitude Reduction	
	%	dB
0000b	0	0
0001b	4.76	-0.42
0010b	9.52	-0.87
0011b	14.28	-1.34
0100b	19.04	-1.83
0101b	23.8	-2.36
0110b	28.56	-2.92
0111b	33.32	-3.52
1000b	38.08	-4.16
1001b	42.85	-4.86
1010b	47.61	-5.61
1011b	52.38	-6.44
1100b	57.14	-7.35
1101b	61.9	-8.38
1110b	66.66	-9.54
1111b	71.42	-10.87

Table 67. SWING Bits of SERDES_CFGTX*n*_CNTL

SWING Bits	Amplitude (mV_{dfpp})
000b	125
001b	250
010b	500
011b	625
100b	750
101b	1000
110b	1125
111b	1250

5.17 SERDES Macro Configuration Register *n* (SERDES_CFG*n*_CNTL)

There are four of these registers, to support four ports (see Table 68). The general form for a SERDES transmit channel configuration register is summarized by Figure 76 and Table 69. See Section 2.3.2.1 for a complete explanation of the programming of this register.

Table 68. SERDES_CFG*n*_CNTL Registers and the Associated Ports

Register	Address Offset	Associated Port
SERDES_CFG0_CNTL	0120h	Port 0, Port 1, Port 2, and Port 3
SERDES_CFG1_CNTL	0124h	Not Used. Program as 0x00000000
SERDES_CFG2_CNTL	0128h	Not Used. Program as 0x00000000
SERDES_CFG3_CNTL	012Ch	Not Used. Program as 0x00000000

Figure 76. SERDES Macro Configuration Register *n* (SERDES_CFG*n*_CNTL)

31	16
Reserved	
R-0	

15	10	9	8	7	6	5	1	0
Reserved		LB		Reserved		MPY		ENPLL
R-0		R/W-0		R-0		R/W-0		R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 69. SERDES Macro Configuration Register *n* (SERDES_CFG*n*_CNTL) Field Descriptions

Bit	Field	Value	Description
31-10	Reserved	0	Reserved
9-8	LB		Loop bandwidth. Specify loop bandwidth settings. Jitter on the reference clock will degrade both the transmit eye and receiver jitter tolerance thereby impairing system performance. Performance of the integrated PLL can be optimized according to the jitter characteristics of the reference clock via the LB field.
		00b	Frequency dependent bandwidth. The PLL bandwidth is set to 1/12 of the frequency of RIOCLK/$\overline{\text{RIOCLK}}$. This setting is suitable for most systems that input the reference clock via a low jitter input cell, and is required for standards compliance
		01b	Reserved
		10b	Low bandwidth. The PLL bandwidth is set to 1/20 of the frequency of RIOCLK/$\overline{\text{RIOCLK}}$, or 3MHz (whichever is larger). In systems where the reference clock is directly input via a low jitter input cell, but is of lower quality, this setting may offer better performance. It will reduce the amount of reference clock jitter transferred through the PLL. However, it also increases the susceptibility to loop noise generated within the PLL itself. It is difficult to predict whether the improvement in the former will more than offset the degradation in the latter.
		11b	High bandwidth. The PLL bandwidth is set to 1/8 of the frequency of RIOCLK/$\overline{\text{RIOCLK}}$. This is the setting appropriate for systems where the reference clock is cleaned through an ultra low jitter LC-based PLL. Standards compliance will be achieved even if the reference clock input to the cleaner PLL is outside the specification for the standard.
7-6	Reserved	0	Reserved

Table 69. SERDES Macro Configuration Register *n* (SERDES_CFG*n*_CNTL) Field Descriptions (continued)

Bit	Field	Value	Description
5-1	MPY		PLL multiply. Select PLL multiply factors between 4 and 60.
		00000b	4x
		00001b	5x
		00010b	6x
		00011b	Reserved
		00100b	8x
		00101b	10x
		00110b	12x
		00111b	12.5x
		01000b	15x
		01001b	20x
		01010b	25x
		01011b	Reserved
		01100b	Reserved
		01111b	Reserved
		1xxxxb	Reserved
0	ENPLL		Enable PLL
		0	PLL disabled
		1	PLL enabled

5.18 DOORBELL*n* Interrupt Condition Status Register (DOORBELL*n*_ICSR)

The four doorbell interrupts are mapped to these registers (see Table 70). The general form of a doorbell interrupt condition status register is shown in Figure 77 and described in Table 71. For additional programming information, see Section 4.3.1 and Section 2.3.6.

Table 70. DOORBELL*n*_ICSR Registers

Register	Address Offset
DOORBELL0_ICSR	0200h
DOORBELL1_ICSR	0210h
DOORBELL2_ICSR	0220h
DOORBELL3_ICSR	0230h

Figure 77. Doorbell *n* Interrupt Condition Status Register (DOORBELL*n*_ICSR)

31															16
Reserved															
R-0															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICS15	ICS14	ICS13	ICS12	ICS11	ICS10	ICS9	ICS8	ICS7	ICS6	ICS5	ICS4	ICS3	ICS2	ICS1	ICS0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 71. DOORBELL*n* Interrupt Condition Status Register (DOORBELL*n*_ICSR) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	These read-only bits return 0s when read.
15-0	ICS*x* (x = 15 to 0)		Doorbell *n* interrupt condition status bit
		0	Bit *x* of the doorbell information value is 0.
		1	Bit *x* of the doorbell information value is 1, generating an interrupt request.

5.19 DOORBELL*n* Interrupt Condition Clear Register (DOORBELL*n*_ICCR)

The four doorbells interrupts that are mapped are cleared by this register (see Table 72). The general form of a doorbell interrupt condition clear register is shown in Figure 78 and described in Table 73. For additional programming information, see Section 4.4.1 and Section 2.3.6.

Table 72. DOORBELL*n*_ICCR Registers

Register	Address Offset
DOORBELL0_ICCR	0208h
DOORBELL1_ICCR	0218h
DOORBELL2_ICCR	0228h
DOORBELL3_ICCR	0238h

Figure 78. Doorbell *n* Interrupt Condition Clear Register (DOORBELL*n*_ICCR)

31															16
Reserved															
R-0															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICC15	ICC14	ICC13	ICC12	ICC11	ICC10	ICC9	ICC8	ICC7	ICC6	ICC5	ICC4	ICC3	ICC2	ICC1	ICC0
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: W = Write only; R = Read only; -*n* = Value after reset

Table 73. DOORBELL*n* Interrupt Condition Clear Register (DOORBELL*n*_ICCR) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	These read-only bits return 0s when read.
15-0	ICC*x* (x = 15 to 0)		Doorbell *n* interrupt condition clear bit
		0	No effect
		1	Clear bit *x* of the corresponding doorbell interrupt condition status register (ICSR).

5.20 RX CPPI Interrupt Status Register (RX_CPPI_ICSR)

The bits in this register indicate any active interrupt requests from RX buffer descriptor queues. The RX CPPI interrupt status register (RX_CPPI_ICSR) is shown in Figure 79 and described in Table 74. For additional programming information, see Section 4.3.2.

Figure 79. RX CPPI Interrupt Condition Status Register (RX_CPPI_ICSR) - Address Offset 0240h

31															16
Reserved															
R-0															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICS15	ICS14	ICS13	ICS12	ICS11	ICS10	ICS9	ICS8	ICS7	ICS6	ICS5	ICS4	ICS3	ICS2	ICS1	ICS0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 74. RX CPPI Interrupt Condition Status Register (RX_CPPI_ICSR) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	These read-only bits return 0 when read.
15-0	ICS*x* (x = 15 to 0)		RX CPPI interrupt status
		0	RX buffer descriptor queue *x* has not generated an interrupt request.
		1	RX buffer descriptor queue *x* has generated an interrupt request.

5.21 RX CPPI Interrupt Clear Register (RX_CPPI_ICCR)

This register is used to clear bits in RX_CPPI_ICSR to acknowledge interrupts from the RX buffer descriptor queues. The RX CPPI interrupt clear register (RX_CPPI_ICCR) is shown in Figure 80 and described in Table 75. For additional programming information, see Section 4.3.2.

Figure 80. RX CPPI Interrupt Condition Clear Register (RX_CPPI_ICCR) - Address Offset 0248h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICC15	ICC14	ICC13	ICC12	ICC11	ICC10	ICC9	ICC8	ICC7	ICC6	ICC5	ICC4	ICC3	ICC2	ICC1	ICC0
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: R = Read only; W = Write only; -*n* = Value after reset

Table 75. RX CPPI Interrupt Condition Clear Register (RX_CPPI_ICCR) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	These read-only bits return 0 when read.
15-0	ICC*x* (x = 15 to 0)		RX CPPI interrupt clear
		0	No effect
		1	Clear bit *x* of RX_CPPI_ICSR.

5.22 TX CPPI Interrupt Status Register (TX_CPPI_ICSR)

The bits in this register indicate any active interrupt requests from TX buffer descriptor queues. TX_CPPI_ICSR is shown in Figure 81 and described in Table 76.

Figure 81. TX CPPI Interrupt Condition Status Register (TX_CPPI_ICSR) - Address Offset 0250h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICS15	ICS14	ICS13	ICS12	ICS11	ICS10	ICS9	ICS8	ICS7	ICS6	ICS5	ICS4	ICS3	ICS2	ICS1	ICS0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 76. TX CPPI Interrupt Condition Status Register (TX_CPPI_ICSR) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	These read-only bits return 0 when read.
15-0	ICS*x* (x = 15 to 0)		TX CPPI interrupt status
		0	TX buffer descriptor queue *x* has not generated an interrupt request.
		1	TX buffer descriptor queue *x* has generated an interrupt request.

5.23 TX CPPI Interrupt Clear Register (TX_CPPI_ICCR)

This register is used to clear bits in TX_CPPI_ICSR to acknowledge interrupts from the TX buffer descriptor queues. TX_CPPI_ICCR is shown in Figure 82 and described in Table 77.

Figure 82. TX CPPI Interrupt Condition Clear Register (TX_CPPI_ICCR) - Address Offset 0258h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICC15	ICC14	ICC13	ICC12	ICC11	ICC10	ICC9	ICC8	ICC7	ICC6	ICC5	ICC4	ICC3	ICC2	ICC1	ICC0
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: R = Read only; W = Write only; -*n* = Value after reset

Table 77. TX CPPI Interrupt Condition Clear Register (TX_CPPI_ICCR) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	These read-only bits return 0 when read.
15-0	ICC*x* (x = 15 to 0)		TX CPPI interrupt clear
		0	No effect
		1	Clear bit *x* of TX_CPPI_ICSR.

5.24 LSU Interrupt Condition Status Register (LSU_ICSR)

Each of the status bits in this register indicates the occurrence of a particular type of transaction interrupt condition for a particular LSU. LSU_ICSR is shown in Figure 83 and described in Table 78. For additional programming information, see Section 4.3.3.

Figure 83. LSU Interrupt Condition Status Register (LSU_ICSR) - Address Offset 0260h

<---------- Bits for LSU4 ---------->								<---------- Bits for LSU3 ---------->							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ICS31	ICS30	ICS29	ICS28	ICS27	ICS26	ICS25	ICS24	ICS23	ICS22	ICS21	ICS20	ICS19	ICS18	ICS17	ICS16
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
<---------- Bits for LSU2 ---------->								<---------- Bits for LSU1 ---------->							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICS15	ICS14	ICS13	ICS12	ICS11	ICS10	ICS9	ICS8	ICS7	ICS6	ICS5	ICS4	ICS3	ICS2	ICS1	ICS0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = Value after reset

Table 78. LSU Interrupt Condition Status Register (LSU_ICSR) Field Descriptions

Bit	Field	Value	Description
31	ICS31	0	LSU4 interrupt condition not detected.
		1	LSU4 interrupt condition detected. Packet not sent due to unavailable outbound credit at given priority.
30	ICS30	0	LSU4 interrupt condition not detected.
		1	LSU4 interrupt condition detected. Retry Doorbell response received or Atomic test-and-swap was not allowed (semaphore in use).
29	ICS29	0	LSU4 interrupt condition not detected.
		1	LSU4 interrupt condition detected. Transaction was not sent due to DMA data transfer error.
28	ICS28	0	LSU4 interrupt condition not detected.
		1	LSU4 interrupt condition detected. Transaction was not sent due to unsupported transaction type or invalid field encoding.
27	ICS27	0	LSU4 interrupt condition not detected.
		1	LSU4 interrupt condition detected. Non-posted transaction received ERROR response, or error in response payload.
26	ICS26	0	LSU4 interrupt condition not detected.
		1	LSU4 interrupt condition detected. Transaction was not sent due to Xoff condition.
25	ICS25	0	LSU4 interrupt condition not detected.
		1	LSU4 interrupt condition detected. Transaction timeout occurred.
24	ICS24	0	LSU4 interrupt condition not detected.
		1	LSU4 interrupt condition detected. Transaction complete, No errors (posted/non-posted). Enable for this interrupt is ultimately controlled by the Interrupt Req bit of LSU4_REG4. This allows enabling/disabling on a per request basis. For optimum LSU performance, interrupt pacing should not be used on the LSU interrupts.
23	ICS23	0	LSU3 interrupt condition not detected.
		1	LSU3 interrupt condition detected. Packet not sent due to unavailable outbound credit at given priority.
22	ICS22	0	LSU3 interrupt condition not detected.
		1	LSU3 interrupt condition detected. Retry Doorbell response received or Atomic test-and-swap was not allowed (semaphore in use).
21	ICS21	0	LSU3 interrupt condition not detected.
		1	LSU3 interrupt condition detected. Transaction was not sent due to DMA data transfer error.
20	ICS20	0	LSU3 interrupt condition not detected.
		1	LSU3 interrupt condition detected. Transaction was not sent due to unsupported transaction type or invalid field encoding.

Table 78. LSU Interrupt Condition Status Register (LSU_ICSR) Field Descriptions (continued)

Bit	Field	Value	Description
19	ICS19	0	LSU3 interrupt condition not detected.
		1	LSU3 interrupt condition detected. Non-posted transaction received ERROR response, or error in response payload.
18	ICS18	0	LSU3 interrupt condition not detected.
		1	LSU3 interrupt condition detected. Transaction was not sent due to Xoff condition.
17	ICS17	0	LSU3 interrupt condition not detected.
		1	LSU3 interrupt condition detected. Transaction timeout occurred.
16	ICS16	0	LSU3 interrupt condition not detected.
		1	LSU3 interrupt condition detected. Transaction complete, No errors (posted/non-posted). Enable for this interrupt is ultimately controlled by the Interrupt Req bit of LSU3_REG4. This allows enabling/disabling on a per request basis. For optimum LSU performance, interrupt pacing should not be used on the LSU interrupts.
15	ICS15	0	LSU2 interrupt condition not detected.
		1	LSU2 interrupt condition detected. Packet not sent due to unavailable outbound credit at given priority.
14	ICS14	0	LSU2 interrupt condition not detected.
		1	LSU2 interrupt condition detected. Retry Doorbell response received or Atomic test-and-swap was not allowed (semaphore in use).
13	ICS13	0	LSU2 interrupt condition not detected.
		1	LSU2 interrupt condition detected. Transaction was not sent due to DMA data transfer error.
12	ICS12	0	LSU2 interrupt condition not detected.
		1	LSU2 interrupt condition detected. Transaction was not sent due to unsupported transaction type or invalid field encoding.
11	ICS11	0	LSU2 interrupt condition not detected.
		1	LSU2 interrupt condition detected. Non-posted transaction received ERROR response, or error in response payload.
10	ICS10	0	LSU2 interrupt condition not detected.
		1	LSU2 interrupt condition detected. Transaction was not sent due to Xoff condition.
9	ICS9	0	LSU2 interrupt condition not detected.
		1	LSU2 interrupt condition detected. Transaction timeout occurred.
8	ICS8	0	LSU2 interrupt condition not detected.
		1	LSU2 interrupt condition detected. Transaction complete, No errors (posted/non-posted). Enable for this interrupt is ultimately controlled by the Interrupt Req bit of LSU2_REG4. This allows enabling/disabling on a per request basis. For optimum LSU performance, interrupt pacing should not be used on the LSU interrupts.
7	ICS7	0	LSU1 interrupt condition not detected.
		1	LSU1 interrupt condition detected. Packet not sent due to unavailable outbound credit at given priority.
6	ICS6	0	LSU1 interrupt condition not detected.
		1	LSU1 interrupt condition detected. Retry Doorbell response received or Atomic test-and-swap was not allowed (semaphore in use).
5	ICS5	0	LSU1 interrupt condition not detected.
		1	LSU1 interrupt condition detected. Transaction was not sent due to DMA data transfer error.
4	ICS4	0	LSU1 interrupt condition not detected.
		1	LSU1 interrupt condition detected. Transaction was not sent due to unsupported transaction type or invalid field encoding.
3	ICS3	0	LSU1 interrupt condition not detected.
		1	LSU1 interrupt condition detected. Non-posted transaction received ERROR response, or error in response payload.
2	ICS2	0	LSU1 interrupt condition not detected.
		1	LSU1 interrupt condition detected. Transaction was not sent due to Xoff condition.

Table 78. LSU Interrupt Condition Status Register (LSU_ICSR) Field Descriptions (continued)

Bit	Field	Value	Description
1	ICS1	0	LSU1 interrupt condition not detected.
		1	LSU1 interrupt condition detected. Transaction timeout occurred.
0	ICS0	0	LSU1 interrupt condition not detected.
		1	LSU1 interrupt condition detected. Transaction complete, No errors (posted/non-posted). Enable for this interrupt is ultimately controlled by the Interrupt Req bit of LSU1_REG4. This allows enabling/disabling on a per request basis. For optimum LSU performance, interrupt pacing should not be used on the LSU interrupts.

5.25 LSU Interrupt Condition Clear Register (LSU_ICCR)

Setting a bit in this register clears the corresponding bit in LSU_ICSR, to acknowledge the interrupt. LSU_ICCR is shown in Figure 84 and described in Table 79. For additional programming information, see Section 4.3.3.

Figure 84. LSU Interrupt Condition Clear Register (LSU_ICCR) - Address Offset 0268h

<---------- Bits for LSU4 ---------->								<---------- Bits for LSU3 ---------->							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ICC31	ICC30	ICC29	ICC28	ICC27	ICC26	ICC25	ICC24	ICC23	ICC22	ICC21	ICC20	ICC19	ICC18	ICC17	ICC16
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0
<---------- Bits for LSU2 ---------->								<---------- Bits for LSU1 ---------->							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICC15	ICC14	ICC13	ICC12	ICC11	ICC10	ICC9	ICC8	ICC7	ICC6	ICC5	ICC4	ICC3	ICC2	ICC1	ICC0
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: W = Write only; -*n* = Value after reset

Table 79. LSU Interrupt Condition Clear Register (LSU_ICCR) Field Descriptions

Bit	Field	Value	Description
31-0	ICC*x* (x = 31 to 0)	0	No effect
		1	Clear bit *x* of the LSU interrupt condition status register (LSU_ICSR).

5.26 Error, Reset, and Special Event Interrupt Condition Status Register (ERR_RST_EVNT_ICSR)

Each of the non-reserved bits in this register indicate the status of a particular interrupt condition in one or more of the SRIO ports. ERR_RST_EVNT_ICSR is shown in Figure 85 and described in Table 80. For additional programming information, see Section 4.3.4.

Figure 85. Error, Reset, and Special Event Interrupt Condition Status Register (ERR_RST_EVNT_ICSR) - Address Offset 0270h

31	17	16
Reserved		ICS16
R-0		R-0

15	12	11	10	9	8	7	3	2	1	0
Reserved		ICS11	ICS10	ICS9	ICS8	Reserved		ICS2	ICS1	ICS0
R-0		R-0	R-0	R-0	R-0	R-0		R-0	R-0	R-0

LEGEND: R = Read only; W = Write only; -*n* = Value after reset

Table 80. Error, Reset, and Special Event Interrupt Condition Status Register (ERR_RST_EVNT_ICSR) Field Descriptions

Bit	Field	Value	Description
31-17	Reserved	0	These reserved bits return 0s when read.
16	ICS16	0	Device reset interrupt not received from any port
		1	Device reset interrupt received from any port
15-12	Reserved	0	These reserved bits return 0s when read.
11	ICS11	0	Error not detected on port 3
		1	Error detected on port 3
10	ICS10	0	Error not detected on port 2
		1	Error detected on port 2
9	ICS9	0	Error not detected on port 1
		1	Error detected on port 1
8	ICS8	0	Error not detected on port 0
		1	Error detected on port 0
7-3	Reserved	0	These reserved bits return 0s when read.
2	ICS2	0	Logical layer error management event capture not detected
		1	Logical layer error management event capture detected
1	ICS1	0	Port-write-in request not received on any port
		1	Port-write-in request received on any port
0	ICS0	0	Multi-cast event control symbol interrupt not received on any port
		1	Multi-cast event control symbol interrupt received on any port

5.27 Error, Reset, and Special Event Interrupt Condition Clear Register (ERR_RST_EVNT_ICCR)

Each bit in this register is used to clear the corresponding status bit in ERR_RST_EVNT_ICSR. The field of ERR_RST_EVNT_ICCR are shown in Figure 86 and described in Table 81. For additional programming information, see Section 4.3.4.

Figure 86. Error, Reset, and Special Event Interrupt Condition Clear Register (ERR_RST_EVNT_ICCR) - Address Offset 0278h

31	17	16
Reserved		ICC16
R-0		W-0

15	12	11	10	9	8	7	3	2	1	0
Reserved		ICC11	ICC10	ICC9	ICC8	Reserved		ICC2	ICC1	ICC0
R-0		W-0	W-0	W-0	W-0	R-0		W-0	W-0	W-0

LEGEND: R = Read only; W = Write only; -*n* = Value after reset

Table 81. Error, Reset, and Special Event Interrupt Condition Clear Register (ERR_RST_EVNT_ICCR) Field Descriptions

Bit	Field	Value	Description
31-17	Reserved	0	These read-only bits return 0s when read.
16	ICC16	0	No effect
		1	Clear bit 16 of ERR_RST_EVNT_ICSR.
15-12	Reserved	0	These read-only bits return 0s when read.
11-8	ICC*x* (x = 11 to 8)	0	No effect
		1	Clear bit *x* of ERR_RST_EVNT_ICSR.
7-3	Reserved	0	These read-only bits return 0s when read.
2-0	ICC*y* (y = 2 to 0)	0	No effect
		1	Clear bit *y* of ERR_RST_EVNT_ICSR.

5.28 DOORBELL*n* Interrupt Condition Routing Registers (DOORBELL*n*_ICRR and DOORBELL*n*_ICRR2)

When doorbell packets are received by the SRIO peripheral, these ICRRs route doorbell interrupt requests from the associated doorbell ICSR to user-selected interrupt destinations. Each of the four doorbells can be mapped to these registers (see Table 82). The general field description in Table 83 applies to an ICRx field of either register. For additional programming information, see Section 4.4.1 and Section 2.3.6.

Table 82. DOORBELL*n*_ICRR Registers

Register	Address Offset
DOORBELL0_ICRR	0280h
DOORBELL0_ICRR2	0284h
DOORBELL1_ICRR	0290h
DOORBELL1_ICRR2	0294h
DOORBELL2_ICRR	02A0h
DOORBELL2_ICRR2	02A4h
DOORBELL3_ICRR	02B0h
DOORBELL3_ICRR3	02B4h

Figure 87. Doorbell *n* Interrupt Condition Routing Registers

Doorbell n Interrupt Condition Routing Register (DOORBELLn_ICRR)

31–28	27–24	23–20	19–16
ICR7	ICR6	ICR5	ICR4
R/W-0h	R/W-0h	R/W-0h	R/W-0h

15–12	11–8	7–4	3–0
ICR3	ICR2	ICR1	ICR0
R/W-0000	R/W-0000	R/W-0000	R/W-0000

Doorbell n Interrupt Condition Routing Register 2 (DOORBELLn_ICRR2)

31–28	27–24	23–20	19–16
ICR15	ICR14	ICR13	ICR12
R/W-0000	R/W-0000	R/W-0000	R/W-0000

15–12	11–8	7–4	3–0
ICR11	ICR10	ICR9	ICR8
R/W-0000	R/W-0000	R/W-0000	R/W-0000

LEGEND: R/W = Read/Write; -*n* = Value after reset

Table 83. DOORBELL*n* Interrupt Condition Routing Register Field Descriptions

Field	Value	Description
ICRx (x = 0 to 15)		Interrupt condition routing. Routes the interrupt request from doorbell *n*, bit x to one of eight interrupt destinations (INTDST0-INTDST7). For example, if ICS6 = 1 in DOORBELL2_ICSR and ICR6 = 0010b in DOORBELL2_ICRR, the interrupt request from doorbell 2, bit 6 is sent to interrupt destination 2.
	0000b	INTDST0
	0001b	INTDST1
	0010b	INTDST2
	0011b	INTDST3
	0100b	INTDST4
	0101b	INTDST5
	0110b	INTDST6
	0111b	INTDST7
	1xxxb	Reserved

5.29 RX CPPI Interrupt Condition Routing Registers (RX_CPPI_ICRR and RX_CPPI_ICRR2)

Figure 88 and Table 84 summarize the ICRRs for the RXU. These registers route queue interrupts to interrupt destinations. For example, if ICS6 = 1 in RX_CPPI_ICSR and ICR6 = 0010b in RX_CPPI_ICRR, the interrupt request from RX buffer descriptor queue 6 is sent to interrupt destination 2. For additional programming see Section 4.4.1.1.

Figure 88. RX CPPI Interrupt Condition Routing Registers

RX CPPI Interrupt Condition Routing Register (RX_CPPI_ICRR) (Address Offset 02C0h)

31–28	27–24	23–20	19–16
ICR7	ICR6	ICR5	ICR4
R/W-0000	R/W-0000	R/W-0000	R/W-0000

15–12	11–8	7–4	3–0
ICR3	ICR2	ICR1	ICR0
R/W-0000	R/W-0000	R/W-0000	R/W-0000

RX CPPI Interrupt Condition Routing Register 2 (RX_CPPI_ICRR2) (Address Offset 02C4h)

31–28	27–24	23–20	19–16
ICR15	ICR14	ICR13	ICR12
R/W-0000	R/W-0000	R/W-0000	R/W-0000

15–12	11–8	7–4	3–0
ICR11	ICR10	ICR9	ICR8
R/W-0000	R/W-0000	R/W-0000	R/W-0000

LEGEND: R/W = Read/Write; *-n* = Value after reset

Table 84. RX CPPI Interrupt Condition Routing Register Field Descriptions

Field	Value	Description
ICR*x* (x = 0 to 15)		Interrupt condition routing. Routes the interrupt request from RX buffer descriptor queue *x* to one of eight interrupt destinations (INTDST0-INTDST7).
	0000b	INTDST0
	0001b	INTDST1
	0010b	INTDST2
	0011b	INTDST3
	0100b	INTDST4
	0101b	INTDST5
	0110b	INTDST6
	0111b	INTDST7
	1xxxb	Reserved

5.30 TX CPPI Interrupt Condition Routing Registers (TX_CPPI_ICRR and TX_CPPI_ICRR2)

Figure 89 and Table 85 summarize the ICRRs for the TXU. These registers route queue interrupts to interrupt destinations. For example, if ICS6 = 1 in TX_CPPI_ICSR and ICR6 = 0011b in TX_CPPI_ICRR, the interrupt request from TX buffer descriptor queue 6 is sent to interrupt destination 3. For additional programming see Section 4.4.1.1.

Figure 89. TX CPPI Interrupt Condition Routing Registers

TX CPPI Interrupt Condition Routing Register (TX_CPPI_ICRR) (Address Offset 02D0h)

31–28	27–24	23–20	19–16
ICR7	ICR6	ICR5	ICR4
R/W-0000	R/W-0000	R/W-0000	R/W-0000

15–12	11–8	7–4	3–0
ICR3	ICR2	ICR1	ICR0
R/W-0000	R/W-0000	R/W-0000	R/W-0000

TX CPPI Interrupt Condition Routing Register 2 (TX_CPPI_ICRR2) (Address Offset 02D4h)

31–28	27–24	23–20	19–16
ICR15	ICR14	ICR13	ICR12
R/W-0000	R/W-0000	R/W-0000	R/W-0000

15–12	11–8	7–4	3–0
ICR11	ICR10	ICR9	ICR8
R/W-0000	R/W-0000	R/W-0000	R/W-0000

LEGEND: R/W = Read/Write; *-n* = Value after reset

Table 85. TX CPPI Interrupt Condition Routing Register Field Descriptions

Field	Value	Description
ICR*x* (x = 0 to 15)		Interrupt condition routing. Routes the interrupt request from TX buffer descriptor queue *x* to one of eight interrupt destinations (INTDST0-INTDST7).
	0000b	INTDST0
	0001b	INTDST1
	0010b	INTDST2
	0011b	INTDST3
	0100b	INTDST4
	0101b	INTDST5
	0110b	INTDST6
	0111b	INTDST7
	1xxxb	Reserved

5.31 LSU Interrupt Condition Routing Registers (LSU_ICRR0-LSU_ICRR3)

Figure 90 shows the ICRRs for the LSU interrupt requests, and Table 86 shows the general description for an ICRx field in any of the four registers. These registers route LSU interrupt requests from LSU_ICSR to interrupt destinations. For example, if ICS4 = 1 in LSU_ICSR and ICR4 = 0000b in LSU_ICRR0, LSU1 has generated a transaction-timeout interrupt request, and that request is routed to interrupt destination 0. For additional programming see Section 4.4.1.2.

Figure 90. LSU Interrupt Condition Routing Registers

LSU Interrupt Condition Routing Register 0 (LSU_ICRR0) (Address Offset 02E0h)

31–28	27–24	23–20	19–16
ICR7	ICR6	ICR5	ICR4
R/W-0000	R/W-0000	R/W-0000	R/W-0000

15–12	11–8	7–4	3–0
ICR3	ICR2	ICR1	ICR0
R/W-0000	R/W-0000	R/W-0000	R/W-0000

LSU Interrupt Condition Routing Register 1 (LSU_ICRR1) (Address Offset 02E4h)

31–28	27–24	23–20	19–16
ICR15	ICR14	ICR13	ICR12
R/W-0000	R/W-0000	R/W-0000	R/W-0000

15–12	11–8	7–4	3–0
ICR11	ICR10	ICR9	ICR8
R/W-0000	R/W-0000	R/W-0000	R/W-0000

LSU Interrupt Condition Routing Register 2 (LSU_ICRR2) (Address Offset 02E8h)

31–28	27–24	23–20	19–16
ICR23	ICR22	ICR21	ICR20
R/W-0000	R/W-0000	R/W-0000	R/W-0000

15–12	11–8	7–4	3–0
ICR19	ICR18	ICR17	ICR16
R/W-0000	R/W-0000	R/W-0000	R/W-0000

LSU Interrupt Condition Routing Register 3 (LSU_ICRR3) (Address Offset 02ECh)

31–28	27–24	23–20	19–16
ICR31	ICR30	ICR29	ICR28
R/W-0000	R/W-0000	R/W-0000	R/W-0000

15–12	11–8	7–4	3–0
ICR27	ICR26	ICR25	ICR24
R/W-0000	R/W-0000	R/W-0000	R/W-0000

LEGEND: R/W = Read/Write; -*n* = Value after reset

Table 86. LSU Interrupt Condition Routing Register Field Descriptions

Field	Value	Description
ICR*x* (x = 0 to 31)		Interrupt condition routing. Routes the associated LSU interrupt request to one of eight interrupt destinations (INTDST0-INTDST7). Bits ICR0-ICR7 are for LSU1; bits ICR8-ICR15, for LSU2; bits ICR16-ICR23, for LSU3; bits ICR24-ICR31, for LSU4.
	0000b	INTDST0
	0001b	INTDST1
	0010b	INTDST2
	0011b	INTDST3
	0100b	INTDST4
	0101b	INTDST5
	0110b	INTDST6
	0111b	INTDST7
	1xxxb	Reserved

5.32 Error, Reset, and Special Event Interrupt Condition Routing Registers (ERR_RST_EVNT_ICRR, ERR_RST_EVNT_ICRR2, and ERR_RST_EVNT_ICRR3)

The ICRRs shown in Figure 91 route port interrupt requests from ERR_RST_EVNT_ICSR to interrupt destinations. For example, if ICS8 = 1 in ERR_RST_EVNT_ICSR and ICR8 = 0001b in ERR_RST_EVNT_ICRR2, port 0 has generated an error interrupt request, and that request is routed to interrupt destination 1. Table 87 gives a general description for an ICRx field in any of the three registers. For additional programming see Section 4.4.1.3.

Figure 91. Error, Reset, and Special Event Interrupt Condition Routing Registers

Error, Reset, and Special Event ICRR (ERR_RST_EVNT_ICRR) (Address Offset 02F0h)

31	12	11	8	7	4	3	0
Reserved		ICR2		ICR1		ICR0	
R-0		R/W-0000		R/W-0000		R/W-0000	

Error, Reset, & Special Event ICRR 2 (ERR_RST_EVNT_ICRR2) (Address Offset 02F4h)

31	16
Reserved	
R-0	

15	12	11	8	7	4	3	0
ICR11		ICR10		ICR9		ICR8	
R/W-0000		R/W-0000		R/W-0000		R/W-0000	

Error, Reset, and Special Event ICRR 3 (ERR_RST_EVNT_ICRR3) (Address Offset 02F8h)

31	4	3	0
Reserved		ICR16	
R-0		R/W-0000	

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 87. Error, Reset, and Special Event Interrupt Condition Routing Register Field Descriptions

Field	Value	Description
ICR*x* (x = 0 to 2, 8 to 11, and 16)		Interrupt condition routing. Routes the associated port interrupt request to one of eight interrupt destinations (INTDST0-INTDST7).
	0000b	INTDST0
	0001b	INTDST1
	0010b	INTDST2
	0011b	INTDST3
	0100b	INTDST4
	0101b	INTDST5
	0110b	INTDST6
	0111b	INTDST7
	1xxxb	Reserved

5.33 Interrupt Status Decode Register (INTDSTn_DECODE)

There are eight of these registers, one for each interrupt destination (see Table 88). This type of register is shown in Figure 92 and described in Table 89. Interrupt sources are mapped to an interrupt decode register only if the ICRRs routes the interrupt source to the corresponding physical interrupt. Each status decode bit is a logical OR of multiple interrupt sources that are mapped to the same bit. For additional programming see Section 4.5.

Table 88. INTDST*n*_DECODE Registers and the Associated Interrupt Destinations

Register	Address Offset	Associated Interrupt Destination
INTDST0_DECODE	0300h	INTDST0
INTDST1_DECODE	0304h	INTDST1
INTDST2_DECODE	0308h	INTDST2
INTDST3_DECODE	030Ch	INTDST3
INTDST4_DECODE	0310h	INTDST4
INTDST5_DECODE	0314h	INTDST5
INTDST6_DECODE	0318h	INTDST6
INTDST7_DECODE	031Ch	INTDST7

Figure 92. Interrupt Status Decode Register (INTDST*n*_DECODE)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ISD31	ISD30	ISD29	ISD28	ISD27	ISD26	ISD25	ISD24	ISD23	ISD22	ISD21	ISD20	ISD19	ISD18	ISD17	ISD16
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ISD15	ISD14	ISD13	ISD12	ISD11	ISD10	ISD9	ISD8	ISD7	ISD6	ISD5	ISD4	ISD3	ISD2	ISD1	ISD0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read, W = Write, -*n* = Value after reset

Table 89. Interrupt Status Decode Register (INTDST*n*_DECODE) Field Descriptions

Bit	Field	Value	Description
31	ISD31	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • An LSU (check LSU_ICSR) • TX buffer descriptor queue 0 (bit 0 of TX_CPPI_ICSR) • RX buffer descriptor queue 0 (bit 0 of RX_CPPI_ICSR)
30	ISD30	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • A port (check ERR_RST_EVNT_ICSR) • TX buffer descriptor queue 1 (bit 1 of TX_CPPI_ICSR) • RX buffer descriptor queue 1 (bit 1 of RX_CPPI_ICSR)
29	ISD29	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • TX buffer descriptor queue 2 (bit 2 of TX_CPPI_ICSR) • RX buffer descriptor queue 2 (bit 2 of RX_CPPI_ICSR)
28	ISD28	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • TX buffer descriptor queue 3 (bit 3 of TX_CPPI_ICSR) • RX buffer descriptor queue 3 (bit 3 of RX_CPPI_ICSR)

Table 89. Interrupt Status Decode Register (INTDST*n*_DECODE) Field Descriptions (continued)

Bit	Field	Value	Description
27	ISD27	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • TX buffer descriptor queue 4 (bit 4 of TX_CPPI_ICSR) • RX buffer descriptor queue 4 (bit 4 of RX_CPPI_ICSR)
26	ISD26	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • TX buffer descriptor queue 5 (bit 5 of TX_CPPI_ICSR) • RX buffer descriptor queue 5 (bit 5 of RX_CPPI_ICSR)
25	ISD25	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • TX buffer descriptor queue 6 (bit 6 of TX_CPPI_ICSR) • RX buffer descriptor queue 6 (bit 6 of RX_CPPI_ICSR)
24	ISD24	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • TX buffer descriptor queue 7 (bit 7 of TX_CPPI_ICSR) • RX buffer descriptor queue 7 (bit 7 of RX_CPPI_ICSR)
23	ISD23	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • TX buffer descriptor queue 8 (bit 8 of TX_CPPI_ICSR) • RX buffer descriptor queue 8 (bit 8 of RX_CPPI_ICSR)
22	ISD22	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • TX buffer descriptor queue 9 (bit 9 of TX_CPPI_ICSR) • RX buffer descriptor queue 9 (bit 9 of RX_CPPI_ICSR)
21	ISD21	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • TX buffer descriptor queue 10 (bit 10 of TX_CPPI_ICSR) • RX buffer descriptor queue 10 (bit 10 of RX_CPPI_ICSR)
20	ISD20	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • TX buffer descriptor queue 11 (bit 11 of TX_CPPI_ICSR) • RX buffer descriptor queue 11 (bit 11 of RX_CPPI_ICSR)
19	ISD19	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • TX buffer descriptor queue 12 (bit 12 of TX_CPPI_ICSR) • RX buffer descriptor queue 12 (bit 12 of RX_CPPI_ICSR)
18	ISD18	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • TX buffer descriptor queue 13 (bit 13 of TX_CPPI_ICSR) • RX buffer descriptor queue 13 (bit 13 of RX_CPPI_ICSR)
17	ISD17	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • TX buffer descriptor queue 14 (bit 14 of TX_CPPI_ICSR) • RX buffer descriptor queue 14 (bit 14 of RX_CPPI_ICSR)
16	ISD16	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • TX buffer descriptor queue 15 (bit 15 of TX_CPPI_ICSR) • RX buffer descriptor queue 15 (bit 15 of RX_CPPI_ICSR)

Table 89. Interrupt Status Decode Register (INTDST*n*_DECODE) Field Descriptions (continued)

Bit	Field	Value	Description
15	ISD15	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • Doorbell 0, bit 15 (bit 15 of DOORBELL0_ICSR) • Doorbell 1, bit 15 (bit 15 of DOORBELL1_ICSR) • Doorbell 2, bit 15 (bit 15 of DOORBELL2_ICSR) • Doorbell 3, bit 15 (bit 15 of DOORBELL3_ICSR)
14	ISD14	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • Doorbell 0, bit 14 (bit 14 of DOORBELL0_ICSR) • Doorbell 1, bit 14 (bit 14 of DOORBELL1_ICSR) • Doorbell 2, bit 14 (bit 14 of DOORBELL2_ICSR) • Doorbell 3, bit 14 (bit 14 of DOORBELL3_ICSR)
13	ISD13	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • Doorbell 0, bit 13 (bit 13 of DOORBELL0_ICSR) • Doorbell 1, bit 13 (bit 13 of DOORBELL1_ICSR) • Doorbell 2, bit 13 (bit 13 of DOORBELL2_ICSR) • Doorbell 3, bit 13 (bit 13 of DOORBELL3_ICSR)
12	ISD12	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • Doorbell 0, bit 12 (bit 12 of DOORBELL0_ICSR) • Doorbell 1, bit 12 (bit 12 of DOORBELL1_ICSR) • Doorbell 2, bit 12 (bit 12 of DOORBELL2_ICSR) • Doorbell 3, bit 12 (bit 12 of DOORBELL3_ICSR)
11	ISD11	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • Doorbell 0, bit 11 (bit 11 of DOORBELL0_ICSR) • Doorbell 1, bit 11 (bit 11 of DOORBELL1_ICSR) • Doorbell 2, bit 11 (bit 11 of DOORBELL2_ICSR) • Doorbell 3, bit 11 (bit 11 of DOORBELL3_ICSR)
10	ISD10	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • Doorbell 0, bit 10 (bit 10 of DOORBELL0_ICSR) • Doorbell 1, bit 10 (bit 10 of DOORBELL1_ICSR) • Doorbell 2, bit 10 (bit 10 of DOORBELL2_ICSR) • Doorbell 3, bit 10 (bit 10 of DOORBELL3_ICSR)
9	ISD9	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • Doorbell 0, bit 9 (bit 9 of DOORBELL0_ICSR) • Doorbell 1, bit 9 (bit 9 of DOORBELL1_ICSR) • Doorbell 2, bit 9 (bit 9 of DOORBELL2_ICSR) • Doorbell 3, bit 9 (bit 9 of DOORBELL3_ICSR)
8	ISD8	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • Doorbell 0, bit 8 (bit 8 of DOORBELL0_ICSR) • Doorbell 1, bit 8 (bit 8 of DOORBELL1_ICSR) • Doorbell 2, bit 8 (bit 8 of DOORBELL2_ICSR) • Doorbell 3, bit 8 (bit 8 of DOORBELL3_ICSR)

Table 89. Interrupt Status Decode Register (INTDST*n*_DECODE) Field Descriptions (continued)

Bit	Field	Value	Description
7	ISD7	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • Doorbell 0, bit 7 (bit 7 of DOORBELL0_ICSR) • Doorbell 1, bit 7 (bit 7 of DOORBELL1_ICSR) • Doorbell 2, bit 7 (bit 7 of DOORBELL2_ICSR) • Doorbell 3, bit 7 (bit 7 of DOORBELL3_ICSR)
6	ISD6	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • Doorbell 0, bit 6 (bit 6 of DOORBELL0_ICSR) • Doorbell 1, bit 6 (bit 6 of DOORBELL1_ICSR) • Doorbell 2, bit 6 (bit 6 of DOORBELL2_ICSR) • Doorbell 3, bit 6 (bit 6 of DOORBELL3_ICSR)
5	ISD5	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • Doorbell 0, bit 5 (bit 5 of DOORBELL0_ICSR) • Doorbell 1, bit 5 (bit 5 of DOORBELL1_ICSR) • Doorbell 2, bit 5 (bit 5 of DOORBELL2_ICSR) • Doorbell 3, bit 5 (bit 5 of DOORBELL3_ICSR)
4	ISD4	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • Doorbell 0, bit 4 (bit 4 of DOORBELL0_ICSR) • Doorbell 1, bit 4 (bit 4 of DOORBELL1_ICSR) • Doorbell 2, bit 4 (bit 4 of DOORBELL2_ICSR) • Doorbell 3, bit 4 (bit 4 of DOORBELL3_ICSR)
3	ISD3	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • Doorbell 0, bit 3 (bit 3 of DOORBELL0_ICSR) • Doorbell 1, bit 3 (bit 3 of DOORBELL1_ICSR) • Doorbell 2, bit 3 (bit 3 of DOORBELL2_ICSR) • Doorbell 3, bit 3 (bit 3 of DOORBELL3_ICSR)
2	ISD2	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • Doorbell 0, bit 2 (bit 2 of DOORBELL0_ICSR) • Doorbell 1, bit 2 (bit 2 of DOORBELL1_ICSR) • Doorbell 2, bit 2 (bit 2 of DOORBELL2_ICSR) • Doorbell 3, bit 2 (bit 2 of DOORBELL3_ICSR)
1	ISD1	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • Doorbell 0, bit 1 (bit 1 of DOORBELL0_ICSR) • Doorbell 1, bit 1 (bit 1 of DOORBELL1_ICSR) • Doorbell 2, bit 1 (bit 1 of DOORBELL2_ICSR) • Doorbell 3, bit 1 (bit 1 of DOORBELL3_ICSR)
0	ISD0	0	No interrupt request routed to this bit.
		1	Interrupt request detected. Possible interrupt sources: • Doorbell 0, bit 0 (bit 0 of DOORBELL0_ICSR) • Doorbell 1, bit 0 (bit 0 of DOORBELL1_ICSR) • Doorbell 2, bit 0 (bit 0 of DOORBELL2_ICSR) • Doorbell 3, bit 0 (bit 0 of DOORBELL3_ICSR)

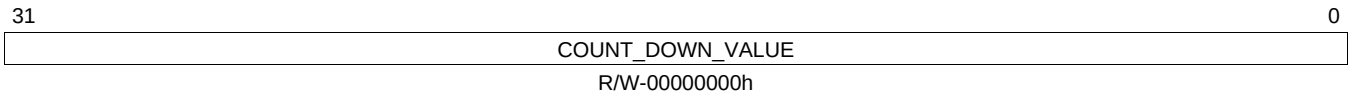
5.34 INTDSTn Interrupt Rate Control Register (INTDSTn_RATE_CNTL)

There are eight interrupt rate control registers, one for each interrupt destination (see Table 90). Figure 93 and Table 91 provide a general description for an interrupt rate control register. These registers are used to set the rate at which an interrupt can be generated for each interrupt destination. A write to one of the registers reloads a counter and immediately starts the counter decrementing. When the counter value reaches 0 (after counting down or after a CPU write of 0), the interrupt logic generates a single interrupt pulse if any bits in the corresponding ICSR are set (or become set after the zero count is reached). For additional programming see Section 4.7.

Table 90. INTDSTn_RATE_CNTL Registers and the Associated Interrupt Destinations

Register	Address Offset	Associated Interrupt Destination
INTDST0_RATE_CNTL	0320h	INTDST0
INTDST1_RATE_CNTL	0324h	INTDST1
INTDST2_RATE_CNTL	0328h	INTDST2
INTDST3_RATE_CNTL	032Ch	INTDST3
INTDST4_RATE_CNTL	0330h	INTDST4
INTDST5_RATE_CNTL	0334h	INTDST5
INTDST6_RATE_CNTL	0338h	INTDST6
INTDST7_RATE_CNTL	033Ch	INTDST7

Figure 93. INTDSTn Interrupt Rate Control Register (INTDSTn_RATE_CNTL)



LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 91. INTDSTn Interrupt Rate Control Register (INTDSTn_RATE_CNTL) Field Descriptions

Bit	Field	Value	Description
31-0	COUNT_DOWN_VALUE	00000000h to FFFFFFFFh	The value written to this field is immediately transferred to the interrupt rate counter, which starts counting down (or causes an interrupt if 0 is written).

5.35 LSU*n* Control Register 0 (LSU*n*_REG0)

There are four of these registers, one for each LSU (see Table 92). The general description for an LSU control register 0 is shown in Figure 94 and described in Table 93. For additional programming see Section 2.3.3.

Table 92. LSU*n*_REG0 Registers and the Associated LSUs

Register	Address Offset	Associated LSU
LSU1_REG0	0400h	LSU1
LSU2_REG0	0420h	LSU2
LSU3_REG0	0440h	LSU3
LSU4_REG0	0460h	LSU4

Figure 94. LSU*n* Control Register 0 (LSU*n*_REG0)

31	0
ADDRESS_MSB	
R/W-00h	

LEGEND: R/W = Read/Write; *-n* = Value after reset

Table 93. LSU*n* Control Register 0 (LSU*n*_REG0) Field Descriptions

Bit	Field	Value	Description
31-0	ADDRESS_MSB	00000000h to FFFFFFFFh	32-bit most significant bits of an extended address specified through LSU*n*.

5.36 LSU*n* Control Register 1 (LSU*n*_REG1)

There are four of these registers, one for each LSU (see Table 94). This register's content is shown in Figure 95 and described in Table 95. For additional programming see Section 2.3.3.

Table 94. LSU*n*_REG1 Registers and the Associated LSUs

Register	Address Offset	Associated LSU
LSU1_REG1	0404h	LSU1
LSU2_REG1	0424h	LSU2
LSU3_REG1	0444h	LSU3
LSU4_REG1	0464h	LSU4

Figure 95. LSU*n* Control Register 1 (LSU*n*_REG1)

31	0
ADDRESS_LSB/CONFIG_OFFSET	
R/W-00000000h	

LEGEND: R/W = Read/Write; -*n* = Value after reset

Table 95. LSU*n* Control Register 1 (LSU*n*_REG1) Field Descriptions

Bit	Field	Value	Description
31-0	ADDRESS_LSB/CONFIG_OFFSET	00000000h to FFFFFFFFh	**For packet types 2, 5, and 6:** The 32-bit destination address or the 32 least significant bits of an extended destination address. This value is used in conjunction with BYTE_COUNT to create a 64-bit aligned RapidIO packet header address.
			For packet type 8 (maintenance packet):
		00000000h to 00FFFFFFh	The right-aligned 24-bit register configuration offset. This value is used in conjunction with BYTE_COUNT to create a 64-bit aligned RapidIO packet header Config_offset value. The 2 LSBs of this field must be 0s because the smallest configuration access is 4 bytes.

5.37 LSU*n* Control Register 2 (LSU*n*_REG2)

There are four of these registers, one for each LSU (see Table 96). LSU*n*_REG2 is shown in Figure 96 and described in Table 97. For additional programming see Section 2.3.3.

Table 96. LSU*n*_REG2 Registers and the Associated LSUs

Register	Address Offset	Associated LSU
LSU1_REG2	0408h	LSU1
LSU2_REG2	0428h	LSU2
LSU3_REG2	0448h	LSU3
LSU4_REG2	0468h	LSU4

Figure 96. LSU*n* Control Register 2 (LSU*n*_REG2)

31	0
DSP_ADDRESS	
R/W-00000000h	

LEGEND: R/W = Read/Write; -*n* = Value after reset

Table 97. LSU*n* Control Register 2 (LSU*n*_REG2) Field Descriptions

Bit	Field	Value	Description
31-0	DSP_ADDRESS	00000000h to FFFFFFFFh	32-bit DSP byte address for the source of the LSU transaction

5.38 LSU*n* Control Register 3 (LSU*n*_REG3)

There are four of these registers, one for each LSU (see Table 98). LSU*n*_REG3 is shown in Figure 97 and described in Table 99. For additional programming see Section 2.3.3.

Table 98. LSU*n*_REG3 Registers and the Associated LSUs

Register	Address Offset	Associated LSU
LSU1_REG3	040Ch	LSU1
LSU2_REG3	042Ch	LSU2
LSU3_REG3	044Ch	LSU3
LSU4_REG3	046Ch	LSU4

Figure 97. LSU*n* Control Register 3 (LSU*n*_REG3)

31	16
Reserved	
R-0000h	

15	12	11	0
Reserved		BYTE_COUNT	
R-0h		R/W-000h	

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 99. LSU*n* Control Register 3 (LSU*n*_REG3) Field Descriptions

Bit	Field	Value	Description
31-12	Reserved	00000h	These read-only bits return 0s when read.
11-0	BYTE_COUNT	000h-FFFh	Number of data bytes to read or write, up to 4K bytes. This value is used in conjunction with the specified RapidIO address to create the data size and word pointer fields in the RapidIO packet header.

5.39 LSU*n* Control Register 4 (LSU*n*_REG4)

There are four of these registers, one for each LSU (see Table 100). LSU*n*_REG4 is shown in Figure 98 and described in Table 101. For additional programming see Section 2.3.3.

Table 100. LSU*n*_REG4 Registers and the Associated LSUs

Register	Address Offset	Associated LSU
LSU1_REG4	0410h	LSU1
LSU2_REG4	0430h	LSU2
LSU3_REG4	0450h	LSU3
LSU4_REG4	0470h	LSU4

Figure 98. LSU*n* Control Register 4 (LSU*n*_REG4)

31–30	29–28	27–26	25–24	23
OUTPORTID	PRIORITY	XAMSB	ID_SIZE	DESTID
R/W-00	R/W-00	R/W-00	R/W-00	R/W-0000h

8	7–1	0
DESTID	Reserved	INTERRUPT_REQ
R/W-0000h	R-00h	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 101. LSU*n* Control Register 4 (LSU*n*_REG4) Field Descriptions

Bit	Field	Value	Description
31-30	OUTPORTID	00b-11b	Indicates the number of the output port (0, 1, 2, or 3) from which the packet is to be transmitted. Specified by the CPU along with the node ID. The output port value is not included in the RapidIO header.
29-28	PRIORITY	00b-11b	Supplies the prio field of the RapidIO packet header to indicate packet priority. To avoid system deadlock, it is recommended that request packets not be sent with priority level 3. It is the responsibility of the software to assign the appropriate outgoing priority.
27-26	XAMSB	00b-11b	Supplies the xamsb field of the RapidIO packet header to specify the 2 MSBs of the extended RapidIO address.
25-24	ID_SIZE		Supplies the tt field of the RapidIO packet header to specify whether 8-bit or 16-bit DeviceIDs are used.
		00b	8 bit device IDs
		01b	16 bit device IDs
		1xb	Reserved
23-8	DESTID	0000h	Supplies the destination ID field of the RapidIO packet header to specifying target device.
7-1	Reserved	00h	These read-only bits return 0s when read.
0	INTERRUPT_REQ		Indicates whether the CPU requests an interrupt upon completion of the LSU command. This is a CPU-controlled request bit and is typically used in conjunction with non-posted commands to alert the CPU when the requested data/status is present.
		0	Interrupt not requested upon completion of command
		1	Interrupt requested upon completion of command

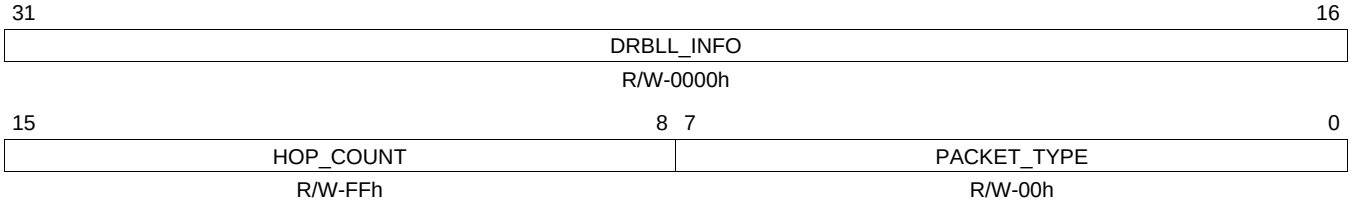
5.40 LSU*n* Control Register 5 (LSU*n*_REG5)

There are four of these registers, one for each LSU (see Table 102). LSU*n*_REG5 is shown in Figure 99 and described in Table 103. For additional programming see Section 2.3.3.

Table 102. LSU*n*_REG5 Registers and the Associated LSUs

Register	Address Offset	Associated LSU
LSU1_REG5	0414h	LSU1
LSU2_REG5	0434h	LSU2
LSU3_REG5	0454h	LSU3
LSU4_REG5	0474h	LSU4

Figure 99. LSU*n* Control Register 5 (LSU*n*_REG5)



LEGEND: R/W = Read/Write; *-n* = Value after reset

Table 103. LSU*n* Control Register 5 (LSU*n*_REG5) Field Descriptions

Bit	Field	Value	Description
31-16	DRBLL_INFO	0000h-FFFFh	RapidIO doorbell info field for type 10 packets. (see Table 23)
15-8	HOP_COUNT	00h-FFh	RapidIO hop count field specified for type 8 (maintenance) packets
7-0	PACKET_TYPE	00h-FFh	The 4 MSBs specify the ftype field for all packet types, and the 4 LSBs specify the trans field for packet types 2, 5, and 8. See Section 2.1.2.4

5.41 LSU*n* Control Register 6 (LSU*n*_REG6)

There are four of these registers, one for each LSU (see Table 104). LSU*n*_REG6 is shown in Figure 100 and described in Table 105. For additional programming see Section 2.3.3.

Table 104. LSU*n*_REG6 Registers and the Associated LSUs

Register	Address Offset	Associated LSU
LSU1_REG6	0418h	LSU1
LSU2_REG6	0438h	LSU2
LSU3_REG6	0458h	LSU3
LSU4_REG6	0478h	LSU4

Figure 100. LSU*n* Control Register 6 (LSU*n*_REG6)

31	16
Reserved	
R-0000h	

15	5	4	1	0
Reserved		COMPLETION_CODE		BSY
R-000h		R-0000		R-0

LEGEND: R = Read only; -*n* = Value after reset

Table 105. LSU*n* Control Register 6 (LSU*n*_REG6) Field Descriptions

Bit	Field	Value	Description
31-5	Reserved	0000h	These read-only bits return 0s when read.
4-1	COMPLETION_CODE		Indicates the status of the pending command.
		0000b	Transaction complete, no errors (posted/non-posted)
		0001b	Transaction timeout occurred on non-posted transaction
		0010 b	Transaction complete, packet not sent due to flow control blockade (Xoff)
		0011b	Transaction complete, non-posted response packet (type 8 and 13) contained ERROR status, or response payload length was in error
		0100b	Transaction complete, packet not sent due to unsupported transaction type or invalid programming encoding for one or more LSU register fields
		0101b	DMA data transfer error
		0110b	"Retry" DOORBELL response received, or Atomic test-and-swap was not allowed (semaphore in use)
		0111b	Transaction complete, packet not sent due to unavailable outbound credit at given priority
		1xxxb	Reserved
0	BSY		Indicates status of the writeable LSU registers
		0	LSU registers available (writable) for next set of transfer descriptors
		1	LSU registers busy with current transfer

5.42 LSU*n* Congestion Control Flow Mask Register (LSU*n*_FLOW_MASKS)

There are four of these registers, one for each LSU (see Table 106). The fields of an LSU*n*_FLOW_MASKS register are summarized by Figure 101 and described in Table 107. The 16 bits within each FLOW_MASK field are summarized by Figure 102 and Table 108. For additional programming see Section 2.3.8.

Table 106. LSU*n*_FLOW_MASKS Registers and the Associated LSUs

Register	Address Offset	LSU
LSU1_FLOW_MASKS	041Ch	LSU1
LSU2_FLOW_MASKS	043Ch	LSU2
LSU3_FLOW_MASKS	045Ch	LSU3
LSU4_FLOW_MASKS	047Ch	LSU4

Figure 101. LSU*n* Congestion Control Flow Mask Register (LSU*n*_FLOW_MASKS)

31	16	15	0
Reserved		FLOW_MASK	
R-0000h		R/W-FFFFh	

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 107. LSU*n* Congestion Control Flow Mask Register (LSU*n*_FLOW_MASKS) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0000h	These read-only bits return 0s when read.
15-0	FLOW_MASK	0000-FFFFh	Flow mask for LSU*n*

Figure 102. LSU*n* FLOW_MASK Fields

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FL15	FL14	FL13	FL12	FL11	FL10	FL9	FL8	FL7	FL6	FL5	FL4	FL3	FL2	FL1	FL0
R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1

LEGEND: R = Read; W = Write; -*n* = Value after reset

Table 108. LSU*n* FLOW_MASK Fields

Bit	Field	Value	Description
15	FL15	0	LSU*n* does not support Flow 15 from table entry
		1	LSU*n* supports Flow 15 from table entry
14	FL14	0	LSU*n* does not support Flow 14 from table entry
		1	LSU*n* supports Flow 14 from table entry
13	FL13	0	LSU*n* does not support Flow 13 from table entry
		1	LSU*n* supports Flow 13 from table entry
12	FL12	0	LSU*n* does not support Flow 12 from table entry
		1	LSU*n* supports Flow 12 from table entry
11	FL11	0	LSU*n* does not support Flow 11 from table entry
		1	LSU*n* supports Flow 11 from table entry
10	FL10	0	LSU*n* does not support Flow 10 from table entry
		1	LSU*n* supports Flow 10 from table entry
9	FL9	0	LSU*n* does not support Flow 9 from table entry
		1	LSU*n* supports Flow 9 from table entry
8	FL8	0	LSU*n* does not support Flow 8 from table entry
		1	LSU*n* supports Flow 8 from table entry

Table 108. LSU*n* FLOW_MASK Fields (continued)

Bit	Field	Value	Description
7	FL7	0	LSU*n* does not support Flow 7 from table entry
		1	LSU*n* supports Flow 7 from table entry
6	FL6	0	LSU*n* does not support Flow 6 from table entry
		1	LSU*n* supports Flow 6 from table entry
5	FL5	0	LSU*n* does not support Flow 5 from table entry
		1	LSU*n* supports Flow 5 from table entry
4	FL4	0	LSU*n* does not support Flow 4 from table entry
		1	LSU*n* supports Flow 4 from table entry
3	FL3	0	LSU*n* does not support Flow 3 from table entry
		1	LSU*n* supports Flow 3 from table entry
2	FL2	0	LSU*n* does not support Flow 2 from table entry
		1	LSU*n* supports Flow 2 from table entry
1	FL1	0	LSU*n* does not support Flow 1 from table entry
		1	LSU*n* supports Flow 1 from table entry
0	FL0	0	LSU*n* does not support Flow 0 from table entry
		1	LSU*n* supports Flow 0 from table entry

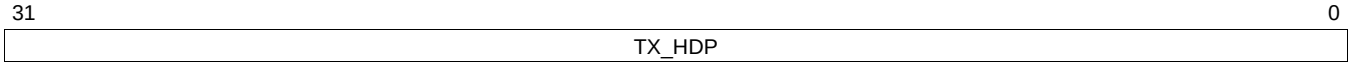
5.43 Queue *n* Transmit DMA Head Descriptor Pointer Register (QUEUE*n*_TXDMA_HDP)

There are sixteen of these registers (see Table 109). QUEUE*n*_TXDMA_HDP is shown in Figure 103 and described in Table 110. For additional programming information, see Section 2.3.4.2 .

Table 109. QUEUE*n*_TXDMA_HDP Registers

Register	Address Offset
QUEUE0_TXDMA_HDP	0500h
QUEUE1_TXDMA_HDP	0504h
QUEUE2_TXDMA_HDP	0508h
QUEUE3_TXDMA_HDP	050Ch
QUEUE4_TXDMA_HDP	0510h
QUEUE5_TXDMA_HDP	0514h
QUEUE6_TXDMA_HDP	0518h
QUEUE7_TXDMA_HDP	051Ch
QUEUE8_TXDMA_HDP	0520h
QUEUE9_TXDMA_HDP	0524h
QUEUE10_TXDMA_HDP	0528h
QUEUE11_TXDMA_HDP	052Ch
QUEUE12_TXDMA_HDP	0530h
QUEUE13_TXDMA_HDP	0534h
QUEUE14_TXDMA_HDP	0538h
QUEUE15_TXDMA_HDP	053Ch

Figure 103. Queue *n* Transmit DMA Head Descriptor Pointer Register (QUEUE*n*_TXDMA_HDP)



R/W-00000000h

LEGEND: R/W = Read/Write; -*n* = Value after reset

Table 110. Queue *n* Transmit DMA Head Descriptor Pointer Register (QUEUE*n*_TXDMA_HDP) Field Descriptions

Bit	Field	Value	Description
31-0	TX_HDP	00000000h to FFFFFFFCh	This field is the memory address for the first buffer descriptor in the transmit queue. This field is written by the DSP core to initiate queue transmit operations and is zeroed by the port when all packets in the queue have been transmitted. An error condition results if the DSP core writes this field when the current field value is nonzero. The address must be 32-bit word aligned (the 2 LSBs must be 0s).

5.44 Queue *n* Transmit DMA Completion Pointer Register (QUEUE*n*_TXDMA_CP)

There are sixteen of these registers (see Table 111). QUEUE*n*_TXDMA_CP is shown in Figure 104 and described in Table 112. For additional programming information, see Section 2.3.4.2 .

Table 111. QUEUE*n*_TXDMA_CP Registers

Register	Address Offset
QUEUE0_TXDMA_CP	0580h
QUEUE1_TXDMA_CP	0584h
QUEUE2_TXDMA_CP	0588h
QUEUE3_TXDMA_CP	058Ch
QUEUE4_TXDMA_CP	0590h
QUEUE5_TXDMA_CP	0594h
QUEUE6_TXDMA_CP	0598h
QUEUE7_TXDMA_CP	059Ch
QUEUE8_TXDMA_CP	05A0h
QUEUE9_TXDMA_CP	05A4h
QUEUE10_TXDMA_CP	05A8h
QUEUE11_TXDMA_CP	05ACh
QUEUE12_TXDMA_CP	05B0h
QUEUE13_TXDMA_CP	05B4h
QUEUE14_TXDMA_CP	05B8h
QUEUE15_TXDMA_CP	05BCh

Figure 104. Queue *n* Transmit DMA Completion Pointer Register (QUEUE*n*_TXDMA_CP)

31	0
TX_CP	
R/W-00000000h	

LEGEND: R/W = Read/Write; -*n* = Value after reset

Table 112. Queue Transmit DMA Completion Pointer Registers (QUEUE*n*_TXDMA_CP) Field Descriptions

Bit	Field	Value	Description
31-0	TX_CP	00000000h to FFFFFFFFh	This field is the memory address for the transmit queue completion pointer. This register is written by the DSP core with the buffer descriptor address for the last buffer processed by the host during interrupt processing. The port uses the value written to determine if the interrupt should be de-asserted.

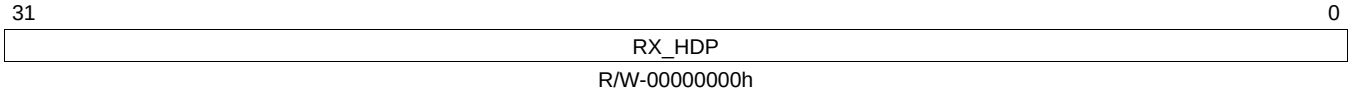
5.45 Queue *n* Receive DMA Head Descriptor Pointer Register (QUEUE*n*_RXDMA_HDP)

There are sixteen of these registers (see Table 113). QUEUE*n*_RXDMA_HDP is shown in Figure 105 and described in Table 114. For additional programming information, see Section 2.3.4.1 .

Table 113. QUEUE*n*_RXDMA_HDP Registers

Register	Address Offset
QUEUE0_RXDMA_HDP	0600h
QUEUE1_RXDMA_HDP	0604h
QUEUE2_RXDMA_HDP	0608h
QUEUE3_RXDMA_HDP	060Ch
QUEUE4_RXDMA_HDP	0610h
QUEUE5_RXDMA_HDP	0614h
QUEUE6_RXDMA_HDP	0618h
QUEUE7_RXDMA_HDP	061Ch
QUEUE8_RXDMA_HDP	0620h
QUEUE9_RXDMA_HDP	0624h
QUEUE10_RXDMA_HDP	0628h
QUEUE11_RXDMA_HDP	062Ch
QUEUE12_RXDMA_HDP	0630h
QUEUE13_RXDMA_HDP	0634h
QUEUE14_RXDMA_HDP	0638h
QUEUE15_RXDMA_HDP	063Ch

Figure 105. Queue *n* Receive DMA Head Descriptor Pointer Register (QUEUE*n*_RXDMA_HDP)



LEGEND: R/W = Read/Write; -*n* = Value after reset

Table 114. Queue *n* Receive DMA Head Descriptor Pointer Register (QUEUE*n*_RXDMA_HDP) Field Descriptions

Bit	Field	Value	Description
31-0	RX_HDP	00000000h to FFFFFFFCh	RX Queue Head Descriptor Pointer: This field is the memory address for the first buffer descriptor in the channel receive queue. This field is written by the DSP core to initiate queue receive operations and is zeroed by the port when all free buffers have been used. An error condition results if the DSP core writes this field when the current field value is nonzero. The address must be 32-bit word aligned (the 2 LSBs must be 0s).

5.46 Queue *n* Receive DMA Completion Pointer Register (QUEUE*n*_RXDMA_CP)

There are sixteen of these registers (see Table 115). QUEUE*n*_RXDMA_CP is shown in Figure 106 and described in Table 116. For additional programming information, see Section 2.3.4.1 .

Table 115. QUEUE*n*_RXDMA_CP Registers

Register	Address Offset
QUEUE0_RXDMA_CP	0680h
QUEUE1_RXDMA_CP	0684h
QUEUE2_RXDMA_CP	0688h
QUEUE3_RXDMA_CP	068Ch
QUEUE4_RXDMA_CP	0690h
QUEUE5_RXDMA_CP	0694h
QUEUE6_RXDMA_CP	0698h
QUEUE7_RXDMA_CP	069Ch
QUEUE8_RXDMA_CP	06A0h
QUEUE9_RXDMA_CP	06A4h
QUEUE10_RXDMA_CP	06A8h
QUEUE11_RXDMA_CP	06ACh
QUEUE12_RXDMA_CP	06B0h
QUEUE13_RXDMA_CP	06B4h
QUEUE14_RXDMA_CP	06B8h
QUEUE15_RXDMA_CP	06BCh

Figure 106. Queue *n* Receive DMA Completion Pointer Register (QUEUE*n*_RXDMA_CP)

31	0
RX_CP	
R/W-00000000h	

LEGEND: R/W = Read/Write; -*n* = Value after reset

Table 116. Queue *n* Receive DMA Completion Pointer Register (QUEUE*n*_RXDMA_CP) Field Descriptions

Bit	Field	Value	Description
31-0	RX_CP	00000000h to FFFFFFFFh	This field is the memory address for the receive queue completion pointer. This register is written by the DSP core with the buffer descriptor address for the last buffer processed by the DSP core during interrupt processing. The port uses the value written to determine if the interrupt should be de-asserted.

5.47 Transmit Queue Teardown Register (TX_QUEUE_TEAR_DOWN)

Each bit in this register corresponds to one of the 16 TX buffer descriptor queues. If a 1 is written to a bit, the teardown process is initiated for the associated queue. TX_QUEUE_TEAR_DOWN is shown in Figure 107 and described in Table 117.

Figure 107. Transmit Queue Teardown Register (TX_QUEUE_TEAR_DOWN) - Address Offset 0700h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0000h															

15	14	13	12	11	10	9	8
QUEUE15_ TEAR_DWN	QUEUE14_ TEAR_DWN	QUEUE13_ TEAR_DWN	QUEUE12_ TEAR_DWN	QUEUE11_ TEAR_DWN	QUEUE10_ TEAR_DWN	QUEUE9_ TEAR_DWN	QUEUE8_ TEAR_DWN
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

7	6	5	4	3	2	1	0
QUEUE7_ TEAR_DWN	QUEUE6_ TEAR_DWN	QUEUE5_ TEAR_DWN	QUEUE4_ TEAR_DWN	QUEUE3_ TEAR_DWN	QUEUE2_ TEAR_DWN	QUEUE1_ TEAR_DWN	QUEUE0_ TEAR_DWN
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: R = Read only; W = Write only; -*n* = Value after reset

Table 117. Transmit Queue Teardown Register (TX_QUEUE_TEAR_DOWN) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	These read-only bits return 0s when read.
15-0	QUEUE*n*_TEAR_DWN (n = 15 to 0)		Queue *n* tear down
		0	No effect
		1	Tear down Queue *n*.

5.48 Transmit CPPI Supported Flow Mask Registers (TX_CPPI_FLOW_MASKS[0-7])

Each of the eight TX CPPI flow mask registers holds the flow masks for two TX descriptor buffer queues (see Table 118). Figure 108 shows the registers, and Figure 109 shows the general form of a flow mask. Each bit of a flow mask selects or deselects a flow for the associated TX queue (see Table 119). For additional programming information, see Section 2.3.8 .

Table 118. TX_CPPI_FLOW_MASKS Registers and the Associated TX Queues

Register	Address Offset	Associated TX Queues
TX_CPPI_FLOW_MASKS0	0704h	Queues 0 and 1
TX_CPPI_FLOW_MASKS1	0708h	Queues 2 and 3
TX_CPPI_FLOW_MASKS2	070Ch	Queues 4 and 5
TX_CPPI_FLOW_MASKS3	0710h	Queues 6 and 7
TX_CPPI_FLOW_MASKS4	0714h	Queues 8 and 9
TX_CPPI_FLOW_MASKS5	0718h	Queues 10 and 11
TX_CPPI_FLOW_MASKS6	071Ch	Queues 12 and 13
TX_CPPI_FLOW_MASKS7	0720h	Queues 14 and 15

Figure 108. Transmit CPPI Supported Flow Mask Registers

Transmit CPPI Supported Flow Mask Register 0 (TX_CPPI_FLOW_MASKS0)

31	16	15	0
QUEUE1_FLOW_MASK		QUEUE0_FLOW_MASK	
R/W-FFh		R/W-FFh	

Transmit CPPI Supported Flow Mask Register 1 (TX_CPPI_FLOW_MASKS1)

31	16	15	0
QUEUE3_FLOW_MASK		QUEUE2_FLOW_MASK	
R/W-FFh		R/W-FFh	

Transmit CPPI Supported Flow Mask Register 2 (TX_CPPI_FLOW_MASKS2)

31	16	15	0
QUEUE5_FLOW_MASK		QUEUE4_FLOW_MASK	
R/W-FFh		R/W-FFh	

Transmit CPPI Supported Flow Mask Register 3 (TX_CPPI_FLOW_MASKS3)

31	16	15	0
QUEUE7_FLOW_MASK		QUEUE6_FLOW_MASK	
R/W-FFh		R/W-FFh	

Transmit CPPI Supported Flow Mask Register 4 (TX_CPPI_FLOW_MASKS4)

31	16	15	0
QUEUE9_FLOW_MASK		QUEUE8_FLOW_MASK	
R/W-FFh		R/W-FFh	

Transmit CPPI Supported Flow Mask Register 5 (TX_CPPI_FLOW_MASKS5)

31	16	15	0
QUEUE11_FLOW_MASK		QUEUE10_FLOW_MASK	
R/W-FFh		R/W-FFh	

Transmit CPPI Supported Flow Mask Register 6 (TX_CPPI_FLOW_MASKS6)

31	16	15	0
QUEUE13_FLOW_MASK		QUEUE12_FLOW_MASK	
R/W-FFh		R/W-FFh	

Transmit CPPI Supported Flow Mask Register 7 (TX_CPPI_FLOW_MASKS7)

31	16	15	0
QUEUE15_FLOW_MASK		QUEUE14_FLOW_MASK	
R/W-FFh		R/W-FFh	

LEGEND: R/W = Read/Write; *-n* = Value after reset

Figure 109. TX Queue *n* FLOW_MASK Fields

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FL15	FL14	FL13	FL12	FL11	FL10	FL9	FL8	FL7	FL6	FL5	FL4	FL3	FL2	FL1	FL0
R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1

LEGEND: R/W = Read/Write; *-n* = Value after reset

Table 119. TX Queue *n* FLOW_MASK Field Descriptions

Bit	Field	Value	Description
15	FL15	0	Queue *n* does not support Flow 15 from table entry
		1	Queue *n* supports Flow 15 from table entry
14	FL14	0	Queue *n* does not support Flow 14 from table entry
		1	Queue *n* supports Flow 14 from table entry
13	FL13	0	Queue *n* does not support Flow 13 from table entry
		1	Queue *n* supports Flow 13 from table entry

Table 119. TX Queue *n* FLOW_MASK Field Descriptions (continued)

Bit	Field	Value	Description
12	FL12	0	Queue *n* does not support Flow 12 from table entry
		1	Queue *n* supports Flow 12 from table entry
11	FL11	0	Queue *n* does not support Flow 11 from table entry
		1	Queue *n* supports Flow 11 from table entry
10	FL10	0	Queue *n* does not support Flow 10 from table entry
		1	Queue *n* supports Flow 10 from table entry
9	FL9	0	Queue *n* does not support Flow 9 from table entry
		1	Queue *n* supports Flow 9 from table entry
8	FL8	0	Queue *n* does not support Flow 8 from table entry
		1	Queue *n* supports Flow 8 from table entry
7	FL7	0	Queue *n* does not support Flow 7 from table entry
		1	Queue *n* supports Flow 7 from table entry
6	FL6	0	Queue *n* does not support Flow 6 from table entry
		1	Queue *n* supports Flow 6 from table entry
5	FL5	0	Queue *n* does not support Flow 5 from table entry
		1	Queue *n* supports Flow 5 from table entry
4	FL4	0	Queue *n* does not support Flow 4 from table entry
		1	Queue *n* supports Flow 4 from table entry
3	FL3	0	Queue *n* does not support Flow 3 from table entry
		1	Queue *n* supports Flow 3 from table entry
2	FL2	0	Queue *n* does not support Flow 2 from table entry
		1	Queue *n* supports Flow 2 from table entry
1	FL1	0	Queue *n* does not support Flow 1 from table entry
		1	Queue *n* supports Flow 1 from table entry
0	FL0	0	Queue *n* does not support Flow 0 from table entry
		1	Queue *n* supports Flow 0 from table entry

5.49 Receive Queue Teardown Register (RX_QUEUE_TEAR_DOWN)

Each of this register's bits corresponds to one of the 16 RX buffer descriptor queues. If a 1 is written to a bit, the teardown process is started for the associated queue. RX_QUEUE_TEAR_DOWN is shown in Figure 110 and described in Table 120. For additional programming information, see Section 2.3.4.1 .

Figure 110. Receive Queue Teardown Register (RX_QUEUE_TEAR_DOWN) - Address Offset 0740h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0000h															

15	14	13	12	11	10	9	8
QUEUE15_ TEAR_DWN	QUEUE14_ TEAR_DWN	QUEUE13_ TEAR_DWN	QUEUE12_ TEAR_DWN	QUEUE11_ TEAR_DWN	QUEUE10_ TEAR_DWN	QUEUE9_ TEAR_DWN	QUEUE8_ TEAR_DWN
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

7	6	5	4	3	2	1	0
QUEUE7_ TEAR_DWN	QUEUE6_ TEAR_DWN	QUEUE5_ TEAR_DWN	QUEUE4_ TEAR_DWN	QUEUE3_ TEAR_DWN	QUEUE2_ TEAR_DWN	QUEUE1_ TEAR_DWN	QUEUE0_ TEAR_DWN
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: R = Read only; W = Write; -*n* = Value after reset

Table 120. Receive Queue Teardown Register (RX_QUEUE_TEAR_DOWN) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0000h	These read-only bits return 0s when read.
15-0	QUEUE*n*_TEAR_DWN (n = 15 to 0)		Queue *n* tear down
		0	No effect
		1	Tear down Queue *n*.

5.50 Receive CPPI Control Register (RX_CPPI_CNTL)

Each bit in this register indicates whether the associated RX buffer descriptor queue must receive messages in the order the source device attempts to transmit them. RX_CPPI_CNTL is shown in Figure 111 and described in Table 121. For additional programming information, see Section 2.3.4.1 .

Figure 111. Receive CPPI Control Register (RX_CPPI_CNTL) - Address Offset 0744h

31	30	29	28	27	26	25	24
Reserved							
R-00h							

23	22	21	20	19	18	17	16
Reserved							
R-00h							

15	14	13	12	11	10	9	8
QUEUE15_ IN_ORDER	QUEUE14_ IN_ORDER	QUEUE13_ IN_ORDER	QUEUE12_ IN_ORDER	QUEUE11_ IN_ORDER	QUEUE10_ IN_ORDER	QUEUE9_ IN_ORDER	QUEUE8_ IN_ORDER
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
QUEUE7_ IN_ORDER	QUEUE6_ IN_ORDER	QUEUE5_ IN_ORDER	QUEUE4_ IN_ORDER	QUEUE3_ IN_ORDER	QUEUE2_ IN_ORDER	QUEUE1_ IN_ORDER	QUEUE0_ IN_ORDER
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 121. Receive CPPI Control Register (RX_CPPI_CNTL) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0000h	Reserved
15-0	QUEUE*n*_IN_ORDER (n = 15 to 0)		Queue*n* in order
		0	Allows out-of-order message reception
		1	Requires in-order message reception. Used for applications with dedicated source-destination flows.

5.51 Transmit CPPI Weighted Round-Robin Control Registers (TX_QUEUE_CNTL[0-3])

The transmission order among TX buffer descriptor queues is based on the programmable weighted round-robin scheme explained in Section 2.3.4.2. As part of this scheme, software must program the 16 mappers to determine the order in which the queues are serviced and how many messages are handled in each queue during each time around the round-robin cycle. The mappers are programmed with the registers shown in Figure 112. The register fields are described in Table 122. For additional programming information, see Section 2.3.4.2 .

Figure 112. Transmit CPPI Weighted Round-Robin Control Registers

TX_QUEUE_CNTL0 - Address Offset 07E0h

<---------- TX_Queue_Map3 ---------->		<---------- TX_Queue_Map2 ---------->	
31 28	27 24	23 20	19 16
Number of Msgs	Queue Pointer	Number of Msgs	Queue Pointer
R/W-0h	R/W-3h	R/W-0	R/W-2h

<---------- TX_Queue_Map1 ---------->		<---------- TX_Queue_Map0 ---------->	
15 12	11 8	7 4	3 0
Number of Msgs	Queue Pointer	Number of Msgs	Queue Pointer
R/W-0h	R/W-1h	R/W-0h	R/W-0h

TX_QUEUE_CNTL1 - Address Offset 07E4h

<---------- TX_Queue_Map7 ---------->		<---------- TX_Queue_Map6 ---------->	
31 28	27 24	23 20	19 16
Number of Msgs	Queue Pointer	Number of Msgs	Queue Pointer
R/W-0	R/W-7h	R/W-0h	R/W-6h

<---------- TX_Queue_Map5 ---------->		<---------- TX_Queue_Map4 ---------->	
15 12	11 8	7 4	3 0
Number of Msgs	Queue Pointer	Number of Msgs	Queue Pointer
R/W-0h	R/W-5h	R/W-0h	R/W-4h

TX_QUEUE_CNTL2 - Address Offset 07E8h

<---------- TX_Queue_Map11 ---------->		<---------- TX_Queue_Map10 ---------->	
31 28	27 24	23 20	19 16
Number of Msgs	Queue Pointer	Number of Msgs	Queue Pointer
R/W-0h	R/W-Bh	R/W-0h	R/W-Ah

<---------- TX_Queue_Map9 ---------->		<---------- TX_Queue_Map8 ---------->	
15 12	11 8	7 4	3 0
Number of Msgs	Queue Pointer	Number of Msgs	Queue Pointer
R/W-0h	R/W-9h	R/W-0h	R/W-8h

TX_QUEUE_CNTL3 - Address Offset 07ECh

<---------- TX_Queue_Map15 ---------->		<---------- TX_Queue_Map14 ---------->	
31 28	27 24	23 20	19 16
Number of Msgs	Queue Pointer	Number of Msgs	Queue Pointer
R/W-0h	R/W-Fh	R/W-0h	R/W-Eh

<---------- TX_Queue_Map13 ---------->		<---------- TX_Queue_Map12 ---------->	
15 12	11 8	7 4	3 0
Number of Msgs	Queue Pointer	Number of Msgs	Queue Pointer
R/W-0h	R/W-Dh	R/W-0h	R/W-Ch

Table 122. Transmit CPPI Weighted Round-Robin Control Register Field Descriptions

Field Pair	Register[Bits]	Field	Value	Description
TX_Queue_Map0	TX_QUEUE_CNTL0[3-0]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL0[7-4]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map1.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages
TX_Queue_Map1	TX_QUEUE_CNTL0[11-8]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL0[15-12]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map2.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages
TX_Queue_Map2	TX_QUEUE_CNTL0[19-16]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL0[23-20]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map3.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages
TX_Queue_Map3	TX_QUEUE_CNTL0[27-24]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL0[31-28]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map4.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages
TX_Queue_Map4	TX_QUEUE_CNTL1[3-0]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL1[7-4]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map5.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages

Table 122. Transmit CPPI Weighted Round-Robin Control Register Field Descriptions (continued)

Field Pair	Register[Bits]	Field	Value	Description
TX_Queue_Map5	TX_QUEUE_CNTL1[11-8]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL1[15-12]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map6.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages
TX_Queue_Map6	TX_QUEUE_CNTL1[19-16]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL1[23-20]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map7.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages
TX_Queue_Map7	TX_QUEUE_CNTL1[27-24]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL1[31-28]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map8.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages
TX_Queue_Map8	TX_QUEUE_CNTL2[3-0]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL2[7-4]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map9.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages
TX_Queue_Map9	TX_QUEUE_CNTL2[11-8]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL2[15-12]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map10.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages

Table 122. Transmit CPPI Weighted Round-Robin Control Register Field Descriptions (continued)

Field Pair	Register[Bits]	Field	Value	Description
TX_Queue_Map10	TX_QUEUE_CNTL2[19-16]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL2[23-20]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map11.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages
TX_Queue_Map11	TX_QUEUE_CNTL2[27-24]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL2[31-28]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map12.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages
TX_Queue_Map12	TX_QUEUE_CNTL3[3-0]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL3[7-4]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map13.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages
TX_Queue_Map13	TX_QUEUE_CNTL3[11-8]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL3[15-12]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map14.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages
TX_Queue_Map14	TX_QUEUE_CNTL3[19-16]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL3[23-20]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map15.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages

Table 122. Transmit CPPI Weighted Round-Robin Control Register Field Descriptions (continued)

Field Pair	Register[Bits]	Field	Value	Description
TX_Queue_Map15	TX_QUEUE_CNTL3[27-24]	Queue Pointer	0h to Fh	Pointer to a queue. This pointer can be programmed to point to any one of the 16 TX buffer descriptor queues.
	TX_QUEUE_CNTL3[31-28]	Number of Msgs	0h to Fh	Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map0.
			0h	1 message
			1h	2 messages
			...	...
			Fh	16 messages

5.52 Mailbox to Queue Mapping Registers (RXU_MAP_L*n* and RXU_MAP_H*n*)

Messages addressed to any of the 64 mailbox locations can be received on any of the RapidIO ports simultaneously. Packets are handled sequentially in order of receipt. A block of 32 mappers directs the inbound messages to the appropriate RX queues. After a device reset, software must configure each of the mappers to map incoming messages with selected mailbox and letter numbers to the desired queue. For a given mapper n, a pair of mailbox to queue mapping registers fully define the configuration for that mapper: a low register (RXU_MAP_L*n*) and a high register (RXU_MAP_H*n*). Table 123 lists all of these register pairs and the associated RX mappers. The general form of an RXU_MAP register pair is summarized by Figure 113, Table 124, and Table 125. For additional programming information, see Section 2.3.4.1 .

Table 123. Mailbox to Queue Mapping Registers and the Associated RX Mappers

Register	Address Offset	Associated RX Mapper
RXU_MAP_L0	0800h	Mapper 0
RXU_MAP_H0	0804h	Mapper 0
RXU_MAP_L1	0808h	Mapper 1
RXU_MAP_H1	080Ch	Mapper 1
RXU_MAP_L2	0810h	Mapper 2
RXU_MAP_H2	0814h	Mapper 2
RXU_MAP_L3	0818h	Mapper 3
RXU_MAP_H3	081Ch	Mapper 3
RXU_MAP_L4	0820h	Mapper 4
RXU_MAP_H4	0824h	Mapper 4
RXU_MAP_L5	0828h	Mapper 5
RXU_MAP_H5	082Ch	Mapper 5
RXU_MAP_L6	0830h	Mapper 6
RXU_MAP_H6	0834h	Mapper 6
RXU_MAP_L7	0838h	Mapper 7
RXU_MAP_H7	083Ch	Mapper 7
RXU_MAP_L8	0840h	Mapper 8
RXU_MAP_H8	0844h	Mapper 8
RXU_MAP_L9	0848h	Mapper 9
RXU_MAP_H9	084Ch	Mapper 9
RXU_MAP_L10	0850h	Mapper 10
RXU_MAP_H10	0854h	Mapper 10
RXU_MAP_L11	0858h	Mapper 11
RXU_MAP_H11	085Ch	Mapper 11
RXU_MAP_L12	0860h	Mapper 12

Table 123. Mailbox to Queue Mapping Registers and the Associated RX Mappers (continued)

Register	Address Offset	Associated RX Mapper
RXU_MAP_H12	0864h	Mapper 12
RXU_MAP_L13	0868h	Mapper 13
RXU_MAP_H13	086Ch	Mapper 13
RXU_MAP_L14	0870h	Mapper 14
RXU_MAP_H14	0874h	Mapper 14
RXU_MAP_L15	0878h	Mapper 15
RXU_MAP_H15	087Ch	Mapper 15
RXU_MAP_L16	0880h	Mapper 16
RXU_MAP_H16	0884h	Mapper 16
RXU_MAP_L17	0888h	Mapper 17
RXU_MAP_H17	088Ch	Mapper 17
RXU_MAP_L18	0890h	Mapper 18
RXU_MAP_H18	0894h	Mapper 18
RXU_MAP_L19	0898h	Mapper 19
RXU_MAP_H19	089Ch	Mapper 19
RXU_MAP_L20	08A0h	Mapper 20
RXU_MAP_H20	08A4h	Mapper 20
RXU_MAP_L21	08A8h	Mapper 21
RXU_MAP_H21	08ACh	Mapper 21
RXU_MAP_L22	08B0h	Mapper 22
RXU_MAP_H22	08B4h	Mapper 22
RXU_MAP_L23	08B8h	Mapper 23
RXU_MAP_H23	08BCh	Mapper 23
RXU_MAP_L24	08C0h	Mapper 24
RXU_MAP_H24	08C4h	Mapper 24
RXU_MAP_L25	08C8h	Mapper 25
RXU_MAP_H25	08CCh	Mapper 25
RXU_MAP_L26	08D0h	Mapper 26
RXU_MAP_H26	08D4h	Mapper 26
RXU_MAP_L27	08D8h	Mapper 27
RXU_MAP_H27	08DCh	Mapper 27
RXU_MAP_L28	08E0h	Mapper 28
RXU_MAP_H28	08E4h	Mapper 28
RXU_MAP_L29	08E8h	Mapper 29
RXU_MAP_H29	08ECh	Mapper 29
RXU_MAP_L30	08F0h	Mapper 30
RXU_MAP_H30	08F4h	Mapper 30
RXU_MAP_L31	08F8h	Mapper 31
RXU_MAP_H31	08FCh	Mapper 31

Figure 113. Mailbox to Queue Mapping Register Pair

Mailbox to Queue Mapping Register L *n* (RXU_MAP_L *n*)

31	30	29	24	23	22	21	16
LETTER_MASK		MAILBOX_MASK		LETTER		MAILBOX	
R/W-11		R/W-111111		R/W-00		R/W-000000	

15	0
SOURCEID	
R/W-0000h	

Mailbox to Queue Mapping Register H *n* (RXU_MAP_H *n*)

31	16
Reserved	
R-0	

15	10	9	8	7	6	5	2	1	0
Reserved		TT		Reserved		QUEUE_ID		PROMISCUOUS	SEGMENT_ MAPPING
R-0		R/W-01		R-00		R/W-0000		R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 124. Mailbox-to-Queue Mapping Register L*n* (RXU_MAP_L*n*) Field Descriptions

Bit	Field	Value	Description
31-30	LETTER_MASK	00b-11b	Letter mask. Each 0 in this field indicates a don't care bit in the letter number. This allows mapper *n* to handle a set or range of letter numbers rather than only one.
29-24	MAILBOX_MASK		Mailbox mask. Each 0 in this field indicates a don't care bit in the mailbox number. This allows mapper *n* to handle a set or range of mailbox numbers rather than only one.
			For a single-segment message:
		000000b-111111b	6-bit mailbox mask value
			For a multi-segment message:
		xxxx00b-xxxx11b	3-bit mailbox mask value
23-22	LETTER	00b-11b	Letter number. If LETTER_MASK = 11b, this is the only letter number handled by mapper *n*. If LETTER_MASK is not 11b, mapper *n* handles the set of letter numbers formed with the mask bit(s).
21-16	MAILBOX		Mailbox number. If MAILBOX_MASK = 111111b, this is the only mailbox number handled by mapper *n*. If MAILBOX_MASK is not all 1s, mapper *n* handles the set of mailbox numbers formed with the mask bit(s).
			For a single-segment message:
		000000b-111111b	6-bit mailbox number (0 to 63)
			For a multi-segment message:
		xxxx00b-xxxx11b	3-bit mailbox number (0 to 3)
15-0	SOURCEID	0000h-FFFFh	Source identification number. The SOURCEID field is used to indicate which external device has access to mapper *n* and its corresponding queue. A comparison is performed between the sourceID of the incoming message packet and the SOURCEID field. If the values do not match, an ERROR response is sent to the sender, and the transaction is logged in the logical layer error management capture registers.

Table 125. Mailbox-to-Queue Mapping Register H*n* (RXU_MAP_H*n*) Field Descriptions

Bit	Field	Value	Description
31-10	Reserved	0	These read-only bits return 0s when read.
9-8	TT		Transport type
		0	During the sourceID comparison, the incoming sourceID is compared with the 8 LSBs of the SOURCEID field of RXU_MAP_L*n*.
		1	During the sourceID comparison, the incoming sourceID is compared with all 16 bits of the SOURCEID field of RXU_MAP_L*n*.
7-6	Reserved	0	These read-only bits return 0s when read.
5-2	QUEUE_ID	0-15	Queue identification number. This field selects which of the 16 RX buffer queues is associated with mapper *n*.
1	PROMISCUOUS		Promiscuous access
		0	Mapper *n* checks the incoming sourceID (access is restricted to one sender). When determining which transactions to service, the mapper checks the sourceID in addition to the mailbox and letter qualifications.
		1	Mapper *n* ignores the incoming sourceID (access is available to any sender). When determining which transactions to service, the mapper checks only the mailbox and letter qualifications.
0	SEGMENT_MAPPING		Segment mapping
		0	Single-segment messaging. Up to 64 mailboxes are available. All six bits of the MAILBOX and MAILBOX_MASK fields of RXU_MAP_L*n* are valid.
		1	Multi-segment messaging. Up to 4 mailboxes are available. Only the 2 LSBs of the MAILBOX and MAILBOX_MASK fields of RXU_MAP_L*n* are valid.

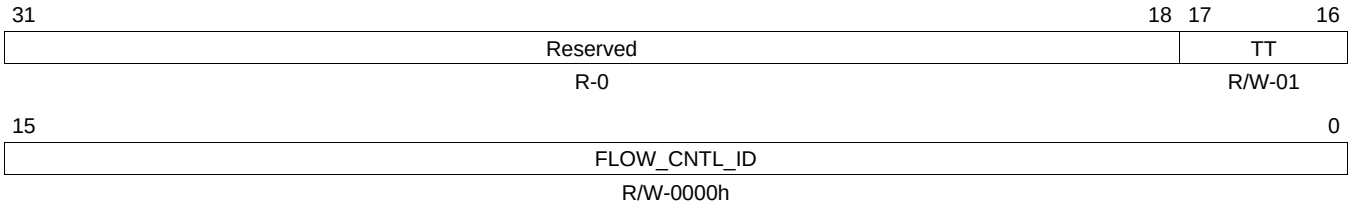
5.53 Flow Control Table Entry Register *n* (FLOW_CNTL*n*)

There are sixteen of these registers (see Table 126). FLOW_CNTL*n* is shown in Figure 114 and described in Table 127. For additional programming information, see Section 2.3.8.

Table 126. FLOW_CNTL*n* Registers

Register	Address Offset
FLOW_CNTL0	0900h
FLOW_CNTL1	0904h
FLOW_CNTL2	0908h
FLOW_CNTL3	090Ch
FLOW_CNTL4	0910h
FLOW_CNTL5	0914h
FLOW_CNTL6	0918h
FLOW_CNTL7	091Ch
FLOW_CNTL8	0920h
FLOW_CNTL9	0924h
FLOW_CNTL10	0928h
FLOW_CNTL11	092Ch
FLOW_CNTL12	0930h
FLOW_CNTL13	0934h
FLOW_CNTL14	0938h
FLOW_CNTL15	093Ch

Figure 114. Flow Control Table Entry Register *n* (FLOW_CNTL*n*)



LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 127. Flow Control Table Entry Register *n* (FLOW_CNTL*n*) Field Descriptions

Bit	Field	Value	Description
31-18	Reserved	0	These read-only bits return 0s when read.
17-16	TT		Transfer type for flow *n*
		00b	8-bit destination IDs
		01b	16-bit destination IDs
		1xb	Reserved
15-0	FLOW_CNTL_ID	0000h-FFFFh	Destination ID for flow *n*. When 8-bit destination IDs are used (TT = 00b), the 8 MSBs of this field are don't care bits.

5.54 Device Identity CAR (DEV_ID)

The device identity CAR (DEV_ID) is shown in Figure 115 and described in Table 128. Writes have no effect to this register. The values are hard coded and will not change from their reset state.

Figure 115. Device Identity CAR (DEV_ID) - Address Offset 1000h

31	16	15	0
DEVICEIDENTITY		DEVICE_VENDORIDENTITY	
R-0000h		R-0030h	

LEGEND: R = Read only; -*n* = Value after reset

This register is writable before BOOT_COMPLETE is set; then it is read only.

Table 128. Device Identity CAR (DEV_ID) Field Descriptions

Bit	Field	Value	Description
31-16	DEVICEIDENTITY	0000h	Identifies the type of device. Vendor specific.
15-0	DEVICE_VENDORIDENTITY	0030h	Device Vendor ID assigned by RapidIO Trade Association.

5.55 Device Information CAR (DEV_INFO)

The device information CAR (DEV_INFO) is shown in Figure 116 and described in Table 129. Writes have no effect to this register. The values are hard coded and will not change from their reset state.

Figure 116. Device Information CAR (DEV_INFO) - Address Offset 1004h

31	0
DEVICEREV	
R-00000000h	

LEGEND: R = Read only; -*n* = Value after reset

This register is writable before BOOT_COMPLETE is set; then it is read only.

Table 129. Device Information CAR (DEV_INFO) Field Descriptions

Bit	Field	Value	Description
31-0	DEVICEREV	00000000h	Vendor supply device revision

5.56 Assembly Identity CAR (ASBLY_ID)

The assembly identity CAR (ASBLY_ID) is shown in Figure 117 and described in Table 130. Writes have no effect to this register. The values are hard coded and will not change from their reset state.

Figure 117. Assembly Identity CAR (ASBLY_ID) - Address Offset 1008h

31	16	15	0
ASSY_IDENTITY		ASSY_VENDORIDENTITY	
R-0000h		R-0030h	

LEGEND: R = Read only; -*n* = Value after reset

Table 130. Assembly Identity CAR (ASBLY_ID) Field Descriptions

Bit	Field	Value	Description
31-16	ASSY_IDENTITY	0000h	Assembly identifier. Vendor specific.
15-0	ASSY_VENDORIDENTITY	0030h	Assembly vendor identifier assigned by RapidIO Trade Association.

5.57 Assembly Information CAR (ASBLY_INFO)

The assembly information CAR (ASBLY_INFO) is shown in Figure 118 and described in Table 131. This register is used by SERDES vendor to designate endpoints among the various function blocks of registers. Writes have no effect to this register. The values are hard coded and will not change from their reset state.

Figure 118. Assembly Information CAR (ASBLY_INFO) - Address Offset 100Ch

31 16	15 0
ASSYREV	EXTENDEDFEATURESPTR
R-0000h	R-0100h

LEGEND: R = Read only; -*n* = Value after reset

Table 131. Assembly Information CAR (ASBLY_INFO) Field Descriptions

Bit	Field	Value	Description
31-16	ASSYREV	0000h	Assembly revision level.
15-0	EXTENDEDFEATURESPTR	0100h	Pointer to first entry in extended features list.

5.58 Processing Element Features CAR (PE_FEAT)

The processing element features CAR (PE_FEAT) is shown in Figure 119 and described in Table 132.

Figure 119. Processing Element Features CAR (PE_FEAT) - Address Offset 1010h

31	30	29	28	27–24
BRIDGE	MEMORY	PROCESSOR	SWITCH	Reserved
R-0	R-0	R-1	R-0	R-0h

23–16
Reserved
R-00h

15–8
Reserved
R-01h

7	6	5	4	3	2–0
FLOW_CONTROL_SUPPORT	RETRANSMIT_SUPPRESS	CRF_SUPPORT	LARGE_SUPPORT	EXTENDED_FEATURES	EXTENDED_ADDRESSING_SUPPORT
R-0	R-0	R-0	R-0	R-1	R-001

LEGEND: R = Read only; -*n* = Value after reset

Table 132. Processing Element Features CAR (PE_FEAT) Field Descriptions

Bit	Field	Value	Description
31	BRIDGE		PE can bridge to another interface. Examples are PCI, proprietary processor buses, DRAM, etc.
30	MEMORY		PE has physically addressable local address space and can be accessed as an endpoint through non-maintenance (i.e., non-coherent read and write) operations. This local address space may be limited to local configuration Registers, or could be on-chip SRAM, etc.
29	PROCESSOR		PE physically contains a local processor or similar device that executes code. A device that bridges to an interface that connects to a processor does not count (see bit 31).
28	SWITCH		PE can bridge to another external RapidIO interface. An internal port to a local endpoint does not count as a switch port. For example, a device with two RapidIO ports and a local endpoint is a two port switch, not a three port switch, regardless of the internal architecture.
27-8	Reserved	0	These read-only bits return 0s when read.
7	FLOW_CONTROL_SUPPORT		PE supports congestion flow control mechanism
		0	PE does not support flow control
		1	PE supports flow control
6	RETRANSMIT_SUPPRESS		PE supports suppression of error recovery on packet CRC errors.
		0	PE does not support suppression
		1	PE supports suppression
5	CRF_SUPPORT		This bit indicates PE support for the Critical Request Flow (CRF) Function.
		0	PE does not support CRF Function
		1	PE supports CRF Function
4	LARGE_SUPPORT		Support of common transport large systems.
		0	PE does not support common transport large systems
		1	PE supports common transport large systems
3	EXTENDED_FEATURES		PE has extended features list; the extended features pointer is valid.

Table 132. Processing Element Features CAR (PE_FEAT) Field Descriptions (continued)

Bit	Field	Value	Description
2-0	EXTENDED_ADDRESSING_SUPPORT		Indicates the number address bits supported by the PE both as a source and target of an operation. All PEs shall at minimum support 34 bit addresses. Encodings other than below are reserved.
		001b	PE supports 34 bit addresses
		011b	PE supports 50 and 34 bit addresses
		101b	PE supports 66 and 34 bit addresses
		111b	PE supports 66, 50 and 34 bit addresses
		Other	Reserved

5.59 Source Operations CAR (SRC_OP)

The source operations CAR (SRC_OP) is shown in Figure 120 and described in Table 133.

Figure 120. Source Operations CAR (SRC_OP) - Address Offset 1018h

31	30	29	28	27	26	25	24
Reserved							
R-0							

23	22	21	20	19	18	17	16
Reserved						IMPLMNT_DEFINED_2	
R-0						R-00	

15	14	13	12	11	10	9	8
READ	WRITE	STREAM_ WRITE	WRITE_WITH_ RESP	DATA_MESS	DOORBELL	Reserved	ATOMIC_TEST_ AND_SWAP
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

7	6	5	4	3	2	1	0
ATOMIC_ INCRMNT	ATOMIC_ DCRMNT	ATOMIC_ SET	ATOMIC_ CLEAR	Reserved	PORT_ WRITE	IMPLMNT_DEFINED_1	
R-0	R-0	R-0	R-0	R-0	R-1	R-00	

LEGEND: R = Read only; -*n* = Value after reset

Table 133. Source Operations CAR (SRC_OP) Field Descriptions

Bit	Field	Value	Description
31-18	Reserved	0	These read-only bits return 0s when read.
17-16	IMPLMNT_DEFINED_2		Defined by the device implementation
15	READ		PE can support a read operation
14	WRITE		PE can support a write operation
13	STREAM_WRITE		PE can support a streaming-write operation
12	WRITE_WITH_RESP		PE can support a write-with-response operation
11	DATA_MESS		PE can support a data message operation
10	DOORBELL		PE can support a doorbell operation
9	Reserved	0	This read-only bit returns 0 when read.
8	ATOMIC_TEST_ AND_SWAP		PE can support an atomic test-and-swap operation
7	ATOMIC_INCRMNT		PE can support an atomic increment operation
6	ATOMIC_DCRMNT		PE can support an atomic decrement operation
5	ATOMIC_SET		PE can support an atomic set operation
4	ATOMIC_CLEAR		PE can support an atomic clear operation
3	Reserved	0	This read-only bit returns 0 when read.
2	PORT_WRITE		PE can support a port-write generation
1-0	IMPLMNT_DEFINED_1		Defined by the device implementation

5.60 Destination Operations CAR (DEST_OP)

The destination operations CAR (DEST_OP) is shown in Figure 121 and described in Table 134.

Figure 121. Destination Operations CAR (DEST_OP) - Address Offset 101Ch

31	30	29	28	27	26	25	24
Reserved							
R-0							

23	22	21	20	19	18	17	16
Reserved						IMPLMNT_DEFINED_2	
R-0						R-00	

15	14	13	12	11	10	9	8
READ	WRITE	STREAM_WRITE	WRITE_WITH_RESP	DATA_MESS	DOORBELL	Reserved	ATOMIC_TEST_AND_SWAP
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

7	6	5	4	3	2	1	0
ATOMIC_INCRMNT	ATOMIC_DCRMNT	ATOMIC_SET	ATOMIC_CLEAR	Reserved	PORT_WRITE	IMPLMNT_DEFINED_1	
R-0	R-0	R-0	R-0	R-0	R-1	R-00	

LEGEND: R = Read only; -*n* = Value after reset

Table 134. Destination Operations CAR (DEST_OP) Field Descriptions

Bit	Field	Value	Description
31-18	Reserved	0	These read-only bits return 0s when read.
17-16	IMPLMNT_DEFINED_2		Defined by the device implementation
15	READ		PE can support a read operation
14	WRITE		PE can support a write operation
13	STREAM_WRITE		PE can support a streaming-write operation
12	WRITE_WITH_RESP		PE can support a write-with-response operation
11	DATA_MESS		PE can support a data message operation
10	DOORBELL		PE can support a doorbell operation
9	Reserved	0	This read-only bit returns 0 when read.
8	ATOMIC_TEST_AND_SWAP		PE can support an atomic test-and-swap operation
7	ATOMIC_INCRMNT		PE can support an atomic increment operation
6	ATOMIC_DCRMNT		PE can support an atomic decrement operation
5	ATOMIC_SET		PE can support an atomic set operation
4	ATOMIC_CLEAR		PE can support an atomic clear operation
3	Reserved	0	This read-only bit returns 0 when read.
2	PORT_WRITE		PE can support a port-write generation
1-0	IMPLMNT_DEFINED_1		Defined by the device implementation

5.61 Processing Element Logical Layer Control CSR (PE_LL_CTL)

The processing element logical layer control CSR (PE_LL_CTL) is shown in Figure 122 and described in Table 135.

Figure 122. Processing Element Logical Layer Control CSR (PE_LL_CTL) - Address Offset 104Ch

31	16
Reserved	
R-0	

15	3	2	0
Reserved		EXTENDED_ ADDRESSING_ CONTROL	
R-0		R/W-001	

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

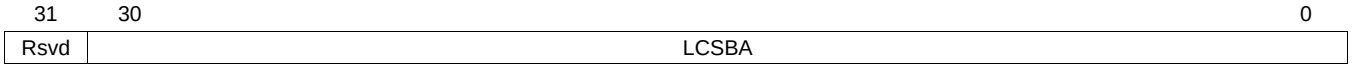
Table 135. Processing Element Logical Layer Control CSR (PE_LL_CTL) Field Descriptions

Bit	Field	Value	Description
31-3	Reserved	0	These read-only bits return 0s when read.
2-0	EXTENDED_ADDRESSING_CONTROL		Controls the number of address bits generated by the PE as a source and processed by the PE as the target of an operation. All other encodings reserved.
		001b	PE supports 34 bit addresses
		010b	PE supports 50 bit addresses
		100b	PE supports 66 bit addresses
		Other	Reserved

5.62 Local Configuration Space Base Address 0 CSR (LCL_CFG_HBAR)

The local configuration space base address 0 CSR (LCL_CFG_HBAR) is shown in Figure 123 and described in Table 136.

Figure 123. Local Configuration Space Base Address 0 CSR (LCL_CFG_HBAR) - Address Offset 1058h



LEGEND: R/W = Read/Write; -*n* = value after reset

Table 136. Local Configuration Space Base Address 0 CSR (LCL_CFG_HBAR) Field Descriptions

Bit	Field	Value	Description
31	Reserved	0	These read-only bits return 0s when read.
30-0	LCSBA	00000000h to FFFFFFFFh	Bits 30 to 15 are reserved for 34-bit addresses, reserved for 50-bit addresses, and bits 66 to 51 of a 66-bit address. Bits 14 to 0 are reserved for 34-bit addresses, bits 50 to 36 of a 50-bit address, and bits 50 to 36 of a 66-bit address.

5.63 Local Configuration Space Base Address 1 CSR (LCL_CFG_BAR)

The local configuration space base address 1 CSR (LCL_CFG_BAR) is shown in Figure 124 and described in Table 137.

Figure 124. Local Configuration Space Base Address 1 CSR (LCL_CFG_BAR) - Address Offset 105Ch

LEGEND: R/W = Read/Write; -*n* = Value after reset

Table 137. Local Configuration Space Base Address 1 CSR (LCL_CFG_BAR) Field Descriptions

Bit	Field	Value	Description
31-0	LCSBA	00000000h to FFFFFFFFh	Bit 31 is reserved for 34-bit addresses, bit 35 of a 50-bit address, and bit 35 of a 66-bit address. Bits 30 to 0 are bits 34 to 3 of a 34-bit address, bits 35 to 3 of a 50-bit address, and bits 35 to 3 of a 66-bit address.

5.64 Base Device ID CSR (BASE_ID)

The base device ID CSR (BASE_ID) is shown in Figure 125 and described in Table 138.

Figure 125. Base Device ID CSR (BASE_ID) - Address Offset 1060h

31	24	23	16
Reserved		BASE_DEVICEID	
R-00h		R/W-FFh	

15	0
LARGE_BASE_DEVICEID	
R/W-FFFFh	

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 138. Base Device ID CSR (BASE_ID) Field Descriptions

Bit	Field	Value	Description
31-24	Reserved	00h	These read-only bits return 0s when read.
23-16	BASE_DEVICEID	00h-FFh	This is the base ID of the device in small common transport system (endpoints only).
15-0	LARGE_BASE_DEVICEID	0000h-FFFFh	This is the base ID of the device in a large common transport system (Only valid for endpoints, and if bit 4 of the PE_FEAT Register is set).

5.65 Host Base Device ID Lock CSR (HOST_BASE_ID_LOCK)

See Section 2.4.2 of the RapidIO *Common Transport Specification* for a description of this register. It provides a lock function that is write-once/reset-able. The host base device ID lock CSR (HOST_BASE_ID_LOCK) is shown in Figure 126 and described in Table 139.

Figure 126. Host Base Device ID Lock CSR (HOST_BASE_ID_LOCK) - Address Offset 1068h

31	16	15	0
Reserved		HOST_BASE_DEVICEID	
R-0000h		R/W-FFFFh	

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 139. Host Base Device ID Lock CSR (HOST_BASE_ID_LOCK) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0000h	These read-only bits return 0s when read.
15-0	HOST_BASE_DEVICEID	0000h-FFFFh	This is the base ID for the Host PE that is initializing this PE.

5.66 Component Tag CSR (COMP_TAG)

The component Tag CSR (COMP_TAG) is shown in Figure 127 and described in Table 140.

Figure 127. Component Tag CSR (COMP_TAG) - Address Offset 106Ch

31	0
COMPONENT_TAG	
R/W-00000000h	

LEGEND: R/W = Read/Write; -*n* = Value after reset

Table 140. Component Tag CSR (COMP_TAG) Field Descriptions

Bit	Field	Value	Description
31-0	COMPONENT_TAG	00000000h to FFFFFFFFh	Software defined component tag for the PE. Useful for devices without device IDs.

5.67 1x/4x LP Serial Port Maintenance Block Header Register (SP_MB_HEAD)

The 1x/4x LP_Serial port maintenance block header register (SP_MB_HEAD) is shown in Figure 128 and described in Table 141.

Figure 128. 1x/4x LP_Serial Port Maintenance Block Header Register (SP_MB_HEAD) - Address Offset 1100h

31 16	15 0
EF_PTR	EF_ID
R-1000h	R-0001h

LEGEND: R = Read only; -*n* = Value after reset

Table 141. 1x/4x LP_Serial Port Maintenance Block Header Register (SP_MB_HEAD) Field Descriptions

Bit	Field	Value	Description
31-16	EF_PTR		Hard wired pointer to the next block in the data structure.
15-0	EF_ID		Hard wired extended features ID.
		0001h	General endpoint device
		0002h	General endpoint device with software assisted error recovery option
		0003h	Switch

5.68 Port Link Time-Out Control CSR (SP_LT_CTL)

The port link time-out control CSR (SP_LT_CTL) is shown in Figure 129 and described in Table 142.

Figure 129. Port Link Time-Out Control CSR (SP_LT_CTL) - Address Offset 1120h

31		8	7	0
TIMEOUT_VALUE				
R/W-FFFFFFh				
TIMEOUT_VALUE			Reserved	
R/W-FFFFFFh			R-00h	

LEGEND: R/W = Read/Write; R = Read only; *-n* = Value after reset

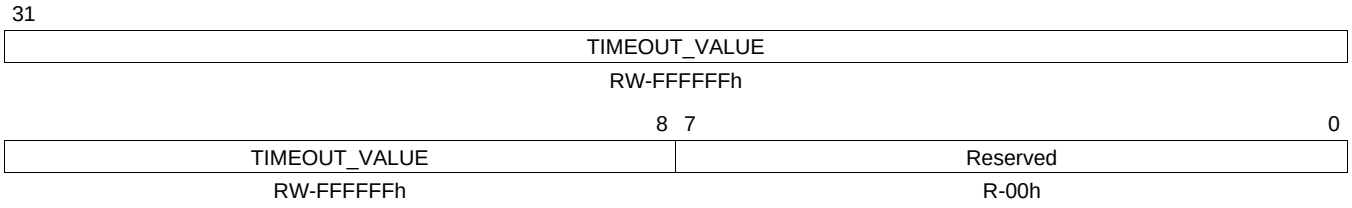
Table 142. Port Link Timeout Control CSR (SP_LT_CTL) Field Descriptions

Bit	Field	Value	Description
31-8	TIMEOUT_VALUE		Timeout value for all ports on the device. This timeout is for link events such as sending a packet to receiving the corresponding ACK. Max value represents 3-6 seconds. Timeout duration = 205 ns * Timeout Value; where Timeout value is the decimal representation of this register value.
		FFFFFFh	3.4 s
		0FFFFFh	215 ms
		00FFFFh	13.4 ms
		000FFFh	839.5 µs
		0000FFh	52.3 µs
		00000Fh	3.1 µs
		000001h	205 ns for simulation only
		000000h	Timer disabled
7-0	Reserved	00h	These read-only bits return 0s when read.

5.69 Port Response Time-Out Control CSR (SP_RT_CTL)

The port response time-out control CSR (SP_RT_CTL) is shown in Figure 130 and described in Table 143 For additional programming information, see Section 2.3.3.3 and Section 2.3.3.

Figure 130. Port Response Time-Out Control CSR (SP_RT_CTL) - Address Offset 1124h



LEGEND: R/W = Read/Write; R = Read only; *-n* = Value after reset

Table 143. Port Response Time-Out Control CSR (SP_RT_CTL) Field Descriptions

Bit	Field	Value	Description
31-8	TIMEOUT_VALUE	000000h to FFFFFFh	Timeout value for all ports on the device. This timeout is for sending a packet to receiving the corresponding response packet. Max value represents 3 to 6 seconds. The timeout duration can be expressed as: Timeout = 15 x ((Prescale Value + 1) x DMA Clock Period x Timeout Value) where Prescale value is set in PER_SET_CNTL (offset 0020h) and the Timeout value is the decimal representation of this register value. For example, given a 400-MHz DMA, a Prescale Value of 4, and a Timeout Value of FFFFFFh, the Timeout duration would be: Timeout = 15 x ((4 + 1) x 2.5 ns x 16777216) = 3.15 s
7-0	Reserved	00h	These read-only bits return 0s when read.

5.70 Port General Control CSR (SP_GEN_CTL)

The port general control CSR (SP_GEN_CTL) is shown in Figure 131 and described in Table 144.

Figure 131. Port General Control CSR (SP_GEN_CTL) - Address Offset 113Ch

31	30	29	28 ... 0
HOST	MASTER_ ENABLE	DISCOVERED	Reserved
R/W-0	R/W-0	R/W-0	R-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 144. Port General Control CSR (SP_GEN_CTL) Field Descriptions

Bit	Field	Value	Description
31	HOST		A host device is a device that is responsible for system exploration, initialization, and maintenance. Agent or slave devices are typically initialized by Host devices.
		0	Agent or Slave device
		1	Host device
30	MASTER_ENABLE		The Master Enable bit controls whether or not a device is allowed to issue requests into the system. If the Master Enable is not set, the device may only respond to requests.
		0	Processing element cannot issue requests
		1	Processing element can issue requests
29	DISCOVERED		This device has been located by the processing element responsible for system configuration.
		0	The device has not been previously discovered
		1	The device has been discovered by another processing element
28-0	Reserved	0	These read-only bits return 0s when read.

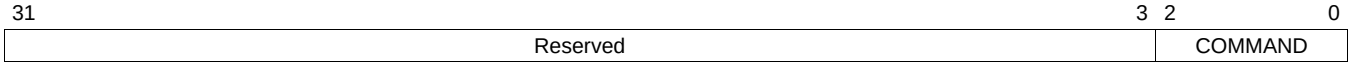
5.71 Port Link Maintenance Request CSR *n* (SP*n*_LM_REQ)

Each of the four ports is supported by a register of this type (see Table 145). SP*n*_LM_REQ is shown in Figure 132 and described in Table 146.

Table 145. SPn_LM_REQ Registers and the Associated Ports

Register	Address Offset	Associated Port
SP0_LM_REQ	1140h	Port 0
SP1_LM_REQ	1160h	Port 1
SP2_LM_REQ	1180h	Port 2
SP3_LM_REQ	11A0h	Port 3

Figure 132. Port Link Maintenance Request CSR *n* (SP*n*_LM_REQ)



LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 146. Port Link Maintenance Request CSR *n* (SP*n*_LM_REQ) Field Descriptions

Bit	Field	Value	Description
31-3	Reserved	0	These read-only bits return 0s when read.
2-0	COMMAND	000b-111b	A write to this register generates a link-request control symbol on the corresponding port interface. Command to be sent in the link-request control symbol. When read, this field returns the last written value.

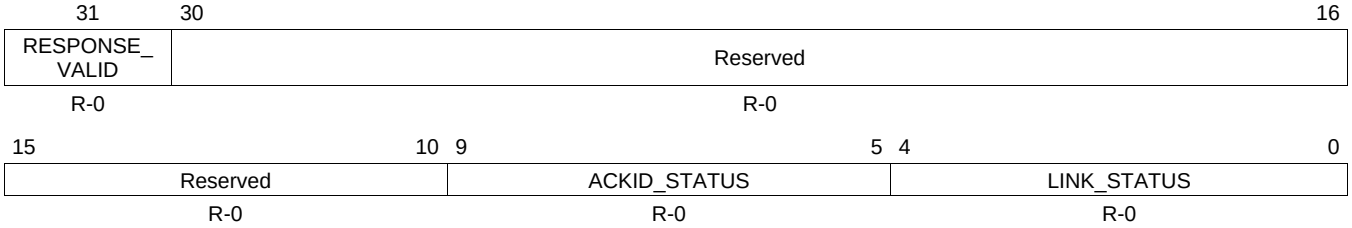
5.72 Port Link Maintenance Response CSR *n* (SP*n*_LM_RESP)

Each of the four ports is supported by a register of this type (see Table 147). The port link maintenance response CSR *n* (SP*n*_LM_RESP) is shown in Figure 133 and described in Table 148.

Table 147. SPn_LM_RESP Registers and the Associated Ports

Register	Address Offset	Associated Port
SP0_LM_RESP	1144h	Port 0
SP1_LM_RESP	1164h	Port 1
SP2_LM_RESP	1184h	Port 2
SP3_LM_RESP	11A4h	Port 3

Figure 133. Port Link Maintenance Response CSR *n* (SP*n*_LM_RESP)



LEGEND: R = Read only; -*n* = Value after reset

Table 148. Port Link Maintenance Response CSR *n* (SP*n*_LM_RESP) Field Descriptions

Bit	Field	Value	Description
31	RESPONSE_VALID		If the link-request causes a link-response, this bit indicates that the link-response has been received and the status fields are valid. If the link-request does not cause a link-response, this bit indicates that the link-request has been transmitted. This bit automatically clears on read.
30-10	Reserved	0	These read-only bits return 0s when read.
9-5	ACKID_STATUS	00000b-11111b	AckID status field from the link-response control symbol
4-0	LINK_STATUS	00000b-11111b	Link status field from the link-response control symbol

5.73 Port Local AckID Status CSR *n* (SP*n*_ACKID_STAT)

Each of the four ports is supported by a register of this type (see Table 149). The port local ackID status CSR *n* (SP*n*_ACKID_STAT) is shown in Figure 134 and described in Table 150.

Table 149. SP*n*_ACKID_STAT Registers and the Associated Ports

Register	Address Offset	Associated Port
SP0_ACKID_STAT	1148h	Port 0
SP1_ACKID_STAT	1168h	Port 1
SP2_ACKID_STAT	1188h	Port 2
SP3_ACKID_STAT	11A8h	Port 3

Figure 134. Port Local AckID Status CSR *n* (SP*n*_ACKID_STAT)

31–29	28–24	23–16
Reserved	INBOUND_ACKID	Reserved
R-0	R/W-0	R-0

15–13	12–8	7–3	4–0
Reserved	OUTSTANDING_ACKID	Reserved	OUTBOUND_ACKID
R-0	R/W-0	R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 150. Port Local AckID Status CSR *n* (SP*n*_ACKID_STAT) Field Descriptions

Bit	Field	Value	Description
31-29	Reserved	0	These read-only bits return 0s when read.
28-24	INBOUND_ACKID	00000b-11111b	Input port next expected ackID value
23-13	Reserved	0	These read-only bits return 0s when read.
12-8	OUTSTANDING_ACKID	00000b-11111b	Output port unacknowledged ackID status. Next expected acknowledge control symbol ackID field that indicates the ackID value expected in the next received acknowledge control symbol.
7-5	Reserved	0	These read-only bits return 0s when read.
4-0	OUTBOUND_ACKID	00000b-11111b	Output port next transmitted ackID value. Software writing this value can force retransmission of outstanding unacknowledged packets in order to manually implement error recovery.

5.74 Port Error and Status CSR *n* (SP*n*_ERR_STAT)

Each of the four ports is supported by a register of this type (see Table 151). The port error and status CSR *n* (SP*n*_ERR_STAT) is shown in Figure 135 and described in Table 152.

Table 151. SP*n*_ERR_STAT Registers and the Associated Ports

Register	Address Offset	Associated Port
SP0_ERR_STAT	1158h	Port 0
SP1_ERR_STAT	1178h	Port 1
SP2_ERR_STAT	1198h	Port 2
SP3_ERR_STAT	11B8h	Port 3

Figure 135. Port Error and Status CSR *n* (SP*n*_ERR_STAT)

31–27	26	25	24
Reserved	OUTPUT_PKT_DROP	OUTPUT_FLD_ENC	OUTPUT_DEGRD_ENC
R-0	R/W-0	R/W-0	R/W-0

23–21	20	19	18	17	16
Reserved	OUTPUT_RETRY_ENC	OUTPUT_RETRIED	OUTPUT_RETRY_STP	OUTPUT_ERROR_ENC	OUTPUT_ERROR_STP
R-0	R/W-0	R-0	R-0	R/W-0	R-0

15–11	10	9	8
Reserved	INPUT_RETRY_STP	INPUT_ERROR_ENC	INPUT_ERROR_STP
R-0	R-0	R/W-0	R-0

7–5	4	3	2	1	0
Reserved	PORT_WRITE_PND	Reserved	PORT_ERROR	PORT_OK	PORT_UNINITIALIZED
R-0	R/W-0	R-0	R/W-0	R-0	R-1

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 152. Port Error and Status CSR *n* (SP*n*_ERR_STAT) Field Descriptions

Bit	Field	Value	Description
31-27	Reserved	0	These read-only bits return 0s when read.
26	OUTPUT_PKT_DROP		Output packet drop
		0	The output port has not discarded a packet.
		1	The output port has discarded a packet for one of the following reasons: • If the OUTPUT_FLD_ENC encountered bit is set, indicating that the failed error threshold has been reached, and the DROP_PACKET_ENABLE bit of the Port Control CSR *n* is enabled, egress output packets can be dropped by the physical layer. Upon discarding a packet, the port sets the OUTPUT_PKT_DROP bit in the Port *n* Error and Status CSR. For details on which packets can be dropped, see the DROP_PACKET_ENABLE and STOP_PORT_FLD_ENC_ENABLE bit field descriptions in Table 154. • For ingress traffic, if there is congestion at the UDI which prevents the a packet from being passed from the RX physical layer to the logical layer within 2^{15} SRIO_CLK cycles, the OUTPUT_PKT_DROP bit is set. In this case, the ERR_IMP_SPECIFIC bit of the Port Error Detect CSR *n* is also set to indicate this condition. No ingress or egress packets are dropped under this scenario.

Table 152. Port Error and Status CSR *n* (SP*n*_ERR_STAT) Field Descriptions (continued)

Bit	Field	Value	Description
25	OUTPUT_FLD_ENC		Output failed condition encountered. Once set, the OUTPUT_FLD_ENC bit remains set until software writes a 1 to it.
		0	The output port has not encountered a failed condition.
		1	The output port has encountered a failed condition. The failed port error threshold has been reached in the Port *n* Error Rate Threshold Register.
24	OUTPUT_DEGRD_ENC		Output degraded condition encountered. Once set, the OUTPUT_DEGRD_ENC bit remains set until software writes a 1 to it.
		0	The output port has not encountered a degraded condition.
		1	The output port has encountered a degraded condition. The degraded port error threshold has been reached in the Port *n* Error Rate Threshold Register.
23-21	Reserved	0	These read-only bits return 0s when read.
20	OUTPUT_RETRY_ENC		Output retry condition encountered. Once set, the OUTPUT_RETRY_ENC bit remains set until software writes a 1 to it.
		0	The output port has not encountered a retry condition.
		1	The output port has encountered a retry condition. This bit is set when bit 18 is set.
19	OUTPUT_RETRIED		Output retried. OUTPUT_RETRIED is a read-only bit.
		0	The output port has not received a packet-retry control symbol.
		1	The output port has received a packet-retry control symbol and cannot make forward progress. This bit is set when bit 18 is set and is cleared when a packet-accepted or packet-not-accepted control symbol is received.
18	OUTPUT_RETRY_STP		Output retry-stopped state. OUTPUT_RETRY_STP is a read-only bit.
		0	The output port has not received a packet-retry control symbol and/or is not in the "output retry-stopped" state.
		1	The output port has received a packet-retry control symbol and is in the "output retry-stopped" state.
17	OUTPUT_ERROR_ENC		Output transmission error encountered. Once set, the OUTPUT_ERROR_ENC bit remains set until software writes a 1 to it.
		0	The output port has not encountered a transmission error.
		1	The output port has encountered (and possibly recovered from) a transmission error. This bit is set when bit 16 is set.
16	OUTPUT_ERROR_STP		Output error-stopped. OUTPUT_ERROR_STP is a read-only bit.
		0	The output port is not in the "output error-stopped" state.
		1	The output port is in the "output error-stopped" state.
15-11	Reserved	0	These read-only bits return 0s when read.
10	INPUT_RETRY_STP		Input retry-stopped state. INPUT_RETRY_STP is a read-only bit.
		0	The input port is not in the "input retry-stopped" state.
		1	The input port is in the "input retry-stopped" state.
9	INPUT_ERROR_ENC		Input port transmission error encountered. Once set, the INPUT_ERROR_ENC bit remains set until software writes a 1 to it.
		0	The input port has not encountered a transmission error.
		1	The input port has encountered (and possibly recovered from) a transmission error. This bit is set when bit 8 is set.
8	INPUT_ERROR_STP		Input error-stopped state. INPUT_ERROR_STP is a read-only bit.
		0	The input port is not in the "input error-stopped" state.
		1	The input port is in the "input error-stopped" state.
7-5	Reserved	0	These read-only bits return 0s when read.

Table 152. Port Error and Status CSR *n* (SP*n*_ERR_STAT) Field Descriptions (continued)

Bit	Field	Value	Description
4	PORT_WRITE_PND		Port-write pending. This bit is only valid if the device is capable of issuing a maintenance port-write transaction. Once set, the PORT_WRITE_PND bit remains set until software writes a 1 to it.
		0	The port has not encountered a condition which required it to initiate a Maintenance Port-write operation.
		1	The port has encountered a condition which required it to initiate a Maintenance Port-write operation.
3	Reserved	0	This read-only bit returns 0 when read.
2	PORT_ERROR		Port unrecoverable error. Once set, the PORT_ERROR bit remains set until software writes a 1 to it.
		0	The input or output port has not encountered an error from which hardware was unable to recover.
		1	The input or output port has encountered an error from which hardware was unable to recover.
1	PORT_OK		Port OK. This bit is a read-only bit.
		0	Port not-OK condition
		1	Port OK condition. The input and output ports are initialized, and the port is exchanging error-free control symbols with the attached device.
0	PORT_UNINITIALIZED		Port uninitialized. PORT_UNINITIALIZED is a read-only bit. This bit and the PORT_OK bit are mutually exclusive.
		0	Input and output ports are initialized.
		1	Input and output ports are not initialized.

5.75 Port Control CSR *n* (SP*n*_CTL)

Each of the four ports is supported by a register of this type (see Table 153). The port control CSR *n* (SP*n*_CTL) is shown in Figure 136 and described in Table 154. There are two modes of operation: single port and multi-port. If you are in multi-port mode (1X_MODE=1 and SP_MODE=01), only 1X width (1 lane) is possible. You can have up to four 1X ports.If you are in single-port mode (1X_MODE=0 and SP_MODE=00), then a 4X width (4 lane) or a 1X width (1 lane, either lane 0 or lane 2) is possible.

Table 153. SP*n*_CTL Registers and the Associated Ports

Register	Address Offset	Associated Port
SP0_CTL	115Ch	Port 0
SP1_CTL	117Ch	Port 1
SP2_CTL	119Ch	Port 2
SP3_CTL	11BCh	Port 3

Figure 136. Port Control CSR *n* (SP*n*_CTL)

31 30	29 27	26 24
PORT_WIDTH	INITIALIZED_PORT_WIDTH	PORT_WIDTH_OVERRIDE
R-01	R-000	R/W-000

23	22	21	20	19	18 16
PORT_ DISABLE	OUTPUT_ PORT_ ENABLE	INPUT_ PORT_ ENABLE	ERROR_ CHECK_ DISABLE	MULTICAST_ PARTICIPANT	Reserved
R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0

15 8
Reserved
R-0

7 4	3	2	1	0
Reserved	STOP_PORT_ FLD_ENC_ ENABLE	DROP_ PACKET_ ENABLE	PORT_ LOCKOUT	PORT_TYPE
R-0	R/W-0	R/W-0	R/W-0	R-1

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 154. Port Control CSR *n* (SP*n*_CTL) Field Descriptions

Bit	Field	Value	Description
31-30	PORT_WIDTH		Port width. This read-only field indicates the hardware width of the port.
		00b	Single-lane port (valid for all ports)
		01b	Four-lane port (valid for port 0 only)
		1xb	Reserved
29-27	INITIALIZED_PORT_WIDTH		Initialized port width. This read-only field indicates the width of the ports after initialization.
		000b	Single-lane port, lane 0
		001b	Single-lane port, lane 2 (See RapidIO Serial Spec 1.2, Chapter 4.4.10)
		010b	Four-lane port
		011b-111b	Reserved

Table 154. Port Control CSR *n* (SP*n*_CTL) Field Descriptions (continued)

Bit	Field	Value	Description
26-24	PORT_WIDTH_OVERRIDE		Port width override. This read-only field is available as a software means to override the hardware width.
		000b	No override
		001b	Reserved
		010b	Force single lane, lane 0
		011b	Force single lane, lane 2
		100b-111b	Reserved
23	PORT_DISABLE		Port disable
		0	Port receivers/drivers are enabled.
		1	Port receivers/drivers are disabled and are unable to receive/transmit any packets or control symbols.
22	OUTPUT_PORT_ENABLE		Output port enable
		0	The port is stopped and not enabled to issue any packets except to route or respond to I/O logical maintenance packets, depending upon the functionality of the processing element. Control symbols are not affected and are sent normally.
		1	The port is enabled to issue any packets.
21	INPUT_PORT_ENABLE		Input port receive enable
		0	The port is stopped and only enabled to route or respond I/O logical maintenance packets, depending upon the functionality of the processing element. Other packets generate packet-not-accepted control symbols to force an error condition to be signaled by the sending device. Control symbols are not affected and are received and handled normally.
		1	Port is enabled to respond to any packet.
20	ERROR_CHECK_DISABLE		Error check disable
		0	RapidIO transmission error checking and recovery are enabled.
		1	RapidIO transmission error checking and recovery are disabled. If an error condition occurs, device behavior is undefined.
19	MULTICAST_PARTICIPANT	0	Multicast-event participant enable. This read-only bit is 0 to indicate that multicast-event control symbols cannot be accepted by this port.
18-4	Reserved	0	These read-only bits return 0s when read.
3	STOP_PORT_FLD_ENC_ENABLE		Stop-on-fail enable
		0	Even when the Output Failed-encountered bit is set, the port continues to attempt to transmit packets to the connected device.
		1	When the Output Failed-encountered bit is set, the port sets the Port Error bit in the Port *n* Error and Status CSR and stops attempting to send packets to the connected device. Packets are discarded if the Drop Packet Enable bit is set.
2	DROP_PACKET_ENABLE		Drop packet enable
		0	The output port continues to try to transmit packets that have been rejected due to transmission errors.
		1	The output port drops packets that are acknowledged with a packet-not-accepted control symbol when the error failed threshold is exceeded. If the port "heals", and the current error rate falls below the failed threshold, the output no longer drops packets. (switch devices only)
1	PORT_LOCKOUT		Port lockout
		0	The port is enabled to issue any packets.
		1	The port is stopped and is not enabled to issue or receive any packets. The input port can still follow the training procedure and can still send and respond to link-requests. All received packets return packet-not-accepted control symbols to force an error condition to be signaled by the sending device.

Table 154. Port Control CSR *n* (SP*n*_CTL) Field Descriptions (continued)

Bit	Field	Value	Description
0	PORT_TYPE	1	Port type. This read-only bit indicates that the port is a serial port rather than a parallel port.

5.76 Error Reporting Block Header Register (ERR_RPT_BH)

The Error Reporting Block Header Register (ERR_RPT_BH) is shown in Figure 137 and described in Table 155.

Figure 137. Error Reporting Block Header Register (ERR_RPT_BH) - Address Offset 2000h

31	16	15	0
EF_PTR		EF_ID	
R-0000h		R-0007h	

LEGEND: R = Read only; -*n* = Value after reset

Table 155. Error Reporting Block Header Register (ERR_RPT_BH) Field Descriptions

Bit	Field	Value	Description
31-16	EF_PTR	0000h	Hard-wired pointer to the next block in the data structure. NONE EXISTS
15-0	EF_ID	0007h	Hard-wired Extended Features ID

5.77 Logical/Transport Layer Error Detect CSR (ERR_DET)

This register allows for the detection of logical/transport layer errors. The detectable errors are captured in the fields shown in Figure 138 and described in Table 156. For additional programming information, see Section 3 .

Figure 138. Logical/Transport Layer Error Detect CSR (ERR_DET) - Address Offset 2008h

31	30	29	28	27	26	25	24
IO_ERR_RSPNS	MSG_ERR_RSPNS	Reserved	ERR_MSG_FORMAT	ILL_TRANS_DECODE	Reserved	MSG_REQ_TIMEOUT	PKT_RSPNS_TIMEOUT
R/W-0	R/W-0	R-0	R/W-0	R/W-0	R-0	R/W-0	R/W-0

23	22	21
UNSOLICITED_RSPNS	UNSUPPORTED_TRANS	Reserved
R/W-0	R/W-0	R-0

8
Reserved
R-0

7	6	5	0
RX_CPPI_SECURITY	RX_IO_DMA_ACCESS	Reserved	
R/W-0	R/W-0	R-0	

LEGEND: R = Read; W = Write; -*n* = Value after reset

Table 156. Logical/Transport Layer Error Detect CSR (ERR_DET) Field Descriptions

Bit	Field	Value	Description
31	IO_ERR_RSPNS		I/O error response (endpoint device only)
		0	An LSU did not receive an ERROR response to an I/O logical layer request.
		1	An LSU received an ERROR response to an I/O logical layer request. To clear this bit, write 0 to it.
30	MSG_ERR_RSPNS		Message error response (endpoint device only)
		0	The TXU did not receive an ERROR response to a message logical layer request.
		1	The TXU received an ERROR response to a message logical layer request. To clear this bit, write 0 to it.
29	Reserved	0	This read-only bit returns 0 when read.
28	ERR_MSG_FORMAT		Error in message format (endpoint device only)
		0	The RXU did not receive a message data payload with an invalid size or segment.
		1	The RXU received a message data payload with an invalid size or segment. To clear this bit, write 0 to it.
27	ILL_TRANS_DECODE		Illegal transaction decode (switch or endpoint device)
			For an LSU or the TXU:
		0	The LSU/TXU did not receive illegal fields in the response packet for an IO/message transaction.
		1	The LSU/TXU received illegal fields in the response packet for an IO/message transaction. To clear this bit, write 0 to it.
			For the MAU or the RXU:
		0	The MAU/RXU did not receive illegal fields in the request packet for an IO/message transaction.
		1	The MAU/RXU received illegal fields in the request packet for an IO/message transaction. To clear this bit, write 0 to it.
26	Reserved	0	This read-only bit returns 0 when read.

Table 156. Logical/Transport Layer Error Detect CSR (ERR_DET) Field Descriptions (continued)

Bit	Field	Value	Description
25	MSG_REQ_TIMEOUT		Message request timeout (endpoint device only)
		0	A timeout has not been detected by RXU.
		1	A timeout has been detected by the RXU. A required message request has not been received by the RXU within the specified time-out interval. To clear this bit, write 0 to it.
24	PKT_RSPNS_TIMEOUT		Packet response timeout (endpoint device only)
		0	A timeout has not been detected by an LSU or the TXU.
		1	A timeout has been detected by an LSU or the TXU. A required response has not been received by the LSU/TXU within the specified timeout interval. To clear this bit, write 0 to it.
23	UNSOLICITED_RSPNS		Unsolicited response (switch or endpoint device)
		0	An unsolicited response packet has not been received by an LSU or the TXU.
		1	An unsolicited response packet has been received by an LSU or the TXU. To clear this bit, write 0 to it.
			Unsolicited responses occur when a response is received by a device that doesn't expect it. Each transmitted message request or non-posted direct IO packet is saved in a scoreboard system. The scoreboard monitors and checks off the individual response packets for each request packet. In all practicality, a device isn't going to send a message response unless it received a message request packet. In order for a response to be considered unsolicited, the scoreboard entries have been deleted. There are two scenarios where the scoreboard can be deleted after a request has been sent, but before receiving the response. First, if a request segment fails to get outbound credit, the TX descriptor is completed with a CC= 7 and the scoreboard is erased. Second, if a response timeout occurs, the TX descriptor is completed with a CC=3 and the scoreboard is erased.
22	UNSUPPORTED_TRANS		Unsupported transaction (switch or endpoint device)
		0	The MAU has not received an unsupported transaction.
		1	The MAU has received an unsupported transaction. That is, the MAU received a transaction that is not supported in the destination operations CAR. To clear this bit, write 0 to it.
21-8	Reserved	0	These read-only bits return 0 when read.
7	RX_CPPI_SECURITY		RX CPPI security error
		0	The RXU has not detected an access block.
		1	The RXU has detected an access block. That is, access to one of the RX queues was blocked. To clear this bit, write 0 to it.
6	RX_IO_DMA_ACCESS		RX I/O DMA access error
		0	A DMA access to the MAU has not been blocked.
		1	A DMA access to the MAU was blocked. To clear this bit, write 0 to it.
5-0	Reserved	0	These read-only bits return 0 when read.

5.78 Logical/Transport Layer Error Enable CSR (ERR_EN)

The logical/transport layer error enable CSR (ERR_EN) is shown in Figure 139 and described in Table 157.

Figure 139. Logical/Transport Layer Error Enable CSR (ERR_EN) - Address Offset 200Ch

31	30	29	28	27	26	25	24
IO_ERR_RESP_ENABLE	MSG_ERR_RESP_ENABLE	Reserved (write 0)	ERR_MSG_FORMAT_ENABLE	ILL_TRANS_DECODE_ENABLE	Reserved (write 0)	MSG_REQ_TIMEOUT_ENABLE	PKT_RESP_TIMEOUT_ENABLE
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

23	22	21
UNSOLICITED_RESP_ENABLE	UNSUPPORTED_TRANS_ENABLE	Reserved
R/W-0	R/W-0	R-0

8
Reserved
R-0

7	6	5	0
RX_CPPI_SECURITY_ENABLE	RX_IO_SECURITY_ENABLE	Reserved	
R/W-0	R/W-0	R-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 157. Logical/Transport Layer Error Enable CSR (ERR_EN) Field Descriptions

Bit	Field	Value	Description
31	IO_ERR_RESP_ENABLE		IO error response reporting enable
		0	Disable reporting of an IO error response.
		1	Enable reporting of an IO error response (endpoint device only). Save and lock original request transaction capture information in all Logical/Transport Layer Capture CSRs.
30	MSG_ERR_RESP_ENABLE		Message error response reporting enable
		0	Disable reporting of a message error response.
		1	Enable reporting of a message error response (endpoint device only). Save and lock transaction capture information in all Logical/Transport Layer Capture CSRs.
29	Reserved	0	Always write 0 to this reserved bit.
28	ERR_MSG_FORMAT_ENABLE		Message format error reporting enable
		0	Disable reporting of a message format error.
		1	Enable reporting of a message format error (endpoint device only). Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs.
27	ILL_TRANS_DECODE_ENABLE		Illegal transaction decode error reporting enable
		0	Disable reporting of an illegal transaction decode error.
		1	Enable reporting of an illegal transaction decode error (switch or endpoint device). Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs.
26	Reserved	0	Always write 0 to this reserved bit.
25	MSG_REQ_TIMEOUT_ENABLE		Message request time-out error reporting enable
		0	Disable reporting of a message request time-out error.
		1	Enable reporting of a message request time-out error (endpoint device only). Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs for the last message-segment request packet received.

Table 157. Logical/Transport Layer Error Enable CSR (ERR_EN) Field Descriptions (continued)

Bit	Field	Value	Description
24	PKT_RESP_TIMEOUT_ENABLE		Packet response time-out error reporting enable
		0	Disable reporting of a packet response time-out error.
		1	Enable reporting of a packet response time-out error (endpoint device only). Save and lock original request address in Logical/Transport Layer Address Capture CSRs. Save and lock original request Destination ID in Logical/Transport Layer Device ID Capture CSR.
23	UNSOLICITED_RESP_ENABLE		Unsolicited response error reporting enable
		0	Disable reporting of an unsolicited response error.
		1	Enable reporting of an unsolicited response error (switch or endpoint device). Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs.
22	UNSUPPORTED_TRANS_ENABLE		Unsupported transaction error reporting enable
		0	Disable reporting of an unsupported transaction error.
		1	Enable reporting of an unsupported transaction error (switch or endpoint device). Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs.
21-8	Reserved	0	These read-only bits return 0s when read.
7	RX_CPPI_SECURITY_ENABLE		RX CPPI security error reporting enable
		0	Disable reporting of an attempt at unauthorized access to a RX queue.
		1	Enable reporting of attempt at unauthorized access to a RX queue. Save and Lock capture information in appropriate Logical/Transport Layer Capture CSRs.
6	RX_IO_SECURITY_ENABLE		RX I/O security error reporting enable
		0	Disable reporting of attempt at unauthorized access to a memory location.
		1	Enable reporting of attempt at unauthorized access to a memory location. Save and Lock capture information in appropriate Logical/Transport Layer Capture CSRs.
5-0	Reserved	0	These read-only bits return 0s when read.

5.79 Logical/Transport Layer High Address Capture CSR (H_ADDR_CAPT)

The logical/transport layer high address capture CSR (H_ADDR_CAPT) is shown in Figure 140 and described in Table 158.

Figure 140. Logical/Transport Layer High Address Capture CSR (H_ADDR_CAPT) - Address Offset 2010h

31	0
ADDRESS_63_32	
R-00000000h	

LEGEND: R = Read only; -*n* = Value after reset

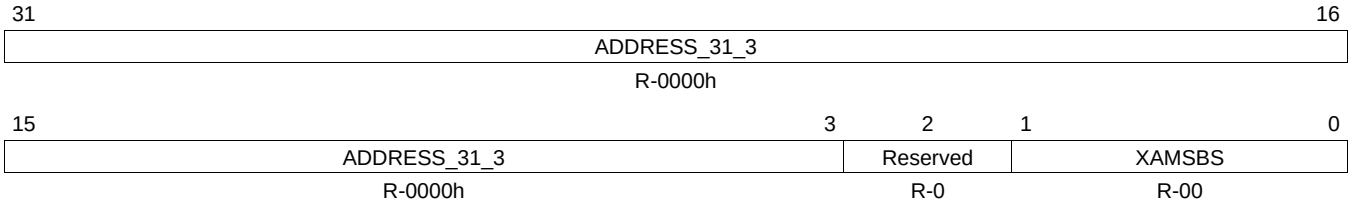
Table 158. Logical/Transport Layer High Address Capture CSR (H_ADDR_CAPT) Field Descriptions

Bit	Field	Value	Description
31-0	ADDRESS_63_32	00000000h to FFFFFFFFh	Most significant 32 bits of the address associated with the error (only valid for devices supporting 66-bit and 50-bit addresses) Bits 31-16 also capture DOORBELL_INFO field for ftype 10 packets.

5.80 Logical/Transport Layer Address Capture CSR (ADDR_CAPT)

The logical/transport layer address capture CSR (ADDR_CAPT) is shown in Figure 141 and described in Table 159.

Figure 141. Logical/Transport Layer Address Capture CSR (ADDR_CAPT) - Address Offset 2014h



LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 159. Logical/Transport Layer Address Capture CSR (ADDR_CAPT) Field Descriptions

Bit	Field	Value	Description
31-3	ADDRESS_31_3	00000000h to 1FFFFFFFh	Least significant 29 bits of the address associated with the error
2	Reserved	0	This read-only bit returns 0 when read. Also captures WDPTR value for ftype 5 packets.
1-0	XAMSBS	00b-11b	Extended address bits of the address associated with the error

5.81 Logical/Transport Layer Device ID Capture CSR (ID_CAPT)

The logical/transport layer device ID capture CSR (ID_CAPT) is shown in Figure 142 and described in Table 160.

Figure 142. Logical/Transport Layer Device ID Capture CSR (ID_CAPT) - Address Offset 2018h

31	24	23	16
MSB_DESTID		DESTID	
R-00h		R-00h	

15	8	7	0
MSB_SOURCEID		SOURCEID	
R-00h		R-00h	

LEGEND: R = Read only; -*n* = Value after reset

Table 160. Logical/Transport Layer Device ID Capture CSR (ID_CAPT) Field Descriptions

Bit	Field	Value	Description
31-24	MSB_DESTID	00h-FFh	Most significant byte of the destinationID associated with the error (large transport systems only)
23-16	DESTID	00h-FFh	The destinationID associated with the error
15-8	MSB_SOURCEID	00h-FFh	Most significant byte of the source ID associated with the error (large transport systems only)
7-0	SOURCEID	00h-FFh	The sourceID associated with the error

5.82 Logical/Transport Layer Control Capture CSR (CTRL_CAPT)

The logical/transport layer control capture CSR (CTRL_CAPT) is shown in Figure 143 and described in Table 161.

Figure 143. Logical/Transport Layer Control Capture CSR (CTRL_CAPT) - Address Offset 201Ch

31 28	27 24	23 16
FTYPE	TTYPE	MSGINFO
R-0h	R-0h	R-00h

15 0
IMP_SPECIFIC
R-0000h

LEGEND: R = Read only; -*n* = Value after reset

Table 161. Logical/Transport Layer Control Capture CSR (CTRL_CAPT) Field Descriptions

Bit	Field	Value	Description
31-28	FTYPE	0h-Fh	Format type associated with the error.
27-24	TTYPE	0h-Fh	Transaction type associated with the error.
23-16	MSGINFO	00h-FFh	Letter, mailbox, and message segment for the last message request received for the mailbox that had an error (message errors only). Also captures the SRCTID of the TX transaction identifying the LSU.
15-0	IMP_SPECIFIC	0000h-FFFFh	Implementation specific information associated with the error.

5.83 Port-Write Target Device ID CSR (PW_TGT_ID)

The port-write target device ID CSR (PW_TGT_ID) is shown in Figure 144 and described in Table 162. For additional programming information, see Section 2.3.5.

Figure 144. Port-Write Target Device ID CSR (PW_TGT_ID) - Address Offset 2028h

31	24	23	16
DEVICEID_MSB		DEVICEID	
R/W-00h		R/W-00h	

15	14	0
ID_LARGE	Reserved	
R/W-0h	R-0000h	

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 162. Port-Write Target Device ID CSR (PW_TGT_ID) Field Descriptions

Bit	Field	Value	Description
31-24	DEVICEID_MSB	00h-FFh	Most significant byte of the port-write target device ID (large transport systems only)
23-16	DEVICEID	00h-FFh	Port-write target device ID
15	ID_LARGE		Port-write large device ID
		0	8-bit device ID for port-write operation
		1	16-bit device ID for port-write operation
14-0	Reserved	0000h	These read-only bits return 0s when read

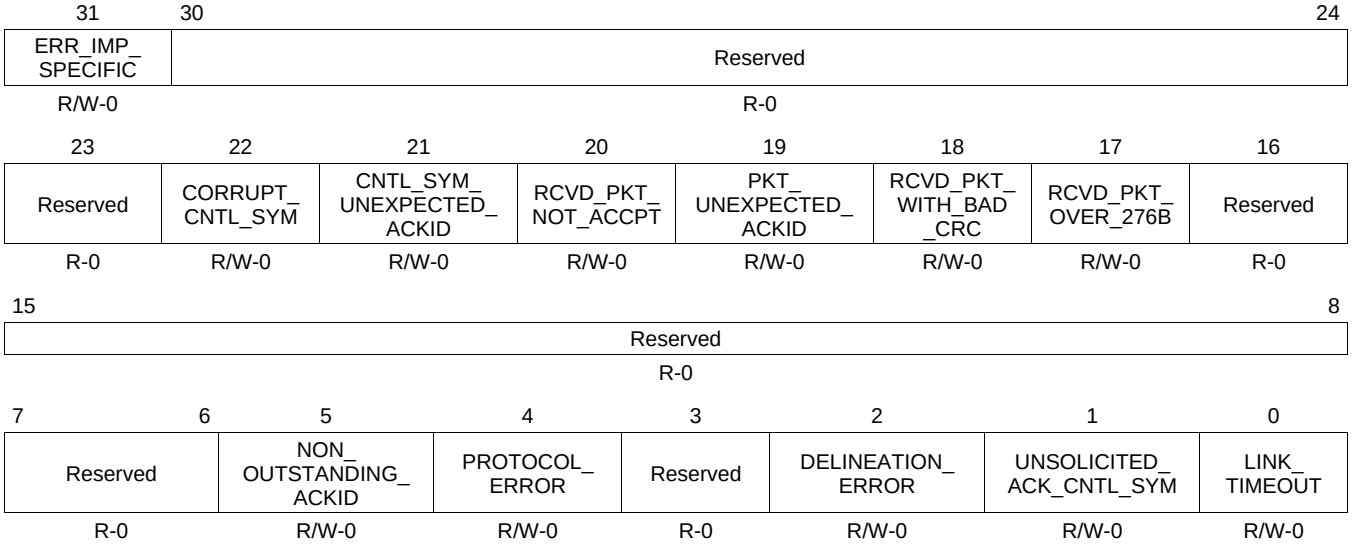
5.84 Port Error Detect CSR *n* (SP*n*_ERR_DET)

Each of the four ports is supported by a register of this type (see Table 163). The port error detect CSR *n* (SP*n*_ERR_DET) is shown in Figure 145 and described in Table 164.

Table 163. SP*n*_ERR_DET Registers and the Associated Ports

Register	Address Offset	Associated Port
SP0_ERR_DET	2040h	Port 0
SP1_ERR_DET	2080h	Port 1
SP2_ERR_DET	20C0h	Port 2
SP3_ERR_DET	2100h	Port 3

Figure 145. Port Error Detect CSR *n* (SP*n*_ERR_DET)



LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 164. Port Error Detect CSR *n* (SP*n*_ERR_DET) Field Descriptions

Bit	Field	Value	Description
31	ERR_IMP_SPECIFIC		Implementation-specific error. This bit covers errors that are a result of an illegal field in a maintenance packet, an illegal destination ID, an unsupported transaction, and an ingress UDI timeout condition. Here, an illegal field in the maintenance packet is basically the maintenance packet having ttypes that are reserved for maintenance packets. An illegal destination ID is a destination ID that is not supported. Unsupported transactions are transactions with a ttype that is reserved.
		0	An implementation specific error has not been detected.
		1	An implementation specific error has been detected.
30-23	Reserved	0	These read-only bits return 0s when read.
22	CORRUPT_CNTL_SYM		Bad CRC in control symbol
		0	The port did not receive a control symbol with a bad CRC value.
		1	The port received a control symbol with a bad CRC value.
21	CNTL_SYM_UNEXPECTED_ACKID		Unexpected ackID in control symbol
		0	The port did not receive an acknowledge control symbol with an unexpected ackID (packet-accepted or packet-retry).
		1	The port received an acknowledge control symbol with an unexpected ackID (packet-accepted or packet-retry). The capture registers do not have valid information during this error detection.

Table 164. Port Error Detect CSR *n* (SP*n*_ERR_DET) Field Descriptions (continued)

Bit	Field	Value	Description
20	RCVD_PKT_NOT_ACCPT		Packet-not-accepted control symbol
		0	The port did not receive a packet-not-accepted acknowledge control symbol.
		1	The port received a packet-not-accepted acknowledge control symbol.
19	PKT_UNEXPECTED_ACKID		Unexpected ackID in packet
		0	The port did not receive a packet with unexpected/out-of-sequence ackID.
		1	The port received a packet with unexpected/out-of-sequence ackID.
18	RCVD_PKT_WITH_BAD_CRC		Bad CRC in packet
		0	The port did not receive a packet with a bad CRC value.
		1	The port received a packet with a bad CRC value.
17	RCVD_PKT_OVER_276B		Oversize packet
		0	The port did not receive packet that exceeds the maximum allowed size.
		1	The port received packet that exceeds the maximum allowed size.
16-6	Reserved	0	These read-only bits return 0s when read.
5	NON_OUTSTANDING_ACKID		Non-outstanding ackID
		0	The port did not receive a link response with a non-outstanding ackID.
		1	The port received a link response with an ackID that is not outstanding. The capture registers do not have valid information during this error detection.
4	PROTOCOL_ERROR		Protocol error
		0	The port did not receive an unexpected packet or control symbol.
		1	The port received an unexpected packet or control symbol.
3	Reserved	0	This read-only bit returns 0 when read.
2	DELINEATION_ERROR		Delineation error
		0	The port did not detect a delineation error.
		1	The port detected a delineation error. The port received an unaligned /SC/ or /PD/ or undefined code-group. The capture registers do not have valid information during this error detection.
1	UNSOLICITED_ACK_CNTL_SYM		Unsolicited acknowledge control symbol
		0	The port did not receive an unexpected acknowledge control symbol.
		1	The port received an unexpected acknowledge control symbol.
0	LINK_TIMEOUT		Link timeout
		0	The port did not experience a link timeout.
		1	The port experienced a link timeout. The port did not receive an acknowledge or link-response control symbol within the specified time-out interval. The capture registers do not have valid information during this error detection.

5.85 Port Error Rate Enable CSR *n* (SP*n*_RATE_EN)

Each of the four ports is supported by a register of this type (see Table 165). The port error rate enable CSR *n* (SP*n*_RATE_EN) is shown in Figure 146 and described in Table 166.

Table 165. SP*n*_RATE_EN Registers and the Associated Ports

Register	Address Offset	Associated Port
SP0_RATE_EN	2044h	Port 0
SP1_RATE_EN	2084h	Port 1
SP2_RATE_EN	20C4h	Port 2
SP3_RATE_EN	2104h	Port 3

Figure 146. Port Error Rate Enable CSR *n* (SP*n*_RATE_EN)

31	30	24
ERR_IMP_SPECIFIC	Reserved	
R/W-0	R-00h	

23	22	21	20	19	18	17	16
Reserved	CORRUPT_CNTL_SYM_EN	CNTL_SYM_UNEXPECTED_ACKID_EN	RCVED_PKT_NOT_ACCPT_EN	PKT_UNEXPECTED_ACKID_EN	RCVED_PKT_WITH_BAD_CRC_EN	RCVED_PKT_OVER_276B_EN	Reserved
R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0

15	8
Reserved	
R-0	

7	6	5	4	3	2	1	0
Reserved		NON_OUTSTANDING_ACKID_EN	PROTOCOL_ERROR_EN	Reserved	DELINEATION_ERROR_EN	UNSOLICITED_ACK_CNTL_SYM_EN	LINK_TIMEOUT_EN
R-0		R/W-0	R/W-0	R-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 166. Port Error Rate Enable CSR *n* (SP*n*_RATE_EN) Field Descriptions

Bit	Field	Value	Description
31	EN_IMP_SPECIFIC		Rate counting enable for implementation-specific errors
		0	Disable error rate counting of implementation specific errors
		1	Enable error rate counting of implementation specific errors.
30-23	Reserved	0	These read-only bits return 0s when read.
22	CORRUPT_CNTL_SYM_ENABLE		Rate counting enable for corrupt control symbols
		0	Disable error rate counting of a corrupt control symbol.
		1	Enable error rate counting of a corrupt control symbol.
21	CNTL_SYM_UNEXPECTED_ACKID_EN		Rate counting enable for control symbols with unexpected ackIDs
		0	Disable error rate counting of an acknowledge control symbol with an unexpected ackID.
		1	Enable error rate counting of an acknowledge control symbol with an unexpected ackID.
20	RCVED_PKT_NOT_ACCPT_EN		Rate counting enable for packet-not-accepted control symbols
		0	Disable error rate counting of received packet-not-accepted control symbols.
		1	Enable error rate counting of received packet-not-accepted control symbols.
19	PKT_UNEXPECTED_ACKID_EN		Rate counting enable for packets with unexpected ackIDs

Table 166. Port Error Rate Enable CSR *n* (SP*n*_RATE_EN) Field Descriptions (continued)

Bit	Field	Value	Description
		0	Disable error rate counting of packets with unexpected/out-of-sequence ackIDs
		1	Enable error rate counting of packets with unexpected/out-of-sequence ackIDs.
18	RCVED_PKT_WITH_BAD_CRC_EN		Rate counting enable for packets with bad CRC
		0	Disable error rate counting of packets with a bad CRC values.
		1	Enable error rate counting of packets with a bad CRC values.
17	RCVED_PKT_OVER_276B_EN		Rate counting enable for oversize packets
		0	Disable error rate counting of packets that exceed the maximum allowed size.
		1	Enable error rate counting of packets that exceed the maximum allowed size.
16-6	Reserved	0	These read-only bits return 0s when read.
5	NON_OUTSTANDING_ACKID_EN		Rate counting enable for non-outstanding ackIDs
		0	Disable error rate counting of link-responses received with an ackID that is not outstanding.
		1	Enable error rate counting of link-responses received with an ackID that is not outstanding.
4	PROTOCOL_ERROR_EN		Rate counting enable for protocol errors
		0	Disable error rate counting of protocol errors.
		1	Enable error rate counting of protocol errors.
3	Reserved	0	This read-only bit returns 0 when read.
2	DELINEATION_ERROR_EN		Rate counting enable for delineation errors
		0	Disable error rate counting of delineation errors.
		1	Enable error rate counting of delineation errors.
1	UNSOLICITED_ACK_CNTL_SYM_EN		Rate counting enable for unsolicited acknowledge control symbols
		0	Disable error rate counting of unsolicited acknowledge control symbols.
		1	Enable error rate counting of unsolicited acknowledge control symbols.
0	LINK_TIMEOUT_EN		Rate counting enable for link time-out errors
		0	Disable error rate counting of link timeout errors.
		1	Enable error rate counting of link timeout errors.

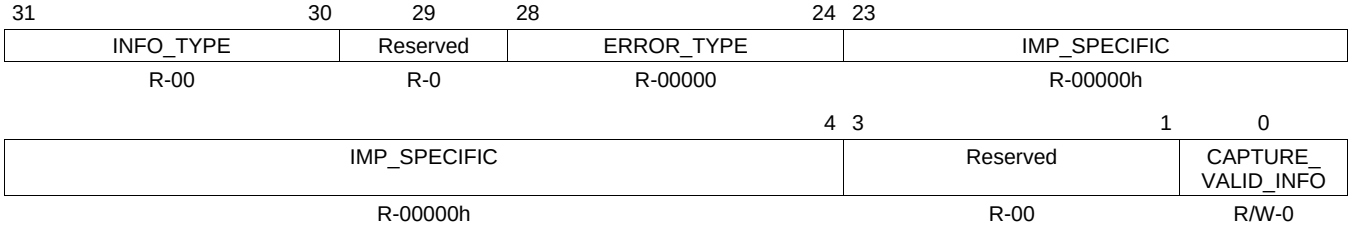
5.86 Port *n* Attributes Error Capture CSR 0 (SP*n*_ERR_ATTR_CAPT_DBG0)

Each of the four ports is supported by a register of this type (see Table 167). The port *n* attributes error capture CSR 0 (SP*n*_ERR_ATTR_CAPT_DBG0) is shown in Figure 147 and described in Table 168.

Table 167. SP*n*_ERR_ATTR_CAPT_DBG0 Registers and the Associated Ports

Register	Address Offset	Associated Port
SP0_ERR_ATTR_CAPT_DBG0	2048h	Port 0
SP1_ERR_ATTR_CAPT_DBG0	2088h	Port 1
SP2_ERR_ATTR_CAPT_DBG0	20C8h	Port 2
SP3_ERR_ATTR_CAPT_DBG0	2108h	Port 3

Figure 147. Port *n* Attributes Error Capture CSR 0 (SP*n*_ERR_ATTR_CAPT_DBG0)



LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 168. Port *n* Attributes Error Capture CSR 0 (SP*n*_ERR_ATTR_CAPT_DBG0) Field Descriptions

Bit	Field	Value	Description
31-30	INFO_TYPE		Type of information logged
		00b	Packet
		01b	Control symbol (only error capture register 0 is valid)
		10b	Implementation specific (capture register contents are implementation specific)
		11b	Reserved
29	Reserved	0	This read-only bit returns 0 when read.
28-24	ERROR_TYPE	00000b-11111b	Encoded value of captured error bit in the Port *n* Error Detect Register
23-4	IMP_SPECIFIC	00000h-FFFFFh	Implementation Dependent Error Information
3-1	Reserved	0	These read-only bits return 0s when read.
0	CAPTURE_VALID_INFO		Valid information captured
		0	The packet/control symbol capture registers do not contain valid information.
		1	The packet/control symbol capture registers contain valid information. For control symbols, only capture register 0 contains meaningful information. A software write of 0 clears this bit and subsequently unlocks all the capture registers of the port.

5.87 Port *n* Error Capture CSR 1 (SP*n*_ERR_CAPT_DBG1)

Each of the four ports is supported by a register of this type (see Table 169). SP*n*_ERR_CAPT_DBG1 is shown in Figure 148 and described in Table 170.

Table 169. SP*n*_ERR_CAPT_DBG1 Registers and the Associated Ports

Register	Address Offset	Associated Port
SP0_ERR_CAPT_DBG1	204Ch	Port 0
SP1_ERR_CAPT_DBG1	208Ch	Port 1
SP2_ERR_CAPT_DBG1	20CCh	Port 2
SP3_ERR_CAPT_DBG1	210Ch	Port 3

Figure 148. Port *n* Error Capture CSR 1 (SP*n*_ERR_CAPT_DBG1)

31	0
CAPTURE0	
R-00000000h	

LEGEND: R = Read only; -*n* = Value after reset

Table 170. Port *n* Error Capture CSR 1 (SP*n*_ERR_CAPT_DBG1) Field Descriptions

Bit	Field	Value	Description
31-0	CAPTURE0	00000000h to FFFFFFFFh	**In the case of a control-symbol error:** Control character and control symbol that correspond to the error **In the case of a packet error:** Bytes 0 to 3 of the packet header that corresponds to the error

5.88 Port *n* Error Capture CSR 2 (SP*n*_ERR_CAPT_DBG2)

Each of the four ports is supported by a register of this type (see Table 171). SP*n*_ERR_CAPT_DBG2 is shown in Figure 149 and described in Table 172.

Table 171. SP*n*_ERR_CAPT_DBG2 Registers and the Associated Ports

Register	Address Offset	Associated Port
SP0_ERR_CAPT_DBG2	2050h	Port 0
SP1_ERR_CAPT_DBG2	2090h	Port 1
SP2_ERR_CAPT_DBG2	20D0h	Port 2
SP3_ERR_CAPT_DBG2	2110h	Port 3

Figure 149. Port *n* Error Capture CSR 2 (SP*n*_ERR_CAPT_DBG2)

31	0
CAPTURE1	
R-00000000h	

LEGEND: R = Read only; -*n* = Value after reset

Table 172. Port *n* Error Capture CSR 2 (SP*n*_ERR_CAPT_DBG2) Field Descriptions

Bit	Field	Value	Description
31-0	CAPTURE1	00000000h to FFFFFFFFh	Bytes 4 to 7 of the packet header that corresponds to the error

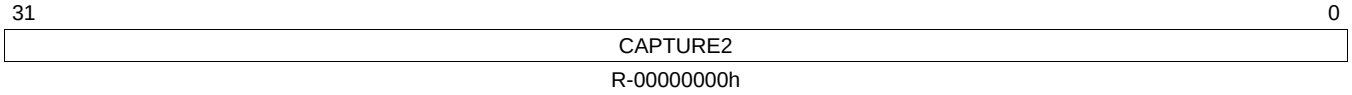
5.89 Port *n* Error Capture CSR 3 (SP*n*_ERR_CAPT_DBG3)

Each of the four ports is supported by a register of this type (see Table 173). SP*n*_ERR_CAPT_DBG3 is shown in Figure 150 and described in Table 174.

Table 173. SP*n*_ERR_CAPT_DBG3 Registers and the Associated Ports

Register	Address Offset	Associated Port
SP0_ERR_CAPT_DBG3	2054h	Port 0
SP1_ERR_CAPT_DBG3	2094h	Port 1
SP2_ERR_CAPT_DBG3	20D4h	Port 2
SP3_ERR_CAPT_DBG3	2114h	Port 3

Figure 150. Port *n* Error Capture CSR 3 (SP*n*_ERR_CAPT_DBG3)



LEGEND: R = Read only; -*n* = Value after reset

Table 174. Port *n* Error Capture CSR 3 (SP*n*_ERR_CAPT_DBG3) Field Descriptions

Bit	Field	Value	Description
31-0	CAPTURE2	00000000h to FFFFFFFFh	Bytes 8 to 11 of the packet header that corresponds to the error

5.90 Port *n* Error Capture CSR 4 (SP*n*_ERR_CAPT_DBG4)

Each of the four ports is supported by a register of this type (see Table 175). The port *n* packet/control symbol error capture CSR 4 (SP*n*_ERR_CAPT_DBG4) is shown in Figure 151 and described in Table 176.

Table 175. SP*n*_ERR_CAPT_DBG4 Registers and the Associated Ports

Register	Address Offset	Associated Port
SP0_ERR_CAPT_DBG4	2058h	Port 0
SP1_ERR_CAPT_DBG4	2098h	Port 1
SP2_ERR_CAPT_DBG4	20D8h	Port 2
SP3_ERR_CAPT_DBG4	2118h	Port 3

Figure 151. Port *n* Error Capture CSR 4 (SP*n*_ERR_CAPT_DBG4)

31	0
CAPTURE3	
R-00000000h	

LEGEND: R = Read only; -*n* = Value after reset

Table 176. Port *n* Error Capture CSR 4 (SP*n*_ERR_CAPT_DBG4) Field Descriptions

Bit	Field	Value	Description
31-0	CAPTURE3	00000000h to FFFFFFFFh	Bytes 12 to 15 of the packet header that corresponds to the error

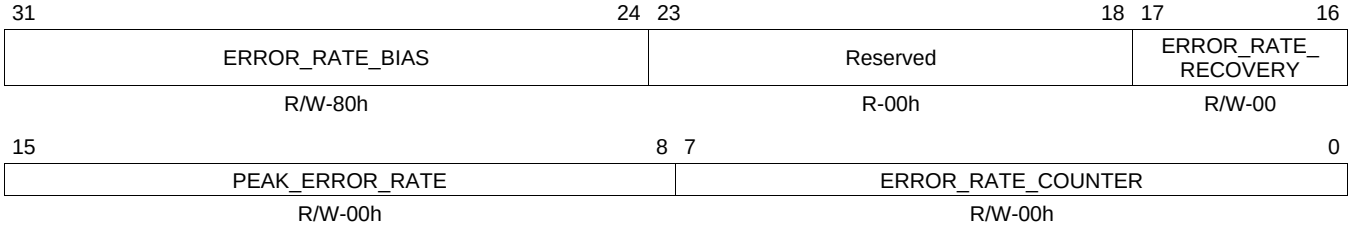
5.91 Port Error Rate CSR *n* (SP*n*_ERR_RATE)

Each of the four ports is supported by a register of this type (see Table 177). SP*n*_ERR_RATE is shown in Figure 152 and described in Table 178.

Table 177. SP*n*_ERR_RATE Registers and the Associated Ports

Register	Address Offset	Associated Port
SP0_ERR_RATE	2068h	Port 0
SP1_ERR_RATE	20A8h	Port 1
SP2_ERR_RATE	20E8h	Port 2
SP3_ERR_RATE	2128h	Port 3

Figure 152. Port Error Rate CSR *n* (SP*n*_ERR_RATE)



LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 178. Port Error Rate CSR *n* (SP*n*_ERR_RATE) Field Descriptions

Bit	Field	Value	Description
31-24	ERROR_RATE_BIAS		These bits provide the error rate bias value.
		00h	Do not decrement the error rate counter
		01h	Decrement every 1ms (nominal)
		03h	Decrement every 10ms (nominal)
		07h	Decrement every 100ms
		0Fh	Decrement every 1s (nominal)
		1Fh	Decrement every 10s (nominal)
		3Fh	Decrement every 100s (nominal)
		7Fh	Decrement every 1000s (nominal)
		FFh	Decrement every 10000s (nominal)
		Other	Reserved
23-18	Reserved	00h	These read-only bits return 0s when read.
17-16	ERROR_RATE_RECOVERY		These bits limit the incrementing of the error rate counter above the failed threshold trigger.
		00b	Only count 2 errors above
		01b	Only count 4 errors above
		10b	Only count 16 errors above
		11b	Do not limit incrementing the error rate count
15-8	PEAK_ERROR_RATE	00h-FFh	This field contains the peak value attained by the error rate counter.
7-0	ERROR_RATE_COUNTER	00h-FFh	These bits maintain a count of the number of transmission errors that have occurred. If this value equals the value contained in the error rate threshold trigger register, then an error will be reported.

5.92 Port Error Rate Threshold CSR *n* (SP*n*_ERR_THRESH)

Each of the four ports is supported by a register of this type (see Table 179). The port error rate threshold CSR *n* (SP*n*_ERR_THRESH) is shown in Figure 153 and described in Table 180.

Table 179. SP*n*_ERR_THRESH Registers and the Associated Ports

Register	Address Offset	Associated Port
SP0_ERR_THRESH	206Ch	Port 0
SP1_ERR_THRESH	20ACh	Port 1
SP2_ERR_THRESH	20ECh	Port 2
SP3_ERR_THRESH	212Ch	Port 3

Figure 153. Port Error Rate Threshold CSR *n* (SP*n*_ERR_THRESH)

31 24	23 16
ERROR_RATE_FAILED_THRESH	ERROR_RATE_DEGRADED_THRESH
R/W-FFh	R/W-FFh

15 0
Reserved
R-0000h

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 180. Port Error Rate Threshold CSR *n* (SP*n*_ERR_THRESH) Field Descriptions

Bit	Field	Value	Description
31-24	ERROR_RATE_FAILED_THRESH		These bits provide the threshold value for reporting an error condition due to a possibly broken link.
		00h	Disable the error rate register
		01h	Set the error reporting threshold to 1
		02h	Set the error reporting threshold to 2
		...	...
		FFh	Set the error reporting threshold to 255
23-16	ERROR_RATE_DEGRADED_THRESH		These bits provide the threshold value for reporting an error condition due to a degrading link.
		00h	Disable the error rate Register
		01h	Set the error reporting threshold to 1
		02h	Set the error reporting threshold to 2
		...	...
		FFh	Set the error reporting threshold to 255
15-0	Reserved	0000h	These read-only bits return 0s when read.

5.93 Port IP Discovery Timer for 4x Mode Register (SP_IP_DISCOVERY_TIMER)

The port IP discovery timer for 4x mode register (SP_IP_DISCOVERY_TIMER) is shown in Figure 154 and described in Table 181.

Figure 154. Port IP Discovery Timer for 4x Mode Register (SP_IP_DISCOVERY_TIMER) - Address Offset 12000h

31–28	27–24	23–20	19–16
DISCOVERY_TIMER	Reserved	PW_TIMER	Reserved
R/W-9h	R-0h	R/W-8h	R-0h

15–0
Reserved
R-0000h

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 181. Port IP Discovery Timer for 4x Mode Register (SP_IP_DISCOVERY_TIMER) Field Descriptions

Bit	Field	Value	Description
31-28	DISCOVERY_TIMER		Discovery timer for 4x mode. The discovery timer allows time for the link partner to enter its DISCOVERY state and if the link partner is supporting 4x mode, for all 4 lanes to be aligned.
		0000b	Reserved
		0001b	0.84 ms
		0010b	0.84 ms x 2 = 1.68 ms
		...	...
		1001b	0.84 ms x 9= 7.56 ms (default)
		...	...
		1111b	0.84 ms x 15= 12.6 ms
27-24	Reserved	0000b	These read-only bits return 0s when read.
23-20	PW_TIMER		Port-write timer. The timer defines a period to repeat sending an error reporting port-write request for software assistance. The timer is stopped by software writing to the error detect registers.
		0000b	Disabled. Port-write is sent once only.
		0001b	107 ms-214 ms
		0010b	214 ms-321 ms
		0100b	428 ms-535 ms
		1000b	856 ms-963 ms (default)
		Other	Reserved
19-0	Reserved	0000h	These read-only bits return 0s when read.

5.94 Port IP Mode CSR (SP_IP_MODE)

The port IP mode CSR (SP_IP_MODE) is shown in Figure 155 and described in Table 182. For additional programming information, see Section 2.3.13.2 .

Figure 155. Port IP Mode CSR (SP_IP_MODE) - Address Offset 12004h

31 30	29	28	27	26	25	24
SP_MODE	IDLE_ERR_DIS	TX_FIFO_BYPASS	PW_DIS	SRC_TGT_ID_DIS	SELF_RST	F8_TGT_ID_DIS
R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-1h

23 8
Reserved
R-0

7 6	5	4	3	2	1	0
Reserved	MLTC_EN	MLTC_IRQ	RST_EN	RST_CS	PW_EN	PW_IRQ
R-0	R/W-0	R/W-0, 1 to clear	R/W-0	R/W-0, 1 to clear	R/W-0	R/W-0, 1 to clear

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 182. Port IP Mode CSR (SP_IP_MODE) Field Descriptions

Bit	Field	Value	Description
31-30	SP_MODE		SRIO port IP mode of operation
		00b	RapidIO *Physical Layer 1x/4x LP-Serial Specification*
		01b	4 ports (1x mode each)
		10b	Reserved
		11b	Reserved
29	IDLE_ERR_DIS		Idle error checking disable
		0	Error checking enabled (default), only |K|, |A| and |R| characters are available. If input receives any other characters in idle sequence, it should enter the Input-Error-stopped state.
		1	Error checking disabled, all not idle or invalid characters during idle sequence are ignored
28	TX_FIFO_BYPASS		Transmit FIFO bypass
		0	The TX_FIFO is operational (default)
		1	The TX_FIFO is bypassed. The txbclk and the sys_clk must be locked during operation, but the phase variation up to 1 clock cycle is allowable. The 4 deep FIFO is used to accommodate the phase difference.
27	PW_DIS		Port-write error reporting disable.
		0	Enable Port-Write Error reporting (default)
		1	Disable Port-Write Error reporting
26	SRC_TGT_ID_DIS		Destination ID Decode Disable-Definition of packet acceptance by the physical layer.
		0	Packet accepted if DESTID = Base ID. When DESTID is not equal to Base ID, the packet is ignored; i.e., it is accepted by RapidIO port but is not forwarded to logical layer.
		1	Packet accepted with any DESTID and forwarded to the logical layer.
25	SELF_RST		Self-reset interrupt enable, when 4 link-request reset control symbols are accepted.
		0	Self reset interrupt disabled (default), interrupt signal is asserted
		1	Self reset interrupt enabled, the reset signal is asserted by the reset controller. When the SELF_RST is set to 1, the SERDES macro resets and all register values from address offset 1000h and higher are returned to default value. All initialized values are lost.
24	F8_TGT_ID_DIS		Physical Layer Disable. This bit only disables the Ftype8 (Maintenance Packet) checking at the physical layer.
		0	Non-matching Ftype8 packets are forwarded to MAC layer.
		1	Regardless of Ftype8 packet destination ID, the physical layer accepts and handles the Maintenance request packets.
23-6	Reserved	0	These read-only bits return 0s when read.

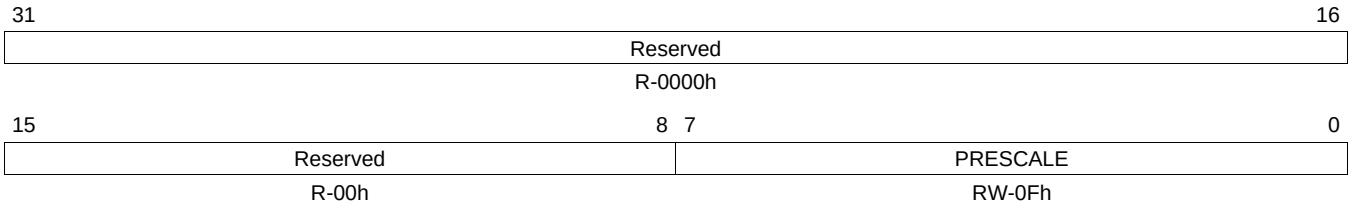
Table 182. Port IP Mode CSR (SP_IP_MODE) Field Descriptions (continued)

Bit	Field	Value	Description
5	MLTC_EN		Multicast-Event Interrupt Enable. If enabled, the interrupt signal is High when the Multicast-Event control symbol is received by any port.
		0	Multicast interrupt disable
		1	Multicast interrupt enable
4	MLTC_IRQ		Multicast-Event Interrupt Status. Once set, the MLTC_IRQ bit remains set until software writes a 1 to it. The mltc_irq output signal is driven by this bit.
		0	The multicast event control symbol has not been received by any of the ports.
		1	The multicast-event control symbol has been received by one of the ports.
3	RST_EN		Reset Interrupt Enable. If enabled, the interrupt signal is High when the 4 reset control symbols are received in a sequence
		0	Reset interrupt disable
		1	Reset interrupt enable
2	RST_CS		Reset received status bit. It is set when Once set, the RST_CS bit remains set until software writes a 1 to it. The rst_irq output signal is driven by this bit.
		0	Four reset control symbols have not been received in a sequence.
		1	Four reset control symbols have been received in a sequence.
1	PW_EN		Port-Write-In Interrupt Enable. If enabled, the interrupt signal is High when the Port-Write-In request is received
		0	Port-Write-In interrupt disable
		1	Port-Write-In interrupt enable
0	PW_IRQ		Port-Write-In request interrupt. Once set, the PW_IRQ bit remains set until software writes a 1 to it. The pw_irq output signal is driven by this bit.
		0	The Port-Write-In request has not been received.
		1	The Port-Write-In request has been received. The payload is captured.

5.95 Port IP Prescaler Register (IP_PRESCAL)

The port IP prescaler register (IP_PRESCAL) is shown in Figure 156 and described in Table 183. This register defines a prescaler for different frequencies of the DMA clock. The purpose of this register is to keep the timers of SP_LT_CTL (offset 01120h), SP0_ERR_RATE through SP3_ERR_RATE (offsets 02068h, 020A8h, 020E8, and 02128h), SP_IP_DISCOVERY_TIMER (offset 12000h), and SP0_SILENCE_TIMER through SP3_SILENCE_TIMER (offsets 14008h, 14108h, 14208h, and 14308h) within the same range for different frequencies of the DMA clock.

Figure 156. Port IP Prescaler Register (IP_PRESCAL) - Address Offset 12008h



LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 183. Port IP Prescaler Register (IP_PRESCAL) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	000000h	These read-only bits return 0s when read.
7-0	PRESCALE		For different frequencies of the DMA clock, use the following formula to get the prescaler value in decimal, where the DMA clock frequency is in MHz:
			{(DMA clock frequency x 16) / 156.25} - 1
		06h	66.67 MHz
		...	...
		09h	100 MHz
		...	...
		0Fh	156.25 MHz
		10h	166.67 MHz
		...	...
		18h	250 MHz
		...	
		21h	333 MHz

5.96 Port-Write-In Capture CSRs (SP_IP_PW_IN_CAPT[0-3])

Four registers are used to capture the incoming 128-bit payload of a Port-Write. These four registers are shown in Figure 157. As can be seen in Table 184, each of the registers captures one of the four 32-bit words of the payload.

Figure 157. Port-Write-In Capture CSRs

Port-Write-In Capture CSR 0 (SP_IP_PW_IN_CAPT0) - Address Offset 12010h

31	0
PW_CAPT0	
R-00000000h	

Port-Write-In Capture CSR 1 (SP_IP_PW_IN_CAPT1) - Address Offset 12014h

31	0
PW_CAPT1	
R-00000000h	

Port-Write-In Capture CSR 0 (SP_IP_PW_IN_CAPT0) - Address Offset 12018h

31	0
PW_CAPT2	
R-00000000h	

Port-Write-In Capture CSR 0 (SP_IP_PW_IN_CAPT0) - Address Offset 1201Ch

31	0
PW_CAPT3	
R-00000000h	

LEGEND: R = Read only; -*n* = Value after reset

Table 184. Port-Write-In Capture CSR Field Descriptions

Field	Value	Description
PW_CAPT0	00000000h to FFFFFFFFh	Word 0 (bits 0 to 31) of the Port-Write payload.
PW_CAPT1	00000000h to FFFFFFFFh	Word 1 (bits 32 to 63) of the Port-Write payload.
PW_CAPT2	00000000h to FFFFFFFFh	Word 2 (bits 64 to 95) of the Port-Write payload.
PW_CAPT3	00000000h to FFFFFFFFh	Word 3 (bits 96 to 127) of the Port-Write payload.

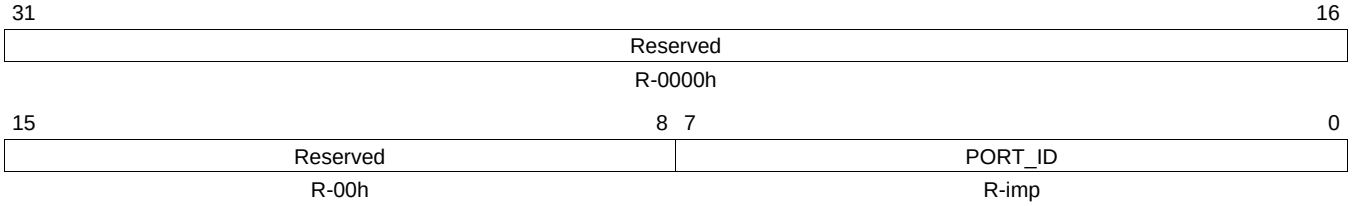
5.97 Port Reset Option CSR *n* (SP*n*_RST_OPT)

Each of the four ports is supported by a register of this type (see Table 185). SP*n*_RST_OPT is shown in Figure 158 and described in Table 186.

Table 185. SP*n*_RST_OPT Registers and the Associated Ports

Register	Address Offset	Associated Port
SP0_RST_OPT	14000h	Port 0
SP1_RST_OPT	14100h	Port 1
SP2_RST_OPT	14200h	Port 2
SP3_RST_OPT	14300h	Port 3

Figure 158. Port Reset Option CSR *n* (SP*n*_RST_OPT)



LEGEND: R = Read only; -*n* = Value after reset; -imp = Value after reset is implementation defined.

Table 186. Port Reset Option CSR *n* (SP*n*_RST_OPT) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	These read-only bits return 0s when read.
7-0	PORT_ID		Port ID defines unique number for port in Switch. The Port ID is used for port-write request. The ID coincides with ISF port of connection. Example: 00_0000_01 _ port 1 (Impl.: IP0, port 1) 00_0001_11 _ port 7 (Impl.: IP1, port 3).

5.98 Port Control Independent Register *n* (SP*n*_CTL_INDEP)

Each of the four ports is supported by a register of this type (see Table 187). The port control independent register *n* (SP*n*_CTL_INDEP) is shown in Figure 159 and described in Table 188.

Table 187. SP*n*_CTL_INDEP Registers and the Associated Ports

Register	Address Offset	Associated Port
SP0_CTL_INDEP	14004h	Port 0
SP1_CTL_INDEP	14104h	Port 1
SP2_CTL_INDEP	14204h	Port 2
SP3_CTL_INDEP	14304h	Port 3

Figure 159. Port Control Independent Register *n* (SP*n*_CTL_INDEP)

31	30	29	28	27	26	25	24
Reserved	TX_FLW	SOFT_REC	Reserved		FORCE_REINIT	TRANS_MODE	
R-0	R/W-0	R/W-0	R-0		W-0	R/W-01	

23	22	21	20	19	18	17	16
DEBUG	SEND_DBG_PKT	ILL_TRANS_EN	ILL_TRANS_ERR	Reserved		MAX_RETRY_EN	MAX_RETRY_ERR
R/W-0	R/W-0	R/W-0	R/W-0	R-0		R/W-0	R/W-0

15	8
MAX_RETRY_THR	
R/W-00h	

7	6	5	0
IRQ_EN	IRQ_ERR	Reserved	
R/W-0	R/W-0	R-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 188. Port Control Independent Register *n* (SP*n*_CTL_INDEP) Field Descriptions

Bit	Field	Value	Description
31	Reserved	0	This read-only bit returns 0 when read.
30	TX_FLW		Transmit Link Flow Control enable
		0	Disables transmit flow control (Enables receive link flow control)
		1	Reserved
29	SOFT_REC		Software controlled error recovery
		0	Transmission of error recovery sequence is performed by the hardware
		1	Transmission of error recovery sequence is performed by the software. By default the transmission error recovery sequence is performed by the hardware. If this bit is set, the hardware recovery is disabled and the hardware transmission logic must wait until software has written the register Port *n* Local ackID Status CSR.
28-27	Reserved	0	These read-only bits return 0s when read.
26	FORCE_REINIT		Force reinitialization process. In 4x mode this bit affects all 4 lanes. This bit is write only, and reads always return 0.
25-24	TRANS_MODE		Describes the transfer mode for each port.
		00b	Reserved (Cut-Through Mode)
		01b	Store & Forward Mode
		1xb	Reserved
23	DEBUG		Mode of operation.
		0	Normal mode
		1	Debug mode. The debug mode unlocks capture registers for write and enable debug packet generator feature.

Table 188. Port Control Independent Register *n* (SP*n*_CTL_INDEP) Field Descriptions (continued)

Bit	Field	Value	Description
22	SEND_DBG_PKT		Send debug packet. Write 1 to force the sending of a debug packet. This bit is set by software and cleared after debug packet is sent. Writes when the bit is set are ignored. Debug mode only.
21	ILL_TRANS_EN		Illegal transfer error reporting enable. If enabled, the Port-Write and interrupt are reported errors.
		0	Disable illegal transfer error reporting.
		1	Enable illegal transfer error reporting.
20	ILL_TRANS_ERR		Illegal Transfer Error. After being set, the ILL_TRANS_ERR bit remains set until written with a 1, or until a value of all 0s is written to the register SP0_ERR_DET.
		0	No error condition detected
		1	One of the following error conditions has been detected: • The received transaction has a reserved value in the tt field. • A reserved field of Maintenance transaction type is detected. • The destination ID is not defined in look-up table. This error is also reported in registers SP0_ERR_DET and ERR_DET.
19-18	Reserved	0	These read-only bits return 0s when read.
17	MAX_RETRY_EN		Maximum retry error reporting enable. If enabled, the Port-Write and interrupt are reported as errors.
		0b	Max retry error report disable
		1b	Max retry error report enable
16	MAX_RETRY_ERR		Maximum retry error. This bit is ignored if max_retry_threshold is 0. Once set, the MAX_RETRY_ERR bit remains set until written with a 1, or until a value of all 0s is written to the register SP0_ERR_DET.
		0	No error condition detected
		1	max_retry_cnt is equal to max_retry_threshold. The Port-Write request and interrupt are generated if enabled. This error is also reported in the register SP0_ERR_DET.
15-8	MAX_RETRY_THR		Maximum Retry Threshold Trigger. These bits provide the threshold value for reporting an error condition due to possibly broken partner behavior.
		00h	Disable the max_retry_error reporting
		01h	Set the max_retry_threshold to 1
		02h	Set the max_retry_threshold to 2
		...	...
		FFh	Set the max_retry_threshold to 255
7	IRQ_EN		Interrupt error reporting enable. If enabled, the interrupt signal is high when the IRQ_ERR is set to 1.
		0	Interrupt error report disable
		1	Interrupt error report enable
6	IRQ_ERR		Interrupt error status
		0	An error has not occurred and/or there is not a Port-Write condition.
		1	An error occurred and there is a Port-Write condition. IRQ_ERR remains at 1 until software writes a 1 to it.
5-0	Reserved	0	These read-only bits return 0s when read.

5.99 Port Silence Timer *n* Register (SP*n*_SILENCE_TIMER)

Each of the four ports is supported by a register of this type (see Table 189). The port silence timer *n* register (SP*n*_SILENCE_TIMER) is shown in Figure 160 and described in Table 190.

Table 189. SP*n*_SILENCE_TIMER Registers and the Associated Ports

Register	Address Offset	Associated Port
SP0_SILENCE_TIMER	14008h	Port 0
SP1_SILENCE_TIMER	14108h	Port 1
SP2_SILENCE_TIMER	14208h	Port 2
SP3_SILENCE_TIMER	14308h	Port 3

Figure 160. Port Silence Timer *n* Register (SP*n*_SILENCE_TIMER)

31	28	27	16
SILENCE_TIMER		Reserved	
R/W-Bh		R-0	

15	0
Reserved	
R-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 190. Port Silence Timer *n* Register (SP*n*_SILENCE_TIMER) Field Descriptions

Bit	Field	Value	Description
31-28	SILENCE_TIMER		Silence timer. Defines the time of the port in the SILENT state.
		0000b	64 ns for debug
		0001b	13.1 µs
		0010b	13.1 µs x 2 = 26.2 µs
		...	...
		1011b	13.1 µs x 11 = 144.1 µs default
		...	...
		1111b	13.1 µs x 15= 196.5 µs
27-0	Reserved	0	These read-only bits return 0s when read.

5.100 Port Multicast-Event Control Symbol Request Register *n* (SP*n*_MULT_EVNT_CS)

Each of the four ports is supported by a register of this type (see Table 191). The port multicast-event control symbol request register *n* (SP*n*_MULT_EVNT_CS) is shown in Figure 161 and described in Table 192.

Table 191. SP*n*_MULT_EVNT_CS Registers and the Associated Ports

Register	Address Offset	Associated Port
SP0_MULT_EVNT_CS	1400Ch	Port 0
SP1_MULT_EVNT_CS	1410Ch	Port 1
SP2_MULT_EVNT_CS	1420Ch	Port 2
SP3_MULT_EVNT_CS	1430Ch	Port 3

Figure 161. Port Multicast-Event Control Symbol Request Register *n* (SP*n*_MULT_EVNT_CS)

31	0
MULT_EVNT_CS	
W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 192. Port Multicast-Event Control Symbol Request Register *n* (SP*n*_MULT_EVNT_CS) Field Descriptions

Bit	Field	Value	Description
31-0	MULT_EVNT_CS		Write to send Control Symbol, data is ignored. Reads return 000000h.

5.101 Port Control Symbol Transmit *n* Register (SP*n*_CS_TX)

Each of the four ports is supported by a register of this type (see Table 193). The port control symbol transmit *n* register (SP*n*_CS_TX) is shown in Figure 162 and described in Table 194.

Table 193. SP*n*_CS_TX Registers and the Associated Ports

Register	Address Offset	Associated Port
SP0_CS_TX	14014h	Port 0
SP1_CS_TX	14114h	Port 1
SP2_CS_TX	14214h	Port 2
SP3_CS_TX	14314h	Port 3

Figure 162. Port Control Symbol Transmit *n* Register (SP*n*_CS_TX)

31–29	28–24	23–19	18–16
STYPE_0	PAR_0	PAR_1	STYPE_1
R/W-0	R/W-0	R/W-0	R/W-0

15–13	12	11–0
CMD	CS_EMB	Reserved
R/W-0	R/W-0	R-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = Value after reset

Table 194. Port Control Symbol Transmit *n* Register (SP*n*_CS_TX) Field Descriptions

Bit	Field	Value	Description
31-29	STYPE_0		Encoding for control symbol that makes use of parameters PAR_0 and PAR_1.
28-24	PAR_0		Used in conjunction with stype0 encoding.
23-19	PAR_1		Used in conjunction with stype0 encoding.
18-16	STYPE_1		Encoding for control symbol that makes use of parameter CMD.
15-13	CMD		Used in conjunction with stype1 encoding to define the link maintenance commands.
12	CS_EMB		When set, forces the outbound flow to insert control symbol into packet. Used in debug mode.
11-0	Reserved	0	These read-only bits return 0s when read.

Appendix A Examples

A.1 SRIO Initialization Example

```
/* Initialization and Open of the SRIO */
    status = CSL_srioInit (&context);
    hSrio = CSL_srioOpen (&srioObj, srioNum, &srioParam, &status);
    if (status != CSL_SOK) {
        printf("SRIO: ... Cannot open SRIO, failed\n");
        return;
    }

/* Create the setup structure */
   srio_Create_Setup (&setup);

/* Setup the SRIO with the selected setup in the last step */
   status = CSL_srioHwSetup (hSrio, &setup);
   if (status != CSL_SOK) {
       printf("SRIO: ... Hardware setup, failed\n");
       return;
   }

/* This routine sets a structure with the required setup of the SRIO */
void srio_Create_Setup (   CSL_SrioHwSetup *pSetup )
{
    Uint32 index;

    /* Peripheral enable */
    pSetup->perEn = 1;

    /* While in shutdown, put memories in sleep mode */
    pSetup->periCntlSetup.swMemSleepOverride = 1;

    /* Enable loopback operation */
    pSetup->periCntlSetup.loopback = 1;

    /* Boot process is over (complete) */
    pSetup->periCntlSetup.bootComplete = 1;

    /* Process TX requests of priority 2, when credit is 1 or greater */
    pSetup->periCntlSetup.txPriority2Wm = CSL_SRIO_TX_PRIORITY_WM_0;

    /* Process TX requests of priority 1, when credit is 1 or greater */
    pSetup->periCntlSetup.txPriority1Wm = CSL_SRIO_TX_PRIORITY_WM_0;

    /* Process TX requests of priority 0, when credit is 1 or greater */
    pSetup->periCntlSetup.txPriority0Wm = CSL_SRIO_TX_PRIORITY_WM_0;

    /* Set internal bus priority to 1 (next to lowest) */
    pSetup->periCntlSetup.busTransPriority = CSL_SRIO_BUS_TRANS_PRIORITY_1;

    /* UDI buffers are port based not proirity based */
    pSetup->periCntlSetup.bufferMode = CSL_SRIO_1X_MODE_PORT;

    /* DSP clock is of 333 MHz from TPCC document section 2.3 */
    pSetup->periCntlSetup.prescalar = CSL_SRIO_CLK_PRESCALE_6;

    /* Enable clocks to all domains */
    pSetup->gblEn = 1;

    /* Enable clock in each domain */
    for (index=0; index<9; index++) { /* 9 domains */
        pSetup->blkEn[index] = 1; /* Enable each of it */
    }

    /* 8-bit id is 0xAB and 16-bit id is 0xBEEF */
    pSetup->deviceId1 = SRIO_SET_DEVICE_ID(SMALL_DEV_ID, LARGE_DEV_ID);

    /* 8-bit id is 0xAB and 16-bit id is 0xBEEF for multi-cast*/
    pSetup->deviceId2 = SRIO_SET_DEVICE_ID(SMALL_DEV_ID, LARGE_DEV_ID);
```

```
/* configure the SERDES registers */

/* SERDES PLL configuration for channel 0 */
pSetup->serDesPllCfg[0].pllEnable = TRUE;
pSetup->serDesPllCfg[0].pllMplyFactor = CSL_SRIO_SERDES_PLL_MPLY_BY_12;

/* SERDES RX channel 0 enable */
pSetup->serDesRxChannelCfg[0].enRx      = TRUE;
pSetup->serDesRxChannelCfg[0].symAlign  = CSL_SRIO_SERDES_SYM_ALIGN_COMMA;
pSetup->serDesRxChannelCfg[0].los       = CSL_SRIO_SERDES_LOS_DET_HIGH_THRESHOLD;
pSetup->serDesRxChannelCfg[0].clockDataRecovery = 0x01;  /* first order */
pSetup->serDesRxChannelCfg[0].equalizer = 0x01;

/* SERDES TX channel 0 enable */
pSetup->serDesTxChannelCfg[0].enTx = TRUE;
pSetup->serDesTxChannelCfg[0].enableFixedPhase = TRUE;

/* SERDES RX channel 1 enable */
pSetup->serDesRxChannelCfg[1].enRx      = TRUE;
pSetup->serDesRxChannelCfg[1].symAlign  = CSL_SRIO_SERDES_SYM_ALIGN_COMMA;
pSetup->serDesRxChannelCfg[1].los       = CSL_SRIO_SERDES_LOS_DET_HIGH_THRESHOLD;
pSetup->serDesRxChannelCfg[1].clockDataRecovery = 0x01;  /* first order */
pSetup->serDesRxChannelCfg[1].equalizer = 0x01;

/* SERDES TX channel 1 enable */
pSetup->serDesTxChannelCfg[1].enTx = TRUE;
pSetup->serDesTxChannelCfg[1].enableFixedPhase = TRUE;

/* SERDES RX channel 2 enable */
pSetup->serDesRxChannelCfg[2].enRx      = TRUE;
pSetup->serDesRxChannelCfg[2].symAlign  = CSL_SRIO_SERDES_SYM_ALIGN_COMMA;
pSetup->serDesRxChannelCfg[2].los       = CSL_SRIO_SERDES_LOS_DET_HIGH_THRESHOLD;
pSetup->serDesRxChannelCfg[2].clockDataRecovery = 0x01;  /* first order */
pSetup->serDesRxChannelCfg[2].equalizer = 0x01;

/* SERDES TX channel 2 enable */
pSetup->serDesTxChannelCfg[2].enTx = TRUE;
pSetup->serDesTxChannelCfg[2].enableFixedPhase = TRUE;

/* SERDES RX channel 3 enable */
pSetup->serDesRxChannelCfg[3].enRx      = TRUE;
pSetup->serDesRxChannelCfg[3].symAlign  = CSL_SRIO_SERDES_SYM_ALIGN_COMMA;
pSetup->serDesRxChannelCfg[3].los       = CSL_SRIO_SERDES_LOS_DET_HIGH_THRESHOLD;
pSetup->serDesRxChannelCfg[3].clockDataRecovery = 0x01; /* first order */
pSetup->serDesRxChannelCfg[3].equalizer = 0x01;

/* SERDES TX channel 3 enable */
pSetup->serDesTxChannelCfg[3].enTx = TRUE;
pSetup->serDesTxChannelCfg[3].enableFixedPhase = TRUE;

/* Select flow control ID length 16-bit */
pSetup->flowCntlIdLen[0] = 1;

/* Destination ID of flow n, same ids as we are doing loopback */
pSetup->flowCntlId[0] = LARGE_DEV_ID;

/* Sets the number of address bits generated by the PE as a source and
 * processed by the PE as the target of an operation as 34 bits
 */
pSetup->peLlAddrCtrl = CSL_SRIO_ADDR_SELECT_34BIT;

/* Base device configuration */
pSetup->devIdSetup.smallTrBaseDevId = SMALL_DEV_ID;
pSetup->devIdSetup.largeTrBaseDevId = LARGE_DEV_ID;
pSetup->devIdSetup.hostBaseDevId = LARGE_DEV_ID;

/* Port General configuration */
pSetup->portGenSetup.portLinkTimeout = 0xFFFFF;  /* 215 ms */
pSetup->portGenSetup.portRespTimeout = 0xFFFFF;  /* 215 ms */
pSetup->portGenSetup.hostEn = 0;                 /* It is a slave */
pSetup->portGenSetup.masterEn = 1;     /* This device can issue requests */

/* Port control configuration */
pSetup->portCntlSetup[0].portDis = 0;            /* Do not disable Port 0 */
```

```
    pSetup->portCntlSetup[0].outPortEn = 1;          /* Output on Port 0 enabled */
    pSetup->portCntlSetup[0].inPortEn = 1;           /* Input on Port 0 enabled */
    pSetup->portCntlSetup[0].portWidthOverride = CSL_SRIO_PORT_WIDTH_NO_OVERRIDE; /* 4 line port
*/
    pSetup->portCntlSetup[0].errCheckDis = 0;        /* Err check enabled */
    pSetup->portCntlSetup[0].multicastRcvEn = 1;     /* MultiCast receive enabled */
    pSetup->portCntlSetup[0].stopOnPortFailEn = 1;   /* Stop on fail */
    pSetup->portCntlSetup[0].dropPktEn = 1;          /* Drop PKT */
    pSetup->portCntlSetup[0].portLockoutEn = 0;      /* Send any PKT */

    /* Enable all logical/transport errors */
    pSetup->lgclTransErrEn = CSL_SRIO_IO_ERR_RESP_ENABLE |
                             CSL_SRIO_ILL_TRANS_DECODE_ENABLE |
                             CSL_SRIO_ILL_TRANS_TARGET_ERR_ENABLE |
                             CSL_SRIO_PKT_RESP_TIMEOUT_ENABLE |
                             CSL_SRIO_UNSOLICITED_RESP_ENABLE |
                             CSL_SRIO_UNSUPPORTED_TRANS_ENABLE;

    /* Enable all Port errors */

    pSetup->portErrSetup[0].portErrRateEn = CSL_SRIO_ERR_IMP_SPECIFIC_ENABLE |
                            CSL_SRIO_CORRUPT_CNTL_SYM_ENABLE |
                            CSL_SRIO_CNTL_SYM_UNEXPECTED_ACKID_ENABLE |
                            CSL_SRIO_RCVD_PKT_NOT_ACCPT_ENABLE |
                            CSL_SRIO_PKT_UNEXPECTED_ACKID_ENABLE |
                            CSL_SRIO_RCVD_PKT_WITH_BAD_CRC_ENABLE |
                            CSL_SRIO_RCVD_PKT_OVER_276B_ENABLE |
                            CSL_SRIO_NON_OUTSTANDING_ACKID_ENABLE |
                            CSL_SRIO_PROTOCOL_ERROR_ENABLE |
                            CSL_SRIO_UNSOLICITED_ACK_CNTL_SYM_ENABLE |
                            CSL_SRIO_LINK_TIMEOUT_ENABLE;

    /* Decrement error rate counter every second */
    pSetup->portErrSetup[0].prtErrRtBias = CSL_SRIO_ERR_RATE_BIAS_1S;

    /* Allow only 2 errors after error threshold is reached */
    pSetup->portErrSetup[0].portErrRtRec = CSL_SRIO_ERR_RATE_COUNT_2;

    /* Port error setup */
    pSetup->portErrSetup[0].portErrRtFldThresh = 10;   /* Err threshold = 10 */
    pSetup->portErrSetup[0].portErrRtDegrdThresh = 10; /* Err degrade threshold = 10 */



        /* This configures the SP_IP_MODE register */
    pSetup->portIpModeSet = 0x4400003F;

    /* Configure the SP_IP_PRESCALE register assuming 333 MHz frequency */
    pSetup->portIpPrescalar = 33;

    /* Port-Write Timer. The timer defines a period to repeat sending an error
     * reporting Port-Write request for software assistance. The timer stopped
     * by software writing to the error detect registers 900 ms
     */
    pSetup->pwTimer = CSL_SRIO_PW_TIME_8;

    /* Port control independent error reporting enable. Macros can be ORed
     * to get the value
     */
    pSetup->portCntlIndpEn[0] = 0x01A20180;
}
```

A.2 LSU Programming Example

```
CSL_SrioDirectIO_ConfigXfr lsu_conf;

// This routine programs the LSU registers
void Srio_LsuSetup(CSL_SrioHandle hSrio, int *src, int *dst, int bytecnt, int type, int port, int
lsu_no){

    /* Create an LSU configuration */
    lsu_conf.srcNodeAddr           = (Uint32)&src[0];  /* Source address */
    lsu_conf.dstNodeAddr.addressHi = 0;
```

```
    lsu_conf.dstNodeAddr.addressLo = (Uint32) &dst[0]; /* Destination address */
    lsu_conf.byteCnt               = bytecnt;
    lsu_conf.idSize                = 1;                    /* 16 bit device id */
    lsu_conf.priority              = 0;                    /* PKT priority */
    lsu_conf.xambs                 = 0;                    /* Not an extended address */
    lsu_conf.dstId                 = LARGE_DEV_ID;
    lsu_conf.intrReq               = 1;                    /* 1 for interrupt request, 0 for no
interrupt */
    lsu_conf.pktType               = type;
/*N/SWRITE/NWRITE_R/NREAD/MAINT_RD/WR/Doorbell*/

    lsu_conf.hopCount              = 0;                    /* Valid for maintainance packet */
    lsu_conf.doorbellInfo          = 0;                    /* Not a doorbell pkt */
    lsu_conf.outPortId             = port;                 /* Transmit Port */

    /* configure the LSU and start transmission */
    CSL_srioLsuSetup (hSrio, &lsu_conf, lsu_no);

}
```

A.3 Message Passing Software

A.3.1 Initialization Example

```
SRIO_REGS->QUEUE_RXDMA_HDP = 0;
SRIO_REGS->QUEUE_RXDMA_HDP = 0;
SRIO_REGS->QUEUE_RXDMA_HDP = 0;
SRIO_REGS->QUEUE_RXDMA_HDP = 0;
SRIO_REGS->QUEUE_RXDMA_HDP = 0;
SRIO_REGS->QUEUE_RXDMA_HDP = 0;
SRIO_REGS->QUEUE_RXDMA_HDP = 0;
SRIO_REGS->QUEUE_RXDMA_HDP = 0;
SRIO_REGS->QUEUE_RXDMA_HDP = 0;
SRIO_REGS->QUEUE_RXDMA_HDP = 0;
SRIO_REGS->QUEUE_RXDMA_HDP = 0;
SRIO_REGS->QUEUE_RXDMA_HDP = 0;
SRIO_REGS->QUEUE_RXDMA_HDP = 0;
SRIO_REGS->QUEUE_RXDMA_HDP = 0;
SRIO_REGS->QUEUE_RXDMA_HDP = 0;
SRIO_REGS->QUEUE_RXDMA_HDP = 0;
```

A.3.2 Queue Mapping

```
SRIO_REGS->MAP[0].RXU_MAP_L =     CSL_FMK( SRIO_RXU_MAP_L_LETTER_MASK, 0)|
                CSL_FMK( SRIO_RXU_MAP_L_MAILBOX_MASK, 0x0)|
                CSL_FMK( SRIO_RXU_MAP_L_LETTER, 0)|
                CSL_FMK( SRIO_RXU_MAP_L_MAILBOX, 0)|
                CSL_FMK( SRIO_RXU_MAP_L_SOURCEID, 0xBEEF);

SRIO_REGS->MAP[0].RXU_MAP_H =     CSL_FMK( SRIO_RXU_MAP_H_TT, 1)|
                CSL_FMK( SRIO_RXU_MAP_H_QUEUE_ID, 0)|
                CSL_FMK( SRIO_RXU_MAP_H_PROMISCUOUS, 1)|
                CSL_FMK( SRIO_RXU_MAP_H_SEGMENT_MAPPING, 1); //1 multi-seg
```

A.3.3 RX Buffer Descriptor

```
BuffDescSrioRx is a data section
#define MESSNUM 16

CSL_SrioBuffDesc *pDescBaseRx = (CSL_SrioBuffDesc *)buffDescSrioRx;
CSL_SrioBuffDesc *CurrPtr, *NextBuffAddr;

Uint32    *RxBuffAddr[16] = {   rcvBuff1, rcvBuff2, rcvBuff3, rcvBuff4,
                                rcvBuff5, rcvBuff6, rcvBuff7, rcvBuff8,
                                rcvBuff9, rcvBuff10, rcvBuff11, rcvBuff12,
                                rcvBuff13, rcvBuff14, rcvBuff15, rcvBuff16 };


        // Program Receive desc
        for ( i = 0; i<MESSNUM;i++) {
                  CurrPtr = pDescBaseRx + i;
```

```
                    if(i < (MESSNUM - 1))
                    NextBuffAddr = pDescBaseRx + (i+1);
                    else
                    NextBuffAddr = 0;

                            SetupRxDesc(CurrPtr, NextBuffAddr, RxBuffAddr[i]);
                    }

//This routine setup the Receive Descriptors
void SetupRxDesc(CSL_SrioBuffDesc *CurrPtr, CSL_SrioBuffDesc *NextPtr, int *BuffPtr) {

        CurrPtr->nextDescPtr    =       (int) NextPtr;
        CurrPtr->buffPtr        =       (int )BuffPtr;
        CurrPtr->opt2           =       CSL_FMK( SRIO_RXBUFFDESC_SOP,1 )|
                                        CSL_FMK( SRIO_RXBUFFDESC_EOP,1 )|
                                        CSL_FMK( SRIO_RXBUFFDESC_OWNERSHIP,1 )|
                                        CSL_FMK( SRIO_RXBUFFDESC_EOQ,1 )|
                                        CSL_FMK( SRIO_RXBUFFDESC_TEARDOWN,0 );
}
```

A.3.4 TX Buffer Description

```
BuffDescSrioTx is a data section

CSL_SrioBuffDesc *pDescBaseTx = (CSL_SrioBuffDesc *) buffDescSrioTx;
CSL_SrioBuffDesc *CurrPtr, *NextBuffAddr;

Uint32 *TxBuffAddr[16] = {    xmtBuff1, xmtBuff2, xmtBuff3, xmtBuff4,
                              xmtBuff5, xmtBuff6, xmtBuff7, xmtBuff8,
                              xmtBuff9, xmtBuff10, xmtBuff11, xmtBuff12,
                              xmtBuff13, xmtBuff14, xmtBuff15, xmtBuff16};

for ( i = 0; i<MESSNUM; i++) {
          CurrPtr = pDescBaseTx + i;
          if(i < (MESSNUM - 1))
                        NextBuffAddr = pDescBaseTx + (i+1);
          else
                        NextBuffAddr = 0;

          SetupTxDesc (CurrPtr, NextBuffAddr, TxBuffAddr[i], port, mailbox);
          mailbox++;
}

//This routine setup the Transmit Descriptors
void SetupTxDesc(CSL_SrioBuffDesc *CurrPtr, CSL_SrioBuffDesc *NextPtr, int *BuffPtr, int port,
int mailbox) {

        CurrPtr->nextDescPtr =  (int) NextPtr;
        CurrPtr->buffPtr     =  (int) BuffPtr;
        CurrPtr->opt1        =  CSL_FMK( SRIO_TXBUFFDESC_DEST_ID, 0xBEEF) |
                                CSL_FMK( SRIO_TXBUFFDESC_PRI, 0)|
                                CSL_FMK( SRIO_TXBUFFDESC_TT, 1)|
                                CSL_FMK( SRIO_TXBUFFDESC_PORT_ID, port)|
                                CSL_FMK( SRIO_TXBUFFDESC_SSIZE, SEGSIZE )|
                                CSL_FMK( SRIO_TXBUFFDESC_MAILBOX, mailbox);
        CurrPtr->opt2        =  CSL_FMK( SRIO_TXBUFFDESC_SOP,1 ) |
                                CSL_FMK( SRIO_TXBUFFDESC_EOP,1 )|
                                CSL_FMK( SRIO_TXBUFFDESC_OWNERSHIP,1 )|
                                CSL_FMK( SRIO_TXBUFFDESC_EOQ,1 )|
                                CSL_FMK( SRIO_TXBUFFDESC_TEARDOWN,0 )|
                                CSL_FMK( SRIO_TXBUFFDESC_RETRY_COUNT,0 )
                                CSL_FMK( SRIO_TXBUFFDESC_CC,0 )|
                                CSL_FMK( SRIO_TXBUFFDESC_MESSAGE_LENGTH,MESSLEN);
}
```

A.3.5 Start Message Passing

```
SRIO_REGS->QUEUE_RXDMA_HDP [0] = (int) pDescBaseRx;
SRIO_REGS->QUEUE_TXDMA_HDP [0] = (int) pDescBaseTx;
```

A.4 Interrupt Handling

```
// Read the ICS register
```

```
CSL_srioGetLsuIntrStat(hSrio, &data);

// check transaction complete bits of LSU 0,1,2 and 3
if((data & ( CSL_FMK(SRIO_LSU_ICSR_ICS0, 1))) || (data & ( CSL_FMK(SRIO_LSU_ICSR_ICS8, 1))) ||
(data & ( CSL_FMK(SRIO_LSU_ICSR_ICS16, 1))) || (data & ( CSL_FMK(SRIO_LSU_ICSR_ICS24, 1))))

        {
        IntCount0++;
        }
        SET_TRACE (0x7aaa);

   // Clear the ICS Register by writing ICC Register
     data = CSL_SRIO_LSU_INTR0 | CSL_SRIO_LSU_INTR8 |
CSL_SRIO_LSU_INTR16 | CSL_SRIO_LSU_INTR24;
     CSL_SrioLsuIntrClear (hSrio, data);

    /* Program Rate control register */
     data = 0x1;
     CSL_SrioSetIntdstRateCntl (hSrio, data);
```

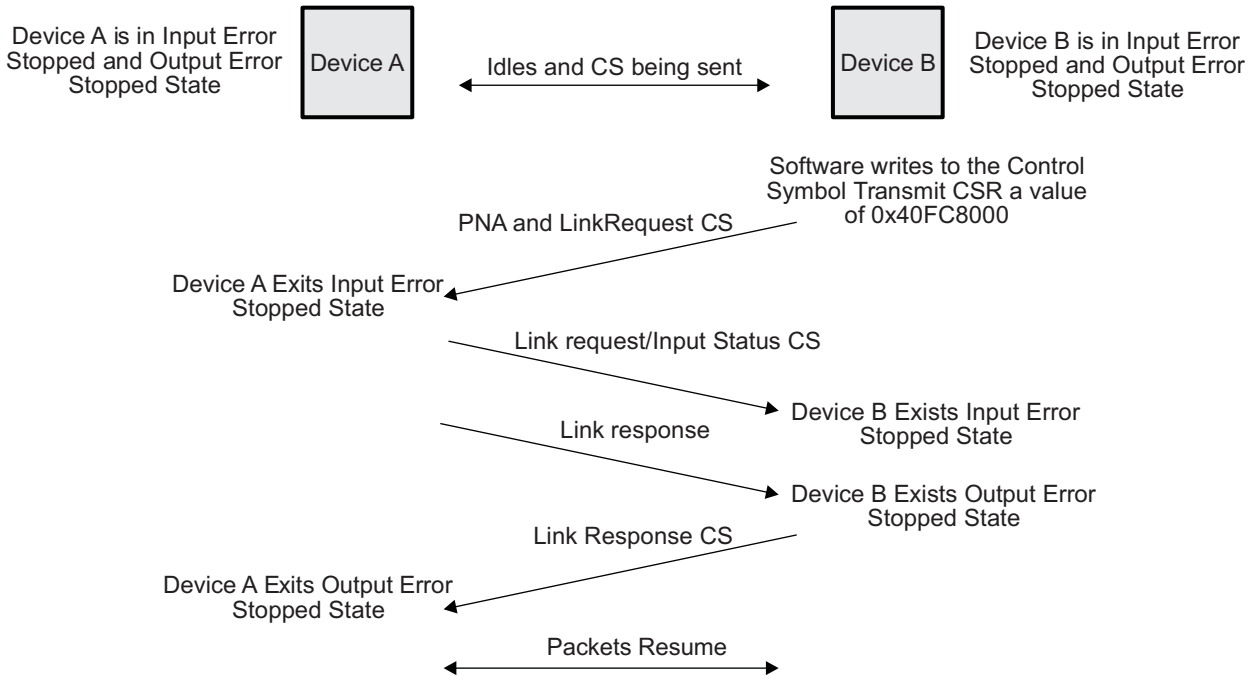
Appendix B Software-Assisted Error Recovery

After link initialization, the typical sequence is for the DSP software to check the PORT_OK bit in the SP(n)_ERR_STAT register before attempting to send packets. Without the PORT_OK bit being set, the two link partners are not initialized and are not able to communicate at all. Additionally, before sending packets it is recommended to check for error conditions and clear any error status bits in the SP(n)_ERR_STAT and SP(n)_ERR_DET registers. Most of these error-indication bits are simply writable to clear as described in their corresponding bit definitions. In some cases after reset, it has been observed that a given link partner may experience temporary bit errors resulting in OUTPUT ERROR-STOPPED and/or INPUT ERROR-STOPPED conditions. In these cases, the hardware may or may not recover from this condition depending on the states involved, severity, and location of the bit errors in the transmitted stream. The software can be used to immediately recover from the input and output error stopped states. It is recommended to add the following step after initialization and before trying to send any packets. This step can be added regardless of whether the device is currently in the output or input error-stopped states.

- The software writes a value of 0x40FC8000 into the local Port n Control Symbol Transmit register (SP(n)_CS_TX).

This software-initiated sequence immediately recovers both ends of the link from both input and output error-stopped conditions. Writing a value of 0x40FC8000 into the Port n Control Symbol Transmit register causes a PNA and link request to be sent in the Stype 0 and 1 fields of the control symbol to the link partner. The PNA causes the link partner to issue a link request to the DSP and the link request causes the link partner to issue a link response. The DSP receiving the link request issues a link response. Figure 163 illustrates the worst-case situation where both device A and B are both in input and output error-stopped states. Note that it is only required to issue the Port n Control Symbol generation for one end of the link.

Figure 163. Software Error Recovery Sequence



The sequence shown in Figure 163 causes both devices to exit all input and output error stopped states. However, it does not align ACKIDs, which need to be synchronized before data or maintenance packets are sent between the link partners. If both link partners are coming out of reset, then the ACKIDs are already aligned and no additional steps are needed. If not, then additional steps must be immediately taken to align ACKIDs. TI's DSP supports the software error recovery registers that make this process straightforward. The following steps can be used to align the ACKIDs if both link partners support these registers:

1. After writing, the SP(n)_CS_TX register = 0x40fc8000 to exit the error states.
2. Each device's SP(n)_LM_RESP register contains the link response data from Step 1. This data indicates the attached link partner's expected inbound ACKID value.
3. The DSP software reads the SP(n)_LM_RESP register and copies the expected partner's inbound ACKID to its own outbound and outstanding ACKID values in the local SP(n)_ACKID_STAT register.
4. The DSP then sends a maintenance packet to the link partner's SP(n)_ACKID_STAT register to program the ACKIDs to match expected values of the DSP. When the maintenance packet is sent, it overwrites all the fields of this register, so the value written should be: outstanding = outbound = DSP's expected inbound ACKID value, and the inbound = 1 + ACKID from Step 3.
5. ACKIDs are now aligned and data packets can be exchanged normally.

Appendix C Revision History

This revision history highlights the technical changes made to the document in this revision.

Table 195. SRIO Revision History

See	Additions/Modifications/Deletions
Table 183	Corrected equation in PRESCALE Field Description

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Audio	www.ti.com/audio	Communications and Telecom	www.ti.com/communications
Amplifiers	amplifier.ti.com	Computers and Peripherals	www.ti.com/computers
Data Converters	dataconverter.ti.com	Consumer Electronics	www.ti.com/consumer-apps
DLP® Products	www.dlp.com	Energy and Lighting	www.ti.com/energy
DSP	dsp.ti.com	Industrial	www.ti.com/industrial
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Security	www.ti.com/security
Logic	logic.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Power Mgmt	power.ti.com	Transportation and Automotive	www.ti.com/automotive
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com	Wireless	www.ti.com/wireless-apps
RF/IF and ZigBee® Solutions	www.ti.com/lprf		
	TI E2E Community Home Page		e2e.ti.com

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265